Sergio Jara Reynoso

Abstract

The purpose of my project was to make my dorm room livelier. My dorm was very dark because it faced sunset and there were other buildings in front of my windows, so sunlight was scarce. I used LEDs connect to a real-time clock module to turn on in the mornings in bright yellow (sunrise), turn off after a few minutes, and then turn on again in the evening in a orange/red tone to mimic sunset Furthermore, I really wanted to have a lightshow in my dorm that would react to music since music is always playing in my dorm. To do this, I used a sound detector module that fed an analog signal from the mic to the Arduino and then have the LEDs react to the volume of the music.

Description:

To achieve my goals, I needed five components: LED strips, sound detector module, real-time clock module, a switch, and the Arduino Uno board. First, I hooked up the LED strip to the Arduino and ran a default sketch to make sure they worked. The three pins for the LEDs are **5V, GND,** and **A2**. It turns out that FastLED and Adafruit_NeoPixel libraries both worked with my LEDs. When I connected two LED strips together, they were not turning on to the brightness of one LED, and that is when I found out that I had to use a capacitator (**CL**) of 100 μF to buffer the current drawn by the LED strips' in case it was a sudden change, which was important for the reacting to sound portion since they had to quickly turn on and off and change colors simultaneously.

When applying the sound detector module, I only needed to use the **Envelope**, **Vcc**, and **GND** pins. The envelope takes the analog signal that the microphone receives and amplifies, and sends it to the Arduino board, into Pin **A0,** and the code for the sound detecting portion takes that input and then reads it and divides it into different components, one of them being volume. If the volume is greater than a specific threshold, then the loop will go through CyclePalette() and the CycleVisual() functions. These functions use the volume information to determine brightness, color, and fade of the LEDs. That

The most difficult part of this project was getting the clock function to work with the LEDs. Originally, I was going to use the millis() function of the Arduino and calculate the time I wanted the lights to turn on in milliseconds. This VERY quickly became a problem because I would have to turn off my computer or move around and that would reset the millis() function and I would need to recalculate the time in order to test it. Furthermore, it was very difficult to write if statements with delay() when I depended on the millis() to continue. I eventually decided to buy a real-time clock module as it would make the entire process a lot easier.

When I received the real-time clock module, another issue arose. I connected the Real-time clock module pins **Vcc, GND, SDA, and SCL** to **5v, GND,** SDA, and SCL. Notice how those last two were not bolded. I spent three days writing different if statements in order to read the clock module's time and make the LEDs turn on at a specific time. For some reason, every time I set the clock to go, the first four LEDs would turn white and stay that way until I disconnected the LEDs. The Serial Monitor would print out the time properly, but the lights were not doing what they should be doing. Furthermore, when I told the program to print out text at certain time, it would print it out just fine. That is when I thought it might be the Arduino board that was the problem. While talking with Darrell, we thought it might be that there is a conflict with the **Adafruit_NeoPixel** library so I started using the **FastLED** library (which is why I have both in my final code). That still did not fix the problem. Then, Darrell discovered that the SDA and SCL pins from the clock had to be connected to pins **A4** and **A5** respectively; however, that is where I originally had the LEDs connected! So, I had to change the pin for the LEDs to **A2** and then connect the SDA and SCL pins to **A4** and **A5** and the lights worked with the clock! When working on fading the LEDs and changing the colors within the time period, I had to use delays that would add up to N.01, where the N equaled the time period allotted. I am still not entirely sure why that worked, but the LEDs were turning on at the times that I wanted them to turn on and turning off when I wanted them to turn off.

Finally, I added a switch, **SF,** that would switch between the clock function and the sound detector function with a flick of a switch. I originally thought about using a button but after talking with Alex, debouncing seemed to a bit too much work. I found a switch in the ES50 lab bags that we received and started playing with it. I had no idea how to use it until Anisha explained that when the switch is flicked towards the ground pin and the information pin, it is HIGH, while if it is on the opposite side, it is LOW so no current will run through the DATA pin. With that information, I stated that if the pin was **HIGH,** to turn the LEDs blue and if it was **LOW**, then the LEDs should be red. This seemed to work just fine, so I had all the different components of my project, I just had to put them together.

I will admit, that I am not much of a coder so although there probably are ways to simplify my code, I just put all three different codes together based on what I wanted. That is why there are multiple LED libraries. Another thing that I would change about my code is to not use delay() because when the LEDs are going through the sunrise/sunset loop, the switch does not affect the circuit. It has to go through the entire loop and all the delays, which can be a bit annoying if I want to change the function right away.

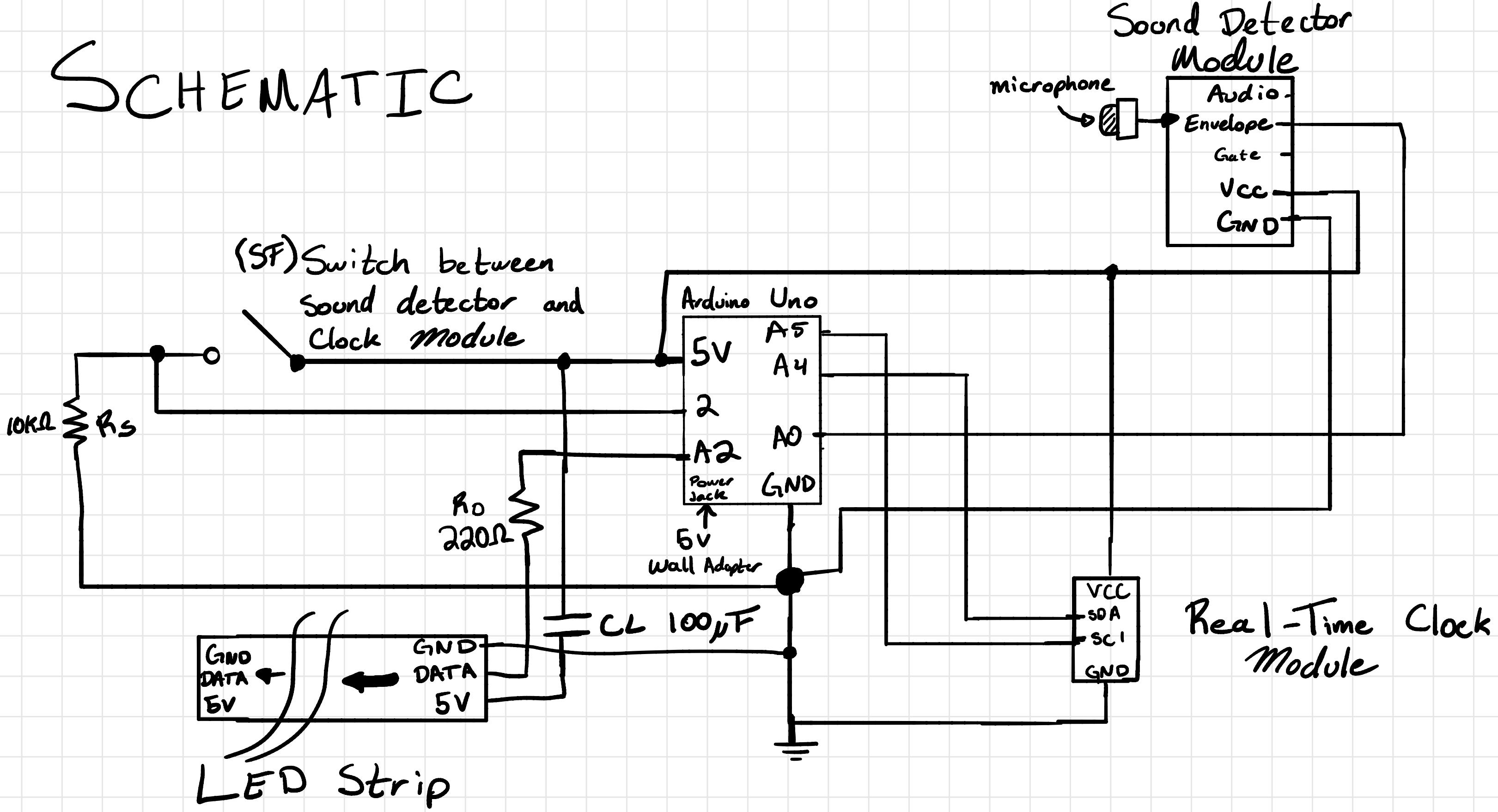
Additional Sources:

Hooking up LEDs
https://www.digikey.com/en/maker/projects/apa102-addressable-led-hookup-guide/093dc573bcf2487ab2146cec7fd870b3

Working with LEDs and Real-Time Clock
https://www.instructables.com/id/DIY-Bed-LEDs-Time-and-Motion-Activated-Video-Tutor/

Sound Reactive -
https://create.arduino.cc/projecthub/buzzandy/music-reactive-led-strip-5645ed

SCHEMATIC


Sound Detector Module
microphone
Audio
Envelope
Gate
Vcc
GND
(SF) Switch between Sound detector and Clock module
Arduino Uno
5V
2
A2
Power Jack
A5
A4
A0
GND
5V Wall Adapter
10KΩ
Rs
Ro
220Ω
CL 100μF
GND
DATA
5V
GND
DATA
5V
LED Strip
VCC
SDA
SCl
GND
Real-Time Clock Module

```
/*  ONLY THE FIRST 8 PAGES ARE MY WORK - Only need to look at the first 8 pages.
I either modified or wrote that code.
    There is a very distinct part in which the code is no longer mine, it is from
one of the links given in the report.

    No need to look beyond page 8 if you want to see only my work.

   This program has two main functions. The first function allows the LEDs to
react to the volume of sound.
    The second function acts almost as an alarm clock with lights. At the
inputted time, the lights will turn on bright yellow
    and get brighter and brighter. Then it fades out and eventually turns off so
the user doesn't have to unplug the lights or turn
    them on.

    These functions are switched between using a single switch that the program
reads as HIGH or LOW and then from there decides what
    function the program should use. On the serial monitor, the time is
constantly printed regardless. I used this for debugging.

    IMPORTANT TO NOTE: Because delay() is used, once a loop starts for the clock
function, it must go all the way through or else
    it will not switch function.
*/


#include <Adafruit_NeoPixel.h>
#include <FastLED.h>
#include <Wire.h>

#ifdef __AVR__
#include <avr/power.h>
#endif
//Constants (change these as necessary)
#define LED_PIN   A2  //Pin for the pixel strand. Can be analog or digital.
#define NUM_LEDS 60  //Change this to the number of LEDs in your strand.
#define LED_HALF  NUM_LEDS/2
#define VISUALS   6   //Change this accordingly if you add/remove a visual in the
switch-case in Visualize()
#define AUDIO_PIN A0  //Pin for the envelope of the sound detector
#define KNOB_PIN  A1  //Pin for the trimpot 10K
#define BUTTON_1  6   //Button 1 cycles color palettes
#define BUTTON_2  5   //Button 2 cycles visualization modes
#define BUTTON_3  4   //Button 3 toggles shuffle mode (automated changing of
color and visual)
```

```
CRGB leds[NUM_LEDS];
int switch_pin = 2;


Adafruit_NeoPixel strand = Adafruit_NeoPixel(NUM_LEDS, LED_PIN, NEO_GRB +
NEO_KHZ800);  //LED strand objetcs

uint16_t gradient = 0; //Used to iterate and loop through each color palette
gradually

uint16_t thresholds[] = {1529, 1019, 764, 764, 764, 1274};

uint8_t palette = 0;
uint8_t visual = 0;
uint8_t volume = 0;
uint8_t last = 0;

float maxVol = 15;
float knob = 1023.0;
float avgBump = 0;
float avgVol = 0;
float shuffleTime = 0;


bool shuffle = false;
bool bump = false;

int8_t pos[NUM_LEDS] = { -2};
uint8_t rgb[NUM_LEDS][3] = {0};


bool right = false;
int8_t dotPos = 0;
float timeBump = 0;
float avgTime = 0;

//Clock Library and Constants

#include "RTClib.h"
bool ledState = false;  //stores whether the LEDs are on or off
RTC_DS3231 rtc;
int t ;
int m;
Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUM_LEDS, LED_PIN, NEO_GRB +
NEO_KHZ800);
```

```
//This is where you can set on and off hours for AM and PM


const int onHourAM = 7, onMinuteAM = 0, offHourAM = 22, offMinuteAM = 0;
const int onHourPM = 7, onMinutePM = 0, offHourPM = 22, offMinutePM = 0;








void setup() {

  Serial.begin(9600);

  //Defines the buttons pins to be input.
  pinMode(BUTTON_1, INPUT); pinMode(BUTTON_2, INPUT); pinMode(BUTTON_3, INPUT);

  //Write a "HIGH" value to the button pins.
  digitalWrite(BUTTON_1, HIGH); digitalWrite(BUTTON_2, HIGH);
digitalWrite(BUTTON_3, HIGH);

  strand.begin(); //Initialize the LED strand object.
  strand.show();  //Show a blank strand, just to get the LED's ready for use.

  pinMode(switch_pin, INPUT);

  FastLED.addLeds<WS2812, LED_PIN, GRB>(leds, NUM_LEDS);

  //clock
  strip.begin();
  strip.setBrightness(64);
  strip.show();

  //clock stuff
  Serial.begin(9600);

  if (!rtc.begin()) {
    Serial.println("Couldn't find RTC");
    while (1);
  }
  rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));


}



void loop() {  //This is where the magic happens. This loop produces each frame
of the visual.

  DateTime now = rtc.now();   //read time of day from RTC
  printTime();  //prints out the current time: helpful for debugging

  t = now.hour();

  m = now.minute();





  if (digitalRead(switch_pin) == HIGH) { //Check for the switch to be on high.
    volume = analogRead(AUDIO_PIN);       //Record the volume level from the
sound detector
    knob = analogRead(KNOB_PIN) / 1023.0; //Record how far the trimpot is twisted

    if (volume < avgVol / 2.0 || volume < 15) volume = 0;

    else avgVol = (avgVol + volume) / 2.0; //If non-zeo, take an "average" of
volumes.

    if (volume > maxVol) maxVol = volume;



    CyclePalette();  //Changes palette for shuffle mode or button press.

    CycleVisual();   //Changes visualization for shuffle mode or button press.

    ToggleShuffle(); //Toggles shuffle mode. Delete this if you didn't use
```

```
buttons.


    if (gradient > thresholds[palette]) {
      gradient %= thresholds[palette] + 1;


      maxVol = (maxVol + volume) / 2.0;
    }


    if (volume - last > 10) avgBump = (avgBump + (volume - last)) / 2.0;


    bump = (volume - last > avgBump * .9);


    if (bump) {
      avgTime = (((millis() / 1000.0) - timeBump) + avgTime) / 2.0;
      timeBump = millis() / 1000.0;
    }

    Visualize();   //Calls the appropriate visualization to be displayed with the
globals as they are.

    gradient++;    //Increments gradient

    last = volume; //Records current volume for next pass

    delay(30);     //Paces visuals so they aren't too fast to be enjoyable
  }

  if (digitalRead(switch_pin) == LOW) { //checks for switch to be low.


    if (((t == onHourAM) && (m >= onMinuteAM) && ( m < onMinuteAM + 2)) || ((t ==
onHourPM) && (m >= onMinutePM) && ( m < onMinutePM + 2)))
    {


      colorWipe(strip.Color(150,   150,   0)); // Yellow, fade

      delay(60000 * .25);

      colorWipe(strip.Color(200,   200,   0)); // yellow dimm
      delay(60000 * .25);

      colorWipe(strip.Color(255,   255,   0)); // Yellow Bright

      delay(60000 * 1.1);

      colorWipe(strip.Color(200,   200,   0)); // Yellow Dimm

      delay(60000 * .25);

      colorWipe(strip.Color(100,   100,   0)); // Yellow, dimmest

      delay(60000 * .25);

    }

    colorWipe(strip.Color(0,   0,   0)); //off

    if (((t == offHourPM) && (m >= offMinutePM) && ( m < offMinutePM + 2)) || ((t == offHourAM) && (m >= offMinuteAM) && ( m < onMinuteAM + 2))) //if time is in the middle of the day - fade off lights, switch off motionStaate, switch on offState
    {


      colorWipe(strip.Color(242,   195,   9)); // Brightest Yellow

      delay(60000 * .25);

      colorWipe(strip.Color(242,   126,   9)); // Bright yellow

      delay(60000 * .25);

      colorWipe(strip.Color(226,   72,   6)); // Bright Yellow

      delay(60000 * .51);

      colorWipe(strip.Color(189,   57,   0)); // Brighter Orange

      delay(60000 * .25);

      colorWipe(strip.Color(142,   43,   0)); // Orange

      delay(60000 * .25);
      colorWipe(strip.Color(100,   0,   0)); // Red

      delay(60000 * .25);

      colorWipe(strip.Color(10,   0,   0)); // Red Dimm

      delay(60000 * .25);




    }
    colorWipe(strip.Color(0,   0,   0)); //off

  }
}


//clock functions to set colors



void printTime()
{
  DateTime now = rtc.now();
  Serial.print(now.hour(), DEC);
  Serial.print(':');
  Serial.print(now.minute(), DEC);
  Serial.print(':');
  Serial.println(now.second(), DEC);
}
//  for (int i = NUM_LEDS; i >= 0; i--) {
//    leds[i] = CRGB ( 255, 0, 0);
//    FastLED.show();
//    delay(40);
//  }


void colorWipe(uint32_t color) {
  for (int i = 0; i < NUM_LEDS; i++) { // For each pixel in strip...
    strip.setPixelColor(i, color);         //  Set pixel's color (in RAM)
    strip.show();                          //  Update strip to match
    //    delay(wait);                           //  Pause for a moment
  }
```

```
}
```

```
// I did not write any of the code below, was part of the sketch that I used!
```

```
//This function calls the appropriate visualization based on the value of "visual"
void Visualize() {
  switch (visual) {
    case 0: return Pulse();
    case 1: return PalettePulse();
    case 2: return Traffic();
    case 3: return Snake();
    case 4: return PaletteDance();
    case 5: return Glitter();
    case 6: return Paintball();
    default: return Pulse();
  }
}

///////////////////////////////////////////////////////////////////////////////////
////////////////////////
//NOTE: The strand displays RGB values as a 32 bit unsigned integer (uint32_t),
which is why ColorPalette()
//      and all associated color functions' return types are uint32_t. This value
is a composite of 3
//      unsigned 8bit integer (uint8_t) values (0-255 for each of red, blue, and
green). You'll notice the
//      function split() (listed below) is used to dissect these 8bit values from
the 32-bit color value.
///////////////////////////////////////////////////////////////////////////////////
////////////////////////



//This function calls the appropriate color palette based on "palette"
//  If a negative value is passed, returns the appropriate palette withe
"gradient" passed.
//  Otherwise returns the color palette with the passed value (useful for fitting
a whole palette on the strand).
uint32_t ColorPalette(float num) {
  switch (palette) {
    case 0: return (num < 0) ? Rainbow(gradient) : Rainbow(num);
    case 1: return (num < 0) ? Sunset(gradient) : Sunset(num);
    case 2: return (num < 0) ? Ocean(gradient) : Ocean(num);
    case 3: return (num < 0) ? PinaColada(gradient) : PinaColada(num);
    case 4: return (num < 0) ? Sulfur(gradient) : Sulfur(num);
    case 5: return (num < 0) ? NoGreen(gradient) : NoGreen(num);
    default: return Rainbow(gradient);
  }
}

///////////////////////////////////////////////////////////////////////////////////
/////////////////////////////
//NOTE: All of these visualizations feature some aspect that affects brightness
based on the volume relative to
//      maxVol, so that louder = brighter. Initially, I did simple proportions
(volume/maxvol), but I found this
//      to be visually indistinct. I then tried an exponential method (raising
the value to the power of
```

```
//      volume/maxvol). While this was more visually satisfying, I've opted for a
balance between the two. You'll
//      notice something like pow(volume/maxVol, 2.0) in the functions below.
This simply squares the ratio of
//      volume to maxVol to get a more exponential curve, but not as exaggerated
as an actual exponential curve.
//      In essence, this makes louder volumes brighter, and lower volumes dimmer,
to be more visually distinct.
////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////



//PULSE
//Pulse from center of the strand
void Pulse() {

  fade(0.75);   //Listed below, this function simply dims the colors a little bit
each pass of loop()

  //Advances the palette to the next noticeable color if there is a "bump"
  if (bump) gradient += thresholds[palette] / 24;

  //If it's silent, we want the fade effect to take over, hence this if-statement
  if (volume > 0) {
    uint32_t col = ColorPalette(-1); //Our retrieved 32-bit color

    //These variables determine where to start and end the pulse since it starts
from the middle of the strand.
    //  The quantities are stored in variables so they only have to be computed
once (plus we use them in the loop).
    int start = LED_HALF - (LED_HALF * (volume / maxVol));
    int finish = LED_HALF + (LED_HALF * (volume / maxVol)) + strand.numPixels() %
2;
    //Listed above, LED_HALF is simply half the number of LEDs on your strand. ↑
this part adjusts for an odd quantity.

    for (int i = start; i < finish; i++) {

      //"damp" creates the fade effect of being dimmer the farther the pixel is
from the center of the strand.
      //  It returns a value between 0 and 1 that peaks at 1 at the center of the
strand and 0 at the ends.
      float damp = sin((i - start) * PI / float(finish - start));

      //Squaring damp creates more distinctive brightness.
      damp = pow(damp, 2.0);

      //Fetch the color at the current pixel so we can see if it's dim enough to
overwrite.
      uint32_t col2 = strand.getPixelColor(i);

      //Takes advantage of one for loop to do the following:
      // Appropriatley adjust the brightness of this pixel using location,
volume, and "knob"
      // Take the average RGB value of the intended color and the existing color,
for comparison
      uint8_t colors[3];
      float avgCol = 0, avgCol2 = 0;
      for (int k = 0; k < 3; k++) {
        colors[k] = split(col, k) * damp * knob * pow(volume / maxVol, 2);
        avgCol += colors[k];
        avgCol2 += split(col2, k);
      }
      avgCol /= 3.0, avgCol2 /= 3.0;

      //Compare the average colors as "brightness". Only overwrite dim colors so
the fade effect is more apparent.
      if (avgCol > avgCol2) strand.setPixelColor(i, strand.Color(colors[0],
colors[1], colors[2]));
    }
  }
  //This command actually shows the lights. If you make a new visualization,
don't forget this!
  strand.show();
}



//PALETTEPULSE
//Same as Pulse(), but colored the entire pallet instead of one solid color
void PalettePulse() {
  fade(0.75);
  if (bump) gradient += thresholds[palette] / 24;
  if (volume > 0) {
    int start = LED_HALF - (LED_HALF * (volume / maxVol));
    int finish = LED_HALF + (LED_HALF * (volume / maxVol)) + strand.numPixels() %
2;
    for (int i = start; i < finish; i++) {
      float damp = sin((i - start) * PI / float(finish - start));
      damp = pow(damp, 2.0);
      //This is the only difference from Pulse(). The color for each pixel isn't
the same, but rather the
      //  entire gradient fitted to the spread of the pulse, with some shifting
from "gradient".
      int val = thresholds[palette] * (i - start) / (finish - start);
      val += gradient;
      uint32_t col = ColorPalette(val);

      uint32_t col2 = strand.getPixelColor(i);
      uint8_t colors[3];
      float avgCol = 0, avgCol2 = 0;
      for (int k = 0; k < 3; k++) {
        colors[k] = split(col, k) * damp * knob * pow(volume / maxVol, 2);
        avgCol += colors[k];
        avgCol2 += split(col2, k);
      }
      avgCol /= 3.0, avgCol2 /= 3.0;
      if (avgCol > avgCol2) strand.setPixelColor(i, strand.Color(colors[0],
colors[1], colors[2]));
    }
  }
  strand.show();
}


//TRAFFIC
//Dots racing into each other
void Traffic() {

  //fade() actually creates the trail behind each dot here, so it's important to
include.
  fade(0.8);

  //Create a dot to be displayed if a bump is detected.
  if (bump) {

    //This mess simply checks if there is an open position (-2) in the pos[]
array.
    int8_t slot = 0;
    for (slot; slot < sizeof(pos); slot++) {
      if (pos[slot] < -1) break;
      else if (slot + 1 >= sizeof(pos)) {
        slot = -3;
        break;
      }
    }

    //If there is an open slot, set it to an initial position on the strand.
    if (slot != -3) {

      //Evens go right, odds go left, so evens start at 0, odds at the largest
position.
      pos[slot] = (slot % 2 == 0) ? -1 : strand.numPixels();

      //Give it a color based on the value of "gradient" during its birth.
      uint32_t col = ColorPalette(-1);
      gradient += thresholds[palette] / 24;
      for (int j = 0; j < 3; j++) {
        rgb[slot][j] = split(col, j);
      }
    }
  }

  //Again, if it's silent we want the colors to fade out.
  if (volume > 0) {

    //If there's sound, iterate each dot appropriately along the strand.
    for (int i = 0; i < sizeof(pos); i++) {

      //If a dot is -2, that means it's an open slot for another dot to take over
eventually.
      if (pos[i] < -1) continue;

      //As above, evens go right (+1) and odds go left (-1)
      pos[i] += (i % 2) ? -1 : 1;

      //Odds will reach -2 by subtraction, but if an even dot goes beyond the LED
strip, it'll be purged.
      if (pos[i] >= strand.numPixels()) pos[i] = -2;

      //Set the dot to its new position and respective color.
      //  I's old position's color will gradually fade out due to fade(), leaving
a trail behind it.
      strand.setPixelColor( pos[i], strand.Color(
                              float(rgb[i][0]) * pow(volume / maxVol, 2.0) * knob,
                              float(rgb[i][1]) * pow(volume / maxVol, 2.0) * knob,
                              float(rgb[i][2]) * pow(volume / maxVol, 2.0) * knob)
                          );
    }
  }
  strand.show(); //Again, don't forget to actually show the lights!
}



//SNAKE
//Dot sweeping back and forth to the beat
void Snake() {
  if (bump) {

    //Change color a little on a bump
    gradient += thresholds[palette] / 30;

    //Change the direction the dot is going to create the illusion of "dancing."
    right = !right;
  }

  fade(0.975); //Leave a trail behind the dot.

  uint32_t col = ColorPalette(-1); //Get the color at current "gradient."

  //The dot should only be moved if there's sound happening.
  //  Otherwise if noise starts and it's been moving, it'll appear to teleport.
  if (volume > 0) {

    //Sets the dot to appropriate color and intensity
    strand.setPixelColor(dotPos, strand.Color(
                           float(split(col, 0)) * pow(volume / maxVol, 1.5) *
knob,
                           float(split(col, 1)) * pow(volume / maxVol, 1.5) *
knob,
                           float(split(col, 2)) * pow(volume / maxVol, 1.5) *
knob)
                        );

    //This is where "avgTime" comes into play.
    //  That variable is the "average" amount of time between each "bump"
detected.
    //  So we can use that to determine how quickly the dot should move so it
matches the tempo of the music.
    //  The dot moving at normal loop speed is pretty quick, so it's the max
speed if avgTime < 0.15 seconds.
    //  Slowing it down causes the color to update, but only change position
every other amount of loops.
    if (avgTime < 0.15)                                               dotPos +=
(right) ? -1 : 1;
```

```
    else if (avgTime >= 0.15 && avgTime < 0.5 && gradient % 2 == 0)   dotPos +=
(right) ? -1 : 1;
    else if (avgTime >= 0.5 && avgTime < 1.0 && gradient % 3 == 0)    dotPos +=
(right) ? -1 : 1;
    else if (gradient % 4 == 0)                                       dotPos +=
(right) ? -1 : 1;
  }

  strand.show(); // Display the lights

  //Check if dot position is out of bounds.
  if (dotPos < 0)    dotPos = strand.numPixels() - 1;
  else if (dotPos >= strand.numPixels())  dotPos = 0;
}



//PALETTEDANCE
//Projects a whole palette which oscillates to the beat, similar to the snake but
a whole gradient instead of a dot
void PaletteDance() {
  //This is the most calculation-intensive visual, which is why it doesn't need
delayed.

  if (bump) right = !right; //Change direction of iteration on bump

  //Only show if there's sound.
  if (volume > avgVol) {

    for (int i = 0; i < strand.numPixels(); i++) {

      float sinVal = abs(sin(
                           (i + dotPos) *
                           (PI / float(strand.numPixels() / 1.25) )
                         ));
      sinVal *= sinVal;
      sinVal *= volume / maxVol;
      sinVal *= knob;

      unsigned int val = float(thresholds[palette] + 1)
                         //map takes a value between -NUM_LEDS and +NUM_LEDS and
returns one between 0 and NUM_LEDS
                         * (float(i + map(dotPos, -1 * (strand.numPixels() - 1),
strand.numPixels() - 1, 0, strand.numPixels() - 1))
                            / float(strand.numPixels()))
                         + (gradient);
      val %= thresholds[palette]; //make sure "val" is within range of the palette

      uint32_t col = ColorPalette(val); //get the color at "val"

      strand.setPixelColor(i, strand.Color(
                             float(split(col, 0))*sinVal,
                             float(split(col, 1))*sinVal,
                             float(split(col, 2))*sinVal)
                          );
    }

    //After all that, appropriately reposition "dotPos."
    dotPos += (right) ? -1 : 1;
  }

  //If there's no sound, fade.
  else  fade(0.8);

  strand.show(); //Show lights.

  //Loop "dotPos" if it goes out of bounds.
  if (dotPos < 0) dotPos = strand.numPixels() - strand.numPixels() / 6;
  else if (dotPos >= strand.numPixels() - strand.numPixels() / 6)  dotPos = 0;
}


//GLITTER
//Creates white sparkles on a color palette to the beat
void Glitter() {

  //This visual also fits a whole palette on the entire strip
  //  This just makes the palette cycle more quickly so it's more visually 
pleasing
  gradient += thresholds[palette] / 204;

  //"val" is used again as the proportional value to pass to ColorPalette() to 
fit the whole palette.
  for (int i = 0; i < strand.numPixels(); i++) {
    unsigned int val = float(thresholds[palette] + 1) *
                       (float(i) / float(strand.numPixels()))
                       + (gradient);
    val %= thresholds[palette];
    uint32_t  col = ColorPalette(val);
    //We want the sparkles to be obvious, so we dim the background color.
    strand.setPixelColor(i, strand.Color(
                           split(col, 0) / 6.0 * knob,
                           split(col, 1) / 6.0 * knob,
                           split(col, 2) / 6.0 * knob)
                        );
  }

  //Create sparkles every bump
  if (bump) {

    //Random generator needs a seed, and micros() gives a large range of values.
    //  micros() is the amount of microseconds since the program started running.
    randomSeed(micros());

    //Pick a random spot on the strand.
    dotPos = random(strand.numPixels() - 1);

    //Draw  sparkle at the random position, with appropriate brightness.
    strand.setPixelColor(dotPos, strand.Color(
                           255.0 * pow(volume / maxVol, 2.0) * knob,
                           255.0 * pow(volume / maxVol, 2.0) * knob,
                           255.0 * pow(volume / maxVol, 2.0) * knob
                         ));
  }
  bleed(dotPos);
  strand.show(); //Show the lights.
}


//PAINTBALL
//Recycles Glitter()'s random positioning; simulates "paintballs" of
//  color splattering randomly on the strand and bleeding together.
void Paintball() {

  //If it's been twice the average time for a "bump" since the last "bump," start 
fading.
  if ((millis() / 1000.0) - timeBump > avgTime * 2.0) fade(0.99);

  //Bleeds colors together. Operates similarly to fade. For more info, see its 
definition below
  bleed(dotPos);

  //Create a new paintball if there's a bump (like the sparkles in Glitter())
  if (bump) {
    //Random generator needs a seed, and micros() gives a large range of values.
    //  micros() is the amount of microseconds since the program started running.
    randomSeed(micros());

    //Pick a random spot on the strip. Random was already reseeded above, so no
real need to do it again.
    dotPos = random(strand.numPixels() - 1);

    //Grab a random color from our palette.
    uint32_t col = ColorPalette(random(thresholds[palette]));

    //Array to hold final RGB values
    uint8_t colors[3];

    //Relates brightness of the color to the relative volume and potentiometer
value.
    for (int i = 0; i < 3; i++) colors[i] = split(col, i) * pow(volume / maxVol,
2.0) * knob;

    //Splatters the "paintball" on the random position.
    strand.setPixelColor(dotPos, strand.Color(colors[0], colors[1], colors[2]));

    //This next part places a less bright version of the same color next to the
left and right of the
    //  original position, so that the bleed effect is stronger and the colors
are more vibrant.
    for (int i = 0; i < 3; i++) colors[i] *= .8;
    strand.setPixelColor(dotPos - 1, strand.Color(colors[0], colors[1],
colors[2]));
    strand.setPixelColor(dotPos + 1, strand.Color(colors[0], colors[1],
colors[2]));
  }
  strand.show(); //Show lights.
}


//////////////////////////////////////////////////////////////////////////////////
///////////////////
//DEBUG CYCLE
//No reaction to sound, merely to see gradient progression of color palettes
//NOT implemented in code as is, but is easily includable in the switch-case.
void Cycle() {
  for (int i = 0; i < strand.numPixels(); i++) {
    float val = float(thresholds[palette]) * (float(i) / float(strand.
numPixels())) + (gradient);
    val = int(val) % thresholds[palette];
    strand.setPixelColor(i, ColorPalette(val));
  }
  strand.show();
  gradient += 32;
}
//////////////////////////////////////////////////////////////////////////////////
///////////////////

//////////</Visual Functions>


//////////<Helper Functions>

void CyclePalette() {

  //IMPORTANT: Delete this whole if-block if you didn't use
buttons//////////////////////////////////

  //If a button is pushed, it sends a "false" reading
  if (!digitalRead(BUTTON_1)) {

    palette++;     //This is this button's purpose, to change the color palette.

    //If palette is larger than the population of thresholds[], start back at 0
    //  This is why it's important you add a threshold to the array if you add a
    //  palette, or the program will cylce back to Rainbow() before reaching it.
    if (palette >= sizeof(thresholds) / 2) palette = 0;

    gradient %= thresholds[palette]; //Modulate gradient to prevent any overflow
that may occur.

    //The button is close to the microphone on my setup, so the sound of pushing
it is
    //  relatively loud to the sound detector. This causes the visual to think a
loud noise
    //  happened, so the delay simply allows the sound of the button to pass
unabated.
    delay(350);

    maxVol = avgVol;  //Set max volume to average for a fresh experience.
  }

//////////////////////////////////////////////////////////////////////////////////
```

```
/////////////////

  //If shuffle mode is on, and it's been 30 seconds since the last shuffle, and
then a modulo
  //  of gradient to get a random decision between palette or visualization
shuffle
  if (shuffle && millis() / 1000.0 - shuffleTime > 30 && gradient % 2) {

    shuffleTime = millis() / 1000.0; //Record the time this shuffle happened.

    palette++;
    if (palette >= sizeof(thresholds) / 2) palette = 0;
    gradient %= thresholds[palette];
    maxVol = avgVol;  //Set the max volume to average for a fresh experience.
  }
}


void CycleVisual() {

  //IMPORTANT: Delete this whole if-block if you didn't use
buttons//////////////////////////////////
  if (!digitalRead(BUTTON_2)) {

    visual++;     //The purpose of this button: change the visual mode

    gradient = 0; //Prevent overflow

    //Resets "visual" if there are no more visuals to cycle through.
    if (visual > VISUALS) visual = 0;
    //This is why you should change "VISUALS" if you add a visual, or the program
loop over it.

    //Resets the positions of all dots to nonexistent (-2) if you cycle to the
Traffic() visual.
    if (visual == 1) memset(pos, -2, sizeof(pos));

    //Gives Snake() and PaletteDance() visuals a random starting point if cycled
to.
    if (visual == 2 || visual == 3) {
      randomSeed(analogRead(0));
      dotPos = random(strand.numPixels());
    }

    //Like before, this delay is to prevent a button press from affecting "maxVol.
```

```
"
    delay(350);

    maxVol = avgVol; //Set max volume to average for a fresh experience
  }


////////////////////////////////////////////////////////////////////////////////////
/////////////////

  //If shuffle mode is on, and it's been 30 seconds since the last shuffle, and
then a modulo
  //  of gradient WITH INVERTED LOGIC to get a random decision between what to
shuffle.
  //  This guarantees one and only one of these shuffles will occur.
  if (shuffle && millis() / 1000.0 - shuffleTime > 30 && !(gradient % 2)) {

    shuffleTime = millis() / 1000.0; //Record the time this shuffle happened.

    visual++;
    gradient = 0;
    if (visual > VISUALS) visual = 0;
    if (visual == 1) memset(pos, -2, sizeof(pos));
    if (visual == 2 || visual == 3) {
      randomSeed(analogRead(0));
      dotPos = random(strand.numPixels());
    }
    maxVol = avgVol;
  }
}


//IMPORTANT: Delete this function  if you didn't use buttons.
/////////////////////////////////////////
void ToggleShuffle() {
  if (!digitalRead(BUTTON_3)) {

    shuffle = !shuffle; //This button's purpose: toggle shuffle mode.

    //This delay is to prevent the button from taking another reading while
you're pressing it
    delay(500);

    //Reset these things for a fresh experience.
    maxVol = avgVol;
    avgBump = 0;
  }
}


//Fades lights by multiplying them by a value between 0 and 1 each pass of loop().
void fade(float damper) {

  //"damper" must be between 0 and 1, or else you'll end up brightening the
lights or doing nothing.

  for (int i = 0; i < strand.numPixels(); i++) {

    //Retrieve the color at the current position.
    uint32_t col = strand.getPixelColor(i);

    //If it's black, you can't fade that any further.
    if (col == 0) continue;

    float colors[3]; //Array of the three RGB values

    //Multiply each value by "damper"
    for (int j = 0; j < 3; j++) colors[j] = split(col, j) * damper;

    //Set the dampened colors back to their spot.
    strand.setPixelColor(i, strand.Color(colors[0] , colors[1], colors[2]));
  }
}


//"Bleeds" colors currently in the strand by averaging from a designated "Point"
void bleed(uint8_t Point) {
  for (int i = 1; i < strand.numPixels(); i++) {

    //Starts by look at the pixels left and right of "Point"
    //  then slowly works its way out
    int sides[] = {Point - i, Point + i};

    for (int i = 0; i < 2; i++) {

      //For each of Point+i and Point-i, the pixels to the left and right, plus
themselves, are averaged together.
      //  Basically, it's setting one pixel to the average of it and its
neighbors, starting on the left and right
      //  of the starting "Point," and moves to the ends of the strand
      int point = sides[i];
      uint32_t colors[] = {strand.getPixelColor(point - 1), strand.
getPixelColor(point), strand.getPixelColor(point + 1)  };

      //Sets the new average values to just the central point, not the left and
right points.
      strand.setPixelColor(point, strand.Color(
                             float( split(colors[0], 0) + split(colors[1], 0) +
split(colors[2], 0) ) / 3.0,
                             float( split(colors[0], 1) + split(colors[1], 1) +
split(colors[2], 1) ) / 3.0,
                             float( split(colors[0], 2) + split(colors[1], 2) +
split(colors[2], 2) ) / 3.0)
                          );
    }
  }
}
//As mentioned above, split() gives you one 8-bit color value
//from the composite 32-bit value that the NeoPixel deals with.
//This is accomplished with the right bit shift operator, ">>"
uint8_t split(uint32_t color, uint8_t i ) {

  //0 = Red, 1 = Green, 2 = Blue

  if (i == 0) return color >> 16;
  if (i == 1) return color >> 8;
  if (i == 2) return color >> 0;
  return -1;
}

//////////</Helper Functions>



//////////<Palette Functions>

//These functions simply take a value and return a gradient color
//  in the form of an unsigned 32-bit integer

//The gradients return a different, changing color for each multiple of 255
//  This is because the max value of any of the 3 RGB values is 255, so it's
//  an intuitive cutoff for the next color to start appearing.
//  Gradients should also loop back to their starting color so there's no jumps
in color.

uint32_t Rainbow(unsigned int i) {
  if (i > 1529) return Rainbow(i % 1530);
  if (i > 1274) return strand.Color(255, 0, 255 - (i % 255));   //violet -> red
  if (i > 1019) return strand.Color((i % 255), 0, 255);         //blue -> violet
  if (i > 764) return strand.Color(0, 255 - (i % 255), 255);    //aqua -> blue
  if (i > 509) return strand.Color(0, 255, (i % 255));          //green -> aqua
  if (i > 255) return strand.Color(255 - (i % 255), 255, 0);    //yellow -> green
  return strand.Color(255, i, 0);                               //red -> yellow
}

uint32_t Sunset(unsigned int i) {
  if (i > 1019) return Sunset(i % 1020);
  if (i > 764) return strand.Color((i % 255), 0, 255 - (i % 255));
//blue -> red
  if (i > 509) return strand.Color(255 - (i % 255), 0, 255);
//purple -> blue
  if (i > 255) return strand.Color(255, 128 - (i % 255) / 2, (i % 255));
//orange -> purple
  return strand.Color(255, i / 2, 0);                                       //red
-> orange
}

uint32_t Ocean(unsigned int i) {
  if (i > 764) return Ocean(i % 765);
  if (i > 509) return strand.Color(0, i % 255, 255 - (i % 255));  //blue -> green
  if (i > 255) return strand.Color(0, 255 - (i % 255), 255);      //aqua -> blue
  return strand.Color(0, 255, i);                                 //green -> aqua
}

uint32_t PinaColada(unsigned int i) {
  if (i > 764) return PinaColada(i % 765);
  if (i > 509) return strand.Color(255 - (i % 255) / 2, (i % 255) / 2, (i % 255)
/ 2);  //red -> half white
  if (i > 255) return strand.Color(255, 255 - (i % 255),
0);                            //yellow -> red
  return strand.Color(128 + (i / 2), 128 + (i / 2), 128 - i /
2);                       //half white -> yellow
}

uint32_t Sulfur(unsigned int i) {
  if (i > 764) return Sulfur(i % 765);
  if (i > 509) return strand.Color(i % 255, 255, 255 - (i % 255));   //aqua ->
yellow
  if (i > 255) return strand.Color(0, 255, i % 255);                 //green ->
aqua
  return strand.Color(255 - i, 255, 0);                              //yellow ->
```

```
green
}

uint32_t NoGreen(unsigned int i) {
  if (i > 1274) return NoGreen(i % 1275);
  if (i > 1019) return strand.Color(255, 0, 255 - (i % 255));         //violet ->
red
  if (i > 764) return strand.Color((i % 255), 0, 255);                //blue ->
violet
  if (i > 509) return strand.Color(0, 255 - (i % 255), 255);          //aqua ->
blue
  if (i > 255) return strand.Color(255 - (i % 255), 255, i % 255);    //yellow ->
aqua
  return strand.Color(255, i, 0);                                     //red ->
yellow
}

//NOTE: This is an example of a non-gradient palette: you will get straight red,
white, or blue
//      This works fine, but there is no gradient effect, this was merely
included as an example.
//      If you wish to include it, put it in the switch-case in ColorPalette()
and add its
//      threshold (764) to thresholds[] at the top.
uint32_t USA(unsigned int i) {
  if (i > 764) return USA(i % 765);
  if (i > 509) return strand.Color(0, 0, 255);      //blue
  if (i > 255) return strand.Color(128, 128, 128);  //white
  return strand.Color(255, 0, 0);                   //red
}

//////////</Palette Functions>
uint32_t Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if (WheelPos < 85) {
    return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  }
  if (WheelPos < 170) {
    WheelPos -= 85;
    return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
  WheelPos -= 170;
  return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}
```