adafruit learning system

Jack-o'-LED·trix

Created by Amelia Tetterton

Last updated on 2018-11-20 05:01:56 PM UTC

Guide Contents

Overview

Use a string of diffused pixels and a push button to create a new take on a jack-o-lantern sure to entertain trick-or-treaters and guests at your next Halloween party.

This is a great beginner project if you're new to soldering or microcontrollers. You can use a real pumpkin (aka "cucurbita pepo") or a fake foam pumpkin (aka a "funkin") to create a reusable and unique piece of decor.

I used an Adafruit Feather 32u4 Bluefruit LE board to drive the 3-pin pixels on my Jack-o'-LED·trix this year. The example code will compile on a number of different boards, including the following:

- Adafruit Metro (https://adafru.it/METROXMETR)
- Teensy-LC (https://adafru.it/CMU)
- Circuit Playground Express (https://adafru.it/wpF)
- Itsy Bitsy M0 (https://adafru.it/BI9)
- Pro Trinket 5V (https://adafru.it/dUf)

Adafruit Parts

2 x 12mm Diffused Thin RGB LED Pixels
You will need 48 pixels.
ADD TO CART

1 x Lithium Ion Polymer Battery
I recommend this 2500mah battery.
ADD TO CART

1 x Adafruit Feather 32u4 Bluefruit LE
Built in battery charging!

ADD TO CART

1 x Premium Male/Male Jumper Wires

Pick your favorite colors!

ADD TO CART

1 x Tactile Switch Buttons (12mm square)

You will need one switch button for this design.

ADD TO CART

Additional Parts & Materials

For this project you will need:

- Pumpkin or Funkin (https://adafru.it/CMV)
- Multi-Colored Heat Shrink (https://adafru.it/dVd)
- Hook-up Wire - 22AWG Solid Core (https://adafru.it/dya)
- Half-size breadboard (https://adafru.it/keP)
- Knife or pumpkin carving tools
- Drill + drill bits (1/8", 1/4", 7/16" & 1/2")
- Pencil
- USB cable - for programming your board & charging your battery
- Soldering station:
    - Soldering iron
    - Soldering iron stand (https://adafru.it/CMW)
    - Solder
    - Wire stripper (https://adafru.it/dDl)
    - Helping hands (https://adafru.it/dxR)
    - Hakko Brass Sponge Solder Tip Cleaner (http://adafru.it/11721)
    - Ventilation, I recommend the USB Rechargeable Mini Solder Fume Extractor (https://adafru.it/CMX) guide from Phillip Burgess (https://adafru.it/iPc)

Code

Arduino IDE

Open your Arduino IDE. If you are new to Arduino, check out Tony DiCola's (https://adafru.it/nYb) guide here: Adafruit Arduino IDE Setup (https://adafru.it/CMS)

This sketch uses the FastLED library. You can learn more about Arduino libraries by reading this guide from Adafruit (https://adafru.it/dit).

Download the FastLED library. See FastLED.io (https://adafru.it/ebm) for more information, the FastLED github (https://adafru.it/ezj), or visit their community on Google+ (https://adafru.it/ebn).

Review the code

You will need to review and update the following before verifying and uploading your code.

The type of LEDs: I used WS2811. You will need to update your sketch by changing this to WS2801 if you are using pixels from Adafruit.

The pins for your LEDs: Check the pins your LEDs are connected to. If you have 4 pin LEDs, like the WS2801 pixels from Adafruit, you will need to uncomment the line defining the *Clock_Pin.*

The color order of your LEDs: Is it RGB or GRB? This may be a bit of trial and error until you're sure.

The size of your matrix: My matrix is 8x6

The button pin: What pin is your pushbutton connected to?

Plug the Adafruit Feather into your computer with a USB cable, verify and upload the following code.

Once your code has been uploaded, set it aside.

```
/*  This is the FastLED library NoisePlusPalette example code incorporating a momentary
    push button to cycle between modes for the Jack-o-LED-trix project.

    You should wire a momentary push button to connect from ground to a digital IO pin.
    In this example, wire the button to ground and pin 11.

    FastLED has a number of built in "palettes" to choose from:
      RainbowColors_p       is all the colors of the rainbow
      PartyColors_p         is all the colors of the rainbow minus greens
      RainbowStripeColors_p is all the colors of the rainbow divided into stripes
      HeatColors_p          is reds and yellows, white, and black
      LavaColors_p          is more reds and orangey colors
      ForestColors_p        is greens and yellows
      OceanColors_p         is lots of blues and aqua colors
      CloudColors_p         is blues and white

    FastLED also provides a number of easy ways to create you own personalized palettes,
    see the "HalloweenPalette_p" to personalize your own colors.

    There are also ways to create other palettes using functions,
    see "SetupCandyCornPalette()" & "SetupBlackAndWhiteStripedPalette()" to personalize
```

```
    your own colors.
*/

#include <FastLED.h>
#include "colorutils.h"
#include "colorpalettes.h"

#define BRIGHTNESS  128

//Define the type of pixels you are using on the next line here.
//If you are using a strand or two of WS2801 pixels from Adafruit, update the chipset type to WS2801.
#define LED_TYPE       WS2811

//You may need to adjust your color order to "GRB" instead of "RGB" below.
#define COLOR_ORDER   RGB

//If you are using 4-pin LEDs, you will need to uncomment and define the CLOCK_PIN
#define DATA_PIN      6
//#define CLOCK_PIN   #

#define BUTTON_PIN   11   // button is connected to pin 11 and GND
#define UPDATES_PER_SECOND 100

//This sketch is set up for a matrix of 8 pixels wide and 6 pixels high. Update to match your matrix.
const uint8_t kMatrixWidth  = 8;
const uint8_t kMatrixHeight = 6;
const bool    kMatrixSerpentineLayout = true;

#define NUM_LEDS (kMatrixWidth * kMatrixHeight)   //no need to define the number of LEDS - here's the mat
#define MAX_DIMENSION ((kMatrixWidth>kMatrixHeight) ? kMatrixWidth : kMatrixHeight)

// The leds
CRGB leds[kMatrixWidth * kMatrixHeight];

uint8_t gHue = 0; // rotating "base color" used by many of the patterns
int ledMode = 0;

// The 16 bit version of our coordinates
static uint16_t x;
static uint16_t y;
static uint16_t z;

// We're using the x/y dimensions to map to the x/y pixels on the matrix.  We'll
// use the z-axis for "time".  speed determines how fast time moves forward.  Try
// 1 for a very slow moving effect, or 60 for something that ends up looking like
// water.
uint16_t speed = 10; // speed is set dynamically once we've started up

// Scale determines how far apart the pixels in our noise matrix are.  Try
// changing these values around to see how it affects the motion of the display.  The
// higher the value of scale, the more "zoomed out" the noise will be.  A value
// of 1 will be so zoomed in, you'll mostly see solid colors.
uint16_t scale = 50; // scale is set dynamically once we've started up

// This is the array that we keep our computed noise values in
uint8_t noise[MAX_DIMENSION][MAX_DIMENSION];

CRGBPalette16 currentPalette( RainbowColors_p );
uint8_t       colorLoop = 0;
```

```
TBlendType    currentBlending;

const TProgmemPalette16 HalloweenPalette_p PROGMEM =
{
  CRGB:: OrangeRed,
  CRGB:: Gold,
  CRGB:: Purple,
  CRGB:: Gold,

  CRGB:: OrangeRed,
  CRGB:: Gold,
  CRGB:: Purple,
  CRGB:: OrangeRed,

  CRGB:: Gold,
  CRGB:: Purple,
  CRGB:: Gold,
  CRGB:: OrangeRed,

  CRGB:: Gold,
  CRGB:: Purple,
  CRGB:: Gold,
  CRGB:: OrangeRed,
};


unsigned long keyPrevMillis = 0;
const unsigned long keySampleIntervalMs = 25;
byte longKeyPressCountMax = 80;    // 80 * 25 = 2000 ms
byte longKeyPressCount = 0;

byte prevKeyState = HIGH;         // button is active low

void setup() {
  delay(3000);

//If you are using 3-pin LEDs, uncomment the next line:
  FastLED.addLeds<LED_TYPE, DATA_PIN, COLOR_ORDER>(leds, NUM_LEDS).setCorrection(TypicalLEDStrip);

//If you are using 4-pin LEDs, uncomment the next line:
  //FastLED.addLeds<LED_TYPE, DATA_PIN, CLOCK_PIN, COLOR_ORDER>(leds, NUM_LEDS).setCorrection(TypicalLEDS

  FastLED.setBrightness(BRIGHTNESS);
  currentBlending;
  pinMode(BUTTON_PIN, INPUT_PULLUP);

  // Initialize our coordinates to some random values
  x = random16();
  y = random16();
  z = random16();
}

// Fill the x/y array of 8-bit noise values using the inoise8 function.
void fillnoise8() {
  // If we're runing at a low "speed", some 8-bit artifacts become visible
  // from frame-to-frame.  In order to reduce this, we can do some fast data-smoothing.
  // The amount of data smoothing we're doing depends on "speed".
  uint8_t dataSmoothing = 0;
  if( speed < 50) {
```

```
    dataSmoothing = 200 - (speed * 4);
  }

  for(int i = 0; i < MAX_DIMENSION; i++) {
    int ioffset = scale * i;
    for(int j = 0; j < MAX_DIMENSION; j++) {
      int joffset = scale * j;

      uint8_t data = inoise8(x + ioffset,y + joffset,z);

      // The range of the inoise8 function is roughly 16-238.
      // These two operations expand those values out to roughly 0..255
      // You can comment them out if you want the raw noise data.
      data = qsub8(data,16);
      data = qadd8(data,scale8(data,39));

      if( dataSmoothing ) {
        uint8_t olddata = noise[i][j];
        uint8_t newdata = scale8( olddata, dataSmoothing) + scale8( data, 256 - dataSmoothing);
        data = newdata;
      }

      noise[i][j] = data;
    }
  }

  z += speed;

  // apply slow drift to X and Y, just for visual variation.
  x += speed / 8;
  y -= speed / 16;
}

void mapNoiseToLEDsUsingPalette()
{
  static uint8_t ihue=0;

  for(int i = 0; i < kMatrixWidth; i++) {
    for(int j = 0; j < kMatrixHeight; j++) {
      // We use the value at the (i,j) coordinate in the noise
      // array for our brightness, and the flipped value from (j,i)
      // for our pixel's index into the color palette.

      uint8_t index = noise[j][i];
      uint8_t bri =   noise[i][j];

      // if this palette is a 'loop', add a slowly-changing base value
      if( colorLoop) {
        index += ihue;
      }

      // brighten up, as the color palette itself often contains the
      // light/dark dynamic range desired
      if( bri > 127 ) {
        bri = 255;
      } else {
        bri = dim8_raw( bri * 2);
      }

      CRGB color = ColorFromPalette( currentPalette, index, bri);
```

```
      leds[XY(i,j)] = color;
    }
  }

  ihue+=1;
}

void loop() {

  byte currKeyState = digitalRead(BUTTON_PIN);

  if ((prevKeyState == LOW) && (currKeyState == HIGH)) {
    shortKeyPress();
  }
  prevKeyState = currKeyState;

  static uint8_t startIndex = 0;
  startIndex = startIndex + 1; /* motion speed */

  // generate noise data
  fillnoise8();

  //The group of colors in a palette are sent through a strip of LEDS in speed and step increments youve
  //You can change the SPEED and STEPS to make things look exactly how you want
  //SPEED refers to how fast the colors move.
  //Try 1 for a very slow moving effect, or 60 for something that ends up looking like water.
  //SCALE refers to how zoomed in we are.
  //Try changing these values around to see how it affects the motion of the display.
  //The higher the value of scale, the more "zoomed out" the noise will be.
  //A value of 1 will be so zoomed in, you'll mostly see solid colors.

  // convert the noise data to colors in the LED array using the current palette
  mapNoiseToLEDsUsingPalette();

  switch (ledMode)  {
  case 0:
  {currentPalette = RainbowColors_p;        speed = 25; scale = 25; colorLoop = 0; }
  break;

  case 1:
  {currentPalette = HeatColors_p;           speed = 20; scale = 50; colorLoop = 2; }
  break;

  case 2:
  {SetupBlackAndWhiteStripedPalette();      speed = 20; scale = 100; colorLoop = 1; }
  break;

  case 3:
  {currentPalette = LavaColors_p;           speed = 10; scale = 25; colorLoop = 0; }
  break;

  case 4:
  {currentPalette = HalloweenPalette_p;     speed = 20; scale = 20; colorLoop = 1; }
  break;

  case 5:
  {SetupCandyCornPalette();                 speed = 15; scale = 30; colorLoop = 1; }
  break;
  }
```

```
  //FillLEDsFromPaletteColors( startIndex);
  LEDS.show();
  delay(1000/speed);
}

void shortKeyPress() {
  ledMode++;
  if (ledMode > 5) {
    ledMode=0;
  }
}

// This function sets up a palette of black and white stripes,
// using code.  Since the palette is effectively an array of
// sixteen CRGB colors, the various fill_* functions can be used
// to set them up.
void SetupBlackAndWhiteStripedPalette()
{
  // 'black out' all 16 palette entries...
  fill_solid( currentPalette, 16, CRGB::Black);
  // and set every fourth one to White or Gray.
  currentPalette[0] = CRGB::White;
  currentPalette[4] = CRGB::Gray;
  currentPalette[8] = CRGB::White;
  currentPalette[12] = CRGB::Gray;

}

// This function sets up a palette of candy corn colors
void SetupCandyCornPalette()
{
    fill_solid( currentPalette, 16, CRGB::Black);
    // set half of the LEDs to the colors selected here. This palette incorporates a lot of black
    currentPalette[0] = CRGB::Orange;
    currentPalette[1] = CRGB::Yellow;
    currentPalette[2] = CRGB::Red;
    currentPalette[3] = CRGB::OrangeRed;

    currentPalette[8] = CRGB::Yellow;
    currentPalette[9] = CRGB::OrangeRed;
    currentPalette[10] = CRGB::Orange;
    currentPalette[11] = CRGB::Red;

}

// Mark's xy coordinate mapping code.  See the XYMatrix for more information on it.
//
uint16_t XY( uint8_t x, uint8_t y)
{
  uint16_t i;
  if( kMatrixSerpentineLayout == false) {
    i = (y * kMatrixWidth) + x;
  }
  if( kMatrixSerpentineLayout == true) {
    if( y & 0x01) {
      // Odd rows run backwards
      uint8_t reverseX = (kMatrixWidth - 1) - x;
      i = (y * kMatrixWidth) + reverseX;
    } else {
```

```
      // Even rows run forwards
      i = (y * kMatrixWidth) + x;
    }
  }
  return i;
}
```

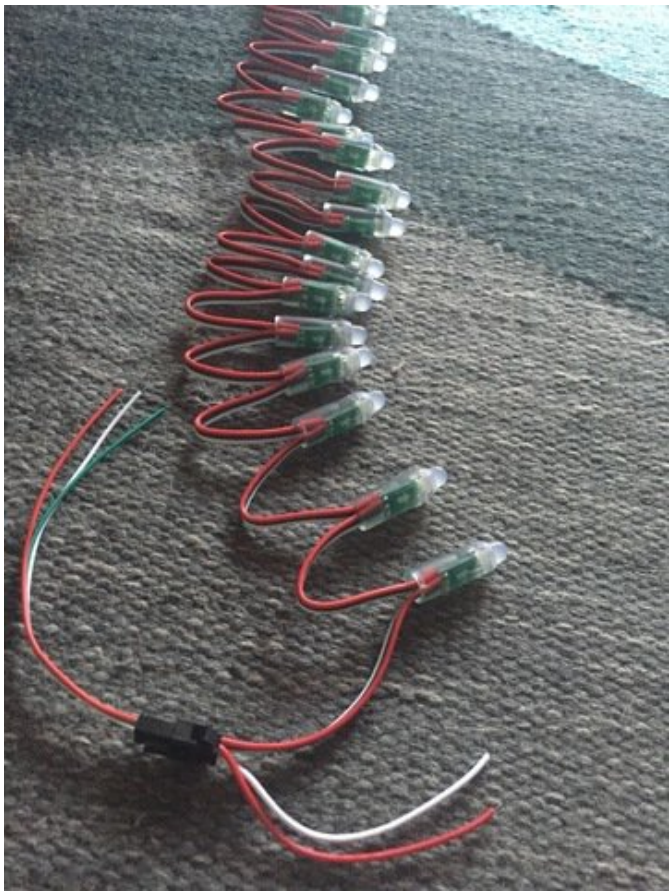
https://adafru.it/D59

https://adafru.it/D59

Electronics Prep

Before you begin this step, you should read up on the following prerequisite guides:

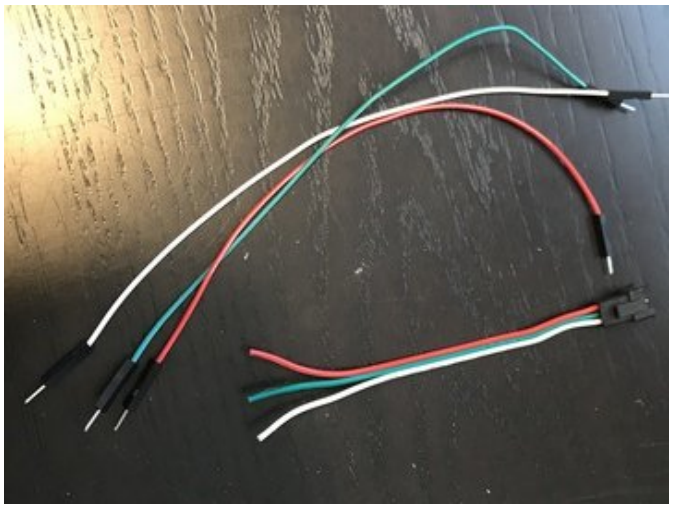
- Adafruit Guide to Excellent Soldering (https://adafru.it/drl)
- 12mm pixels guide (https://adafru.it/dgr)
- All About Batteries (https://adafru.it/tfL)
- DIY On/Off JST Switch Adapter (https://adafru.it/CMT) - the parts for creating this optional adapter are not accounted for in this guide.

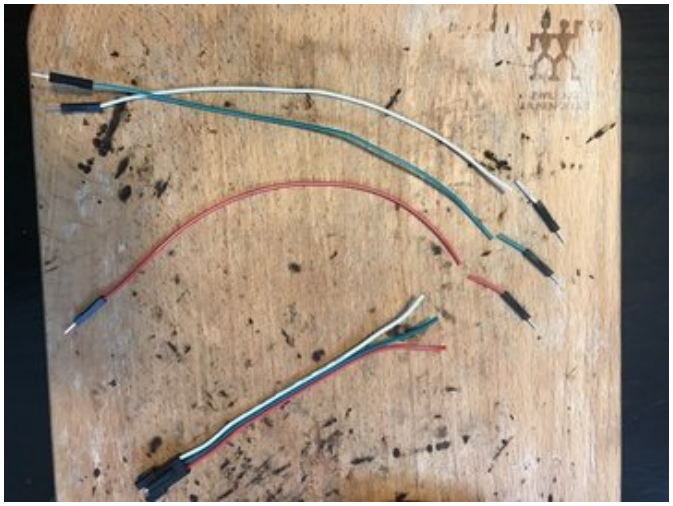


LEDs

Determine the direction the data will travel through your LEDs, marked by an arrow. On my pixels, I see an arrow pointing towards the pixels on one side. Based on this, find the initial point where the data will enter the pixels.

Find the matching 3- or 4- wire connector that fits into this side. If it's connected to the LEDs, remove it.

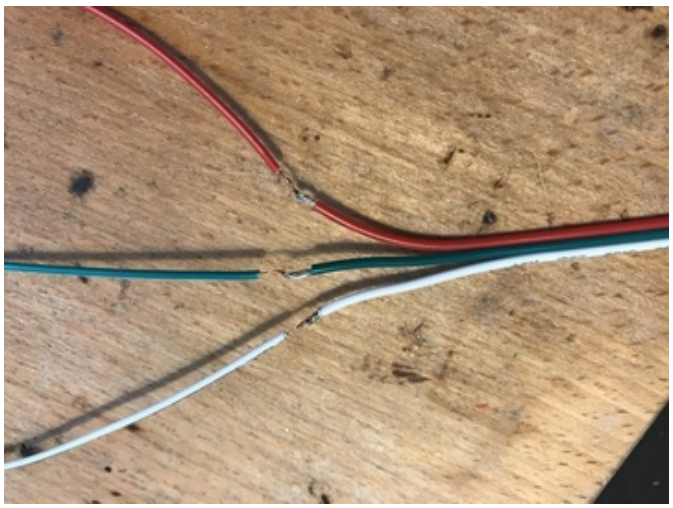




Prep

Locate 3 or 4 jumper wires and some small pieces of heat shrink. I've used matching color wire and the best possible colors I had on hand for heat shrink.

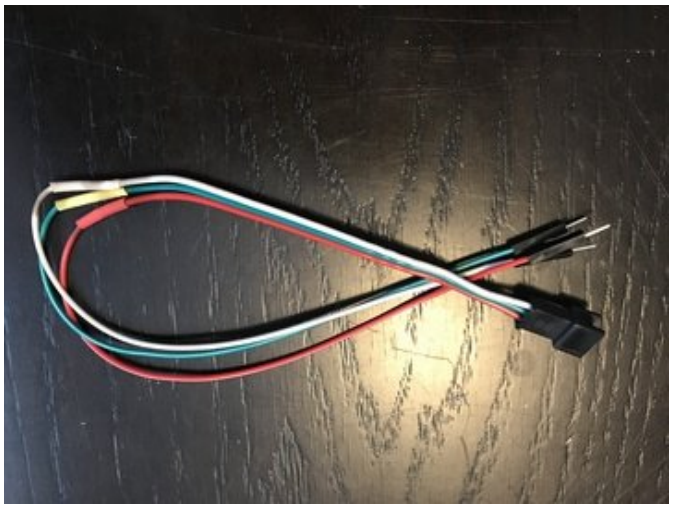
Cut one male side of each jumper wire off and throw out, or add them to your pile of maker confetti.



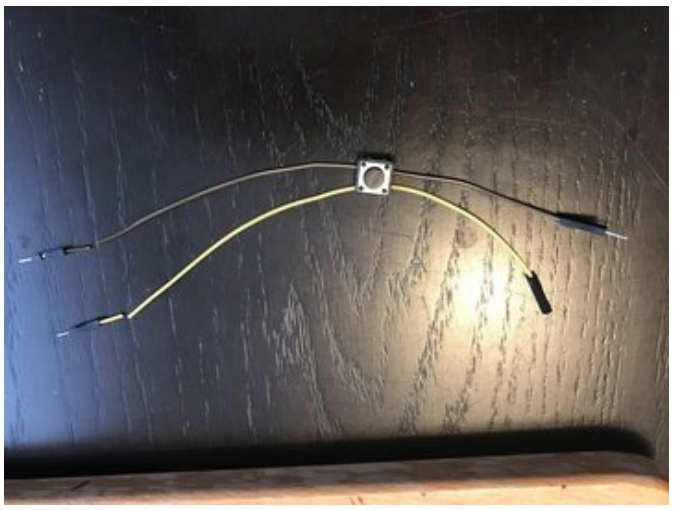
Solder

Strip about 1/2" of the open ends of each of the jumper wires & the open ends of the LED connector wires.

Secure some heat shrink on the wire and use your fingers to twist the the matching wires together.



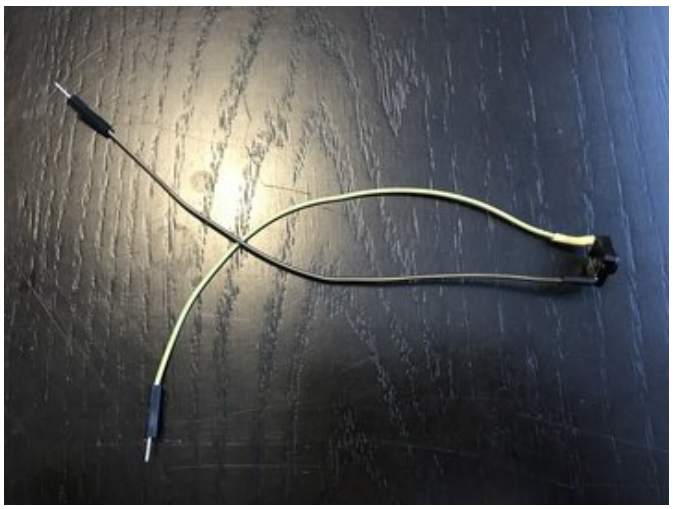
Solder the wires & heat up the heat shrink and set aside.



Button

Locate 2 jumper wires and some small pieces of heat shrink for your button.

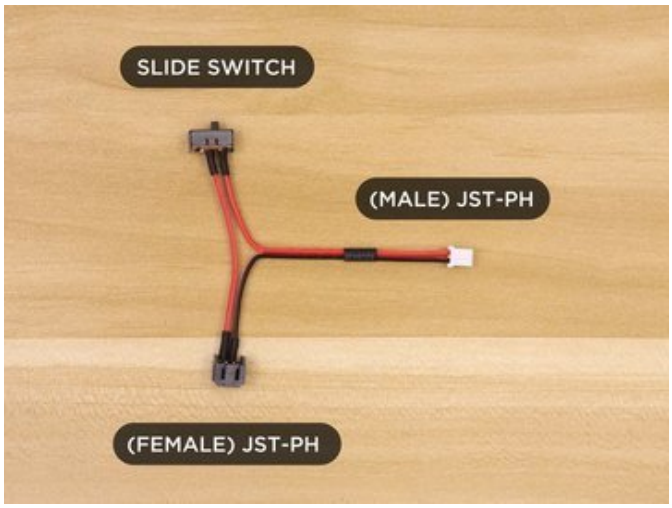
As you did above, cut one male side of each jumper wire off.



Strip about 1/2" of the open end of each jumper wire.

Secure some heat shrink on the wire and use your fingers to twist the wire onto opposite (diagonal) legs of the button.

Solder the wires to the button and heat up the heat shrink to secure it below the button.



Optional

Follow the Ruiz Brothers (https://adafru.it/onA) great learn guide and create a DIY On/Off JST Switch Adapter (https://adafru.it/vjb) for this project.

Photo credit: Ruiz Brothers

Exercise caution using sharp knives & power tools!

Cut the top

Draw a large circle on the top of your pumpkin. If you're feeling smart, throw a nice notch in there. Cut out the hole using your pumpkin carving tools or knife of choice.

Map the holes

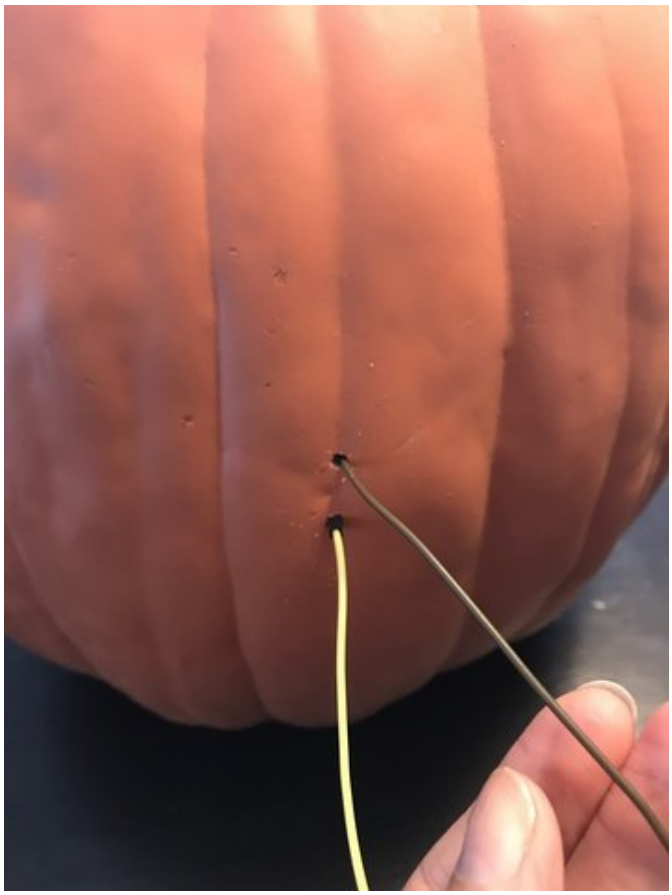
Using a pencil, pick a location as the center spot. Chart holes that are approximately 1.5" apart. Since the surface is spherical, the holes on the top and bottom row will be closer together than 1.5" apart.

Drill

Drill pilot holes with a 1/4" bit and then larger holes with the 7/16" bit. Test the holes with a pixel and go with a larger bit if needed. A 1/2" bit may work better, so it's good to have both on hand.

Position the button

Fold two diagonal button legs back or remove them. Pick a place where you'd like your button to sit on the outside of the pumpkin. Given the length of your wires, you probably won't be able to have it sit too high on the pumpkin. Push the legs of the button into the pumpkin to create a couple small markings.

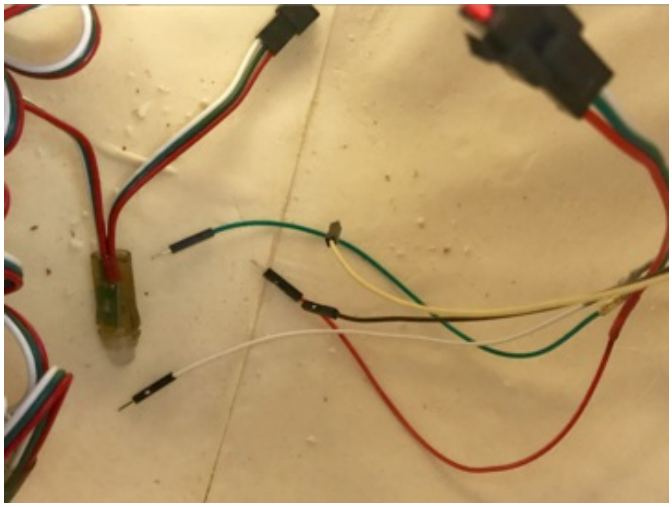


Drill button holes

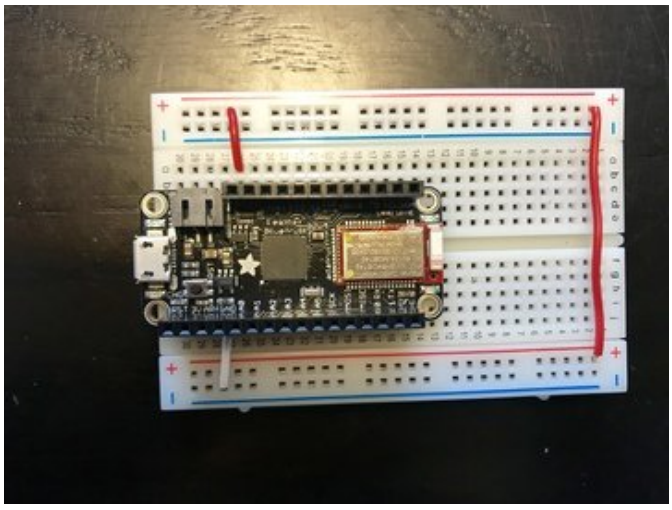
Drill the two button holes using a 1/8" bit. Feed the male jumper cables attached to the button into the holes to that the button sits against the pumpkin.

Clean your pumpkin and insert your pixels into the holes. The first pixel that receives data should be in the top left hand corner. Your pixels should be laid out in a zig-zag, back-and-forth, or serpentine way, so that the 9th pixel is directly below the 8th.

Final Assembly



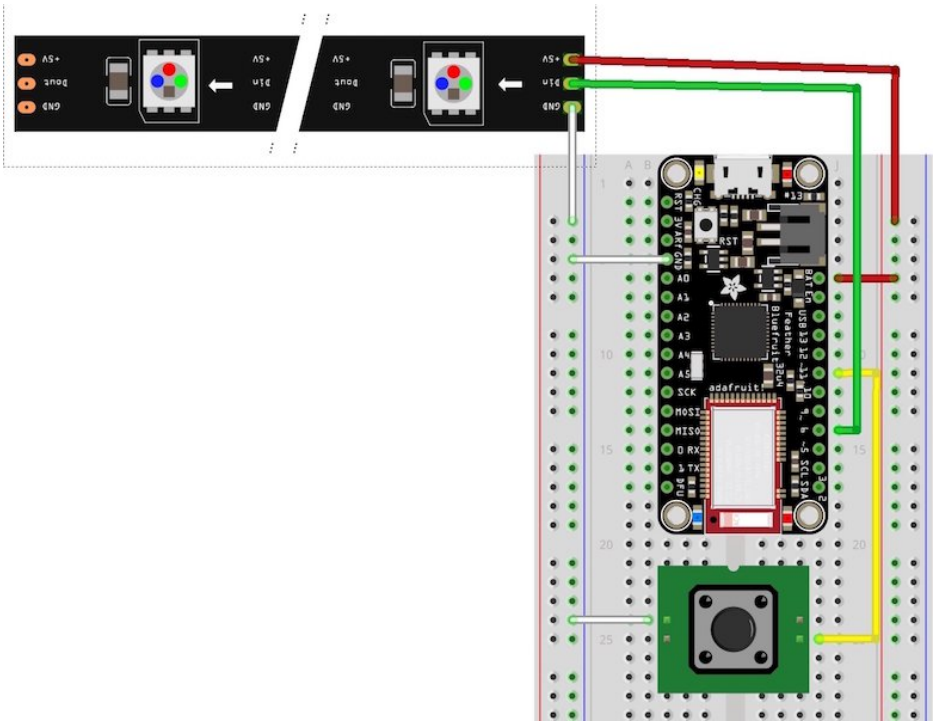
You should now have 5 or 6 jumper wire ends inside your pumpkin, 2 from the button and 3-4 from your pixels.



Breadboard

Prep your breadboard by creating a ground rail and a power rail (a rail is a place where all the same kind of wires hook together).

Circuit Diagram



Review the circuit diagram. This illustration is meant for referencing wired connections. The style of LEDs, length of wire, position, and size of components do not match the actual project.

LEDs

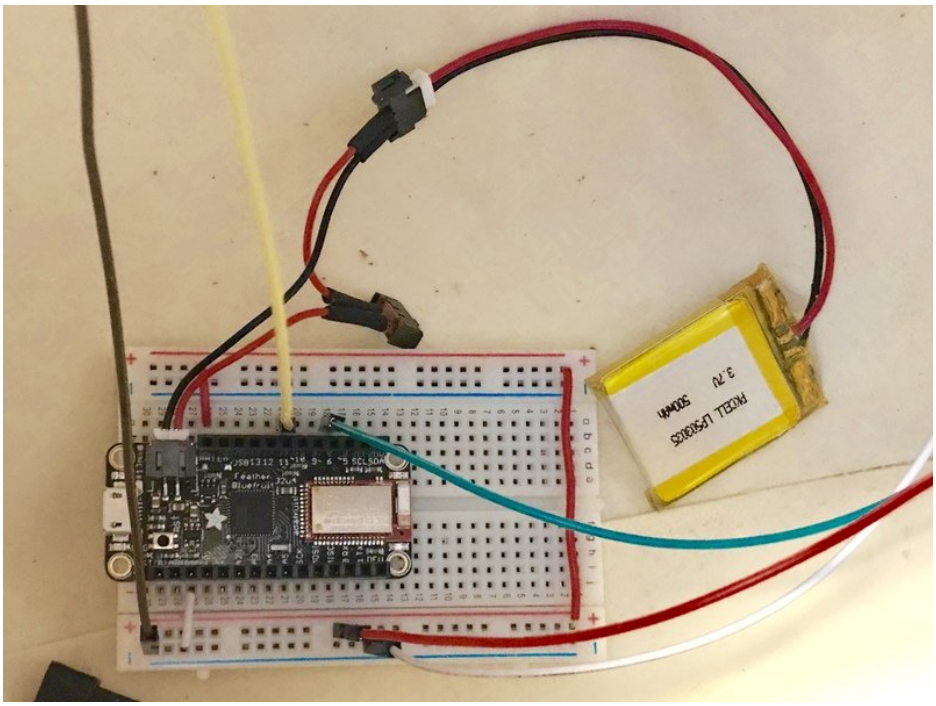
- The **+5v** on the LEDs connects to the **power rail** on the breadboard.
- The **GND** on the LEDs connects to the **ground rail** on the breadboard.
- The **Data In** on the LEDs connects to **pin 6** on the Feather Board.
- If using 4-pin LEDs, such as the WS2801 from Adafruit, the **Clock In** on the LEDs connects to pin 5 on the Feather Board.

Push Button

- One side of the button connects to the **ground rail** on the breadboard.
- The other side of the button (diagonal leg) connects to **pin 11** on the Feather Board.

Battery

- The Lithium-Ion battery plugs into the **JST socket** on the Feather Board



Done!

Power on and you should have lights and different animations with a button press.

If you find something not right, check your connections. Ensure the code was loaded onto the Feather board.