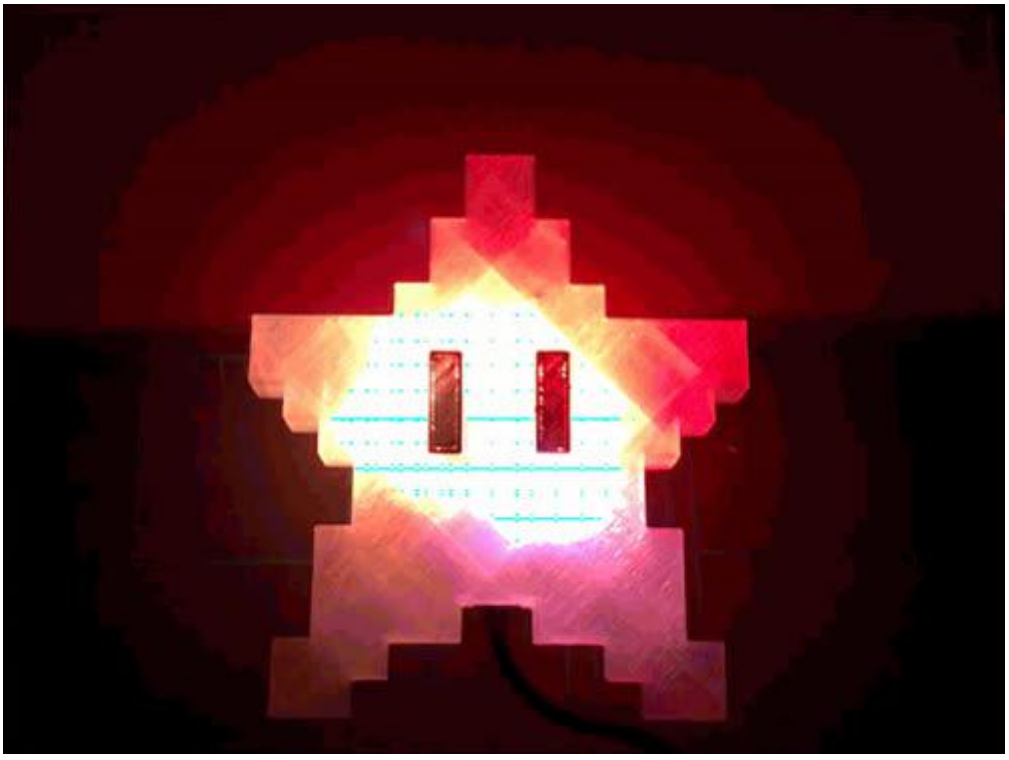
Starduino: 8-Bit Super Mario Tree Topper

Created by John Park



https://learn.adafruit.com/starduino-neopixel-8-bit-mario-star-tree-topper

Last updated on 2022-12-01 02:39:44 PM EST

Table of Contents

Overview

This guide was written for the Gemma v2 boards. It can also be done with the Gemma M0. We recommend the Gemma M0 as it is easier to use and is more compatible with modern computers! The wiring is the same.

Let's build an 8-bit tree topper! This project uses the GEMMA to run a NeoPixel ring stuffed inside a 3D printed Mario star. You start out by printing four star model parts. Then, you'll build and program the circuit. Next comes assembly, and finally you'll power it up and place it atop your tree! (Thanks to Artie Beavis () / AtmelMakes () for the name!)

Before you start, you should be familiar with using your 3D printer, and read up on these guides:

- Introducing GEMMA () or Adafruit Gemma M0 ()
- NeoPixel Uberguide ()
- Adafruit Guide to Excellent Soldering ()

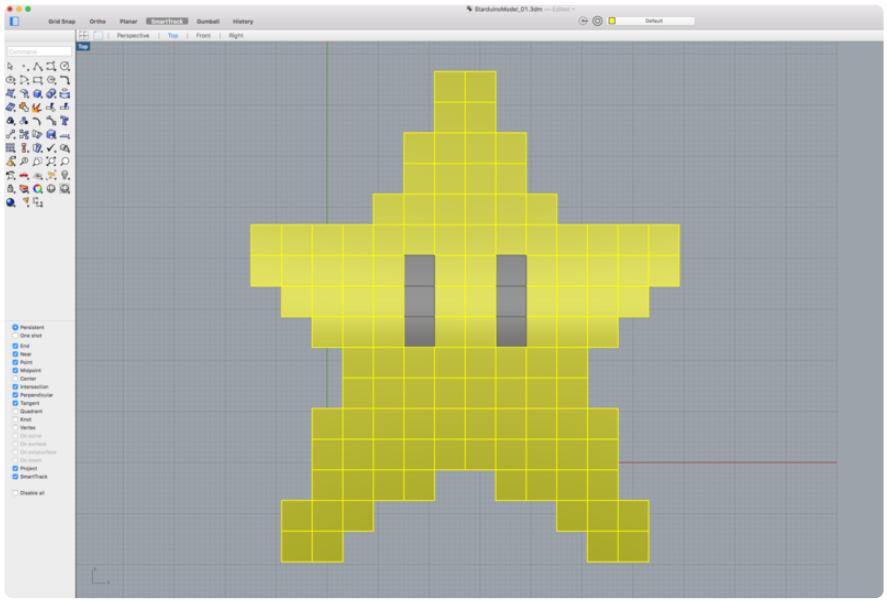
You'll need:

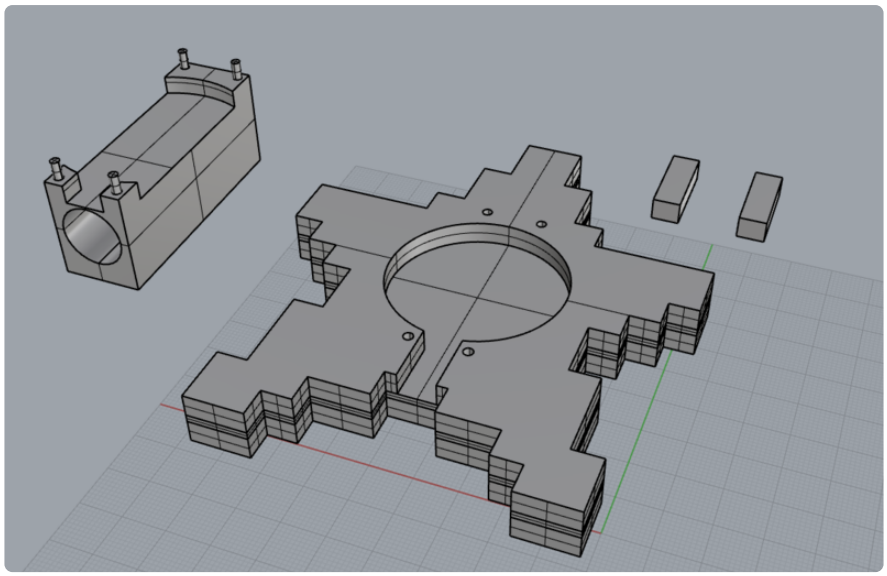
- 3D printer, such as Printrbot (http://adafru.it/1760)
- Translucent filament (http://adafru.it/2451) and black filament (http://adafru.it/2060) for 3D printer
- Adafruit Gemma M0 () or Adafruit GEMMA v2 ()
- RGB NeoPixel (http://adafru.it/1463) LEDs 16 x ring
- Soldering iron and solder
- Solid core (http://adafru.it/1311) or stranded (http://adafru.it/1970) wire (20 to 26 gauge)

- Helping third hand tool
- Wire strippers
- Flush diagonal cutters
- USB cable (http://adafru.it/260) - A/MiniB - 3ft
- 5V 1A USB port power supply (http://adafru.it/501) or USB battery pack (http://adafru.it/1959)

Print the Star

Time to heat and squeeze a length of humble, nondescript plastic filament into a delightful, three dimensional object! The model was built in Rhino using NURBS curves, extrusions, and solid booleans. The star has a cylindrical section designed to fit the GEMMA and NeoPixel ring, as well as a slot for the USB cable. The rectangular base piece has a complimentary section as well as posts to snap into the holes in the star once the electronics are in place, and a tubular section that to slide over the top of your tree.

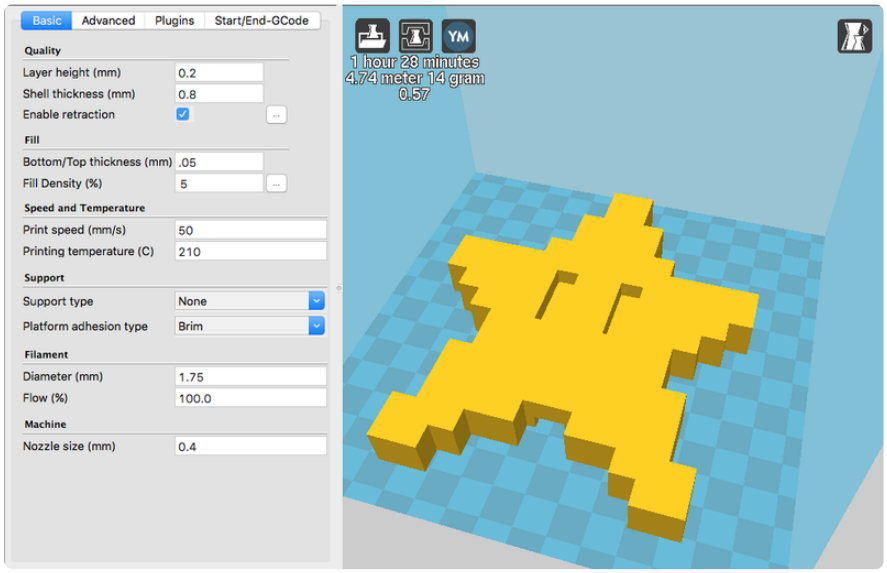




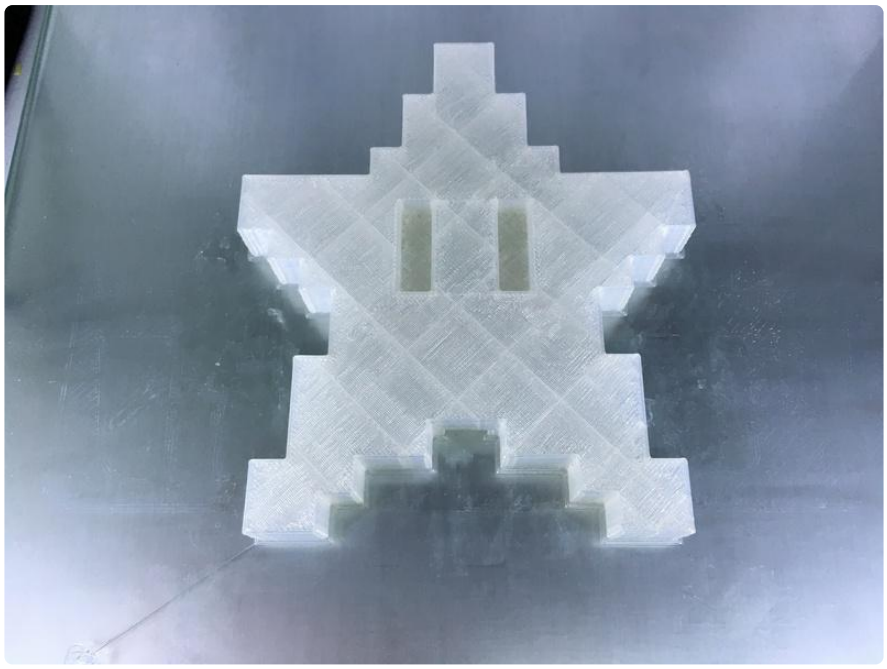
First, you'll need to download the model files from the link below.

Download STL files

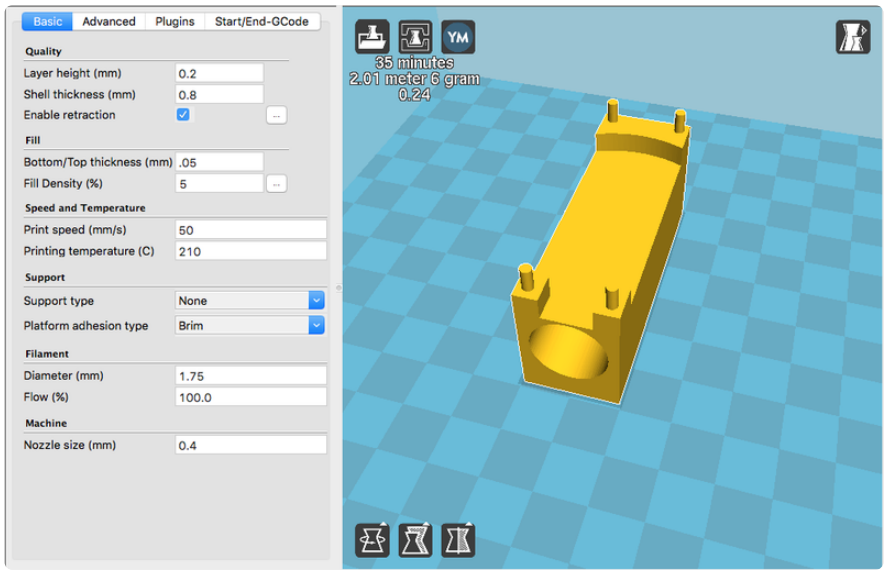
Once downloaded, load the first model, StarduinoBody.stl, into CURA or another 3D printer slicer package.

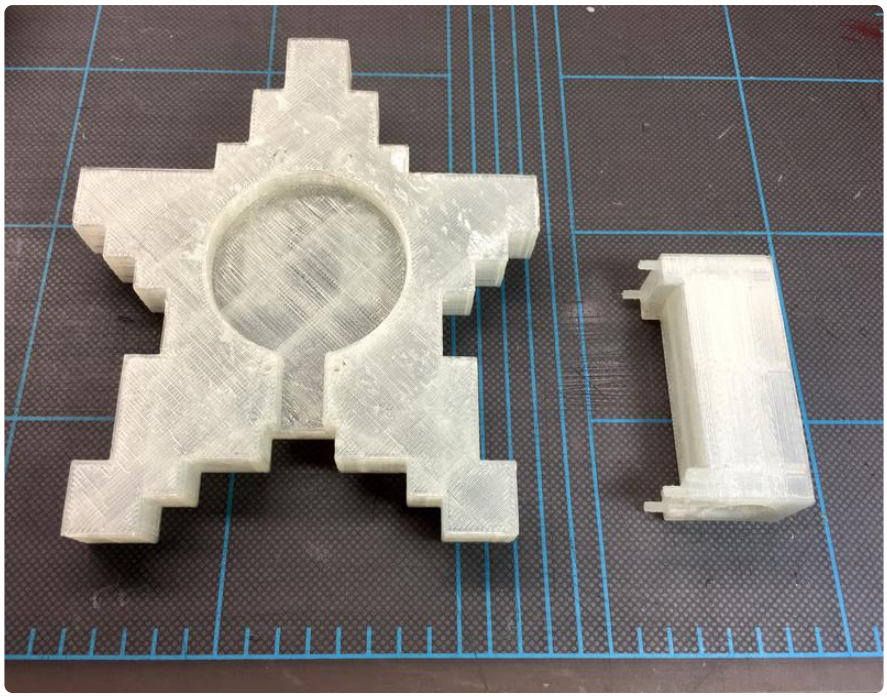


Adjust the settings to suit your printer -- a fairly low resolution print with 5-10% infill works well. Then, load natural/translucent filament into your printer and print!

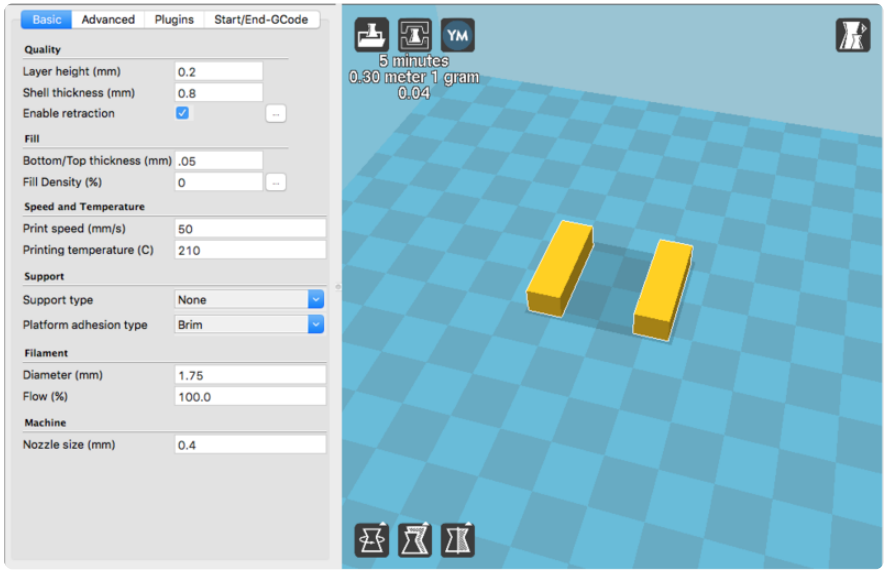


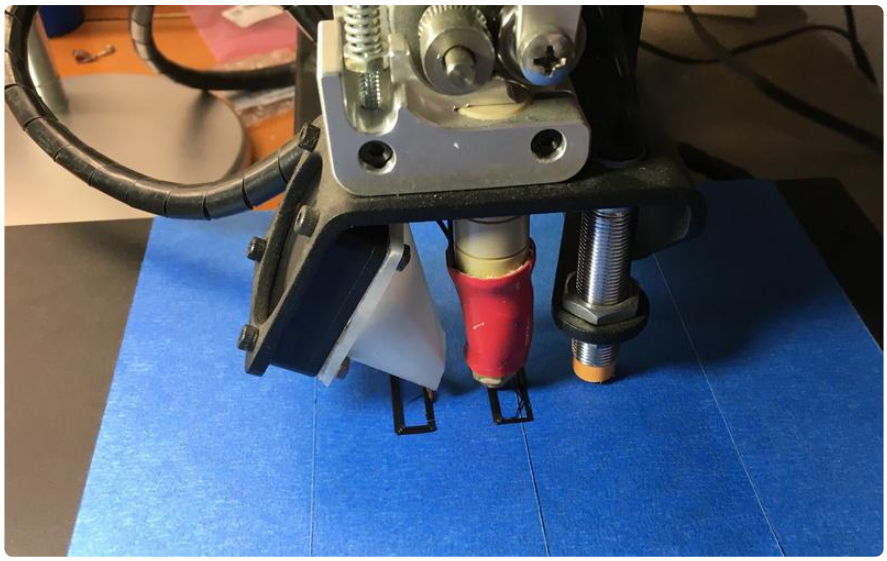
Next, load the StarduinoBase.stl model and print it with the same settings as the body.



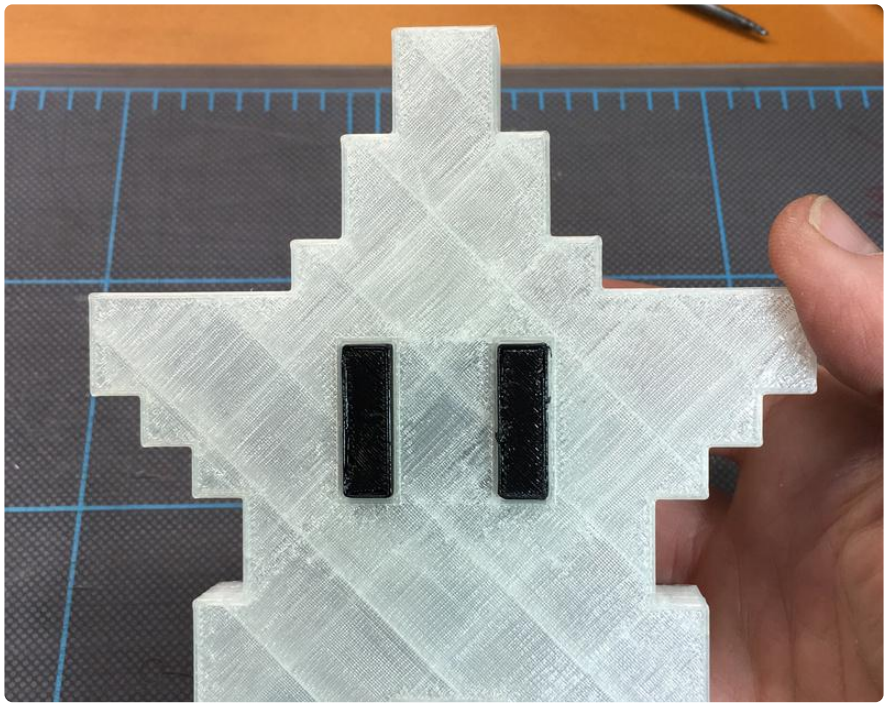


The final parts to print are the eyes. You may want to switch to black filament for these prints. Or, you may print with the same filament as before and then simply ink the eyes with black permanent marker. You can choose to print one eye at a time or both in one print job.





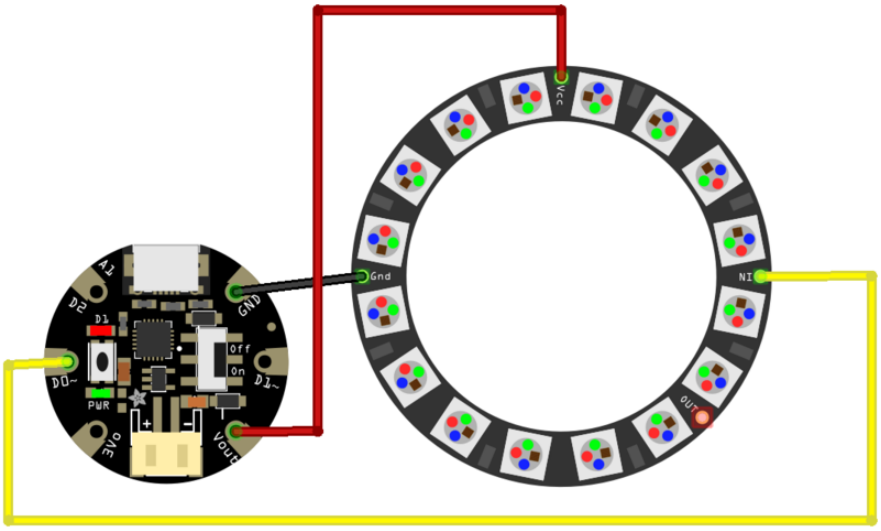
Once you've completed making the eyes, go ahead and press fit them into the star.



Make the Blinky Electronics

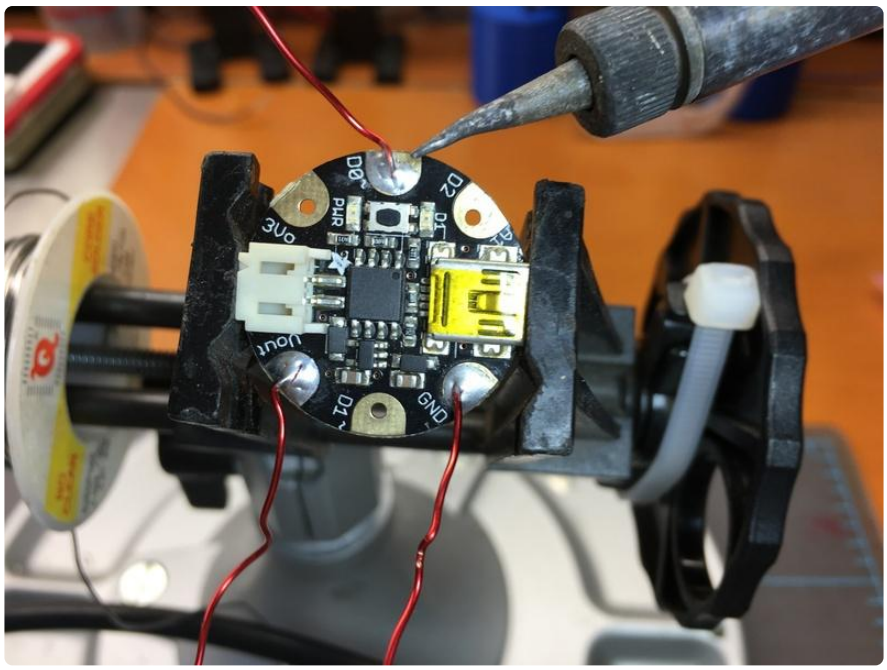
Controlling the NeoPixel ring's individual LEDs from the GEMMA is quite simple. Since the NeoPixels are individually addressable, the only connections needed between the ring and the GEMMA are for voltage, ground, and control signal. That's just three wires!

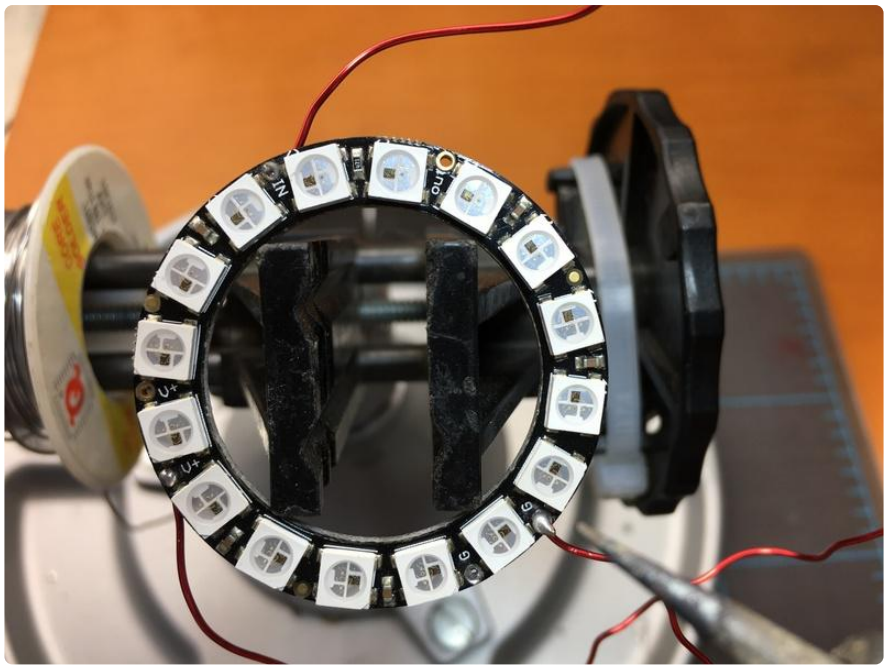
This guide was written for the Gemma v2 board. The pins used are the same for the Gemma M0. We recommend the Gemma M0 as it is easier to use and is more compatible with modern computers!

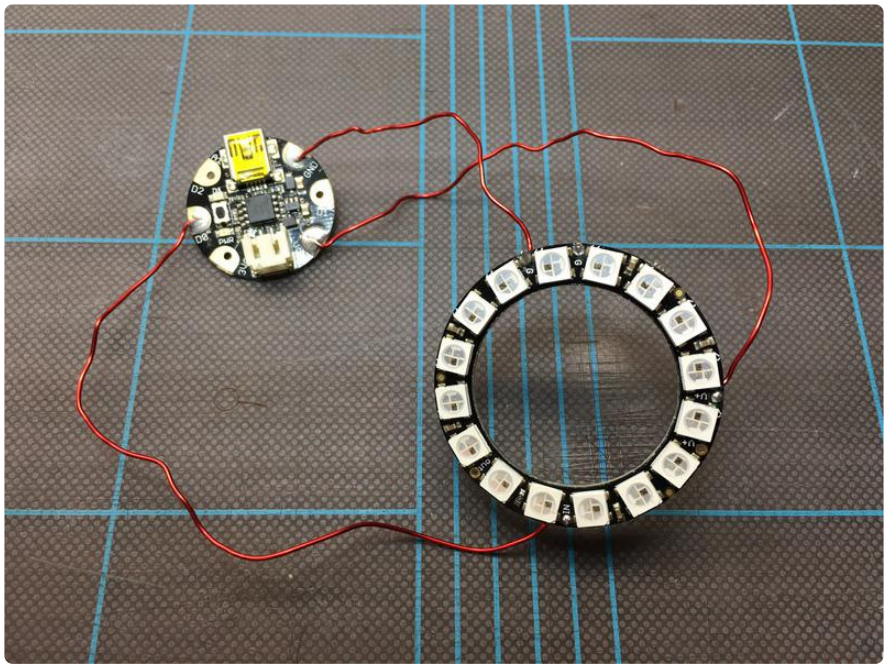


Cut three short lengths of wire (enamel covered motor wire is used here, but you can use any type that you have handy). Strip a bit off their ends, then solder these connections:

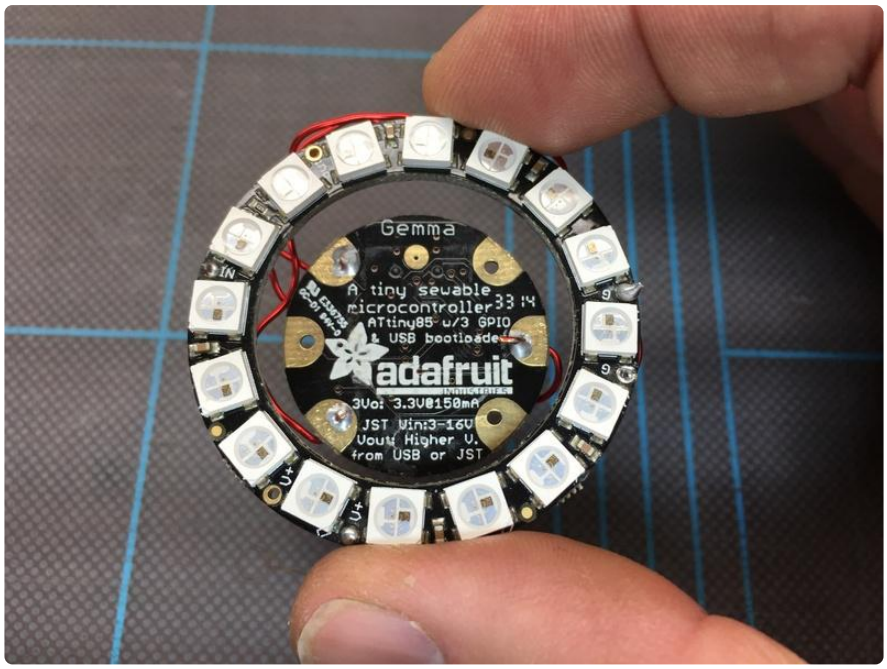
- GEMMA GND -> NeoPixel Gnd
- GEMMA Vout -> NeoPixel Vcc
- GEMMA D0 -> NeoPixel IN







Once sodering is completed, bend and tuck the wiring so that the GEMMA fits neatly inside the NeoPixel ring.



Arduino Code

The Arduino code presented below works equally well on GEMMA: v2 and M0. But if you have an M0 board, consider using the CircuitPython code on the next page of this guide, no Arduino IDE required!

To program GEMMA, make sure you have followed the instructions found in the "Introducing GEMMA" guide ().

Once you've got the GEMMA working, you can play with different example sketches, or code your own blinky pattern.

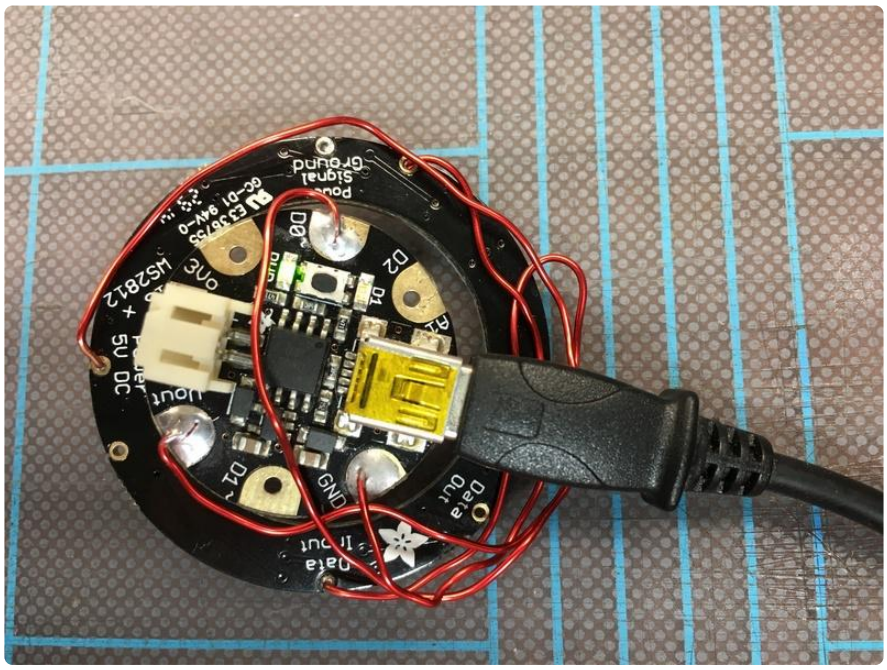
If you'd like an exciting, fiery pattern to run, download and install the FastLED Arduino library found here (), and then copy the following Arduino sketch.

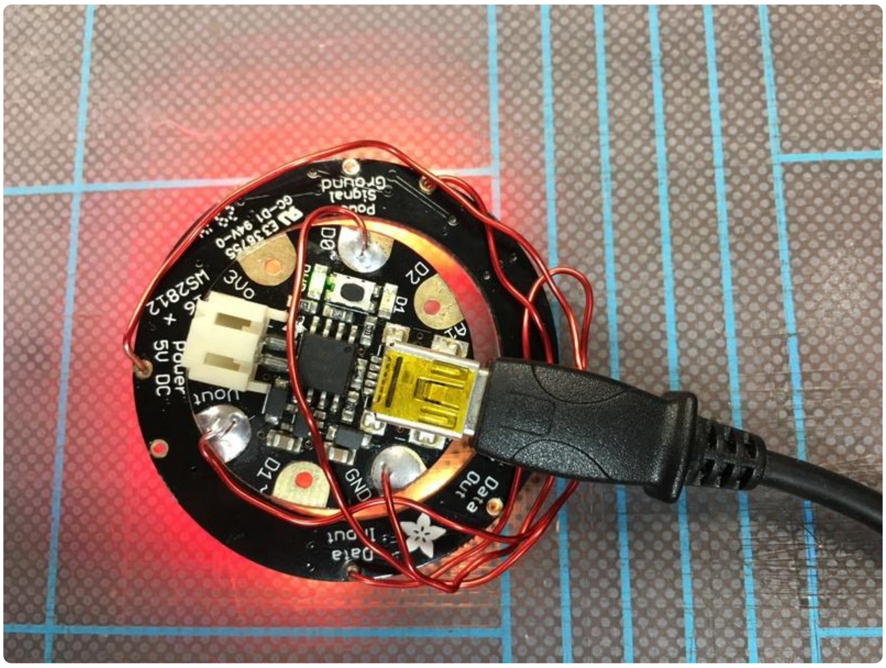
Plug the GEMMA into a USB cable connected to your computer, press and release the reset button on the GEMMA (the red LED on the GEMMA will blink) and then use the Arduino software IDE to upload the sketch to the board.

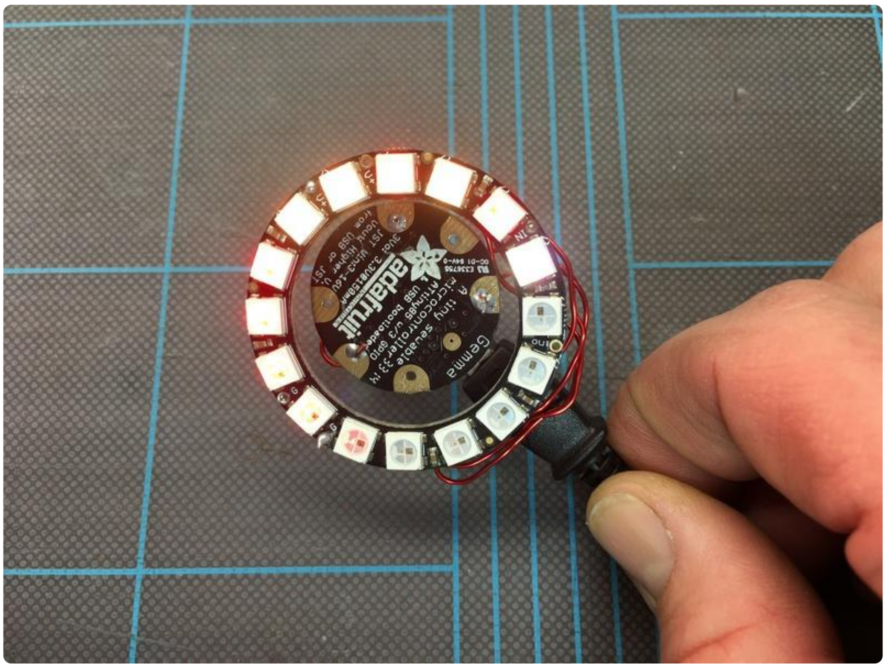
```
// SPDX-FileCopyrightText: 2018 Mikey Sklar for Adafruit Industries
//
// SPDX-License-Identifier: MIT

#include <FastLED.h>

#define LED_PIN     0
#define COLOR_ORDER GRB
#define CHIPSET     WS2811
#define NUM_LEDS    30
```

```
#define BRIGHTNESS  200
#define FRAMES_PER_SECOND 60

bool gReverseDirection = false;

CRGB leds[NUM_LEDS];

void setup() {
  delay(3000); // sanity delay
  FastLED.addLeds<CHIPSET, LED_PIN, COLOR_ORDER>(leds, NUM_LEDS).setCorrection(
TypicalLEDStrip );
  FastLED.setBrightness( BRIGHTNESS );
}

void loop()
{
  // Add entropy to random number generator; we use a lot of it.
  // random16_add_entropy( random());

  Fire2012(); // run simulation frame

  FastLED.show(); // display this frame
  FastLED.delay(1000 / FRAMES_PER_SECOND);
}


// Fire2012 by Mark Kriegsman, July 2012
// as part of "Five Elements" shown here: http://youtu.be/knWiGsmgycY
////
// This basic one-dimensional 'fire' simulation works roughly as follows:
// There's a underlying array of 'heat' cells, that model the temperature
// at each point along the line.  Every cycle through the simulation,
// four steps are performed:
//  1) All cells cool down a little bit, losing heat to the air
//  2) The heat from each cell drifts 'up' and diffuses a little
//  3) Sometimes randomly new 'sparks' of heat are added at the bottom
//  4) The heat from each cell is rendered as a color into the leds array
//     The heat-to-color mapping uses a black-body radiation approximation.
//
// Temperature is in arbitrary units from 0 (cold black) to 255 (white hot).
//
// This simulation scales it self a bit depending on NUM_LEDS; it should look
// "OK" on anywhere from 20 to 100 LEDs without too much tweaking.
//
// I recommend running this simulation at anywhere from 30-100 frames per second,
// meaning an interframe delay of about 10-35 milliseconds.
//
// Looks best on a high-density LED setup (60+ pixels/meter).
//
//
// There are two main parameters you can play with to control the look and
// feel of your fire: COOLING (used in step 1 above), and SPARKING (used
// in step 3 above).
//
// COOLING: How much does the air cool as it rises?
// Less cooling = taller flames.  More cooling = shorter flames.
// Default 50, suggested range 20-100
#define COOLING  55

// SPARKING: What chance (out of 255) is there that a new spark will be lit?
// Higher chance = more roaring fire.  Lower chance = more flickery fire.
// Default 120, suggested range 50-200.
#define SPARKING 120


void Fire2012()
{
// Array of temperature readings at each simulation cell
```

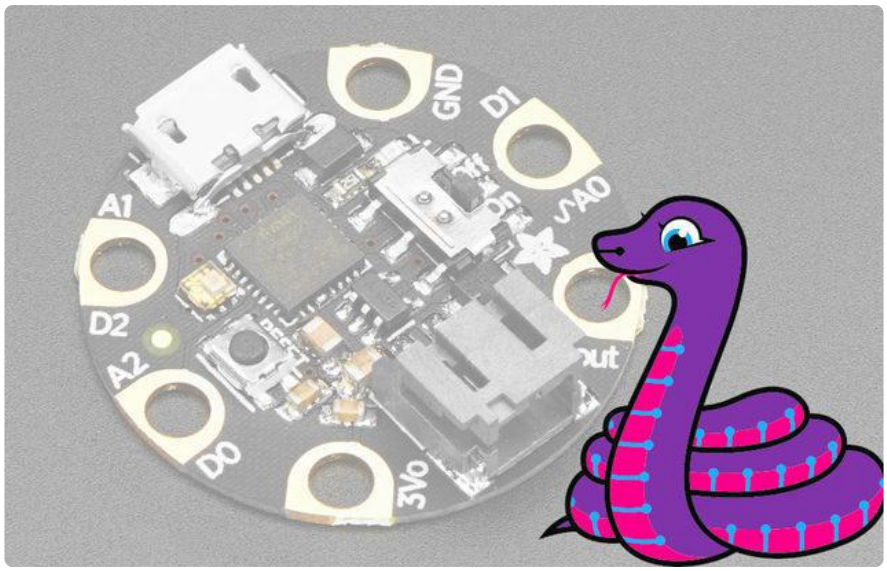
```
  static byte heat[NUM_LEDS];

  // Step 1.  Cool down every cell a little
    for( int i = 0; i < NUM_LEDS; i++) {
      heat[i] = qsub8( heat[i],  random8(0, ((COOLING * 10) / NUM_LEDS) + 2));
    }

    // Step 2.  Heat from each cell drifts 'up' and diffuses a little
    for( int k= NUM_LEDS - 1; k >= 2; k--) {
      heat[k] = (heat[k - 1] + heat[k - 2] + heat[k - 2] ) / 3;
    }

    // Step 3.  Randomly ignite new 'sparks' of heat near the bottom
    if( random8() < SPARKING ) {
      int y = random8(7);
      heat[y] = qadd8( heat[y], random8(160,255) );
    }

    // Step 4.  Map from heat cells to LED colors
    for( int j = 0; j < NUM_LEDS; j++) {
      CRGB color = HeatColor( heat[j]);
      int pixelnumber;
      if( gReverseDirection ) {
        pixelnumber = (NUM_LEDS-1) - j;
      } else {
        pixelnumber = j;
      }
      leds[pixelnumber] = color;
    }
}
```




Ground
Signal
D0
D2
D1
GND
Data Out
Vout
5V DC
Power
3Vo
WS2812


adafruit
Gemma

CircuitPython Code



GEMMA M0 boards can run CircuitPython — a different approach to programming compared to Arduino sketches. In fact, CircuitPython comes factory pre-loaded on GEMMA M0. If you've overwritten it with an Arduino sketch, or just want to learn the basics of setting up and using CircuitPython, this is explained in the Adafruit GEMMA M0 guide ().

These directions are specific to the "M0" GEMMA board. The GEMMA v2 with an 8-bit AVR microcontroller doesn't run CircuitPython...for those boards, use the Arduino sketch on the "Arduino code" page of this guide.

Below is CircuitPython code that works similarly (though not exactly the same) as the Arduino sketch shown on a prior page. To use this, plug the GEMMA M0 into USB...it should show up on your computer as a small flash drive...then edit the file "code.py" with your text editor of choice. Select and copy the code below and paste it into that file, entirely replacing its contents (don't mix it in with lingering bits of old code). When you save the file, the code should start running almost immediately (if not, see notes at the bottom of this page).

If GEMMA M0 doesn't show up as a drive, follow the GEMMA M0 guide link above to prepare the board for CircuitPython.

```
# SPDX-FileCopyrightText: 2018 Mikey Sklar for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import board
import neopixel
import adafruit_fancyled.adafruit_fancyled as fancy
```

```
num_leds = 16       # number of LEDs per strip
saturation = 255    # 0-255, 0 is pure white, 255 is fully saturated color
blend = True        # color blending between palette indices
brightness = 0.8    # brightness the range is 0.0 - 1.0
flicker = 0          # flame flicker

# NeoPixel objects using
leds = neopixel.NeoPixel(board.D0, num_leds)

# Inspired by Fire2012() by Mark Kriegsman and his use of FastLED
# to create a one-dimensional 'fire' simulation
# the heat colors are from the heat palette that FastLED provides
def fire_2018(strip, offset):
    # heat colors
    palette = [0x330000, 0x660000, 0x990000, 0xCC0000, 0xFF0000,
               0xFF3300, 0xFF6600, 0xFF9900, 0xFFCC00, 0xFFFF00,
               0xFFFF33, 0xFFFF66, 0xFFFF99, 0xFFFFCC]

    for i in range(num_leds):
        # FancyLED can handle the gamma adjustment, brightness and RGB settings
        color = fancy.palette_lookup(palette, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=brightness)
        strip[i] = color.pack()

while True:
    fire_2018(leds, flicker)
    flicker += 0.3           # flame flicker, adjust value to control speed
```

Installing Libraries

This code requires two libraries be installed:

- neopixel
- adafruit_fancyled

A factory-fresh board will have the neopixel library already installed. If you've just reloaded the board with CircuitPython, create the "lib" directory and then copy in the neopixel.mpy and adafruit_fancyled folder from the latest release of the Adafruit_CircuitPython_Bundle ().

The FancyLED library () being using in this CircuitPython example is not the same as the FastLED ()used for Arduino. FancyLED has a subset of FastLED features and some different syntax. The FancyLED tutorial provides an excellent overview ().

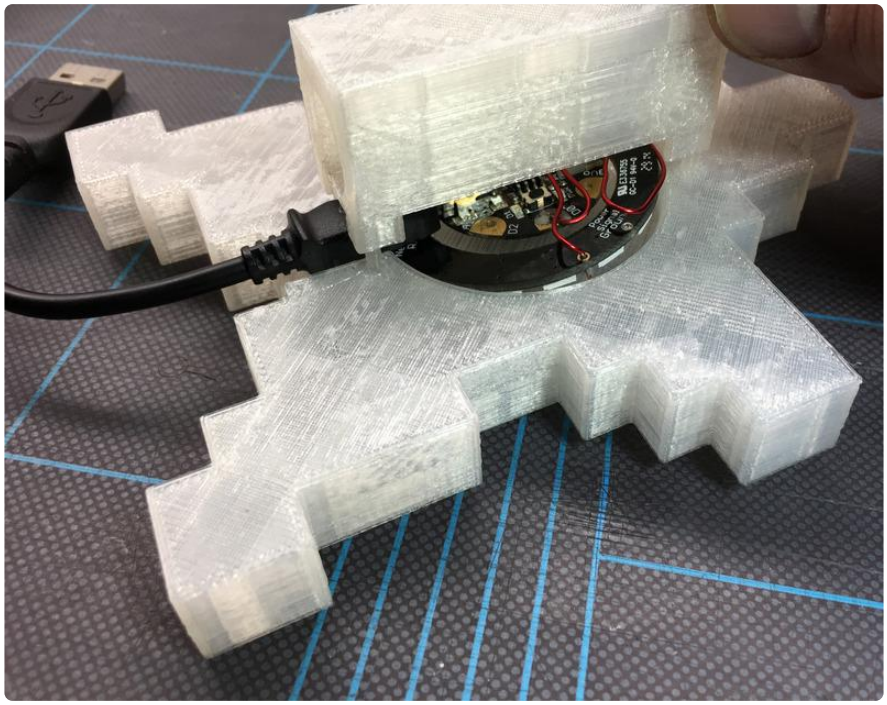
The file system layout on your gemma M0 should look like this:

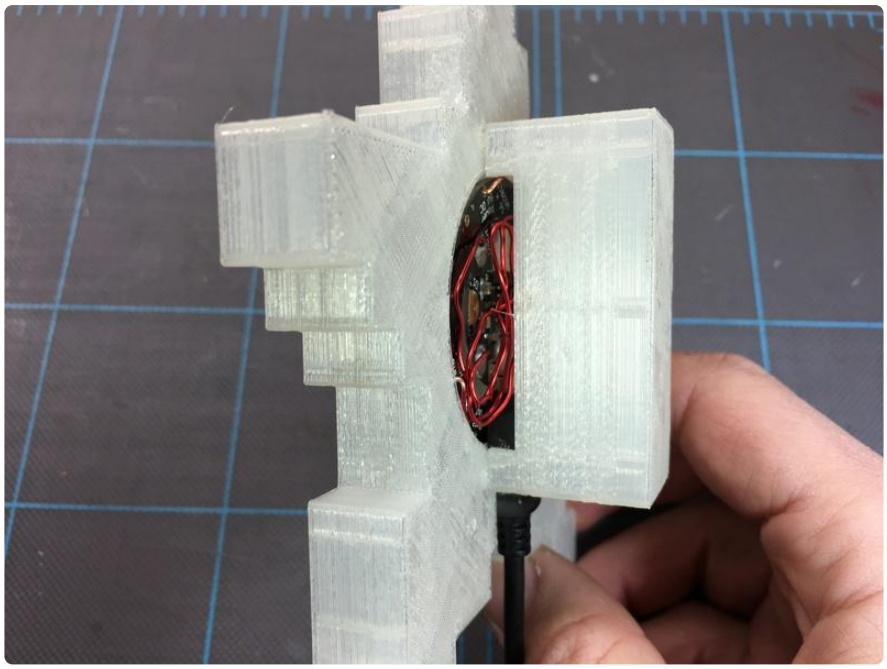
```
$ pwd
/Volumes/CIRCUITPY
$ find .
.
./boot_out.txt
./.fseventsd
```

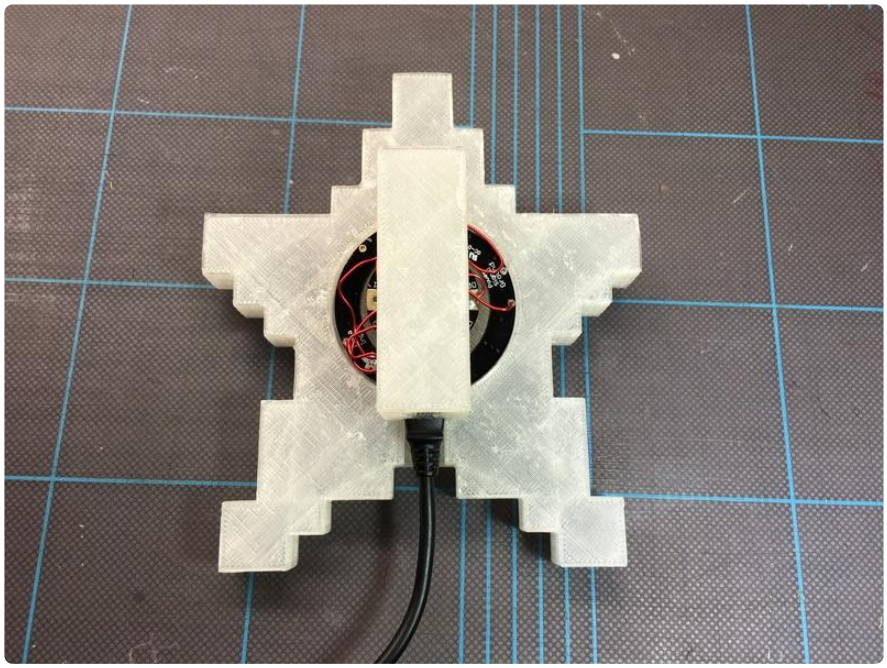
```
./.fseventsd/fseventsd-uuid
./lib
./lib/neopixel.mpy
./lib/adafruit_fancyled
./lib/adafruit_fancyled/adafruit_fancyled.mpy
./lib/adafruit_fancyled/fastled_helpers.mpy
./main.py
```

Assemble the Awesome Starduino

The final step is to put the electronics into the star. Simply place the NeoPixel/GEMMA bundle into the cylindrical recess in the star with the LEDs facing forward, plug in the USB cable, and then place the base model onto the backside, snapping the four posts into place. It should all hold together nicely via friction.







Plug the USB cable into a power source -- either a wall adapter or a battery -- and watch it sparkle!

All that's left is to top your tree with your rad 8-bit Super Mario star!