Techno-Tiki RGB LED Torch

Created by Tony DiCola

https://learn.adafruit.com/techno-tiki-rgb-led-torch

Last updated on 2023-08-29 02:58:31 PM EDT

Table of Contents

Overview

This guide was written for the Gemma v2 board, but can be done with either the original or Gemma M0. We recommend the Gemma M0 or Circuit Playground Express as they are easier to use and are more compatible with modern computers!

Aloha! Are you looking for a beautiful decoration for late summer parties, camping trips, or festivals? The techno-tiki torch project is the perfect way to set the mood with LED torches that glow and animate in any color. These lights are built from around Gemma (http://adafru.it/2470) will work with the original (), v2 () or M0 () combined with a NeoPixel strip (http://adafru.it/1376) placed inside a mason jar that's filled with diffuse material. The jar slides right into a bamboo tiki torch and turns it into a techno-tiki torch that glows with cyber-Hawaiian ambiance. And since this torch has no flames it's safe to use anywhere, even indoors! Light up the dark with the fun colors and style of the techno-tiki torch.

UPDATE October 2017: There's now a version of the techno tiki torch that uses Circuit Playground Express ()! This version has IR remote control built-in using Circuit Playground Express' IR receiver. You don't even need to solder NeoPixels to the board as it uses the 10 pixels built-in to Circuit Playground Express too! Check out the repository for the techno tiki code () to get the Circuit Playground Express version and load it on your board using the Arduino IDE or CircuitPython!

To follow this project it will help to familiarize yourself with the following guides first:

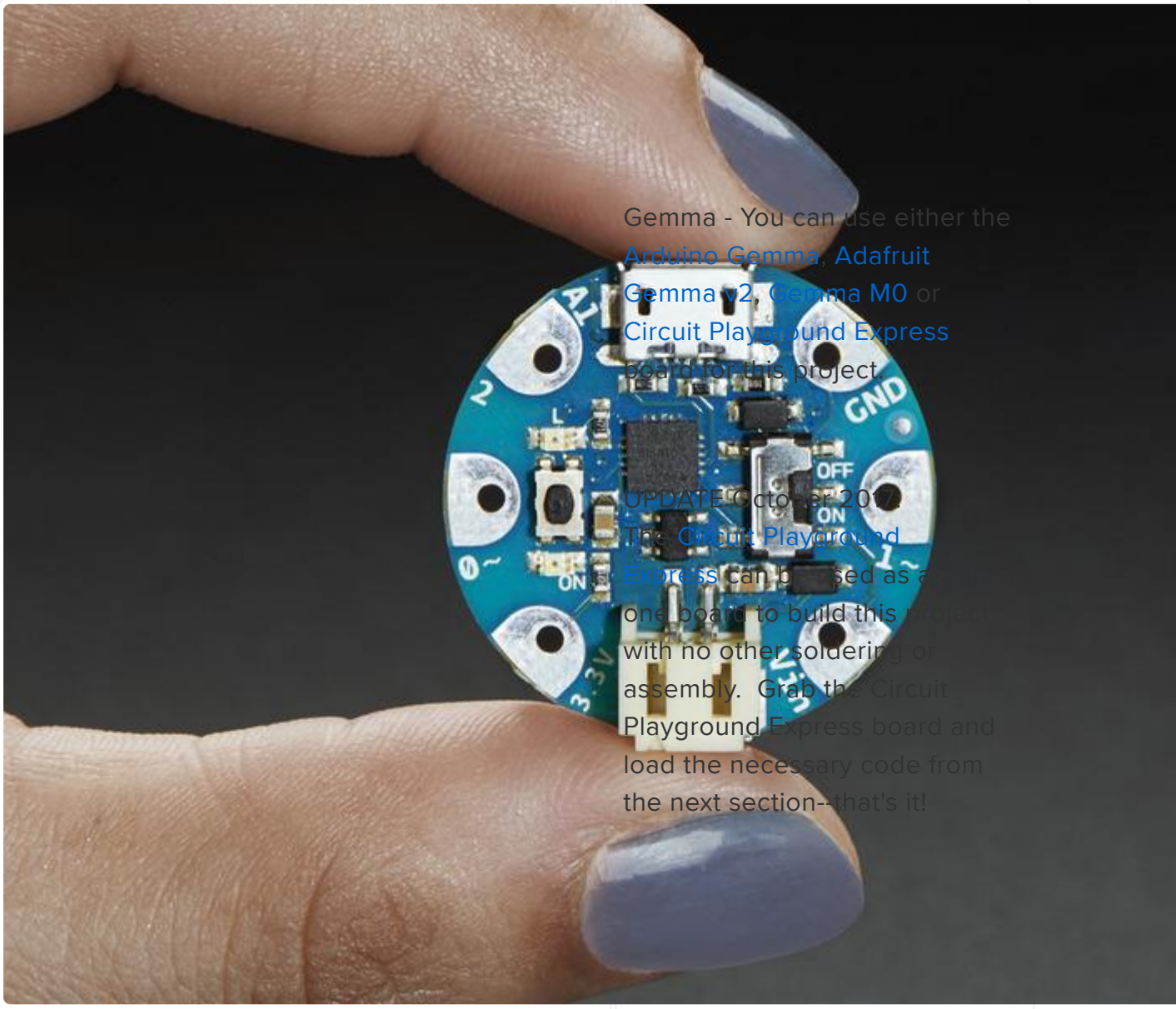
- Adafruit Gemma M0 Guide ()

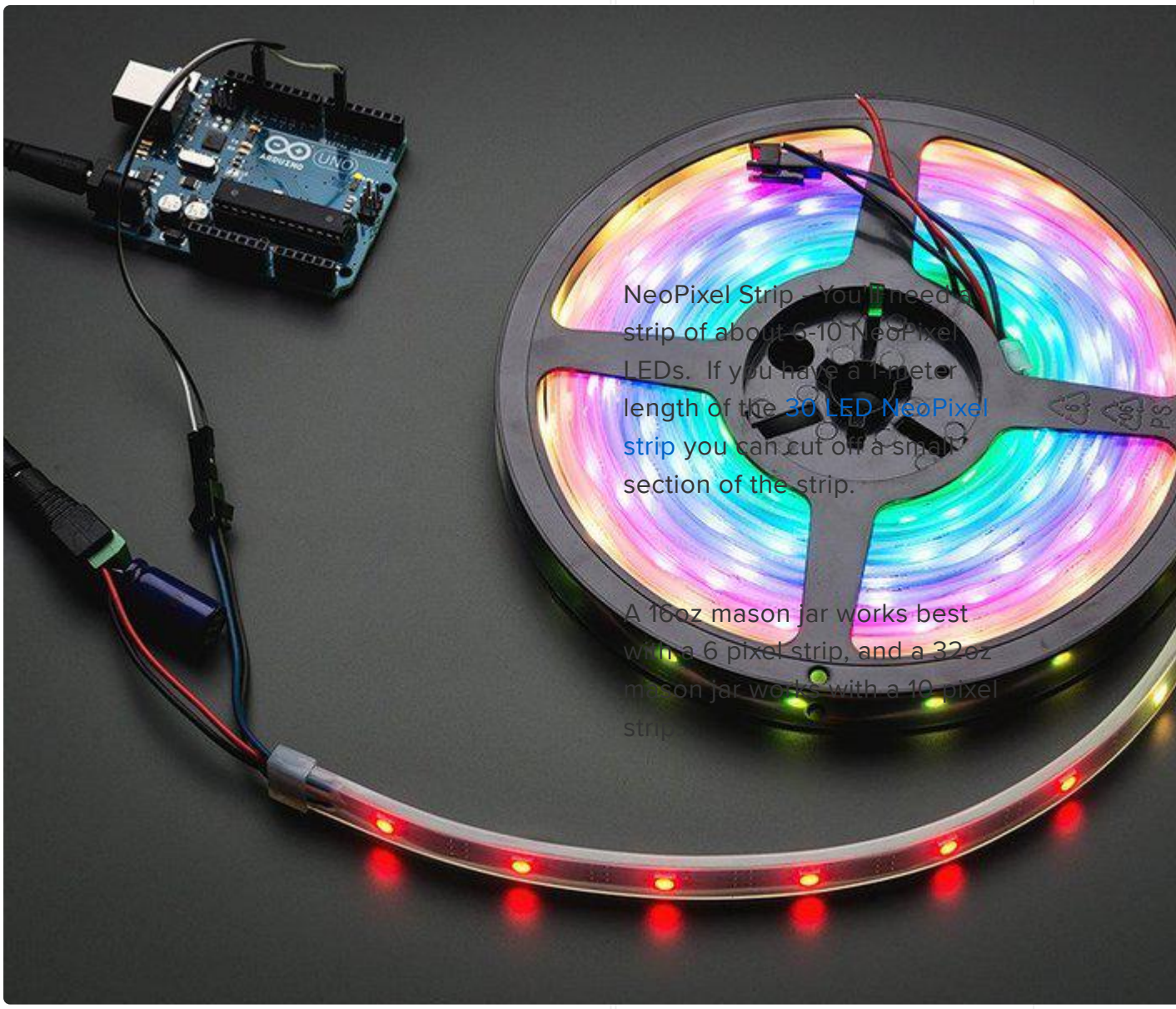
- Adafruit Circuit Playground Express Guide ()
- Adafruit Gemma Guide (original and v2) ()
- Adafruit NeoPixel Uberguide ()
- Adafruit Guide to Excellent Soldering ()

Overall this is an easy project that can be built in an afternoon. There's a small amount of soldering to connect NeoPixels, Gemma, and an IR receiver. Continue on to learn about the parts needed to build the project.

Parts

You'll need the following parts to build this project:



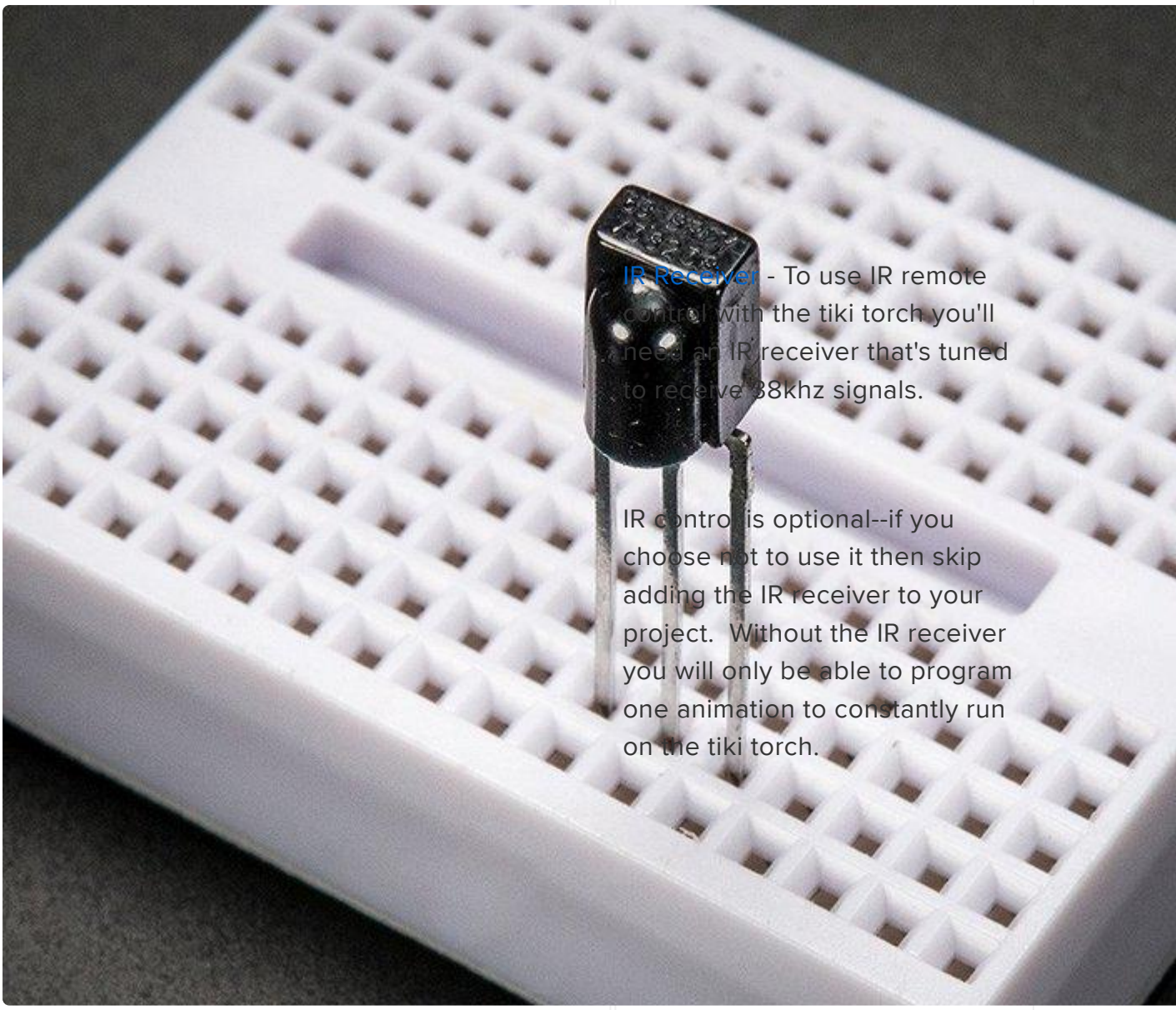


NeoPixel Strip - You'll need a strip of about 6-10 NeoPixel LEDs. If you have a 1-meter length of the 30 LED NeoPixel strip you can cut off a small section of the strip.

A 16oz mason jar works best with a 6 pixel strip, and a 32oz mason jar works with a 10 pixel strip.

Tiki torch - Look for plain bamboo tiki torches like shown to the left as they likely fit mason jars without any modification. Check the outdoor or seasonal section of department or hardware stores for tiki torches. Use a mason jar to check how well it fits in the torch if you can before purchasing.

Tiki brand 'The Original' tiki torches worked best in my testing and didn't require any modifications to hold a 16 or 32oz regular mouth mason jar.



IR Receiver - To use IR remote control with the tiki torch you'll need an IR receiver that's tuned to receive 38khz signals.

IR control is optional--if you choose not to use it then skip adding the IR receiver to your project. Without the IR receiver you will only be able to program one animation to constantly run on the tiki torch.

Remote control - To use IR control for your tiki torch make sure to use this micro remote control. This remote sends NEC IR remote control codes which the project software can recognize because of code size constraints only NEC IR codes are supported by this project.

Battery - A lithium ion battery is recommended as it's small but still able to light power hungry NeoPixels. A 2200 or 2200 mAH battery will run bright animations for about 5 hours, and stay in off/sleep mode up to about a week. Don't use a battery smaller than about 1000 mAH or else you'll only get an hour or less of bright animations.

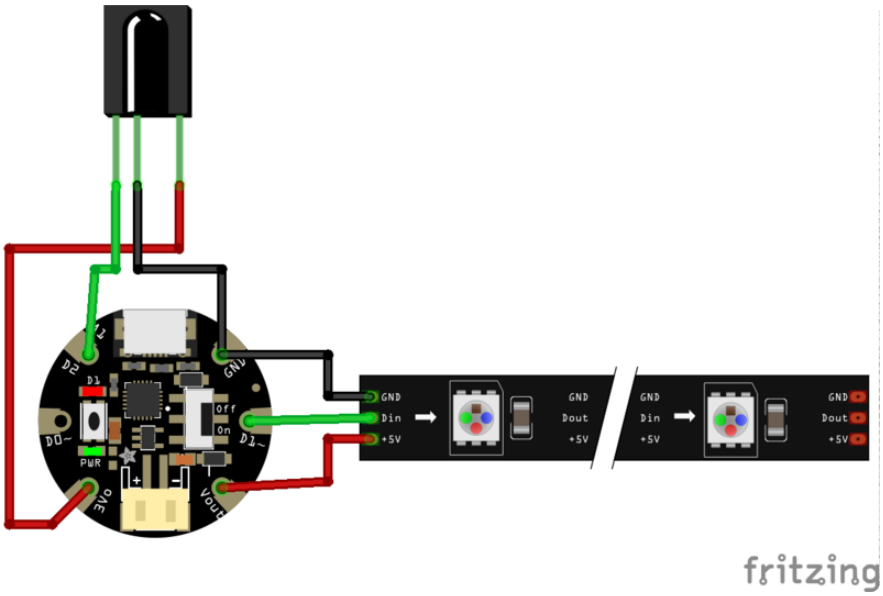
Make sure you have a battery charger too, like the Micro Lipo charger.

Soldering tools - You'll need to solder wires to the IR receiver, NeoPixel strip, and Gemma. You'll also want heatshrink wrap to protect the soldered connections on the IR receiver.

Mason Jar - Look for a regular mouth mason jar in either 16oz or 32oz size. You can typically find mason jars at any grocery, department, or craft store however they are available online too. Clear glass jars work great, but you can experiment with colored or even patterned jars. Don't use a wide mouth jar as it's too large to fit inside a typical tiki torch without modification!

Vase filler material - Look for a small, clear vase filler material like tiny clear beads or plastic rods. The smaller the material the better as it will fill in the space of the jar and better diffuse the light of the pixels. Be careful not to use a metal or conductive filler material that might short electric connections. This crystal fill product worked well in my testing.

Circuit Diagram



This diagram uses the original Gemma but you can also use the Gemma M0 with the exact same wiring!

- NeoPixel strip ground to Gemma GND/ground.
- NeoPixel strip Din to Gemma D1.
- NeoPixel strip +5V to Gemma Vout.
- IR receiver signal pin (left-most pin when receiver is facing you) to Gemma D2.
- IR receiver ground pin (middle pin) to Gemma GND/ground.
- IR receiver power pin (right-most pin when receiver is facing you) to Gemma 3Vo.

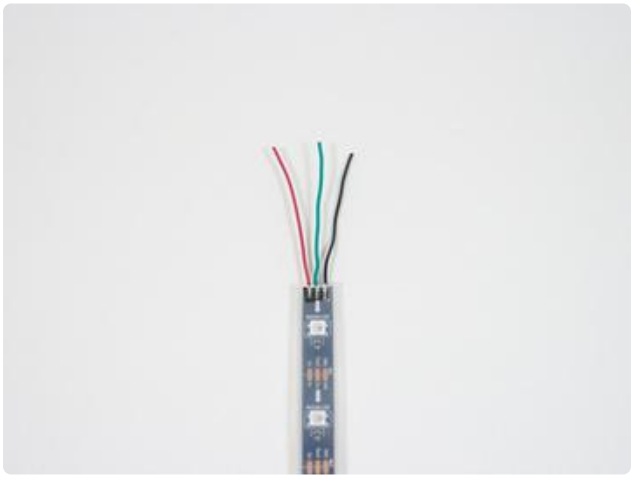
Soldering

Follow the steps below to solder the hardware in the project. If you're new to soldering be sure to read the guide to excellent soldering ().

Note the IR receiver is optional. If you choose not to use the IR receiver then ignore the instructions that deal with assembling and connecting it to the Gemma. Without the IR receiver you will only be able to program one animation to constantly run instead of being able to control animations with the remote control.

Cut your NeoPixel strip down to the size you'd like to use for your torch. For a 16oz mason jar a strip of 6 NeoPixels (from the 30 pixels per meter strip) works great, and

for a 32oz mason jar 10 NeoPixels are best. Make sure to only cut the NeoPixel strip on the line with exposed solder pads between pixels.



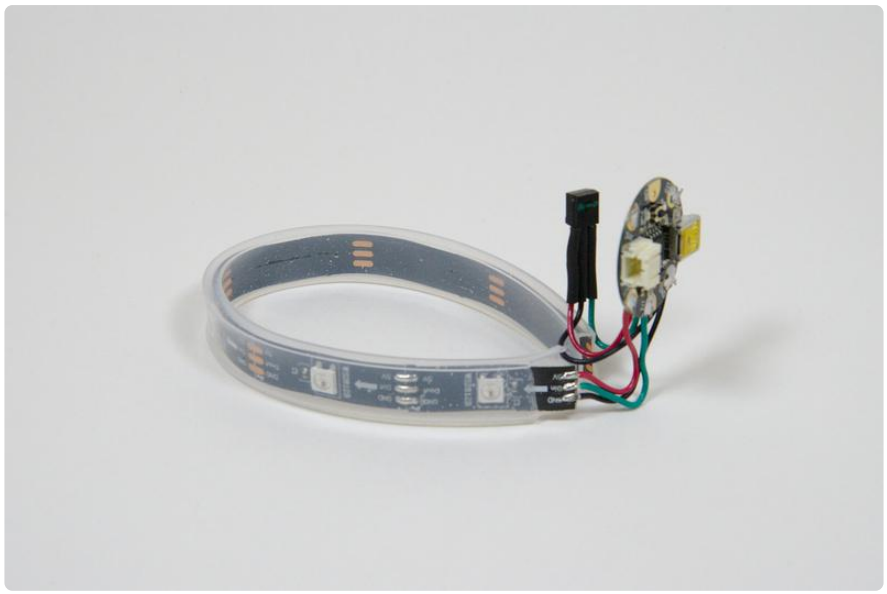
Start by soldering wires that are about 3-4" long to the NeoPixel strip and IR receiver. It's easiest to 'tin' each wire and pin by heating them with the iron and touching with solder to leave a thin layer of solder on the part. Then hold the tinned parts together and heat them with the iron to flow the solder again and fuse the parts.

Make sure to solder to the end of the NeoPixel strip with the arrow pointing towards the pixels (the signal line can only flow in one direction on a NeoPixel strip).

Add heatshrink to the connections on the IR receiver to prevent them from touching and shorting out.

Next solder the NeoPixel strip and IR receiver to the Gemma as shown in the Circuit Diagram section.

Once the parts are soldered together gently bend the IR receiver so it's facing behind the Gemma, and the NeoPixel strip so it forms a U shape as shown below.



Continue on to learn how to install the Arduino sketch that controls the hardware.

Arduino Code

This guide was written for the original Gemma and Gemma v2 boards. If you have a Gemma M0 you must use CircuitPython. We recommend the Gemma M0 or Circuit Playground Express as they are easier to use and are more compatible with modern computers!

Installation

Before you install the software for this project you'll need to make sure you're running the latest Arduino IDE 1.6.x version (). In addition you'll need to install the Adafruit NeoPixel library () using either the library manager () (recommended) or a manual installation ().

Once you have the Arduino IDE and NeoPixel library setup, click the button below to download the Arduino sketches for this project from their home on GitHub ():

You can download these three Arduino sketches:

- Techno_Tiki_No_Remote_Control () - This sketch is for a hardware setup that does not use the IR receiver and remote control.
- Techno_Tiki_With_Remote_Control () - This sketch is for a hardware setup that uses the IR receiver and remote control.

- Techno_Tiki_Circuit_Playground_Express () - This sketch is for a hardware setup using the Circuit Playground Express () board. Be sure to follow the Circuit Playground Express guide () to setup the Arduino IDE to load the board and inst all the necessary Circuit Playground library () first!

Open the appropriate sketch in the Arduino IDE.

Configuration

If you're using a remote control there are some configuration settings you can change at the top of the Techno_Tiki_With_Remote_Control sketch:

```
// SPDX-FileCopyrightText: 2018 Tony DiCola for Adafruit Industries
// SPDX-FileCopyrightText: 2020 Erin St. Blaine for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// Techno-Tiki RGB LED Torch with IR Remote Control
// Created by Tony DiCola
//
// See guide at: https://learn.adafruit.com/techno-tiki-rgb-led-torch/overview
//
// Released under a MIT license: http://opensource.org/licenses/MIT
#include <avr/power.h>
#include <avr/sleep.h>
#include <Adafruit_NeoPixel.h>

// Sketch configuration:
#define PIXEL_PIN      1    // Pin connected to the NeoPixel strip.

#define PIXEL_COUNT    6    // Number of NeoPixels.  At most only about 100 pixels
                            // can be used at a time before it will take too long
                            // to update the pixels and IR remote codes might be
                            // missed.

#define PIXEL_TYPE     NEO_GRB + NEO_KHZ800   // Type of NeoPixel.  Keep this the
default
                                              // if unsure.  See the NeoPixel
library examples
                                              // for more explanation and other
possible values.

#define IR_PIN         2    // Pin connected to the IR receiver.

// Adafruit IR Remote Codes:
//   Button       Code          Button  Code
//   -----------  ------        ------  -----
//   VOL-:        0x0000        0/10+:  0x000C
//   Play/Pause:  0x0001        1:      0x0010
//   VOL+:        0x0002        2:      0x0011
//   SETUP:       0x0004        3:      0x0012
//   STOP/MODE:   0x0006        4:      0x0014
//   UP:          0x0005        5:      0x0015
//   DOWN:        0x000D        6:      0x0016
//   LEFT:        0x0008        7:      0x0018
//   RIGHT:       0x000A        8:      0x0019
//   ENTER/SAVE:  0x0009        9:      0x001A
//   Back:        0x000E
#define COLOR_CHANGE      0x000A   // Button that cycles through color animations.
```

```
#define ANIMATION_CHANGE  0x0008   // Button that cycles through animation types
(only two supported).
#define SPEED_CHANGE      0x0005   // Button that cycles through speed choices.
#define POWER_OFF         0x0000   // Button that turns off/sleeps the pixels.
#define POWER_ON          0x0002   // Button that turns on the pixels.  Must be
pressed twice to register!


// Build lookup table/palette for the color animations so they aren't computed at
runtime.
// The colorPalette two-dimensional array below has a row for each color animation
and a column
// for each step within the animation.  Each value is a 24-bit RGB color.  By
looping through
// the columns of a row the colors of pixels will animate.
const int colorSteps = 8;   // Number of rows/animations.
const int colorCount = 24;  // Number of columns/steps.
const uint32_t colorPalette[colorCount][colorSteps] PROGMEM = {
  // Complimentary colors
  { 0xFF0000, 0xDA2424, 0xB64848, 0x916D6D, 0x6D9191, 0x48B6B6, 0x24DADA,
0x00FFFF }, // Red-cyan
  { 0xFFFF00, 0xDADA24, 0xB6B648, 0x91916D, 0x6D6D91, 0x4848B6, 0x2424DA,
0x0000FF }, // Yellow-blue
  { 0x00FF00, 0x24DA24, 0x48B648, 0x6D916D, 0x916D91, 0xB648B6, 0xDA24DA,
0xFF00FF }, // Green-magenta

  // Adjacent colors (on color wheel).
  { 0xFF0000, 0xFF2400, 0xFF4800, 0xFF6D00, 0xFF9100, 0xFFB600, 0xFFDA00,
0xFFFF00 }, // Red-yellow
  { 0xFFFF00, 0xDAFF00, 0xB6FF00, 0x91FF00, 0x6DFF00, 0x48FF00, 0x24FF00,
0x00FF00 }, // Yellow-green
  { 0x00FF00, 0x00FF24, 0x00FF48, 0x00FF6D, 0x00FF91, 0x00FFB6, 0x00FFDA,
0x00FFFF }, // Green-cyan
  { 0x00FFFF, 0x00DAFF, 0x00B6FF, 0x0091FF, 0x006DFF, 0x0048FF, 0x0024FF,
0x0000FF }, // Cyan-blue
  { 0x0000FF, 0x2400FF, 0x4800FF, 0x6D00FF, 0x9100FF, 0xB600FF, 0xDA00FF,
0xFF00FF }, // Blue-magenta
  { 0xFF00FF, 0xFF00DA, 0xFF00B6, 0xFF0091, 0xFF006D, 0xFF0048, 0xFF0024,
0xFF0000 }, // Magenta-red

  // Other combos.
  { 0xFF0000, 0xDA2400, 0xB64800, 0x916D00, 0x6D9100, 0x48B600, 0x24DA00,
0x00FF00 }, // Red-green
  { 0xFFFF00, 0xDAFF24, 0xB6FF48, 0x91FF6D, 0x6DFF91, 0x48FFB6, 0x24FFDA,
0x00FFFF }, // Yellow-cyan
  { 0x00FF00, 0x00DA24, 0x00B648, 0x00916D, 0x006D91, 0x0048B6, 0x0024DA,
0x0000FF }, // Green-blue
  { 0x00FFFF, 0x24DAFF, 0x48B6FF, 0x6D91FF, 0x916DFF, 0xB648FF, 0xDA24FF,
0xFF00FF }, // Cyan-magenta
  { 0x0000FF, 0x2400DA, 0x4800B6, 0x6D0091, 0x91006D, 0xB60048, 0xDA0024,
0xFF0000 }, // Blue-red
  { 0xFF00FF, 0xFF24DA, 0xFF48B6, 0xFF6D91, 0xFF916D, 0xFFB648, 0xFFDA24,
0xFFFF00 }, // Magenta-yellow

  // Solid colors fading to dark.
  { 0xFF0000, 0xDF0000, 0xBF0000, 0x9F0000, 0x7F0000, 0x5F0000, 0x3F0000,
0x1F0000 }, // Red
  { 0xFF9900, 0xDF8500, 0xBF7200, 0x9F5F00, 0x7F4C00, 0x5F3900, 0x3F2600,
0x1F1300 }, // Orange
  { 0xFFFF00, 0xDFDF00, 0xBFBF00, 0x9F9F00, 0x7F7F00, 0x5F5F00, 0x3F3F00,
0x1F1F00 }, // Yellow
  { 0x00FF00, 0x00DF00, 0x00BF00, 0x009F00, 0x007F00, 0x005F00, 0x003F00,
0x001F00 }, // Green
  { 0x0000FF, 0x0000DF, 0x0000BF, 0x00009F, 0x00007F, 0x00005F, 0x00003F,
0x00001F }, // Blue
  { 0x4B0082, 0x410071, 0x380061, 0x2E0051, 0x250041, 0x1C0030, 0x120020,
0x090010 }, // Indigo
  { 0x8B00FF, 0x7900DF, 0x6800BF, 0x56009F, 0x45007F, 0x34005F, 0x22003F,
0x11001F }, // Violet
```

```
  { 0xFFFFFF, 0xDFDFDF, 0xBFBFBF, 0x9F9F9F, 0x7F7F7F, 0x5F5F5F, 0x3F3F3F,
0x1F1F1F }, // White

  // Rainbow colors.
  { 0xFF0000, 0xFF9900, 0xFFFF00, 0x00FF00, 0x0000FF, 0x4B0082, 0x8B00FF, 0xFFFFFF }
};

// List of animations speeds (in milliseconds).  This is how long an animation
spends before
// changing to the next step.  Higher values are slower.
const uint16_t speeds[5] = { 400, 200, 100, 50, 25 };

// Global state used by the sketch:
Adafruit_NeoPixel strip = Adafruit_NeoPixel(PIXEL_COUNT, PIXEL_PIN, PIXEL_TYPE);
volatile bool receiverFell = false;
uint8_t colorIndex = 0;
uint8_t animationIndex = 0;
uint8_t speedIndex = 2;


void setup(void) {
  // Setup IR receiver pin as an input.
  pinMode(IR_PIN, INPUT);
  // Initialize and clear the NeoPixel strip.
  strip.begin();
  strip.clear();
  strip.show(); // Initialize all pixels to 'off'
  // Enable an interrupt that's called when the IR receiver pin signal falls
  // from high to low.  This indicates a remote control code being received.
  attachInterrupt(0, receiverFalling, FALLING);
}

void loop(void) {
  // Main loop will first update all the pixels based on the current animation.
  for (int i = 0; i < PIXEL_COUNT; ++i) {
    switch (animationIndex) {
    case 0:
    {
      // Animation 0, solid color pulse of all pixels.
      uint8_t currentStep = (millis()/speeds[speedIndex])%(colorSteps*2-2);
      if (currentStep >= colorSteps) {
        currentStep = colorSteps-(currentStep-(colorSteps-2));
      }
      // Note that colors are stored in flash memory so they need to be read
      // using the pgmspace.h functions.
      uint32_t color = pgm_read_dword_near(&colorPalette[colorIndex][currentStep]);
      strip.setPixelColor(i, color);
      break;
    }
    case 1:
    {
      // Animation 1, moving color pulse.  Use position to change brightness.
      uint8_t currentStep = (millis()/speeds[speedIndex]+i)%(colorSteps*2-2);
      if (currentStep >= colorSteps) {
        currentStep = colorSteps-(currentStep-(colorSteps-2));
      }
      // Note that colors are stored in flash memory so they need to be read
      // using the pgmspace.h functions.
      uint32_t color = pgm_read_dword_near(&colorPalette[colorIndex][currentStep]);
      strip.setPixelColor(i, color);
      break;
    }
    }
  }
  // Next check for any IR remote commands.
  handleRemote();
  // Show the updated pixels.
  strip.show();
  // Again check for IR remote commands.
```

```
  handleRemote();
}

void receiverFalling() {
  // Interrupt function that's called when the IR receiver pin falls from high to
  // low and indicates the start of an IR command being received.  Note that
  // interrupts need to be very fast and perform as little processing as possible.
  // This just sets a global variable which the main loop will periodically check
  // and perform appropriate IR command decoding when necessary.
  receiverFell = true;
}

bool readNEC(uint16_t* result) {
  // Check if a NEC IR remote command can be read and decoded from the IR receiver.
  // If the command is decoded then the result is stored in the provided pointer and
  // true is returned.  Otherwise if the command was not decoded then false is
returned.
  // First check that a falling signal was detected and start reading pulses.
  if (!receiverFell) {
    return false;
  }
  // Read the first pulse with a large timeout since it's 9ms long, then
  // read subsequent pulses with a shorter 2ms timeout.
  uint32_t durations[33];
  durations[0] = pulseIn(IR_PIN, HIGH, 20000);
  for (uint8_t i = 1; i < 33; ++i) {
    durations[i] = pulseIn(IR_PIN, HIGH, 5000);
  }
  // Reset any state changed by the interrupt.
  receiverFell = false;
  // Check the received pulses are in a NEC format.
  // First verify the initial pulse is around 4.5ms long.
  if ((durations[0] < 4000) || (durations[1] > 5000)) {
    return false;
  }
  // Now read the bits from subsequent pulses.  Stop if any were a timeout (0
value).
  uint8_t data[4] = {0};
  for (uint8_t i=0; i<32; ++i) {
    if (durations[1+i] == 0) {
      return false; // Timeout
    }
    uint8_t b = durations[1+i] < 1000 ? 0 : 1;
    data[i/8] |= b << (i%8);
  }
  // Verify bytes and their inverse match.  Use the same two checks as the NEC IR
remote
  // library here: https://github.com/adafruit/Adafruit-NEC-remote-control-library
  if ((data[0] == (~data[1] & 0xFF)) && (data[2] == (~data[3] & 0xFF))) {
    *result = data[0] << 8 | data[2];
    return true;
  }
  else if ((data[0] == 0) && (data[1] == 0xBF) && (data[2] == (~data[3] & 0xFF))) {
    *result = data[2];
    return true;
  }
  else {
    // Something didn't match, fail!
    return false;
  }
}

void handleRemote() {
  // Check if an IR remote code was received and perform the appropriate action.
  // First read a code.
  uint16_t irCode;
  if (!readNEC(&irCode)) {
    return;
  }
```

```
  switch(irCode) {
  case COLOR_CHANGE:
    // Color change command, increment the current color animation and wrap
    // back to zero when past the end.
    colorIndex = (colorIndex+1)%colorCount;
    break;
  case ANIMATION_CHANGE:
    // Animation change command, increment the current animation type and wrap
    // back to zero when past the end.
    animationIndex = (animationIndex+1)%2;
    break;
  case SPEED_CHANGE:
    // Speed change command, increment the current speed and wrap back to zero
    // when past the end.
    speedIndex = (speedIndex+1)%5;
    break;
  case POWER_OFF:
    // Enter full power down sleep mode.
    // First turn off the NeoPixels.
    strip.clear();
    strip.show();
    while (true) {
      // Next disable the current falling interrupt and enable a low value
interrupt.
      // This is required because the ATtiny85 can't wake from a falling interrupt
      // and instead can only wake from a high or low value interrupt.
      detachInterrupt(0);
      attachInterrupt(0, receiverFalling, LOW);
      // Now enter full power down sleep mode.
      power_all_disable();
      set_sleep_mode(SLEEP_MODE_PWR_DOWN);
      sleep_mode();
      // Processor is currently asleep and will wake up when the IR receiver pin
      // is at a low value (i.e. when a IR command is received).
      // Sleep resumes here.  When awake power everything back up.
      power_all_enable();
      // Re-enable the falling interrupt.
      detachInterrupt(0);
      attachInterrupt(0, receiverFalling, FALLING);
      // Now wait up to 1 second for a second power on command to be received.
      // This is necessary because the first power on command will wake up the CPU
      // but happens a little too quickly to be reliably read.
      for (int i=0; i<200; ++i) {
        uint16_t irCode;
        if ((readNEC(&irCode) == 1) && (irCode == POWER_ON)) {
          // Second power on command was received, jump out of the power on loop and
          // return to normal operation.
          return;
        }
        delay(5);
      }
      // If no power on command was received within 1 second of awaking then the
      // code will loop back to the top and go to sleep again.
    }
  }
}
```

In particular change the PIXEL_COUNT value from 6 to the appropriate number of NeoPixels used in your hardware (like 10, etc.).

Notice you can change which remote control buttons control the sketch by adjusting the COLOR_CHANGE, ANIMATION_CHANGE, SPEED_CHANGE, POWER_OFF, and P

OWER_ON values at the bottom. For now keep the default values but be aware you can change them based on the remote codes in the comments above.

If you're not using a remote control there's a different set of configuration values you can change in the Techno_Tiki_No_Remote_Control sketch:

Change the PIXEL_COUNT value from 6 to the number of pixels used in your hardware (like 10, etc).

In addition you can adjust the speed and type of animation by changing the SPEED_MS and ANIMATION values. Read the comments to see how these values will change the animation behavior. Remember there is no remote control so the only way to change animation is by adjusting the sketch and uploading it to the hardware again.

You can also choose the animation color in this part of the sketch:

```
// Color animation values.  Each value is a 24-bit RGB color value that will be
displayed
// at that current step in the animation.  Make sure only ONE row is uncommented
below!
const int colorSteps = 8;  // Number of steps in the animation.
const uint32_t colorAnimation[colorSteps] PROGMEM =
  // Complimentary colors
  //{ 0xFF0000, 0xDA2424, 0xB64848, 0x916D6D, 0x6D9191, 0x48B6B6, 0x24DADA,
0x00FFFF }; // Red-cyan
  //{ 0xFFFF00, 0xDADA24, 0xB6B648, 0x91916D, 0x6D6D91, 0x4848B6, 0x2424DA,
0x0000FF }; // Yellow-blue
  //{ 0x00FF00, 0x24DA24, 0x48B648, 0x6D916D, 0x916D91, 0xB648B6, 0xDA24DA,
0xFF00FF }; // Green-magenta

  // Adjacent colors (on color wheel).
  { 0xFF0000, 0xFF2400, 0xFF4800, 0xFF6D00, 0xFF9100, 0xFFB600, 0xFFDA00,
0xFFFF00 }; // Red-yellow
  //{ 0xFFFF00, 0xDAFF00, 0xB6FF00, 0x91FF00, 0x6DFF00, 0x48FF00, 0x24FF00,
0x00FF00 }; // Yellow-green
  //{ 0x00FF00, 0x00FF24, 0x00FF48, 0x00FF6D, 0x00FF91, 0x00FFB6, 0x00FFDA,
0x00FFFF }; // Green-cyan
  //{ 0x00FFFF, 0x00DAFF, 0x00B6FF, 0x0091FF, 0x006DFF, 0x0048FF, 0x0024FF,
0x0000FF }; // Cyan-blue
  //{ 0x0000FF, 0x2400FF, 0x4800FF, 0x6D00FF, 0x9100FF, 0xB600FF, 0xDA00FF,
0xFF00FF }; // Blue-magenta
  //{ 0xFF00FF, 0xFF00DA, 0xFF00B6, 0xFF0091, 0xFF006D, 0xFF0048, 0xFF0024,
0xFF0000 }; // Magenta-red

  // Other combos.
  //{ 0xFF0000, 0xDA2400, 0xB64800, 0x916D00, 0x6D9100, 0x48B600, 0x24DA00,
0x00FF00 }; // Red-green
  //{ 0xFFFF00, 0xDAFF24, 0xB6FF48, 0x91FF6D, 0x6DFF91, 0x48FFB6, 0x24FFDA,
0x00FFFF }; // Yellow-cyan
  //{ 0x00FF00, 0x00DA24, 0x00B648, 0x00916D, 0x006D91, 0x0048B6, 0x0024DA,
0x0000FF }; // Green-blue
  //{ 0x00FFFF, 0x24DAFF, 0x48B6FF, 0x6D91FF, 0x916DFF, 0xB648FF, 0xDA24FF,
0xFF00FF }; // Cyan-magenta
  //{ 0x0000FF, 0x2400DA, 0x4800B6, 0x6D0091, 0x91006D, 0xB60048, 0xDA0024,
0xFF0000 }; // Blue-red
  //{ 0xFF00FF, 0xFF24DA, 0xFF48B6, 0xFF6D91, 0xFF916D, 0xFFB648, 0xFFDA24,
0xFFFF00 }; // Magenta-yellow

  // Solid colors fading to dark.
  //{ 0xFF0000, 0xDF0000, 0xBF0000, 0x9F0000, 0x7F0000, 0x5F0000, 0x3F0000,
```

```
0x1F0000 }; // Red
  //{ 0xFF9900, 0xDF8500, 0xBF7200, 0x9F5F00, 0x7F4C00, 0x5F3900, 0x3F2600,
0x1F1300 }; // Orange
  //{ 0xFFFF00, 0xDFDF00, 0xBFBF00, 0x9F9F00, 0x7F7F00, 0x5F5F00, 0x3F3F00,
0x1F1F00 }; // Yellow
  //{ 0x00FF00, 0x00DF00, 0x00BF00, 0x009F00, 0x007F00, 0x005F00, 0x003F00,
0x001F00 }; // Green
  //{ 0x0000FF, 0x0000DF, 0x0000BF, 0x00009F, 0x00007F, 0x00005F, 0x00003F,
0x00001F }; // Blue
  //{ 0x4B0082, 0x410071, 0x380061, 0x2E0051, 0x250041, 0x1C0030, 0x120020,
0x090010 }; // Indigo
  //{ 0x8B00FF, 0x7900DF, 0x6800BF, 0x56009F, 0x45007F, 0x34005F, 0x22003F,
0x11001F }; // Violet
  //{ 0xFFFFFF, 0xDFDFDF, 0xBFBFBF, 0x9F9F9F, 0x7F7F7F, 0x5F5F5F, 0x3F3F3F,
0x1F1F1F }; // White

  // Rainbow colors.
  //{ 0xFF0000, 0xFF9900, 0xFFFF00, 0x00FF00, 0x0000FF, 0x4B0082, 0x8B00FF,
0xFFFFFF };
```

By default a red-yellow animation will be used with the sketch. However you can comment the red-yellow line and uncomment a different line to change the animation. Remember only one line should be uncommented! If you have multiple lines uncommented then you'll receive an error trying to compile the sketch.

For example to change to the rainbow color animation at the end you would change the section to look like:

```
// Color animation values.  Each value is a 24-bit RGB color value that will be
displayed
// at that current step in the animation.  Make sure only ONE row is uncommented
below!
const int colorSteps = 8;  // Number of steps in the animation.
const uint32_t colorAnimation[colorSteps] PROGMEM =
  // Complimentary colors
  //{ 0xFF0000, 0xDA2424, 0xB64848, 0x916D6D, 0x6D9191, 0x48B6B6, 0x24DADA,
0x00FFFF }; // Red-cyan
  //{ 0xFFFF00, 0xDADA24, 0xB6B648, 0x91916D, 0x6D6D91, 0x4848B6, 0x2424DA,
0x0000FF }; // Yellow-blue
  //{ 0x00FF00, 0x24DA24, 0x48B648, 0x6D916D, 0x916D91, 0xB648B6, 0xDA24DA,
0xFF00FF }; // Green-magenta

  // Adjacent colors (on color wheel).
  //{ 0xFF0000, 0xFF2400, 0xFF4800, 0xFF6D00, 0xFF9100, 0xFFB600, 0xFFDA00,
0xFFFF00 }; // Red-yellow
  //{ 0xFFFF00, 0xDAFF00, 0xB6FF00, 0x91FF00, 0x6DFF00, 0x48FF00, 0x24FF00,
0x00FF00 }; // Yellow-green
  //{ 0x00FF00, 0x00FF24, 0x00FF48, 0x00FF6D, 0x00FF91, 0x00FFB6, 0x00FFDA,
0x00FFFF }; // Green-cyan
  //{ 0x00FFFF, 0x00DAFF, 0x00B6FF, 0x0091FF, 0x006DFF, 0x0048FF, 0x0024FF,
0x0000FF }; // Cyan-blue
  //{ 0x0000FF, 0x2400FF, 0x4800FF, 0x6D00FF, 0x9100FF, 0xB600FF, 0xDA00FF,
0xFF00FF }; // Blue-magenta
  //{ 0xFF00FF, 0xFF00DA, 0xFF00B6, 0xFF0091, 0xFF006D, 0xFF0048, 0xFF0024,
0xFF0000 }; // Magenta-red

  // Other combos.
  //{ 0xFF0000, 0xDA2400, 0xB64800, 0x916D00, 0x6D9100, 0x48B600, 0x24DA00,
0x00FF00 }; // Red-green
  //{ 0xFFFF00, 0xDAFF24, 0xB6FF48, 0x91FF6D, 0x6DFF91, 0x48FFB6, 0x24FFDA,
0x00FFFF }; // Yellow-cyan
  //{ 0x00FF00, 0x00DA24, 0x00B648, 0x00916D, 0x006D91, 0x0048B6, 0x0024DA,
0x0000FF }; // Green-blue
```

```
  //{ 0x00FFFF, 0x24DAFF, 0x48B6FF, 0x6D91FF, 0x916DFF, 0xB648FF, 0xDA24FF,
0xFF00FF }; // Cyan-magenta
  //{ 0x0000FF, 0x2400DA, 0x4800B6, 0x6D0091, 0x91006D, 0xB60048, 0xDA0024,
0xFF0000 }; // Blue-red
  //{ 0xFF00FF, 0xFF24DA, 0xFF48B6, 0xFF6D91, 0xFF916D, 0xFFB648, 0xFFDA24,
0xFFFF00 }; // Magenta-yellow

  // Solid colors fading to dark.
  //{ 0xFF0000, 0xDF0000, 0xBF0000, 0x9F0000, 0x7F0000, 0x5F0000, 0x3F0000,
0x1F0000 }; // Red
  //{ 0xFF9900, 0xDF8500, 0xBF7200, 0x9F5F00, 0x7F4C00, 0x5F3900, 0x3F2600,
0x1F1300 }; // Orange
  //{ 0xFFFF00, 0xDFDF00, 0xBFBF00, 0x9F9F00, 0x7F7F00, 0x5F5F00, 0x3F3F00,
0x1F1F00 }; // Yellow
  //{ 0x00FF00, 0x00DF00, 0x00BF00, 0x009F00, 0x007F00, 0x005F00, 0x003F00,
0x001F00 }; // Green
  //{ 0x0000FF, 0x0000DF, 0x0000BF, 0x00009F, 0x00007F, 0x00005F, 0x00003F,
0x00001F }; // Blue
  //{ 0x4B0082, 0x410071, 0x380061, 0x2E0051, 0x250041, 0x1C0030, 0x120020,
0x090010 }; // Indigo
  //{ 0x8B00FF, 0x7900DF, 0x6800BF, 0x56009F, 0x45007F, 0x34005F, 0x22003F,
0x11001F }; // Violet
  //{ 0xFFFFFF, 0xDFDFDF, 0xBFBFBF, 0x9F9F9F, 0x7F7F7F, 0x5F5F5F, 0x3F3F3F,
0x1F1F1F }; // White

  // Rainbow colors.
  { 0xFF0000, 0xFF9900, 0xFFFF00, 0x00FF00, 0x0000FF, 0x4B0082, 0x8B00FF,
0xFFFFFF };
```

Upload

Once you've configured the sketch for your hardware you're ready to upload it to the Gemma board. Before you upload make sure you've setup the Arduino IDE to program Gemma. In the Tools -> Board menu select the Arduino Gemma board (you can use this option even if you're using the Adafruit Gemma boards). Also in the Tool s -> Programmer menu select the USBtinyISP option.

Make sure the Gemma's USB micro/mini connector is connected to your computer. If your Gemma board has an on/off switch slide it into the on position. Press the reset button on the Gemma and you should see its red light start pulsing as the bootloader waits for a sketch to be uploaded. In the Arduino IDE press the upload button or use the Sketch -> Upload command. After a few moments you should see the sketch uploaded to the hardware (on Linux systems you might see broken pipe errors that can be ignored).

If you receive an error be sure to read the Gemma guide () and confirm you can upload a basic LED blinking sketch.

If you're using more than 6 LEDs you might not see them light up because they can draw more current than some USB ports provide. Unplug the Gemma from your computer and plug in the lithium ion battery. Ensure the Gemma on/off switch is in the on position (if your Gemma has a switch). Verify that the Gemma board turns on

and after it waits a few seconds in the bootloader (with a pulsing red LED) it should turn on all the LEDs and start animating.

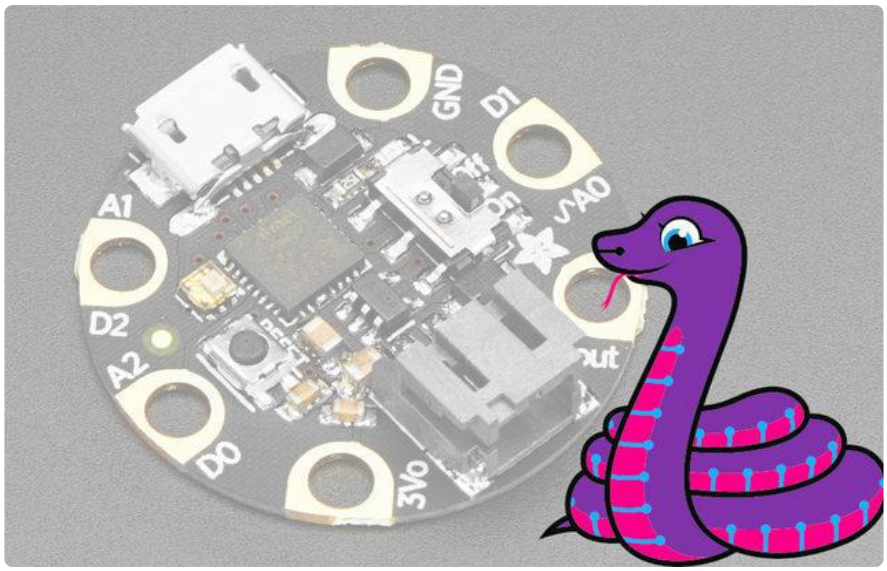
If you're using an IR receiver and remote control check that you can control the hardware behavior with the remote control. Point the remote control at the front of the IR receiver. By default the sketch is programmed to use these buttons on the remote:

- Vol- - This turns off the hardware and puts it into a low power sleep mode where it draws about 11mA of current (compared to ~200+ mA while animating pixels).
- Vol+ - When the hardware is in low power sleep mode press this button twice to turn it back on into full power mode. Remember you need to press the button twice.
- Up Arrow - This cycles through the animation speeds.
- Left Arrow - This cycles through the two different types of animation.
- Right Arrow - This cycles through the different colors of animation.

Try pressing the buttons and verify they work as expected. If the buttons aren't working make sure the remote control has a fresh battery and the plastic tab is removed which protects the battery during shipping. Also ensure the IR receiver is correctly connected to the Gemma (its signal line must be on pin D2). Note that with Circuit Playground Express you need to hold the remote control very close to the board as its IR receiver sensitivity through the glass beads/vase filler isn't as good as a stand-alone sensor.

Once your hardware is working you're ready to move on to finish assembling the project.

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 and CircuitPython Express boards. The original GEMMA 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 "main.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.

There are three different CircuitPython sketches to choose from:

- Techno_Tiki_No_Remote_Control () - This sketch is for a hardware setup that does not use the IR receiver and remote control.

- Techno_Tiki_With_Remote_Control () - This sketch is for a hardware setup that uses the IR receiver and remote control.
- Techno_Tiki_Circuit_Playground_Express () - This sketch is for a hardware setup using the Circuit Playground Express () board. Be sure to follow the Circuit Playground Express guide. ()

Required Libraries

Additional libraries will be necessary to run all of the CircuitPython examples on this page.

- neopixel.mpy - Necessary for all three examples.
- adafruit_irremote.mpy - Necessary for With_Remote_Control and Circuit_Playground_Express
- adafruit_circuitplayground- Necessary only for Circuit_Playground_Express

These libraries are available for download here:

Adafruit CircuitPython Library Bundle

Install Circuit Python Libraries

Now that we have all of the libraries and know which ones this project needs, we'll need to copy them onto the Circuit Playground Express USB drive (which will be named CIRCUITPY after flashing the firmware). In the CIRCUITPY drive, create a new folder and name it "lib". Then, copy the libraries to that "lib" folder. The lib folder should contain neopixel.mpy, adafruit_irremote.mpy and adafruit_circuitplayground .

Techno_Tiki_No_Remote_Control

```
# SPDX-FileCopyrightText: 2018 Mikey Sklar for Adafruit Industries
# SPDX-FileCopyrightText: 2020 Erin St. Blaine for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import neopixel

pixel_count = 6         # Number of NeoPixels
pixel_pin = board.D1    # Pin where NeoPixels are connected

speed = .1              # Animation speed (in seconds).
                        # This is how long to spend in a single animation frame.
```

```
                        # Higher values are slower.
                        # Good values to try are 400, 200, 100, 50, 25, etc.

animation = 0           # Type of animation, can be one of these values:
                        # 0 - Solid color pulse
                        # 1 - Moving color pulse

color_steps = 8         # Number of steps in the animation.

brightness = 1.0        # 0-1, higher number is brighter


# Adjacent colors (on color wheel).
# red yellow
color_animation = ([255, 0, 0], [255, 36, 0], [255, 72, 0], [255, 109, 0],
                   [255, 145, 0], [255, 182, 0], [255, 218, 0], [255, 255, 0])

# Adjacent colors
#([255, 0, 0], [255, 36, 0], [255, 72, 0], [255, 109, 0],
# [255, 145, 0], [255, 182, 0], [255, 218, 0], [255, 255, 0])           # red yellow
#([255, 255, 0], [218, 255, 0], [182, 255, 0], [145, 255, 0],
# [109, 255, 0], [72, 255, 0], [36, 255, 0], [0, 255, 0])               # yello
green
#([0, 255, 0], [0, 255, 36], [0, 255, 72], [0, 255, 109],
# [0, 255, 145], [0, 255, 182], [0, 255, 218], [0, 255, 255])           # green cyan
#([0, 255, 255], [0, 218, 255], [0, 182, 255], [0, 145, 255],
# [0, 109, 255], [0, 72, 255], [0, 36, 255], [0, 0, 255])               # cyan blue
#([0, 0, 255], [36, 0, 255], [72, 0, 255], [109, 0, 255],
# [145, 0, 255], [182, 0, 255], [218, 0, 255], [255, 0, 255])           # blue
magenta
#([255, 0, 255], [255, 0, 218], [255, 0, 182], [255, 0, 145],
# [255, 0, 109], [255, 0, 72], [255, 0, 36], [255, 0, 0])               # magenta
red

# Complimentary colors
#([255, 0, 0], [218, 36, 36], [182, 72, 72], [145, 109, 109],
# [109, 145, 145], [72, 182, 182], [36, 218, 218], [0, 255, 255])       # red cyan
#([255, 255, 0], [218, 218, 36], [182, 182, 72], [145, 145, 109],
# [109, 109, 145], [72, 72, 182], [36, 36, 218], [0, 0, 255])           # yellow
blue
#([0, 255, 0], [36, 218, 36], [72, 182, 72], [109, 145, 109],
# [145, 109, 145], [182, 72, 182], [218, 36, 218], [255, 0, 255])       # green
magenta

# Other combos
#([255, 0, 0], [218, 36, 0], [182, 72, 0], [145, 109, 0],
# [109, 145, 0], [72, 182, 0], [36, 218, 0], [0, 255, 0])               # red green
#([255, 255, 0], [218, 255, 36], [182, 255, 72], [145, 255, 109],
# [109, 255, 145], [72, 255, 182], [36, 255, 218], [0, 255, 255])       # yellow
cyan
#([0, 255, 0], [0, 218, 36], [0, 182, 72], [0, 145, 109],
# [0, 109, 145], [0, 72, 182], [0, 36, 218], [0, 0, 255])               # green blue
#([0, 255, 255], [36, 218, 255], [72, 182, 255], [109, 145, 255],
# [145, 109, 255], [182, 72, 255], [218, 36, 255], [255, 0, 255])       # cyan
magenta
#([0, 0, 255], [36, 0, 218], [72, 0, 182], [109, 0, 145],
# [145, 0, 109], [182, 0, 72], [218, 0, 36], [255, 0, 0])               # blue red
#([255, 0, 255], [255, 36, 218], [255, 72, 182], [255, 109, 145],
# [255, 145, 109], [255, 182, 72], [255, 218, 36], [255, 255, 0])       # magenta
yellow

# Solid colors fading to dark
#([255, 0, 0], [223, 0, 0], [191, 0, 0], [159, 0, 0],
# [127, 0, 0], [95, 0, 0], [63, 0, 0], [31, 0, 0])                      #  red
#([255, 153, 0], [223, 133, 0], [191, 114, 0], [159, 95, 0],
# [127, 76, 0], [95, 57, 0], [63, 38, 0], [31, 19, 0])                  # orange
#([255, 255, 0], [223, 223, 0], [191, 191, 0], [159, 159, 0],
# [127, 127, 0], [95, 95, 0], [63, 63, 0], [31, 31, 0])                 # yellow
#([0, 255, 0], [0, 223, 0], [0, 191, 0], [0, 159, 0],
```

```
# [0, 127, 0], [0, 95, 0], [0, 63, 0], [0, 31, 0])                        # green
#([0, 0, 255], [0, 0, 223], [0, 0, 191], [0, 0, 159],
# [0, 0, 127], [0, 0, 95], [0, 0, 63], [0, 0, 31])                        # blue
#([75, 0, 130], [65, 0, 113], [56, 0, 97], [46, 0, 81],
# [37, 0, 65], [28, 0, 48], [18, 0, 32], [9, 0, 16])                      # indigo
#([139, 0, 255], [121, 0, 223], [104, 0, 191], [86, 0, 159],
# [69, 0, 127], [52, 0, 95], [34, 0, 63], [17, 0, 31])                    # violet
#([255, 255, 255], [223, 223, 223], [191, 191, 191], [159, 159, 159],
# [127, 127, 127], [95, 95, 95], [63, 63, 63], [31, 31, 31])              # white
#([255, 0, 0], [255, 153, 0], [255, 255, 0], [0, 255, 0],
# [0, 0, 255], [75, 0, 130], [139, 0, 255], [255, 255, 255])              # rainbow
colors

# Global state used by the sketch
strip = neopixel.NeoPixel(pixel_pin, pixel_count, brightness=1, auto_write=False)

while True:  # Loop forever...

    # Main loop will update all the pixels based on the animation.
    for i in range(pixel_count):

        # Animation 0, solid color pulse of all pixels.
        if animation == 0:
            current_step = (time.monotonic() / speed) % (color_steps * 2 - 2)
            if current_step >= color_steps:
                current_step = color_steps - (current_step - (color_steps - 2))

        # Animation 1, moving color pulse.  Use position to change brightness.
        elif animation == 1:
            current_step = (time.monotonic() / speed + i) % (color_steps * 2 - 2)
            if current_step >= color_steps:
                current_step = color_steps - (current_step - (color_steps - 2))

        strip[i] = color_animation[int(current_step)]

    # Show the updated pixels.
    strip.show()
```

Techno_Tiki_With_Remote_Control

```
# SPDX-FileCopyrightText: 2018 Tony DiCola for Adafruit Industries
# SPDX-FileCopyrightText: 2018 Mikey Sklar for Adafruit Industries
# SPDX-FileCopyrightText: 2020 Erin St. Blaine for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Techno-Tiki RGB LED Torch with IR Remote Control
# Created by Tony DiCola for Arduino
# Ported to CircuitPython by Mikey Sklar
#
# See guide at: https://learn.adafruit.com/techno-tiki-rgb-led-torch
#
# Released under a MIT license: http://opensource.org/licenses/MIT

import time
import board
import neopixel
import adafruit_irremote
import pulseio
# pylint: disable=global-statement

pixel_pin = board.D1    # Pin where NeoPixels are connected

pixel_count = 6         # Number of NeoPixels
```

```
speed = .1              # Animation speed (in seconds).
                        # This is how long to spend in a single animation frame.
                        # Higher values are slower.
                        # Good values to try are 400, 200, 100, 50, 25, etc.

animation = 1           # Type of animation, can be one of these values:
                        # 0 - Solid color pulse
                        # 1 - Moving color pulse


brightness = 1.0        # 0-1, higher number is brighter

ir_code_min = 60
ir_code_max = 70
pulsein = pulseio.PulseIn(board.D2, maxlen=100, idle_state=True)
decoder = adafruit_irremote.GenericDecode()

# Adafruit IR Remote Codes:
# Button       Code         Button  Code
# -----------  ------       ------  -----
# VOL-:        255            0/10+:    207
# Play/Pause:  127            1:        247
# VOL+:        191            2:        119
# SETUP:       223            3:        183
# STOP/MODE:   159            4:        215
# UP:          95             5:        87
# DOWN:        79             6:        151
# LEFT:        239            7:        231
# RIGHT:       175            8:        103
# ENTER/SAVE:  111            9:        167
# Back:        143

color_change = 175        # Button that cycles through color animations.
animation_change = 239    # Button that cycles through animation types (only two
supported).
speed_change = 95        # Button that cycles through speed choices.
power_off = 255            # Button that turns off the pixels.
power_on = 191            # Button that turns on the pixels.  Must be pressed twice
to register!

# Build lookup table/palette for the color animations so they aren't computed at
runtime.
# The colorPalette two-dimensional array below has a row for each color animation
and a column
# for each step within the animation.  Each value is a 24-bit RGB color.  By
looping through
# the columns of a row the colors of pixels will animate.
color_steps = 8         # Number of steps in the animation.
color_count = 23        # number of columns/steps

color_palette = [
    # Complimentary colors
    ([255, 0, 0], [218, 36, 36], [182, 72, 72], [145, 109, 109],
     [109, 145, 145], [72, 182, 182], [36, 218, 218], [0, 255, 255]),       # red
cyan
    ([255, 255, 0], [218, 218, 36], [182, 182, 72], [145, 145, 109],
     [109, 109, 145], [72, 72, 182], [36, 36, 218], [0, 0, 255]),           #
yellow blue
    ([0, 255, 0], [36, 218, 36], [72, 182, 72], [109, 145, 109],
     [145, 109, 145], [182, 72, 182], [218, 36, 218], [255, 0, 255]),       # green
magenta

    # Adjacent colors (on color wheel).
    ([255, 255, 0], [218, 255, 0], [182, 255, 0], [145, 255, 0],
     [109, 255, 0], [72, 255, 0], [36, 255, 0], [0, 255, 0]),               # yello
green
    ([0, 255, 0], [0, 255, 36], [0, 255, 72], [0, 255, 109],
     [0, 255, 145], [0, 255, 182], [0, 255, 218], [0, 255, 255]),           # green
cyan
```

```
    ([0, 255, 255], [0, 218, 255], [0, 182, 255], [0, 145, 255],
     [0, 109, 255], [0, 72, 255], [0, 36, 255], [0, 0, 255]),               # cyan
blue
    ([0, 0, 255], [36, 0, 255], [72, 0, 255], [109, 0, 255],
     [145, 0, 255], [182, 0, 255], [218, 0, 255], [255, 0, 255]),           # blue
magenta
    ([255, 0, 255], [255, 0, 218], [255, 0, 182], [255, 0, 145],
     [255, 0, 109], [255, 0, 72], [255, 0, 36], [255, 0, 0]),               #
magenta red

    # Other combos
    ([255, 0, 0], [218, 36, 0], [182, 72, 0], [145, 109, 0],
     [109, 145, 0], [72, 182, 0], [36, 218, 0], [0, 255, 0]),               # red
green
    ([255, 255, 0], [218, 255, 36], [182, 255, 72], [145, 255, 109],
     [109, 255, 145], [72, 255, 182], [36, 255, 218], [0, 255, 255]),       #
yellow cyan
    ([0, 255, 0], [0, 218, 36], [0, 182, 72], [0, 145, 109],
     [0, 109, 145], [0, 72, 182], [0, 36, 218], [0, 0, 255]),               # green
blue
    ([0, 255, 255], [36, 218, 255], [72, 182, 255], [109, 145, 255],
     [145, 109, 255], [182, 72, 255], [218, 36, 255], [255, 0, 255]),       # cyan
magenta
    ([0, 0, 255], [36, 0, 218], [72, 0, 182], [109, 0, 145],
     [145, 0, 109], [182, 0, 72], [218, 0, 36], [255, 0, 0]),               # blue
red
    ([255, 0, 255], [255, 36, 218], [255, 72, 182], [255, 109, 145],
     [255, 145, 109], [255, 182, 72], [255, 218, 36], [255, 255, 0]),       #
magenta yellow

    # Solid colors fading to dark.
    ([255, 0, 0], [223, 0, 0], [191, 0, 0], [159, 0, 0],
     [127, 0, 0], [95, 0, 0], [63, 0, 0], [31, 0, 0]),                      # red
    ([255, 153, 0], [223, 133, 0], [191, 114, 0], [159, 95, 0],
     [127, 76, 0], [95, 57, 0], [63, 38, 0], [31, 19, 0]),                  # orange
    ([255, 255, 0], [223, 223, 0], [191, 191, 0], [159, 159, 0],
     [127, 127, 0], [95, 95, 0], [63, 63, 0], [31, 31, 0]),                 # yellow
    ([0, 255, 0], [0, 223, 0], [0, 191, 0], [0, 159, 0],
     [0, 127, 0], [0, 95, 0], [0, 63, 0], [0, 31, 0]),                      # green
    ([0, 0, 255], [0, 0, 223], [0, 0, 191], [0, 0, 159],
     [0, 0, 127], [0, 0, 95], [0, 0, 63], [0, 0, 31]),                      # blue
    ([75, 0, 130], [65, 0, 113], [56, 0, 97], [46, 0, 81],
     [37, 0, 65], [28, 0, 48], [18, 0, 32], [9, 0, 16]),                    # indigo
    ([139, 0, 255], [121, 0, 223], [104, 0, 191], [86, 0, 159],
     [69, 0, 127], [52, 0, 95], [34, 0, 63], [17, 0, 31]),                  # violet
    ([255, 255, 255], [223, 223, 223], [191, 191, 191], [159, 159, 159],
     [127, 127, 127], [95, 95, 95], [63, 63, 63], [31, 31, 31]),            # white
    ([255, 0, 0], [255, 153, 0], [255, 255, 0], [0, 255, 0],
     [0, 0, 255], [75, 0, 130], [139, 0, 255], [255, 255, 255])             #
rainbow colors
]

# List of animations speeds (in seconds).  This is how long an animation spends
before
# changing to the next step.  Higher values are slower.
speeds = [.4, .2, .1, .05, .025]

# Global state used by the sketch
strip = neopixel.NeoPixel(pixel_pin, pixel_count, brightness=brightness,
auto_write=False)
color_index = 0
animation_index = 0
speed_index = 2

def read_NEC():
# Check if a NEC IR remote command is the correct length.
# Save the third decoded value as our unique identifier.
    pulses = decoder.read_pulses(pulsein, max_pulse=5000)
    command = None
```

```
    try:
        if len(pulses) >= ir_code_min and len(pulses) <= ir_code_max:
            code = decoder.decode_bits(pulses)
            if len(code) > 3:
                command = code[2]
    except adafruit_irremote.IRNECRepeatException:  # Catches the repeat signal
        pass
    except adafruit_irremote.IRDecodeException:  # Failed to decode
        pass

    return command

def handle_remote():
    global color_index, animation_index, speed_index

    ir_code = read_NEC()

    if ir_code is None:
        time.sleep(.1)
        return

    if ir_code == color_change:
        color_index = (color_index + 1) % color_count
    elif ir_code == animation_change:
        animation_index = (animation_index + 1) % 2
    elif ir_code == speed_change:
        speed_index = (speed_index + 1) % 5
    elif ir_code == power_off:
        strip.fill([0, 0, 0])
        strip.show()

while True:  # Loop forever...

    # Main loop will update all the pixels based on the animation.
    for i in range(pixel_count):

        # Animation 0, solid color pulse of all pixels.
        if animation_index == 0:
            current_step = (time.monotonic() / speeds[speed_index]) % (color_steps *
2 - 2)
            if current_step >= color_steps:
                current_step = color_steps - (current_step - (color_steps - 2))

        # Animation 1, moving color pulse.  Use position to change brightness.
        elif animation == 1:
            current_step = (time.monotonic() / speeds[speed_index] + i) %
(color_steps * 2 - 2)
            if current_step >= color_steps:
                current_step = color_steps - (current_step - (color_steps - 2))

        strip[i] = color_palette[int(color_index)][int(current_step)]

    # Next check for any IR remote commands.
    handle_remote()

    # Show the updated pixels.
    strip.show()

    # Next check for any IR remote commands.
    handle_remote()
```

Techno_Tiki_Circuit_Playground_Express

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

# Techno-Tiki RGB LED Torch with IR Remote Control
# Created by Tony DiCola for Arduino
# Ported to CircuitPython by Mikey Sklar
#
# See guide at: https://learn.adafruit.com/techno-tiki-rgb-led-torch
#
# Released under a MIT license: http://opensource.org/licenses/MIT

import time
import adafruit_irremote
import board
import neopixel
import pulseio

# pylint: disable=global-statement

brightness = 1        # 0-1, higher number is brighter

# Adafruit IR Remote Codes:
# Button       Code         Button  Code
# -----------  ------       ------  -----
# VOL-:        255            0/10+:    207
# Play/Pause:  127            1:        247
# VOL+:        191            2:        119
# SETUP:       223            3:        183
# STOP/MODE:   159            4:        215
# UP:          95            5:        87
# DOWN:        79            6:        151
# LEFT:        239            7:        231
# RIGHT:       175            8:        103
# ENTER/SAVE:  111            9:        167
# Back:        143

# Adafruit IR Remote Codes:
volume_down = 255
play_pause = 127
volume_up = 191
setup = 223
up_arrow = 95
stop_mode = 159
left_arrow = 239
enter_save = 111
right_arrow = 175
down_arrow = 79
num_1 = 247
num_2 = 119
num_3 = 183
num_4 = 215
num_5 = 87
num_6 = 151
num_7 = 231
num_8 = 103
num_9 = 167

# Define which remote buttons are associated with sketch actions.
color_change = right_arrow      # Button that cycles through color animations.
animation_change = left_arrow   # Button that cycles through animation types (only
two supported).
speed_change = up_arrow         # Button that cycles through speed choices.
```

```
power_off = volume_down         # Button that turns off the pixels.
power_on = volume_up            # Button that turns on the pixels. Must be pressed
twice.

# The colorPalette two-dimensional array below has a row for each color animation
and a column
# for each step within the animation.  Each value is a 24-bit RGB color.  By
looping through
# the columns of a row the colors of pixels will animate.
color_steps = 8         # Number of steps in the animation.
color_count = 8         # number of columns/steps

# Build lookup table/palette for the color animations so they aren't computed at
runtime.
color_palette = [
    # Complimentary colors
    ([255, 0, 0], [218, 36, 36], [182, 72, 72], [145, 109, 109],
     [109, 145, 145], [72, 182, 182], [36, 218, 218], [0, 255, 255]),   # red cyan
    ([255, 255, 0], [218, 218, 36], [182, 182, 72], [145, 145, 109],
     [109, 109, 145], [72, 72, 182], [36, 36, 218], [0, 0, 255]),       # yellow
blue
    ([0, 255, 0], [36, 218, 36], [72, 182, 72], [109, 145, 109],
     [145, 109, 145], [182, 72, 182], [218, 36, 218], [255, 0, 255]),   # green
magenta

    # Adjacent colors (on color wheel).
    ([255, 255, 0], [218, 255, 0], [182, 255, 0], [145, 255, 0],
     [109, 255, 0], [72, 255, 0], [36, 255, 0], [0, 255, 0]),           # yello
green
    ([0, 255, 0], [0, 255, 36], [0, 255, 72], [0, 255, 109],
     [0, 255, 145], [0, 255, 182], [0, 255, 218], [0, 255, 255]),       # green cyan
    ([0, 255, 255], [0, 218, 255], [0, 182, 255], [0, 145, 255],
     [0, 109, 255], [0, 72, 255], [0, 36, 255], [0, 0, 255]),           # cyan blue
    ([0, 0, 255], [36, 0, 255], [72, 0, 255], [109, 0, 255],
     [145, 0, 255], [182, 0, 255], [218, 0, 255], [255, 0, 255]),       # blue
magenta
    ([255, 0, 255], [255, 0, 218], [255, 0, 182], [255, 0, 145],
     [255, 0, 109], [255, 0, 72], [255, 0, 36], [255, 0, 0])            # magenta
red
]

   # Other combos
   #([255, 0, 0], [218, 36, 0], [182, 72, 0], [145, 109, 0],
    #[109, 145, 0], [72, 182, 0], [36, 218, 0], [0, 255, 0]),           # red green
   #([255, 255, 0], [218, 255, 36], [182, 255, 72], [145, 255, 109],
    #[109, 255, 145], [72, 255, 182], [36, 255, 218], [0, 255, 255]),   # yellow
cyan
   #([0, 255, 0], [0, 218, 36], [0, 182, 72], [0, 145, 109],
    #[0, 109, 145], [0, 72, 182], [0, 36, 218], [0, 0, 255]),           # green blue
   #([0, 255, 255], [36, 218, 255], [72, 182, 255], [109, 145, 255],
    #[145, 109, 255], [182, 72, 255], [218, 36, 255], [255, 0, 255]),   # cyan
magenta
   #([0, 0, 255], [36, 0, 218], [72, 0, 182], [109, 0, 145],
    #[145, 0, 109], [182, 0, 72], [218, 0, 36], [255, 0, 0]),           # blue red
   #([255, 0, 255], [255, 36, 218], [255, 72, 182], [255, 109, 145],
    #[255, 145, 109], [255, 182, 72], [255, 218, 36], [255, 255, 0]),   # magenta
yellow

   # Solid colors fading to dark.
   #([255, 0, 0], [223, 0, 0], [191, 0, 0], [159, 0, 0],
    #[127, 0, 0], [95, 0, 0], [63, 0, 0], [31, 0, 0]),                  # red
   #([255, 153, 0], [223, 133, 0], [191, 114, 0], [159, 95, 0],
    #[127, 76, 0], [95, 57, 0], [63, 38, 0], [31, 19, 0]),              # orange
   #([255, 255, 0], [223, 223, 0], [191, 191, 0], [159, 159, 0],
    #[127, 127, 0], [95, 95, 0], [63, 63, 0], [31, 31, 0]),             # yellow
   #([0, 255, 0], [0, 223, 0], [0, 191, 0], [0, 159, 0],
    #[0, 127, 0], [0, 95, 0], [0, 63, 0], [0, 31, 0]),                  # green
   #([0, 0, 255], [0, 0, 223], [0, 0, 191], [0, 0, 159],
    #[0, 0, 127], [0, 0, 95], [0, 0, 63], [0, 0, 31]),                  # blue
```

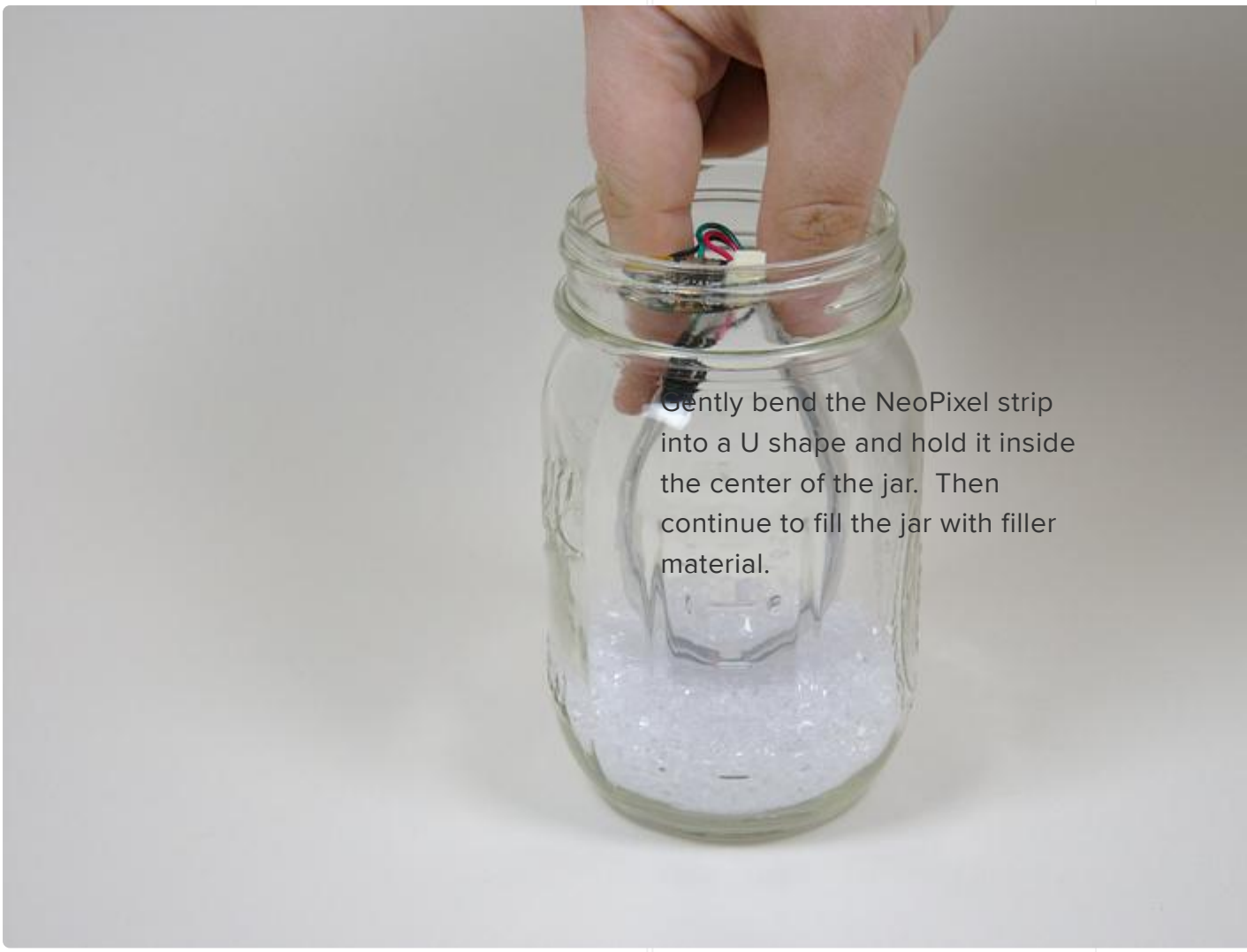
```
   #([75, 0, 130], [65, 0, 113], [56, 0, 97], [46, 0, 81],
    #[37, 0, 65], [28, 0, 48], [18, 0, 32], [9, 0, 16]),                # indigo
   #([139, 0, 255], [121, 0, 223], [104, 0, 191], [86, 0, 159],
    #[69, 0, 127], [52, 0, 95], [34, 0, 63], [17, 0, 31]),              # violet
   #([255, 255, 255], [223, 223, 223], [191, 191, 191], [159, 159, 159],
    #[127, 127, 127], [95, 95, 95], [63, 63, 63], [31, 31, 31]),       # white
   #([255, 0, 0], [255, 153, 0], [255, 255, 0], [0, 255, 0],
    #[0, 0, 255], [75, 0, 130], [139, 0, 255], [255, 255, 255]),       # rainbow
colors

# List of animations speeds (in seconds).  This is how long an animation spends
before
# changing to the next step.  Higher values are slower.
speeds = [.4, .2, .1, .05, .025]

# Global state used by the sketch
color_index = 0
animation_index = 0
speed_index = 2

pixel_pin = board.D1    # Pin where NeoPixels are connected
pixel_count = 10        # Number of NeoPixels

speed = .1              # Animation speed (in seconds).
                        # This is how long to spend in a single animation frame.
                        # Higher values are slower.

animation = 1           # Type of animation, can be one of these values:
                        # 0 - Solid color pulse
                        # 1 - Moving color pulse

# initialize LEDS
strip = neopixel.NeoPixel(board.NEOPIXEL, pixel_count, brightness=brightness,
auto_write=False)

# initialize Remote Control
ir_code_min = 60
ir_code_max = 70
pulsein = pulseio.PulseIn(board.REMOTEIN, maxlen=100, idle_state=True)
decoder = adafruit_irremote.GenericDecode()

def read_nec():
# Check if a NEC IR remote command is the correct length.
# Save the third decoded value as our unique identifier.

    pulses = decoder.read_pulses(pulsein, max_pulse=5000)
    command = None

    try:
        if len(pulses) >= ir_code_min and len(pulses) <= ir_code_max:
            code = decoder.decode_bits(pulses)
            if len(code) > 3:
                command = code[2]
    except adafruit_irremote.IRNECRepeatException:  # Catches the repeat signal
        pass
    except adafruit_irremote.IRDecodeException:  # Failed to decode
        pass

    return command

def handle_remote():
    global color_index, animation_index, speed_index

    ir_code = read_nec()

    if ir_code is None:
        time.sleep(.1)
        return
```

```
    if ir_code == color_change:
        color_index = (color_index + 1) % color_count
    elif ir_code == animation_change:
        animation_index = (animation_index + 1) % 2
    elif ir_code == speed_change:
        speed_index = (speed_index + 1) % 5
    elif ir_code == power_off:
        strip.fill([0, 0, 0])
        strip.show()

while True:  # Loop forever...

    # Main loop will update all the pixels based on the animation.
    for i in range(pixel_count):

        # Animation 0, solid color pulse of all pixels.
        if animation_index == 0:
            current_step = (time.monotonic() / speeds[speed_index]) % (color_steps *
2 - 2)
            if current_step >= color_steps:
                current_step = color_steps - (current_step - (color_steps - 2))

        # Animation 1, moving color pulse.  Use position to change brightness.
        elif animation == 1:
            current_step = (time.monotonic() / speeds[speed_index] + i) %
(color_steps * 2 - 2)
            if current_step >= color_steps:
                current_step = color_steps - (current_step - (color_steps - 2))

        strip[i] = color_palette[int(color_index)][int(current_step)]

    # Next check for any IR remote commands.
    handle_remote()

    # Show the updated pixels.
    strip.show()
```

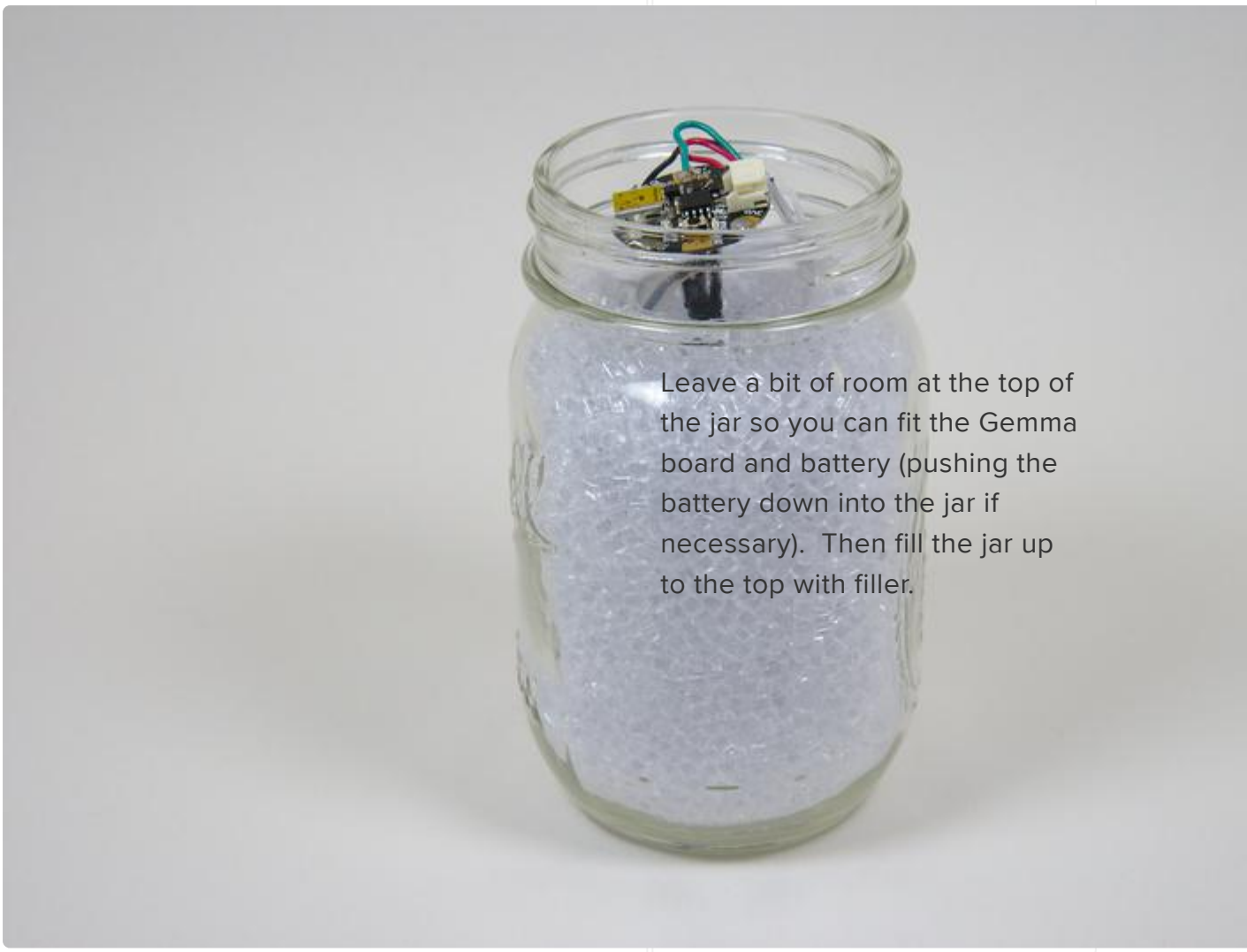
Assembly

Once you've soldered the hardware together and uploaded the sketch code you're ready to assemble the final tiki torch.


Start by putting a layer of filler material inside a mason jar.
To prevent a mess if filler material spills I recommend filling the jars on a tray or in a box.



Gently bend the NeoPixel strip into a U shape and hold it inside the center of the jar. Then continue to fill the jar with filler material.



Once the jar is filled you're ready to put the lid on and screw it shut. However make sure the lid is not conductive or it might short out the connections on the hardware! Most mason jar lids have a white coating on them that is not conductive, however if your lid is shiny metal or you're not sure if it's conductive then cover the lid in tape (like packing tape or even duct tape) to protect it.

Once the jar is sealed up check that you can still control it using the remote control. If the remote control isn't registering double check the IR receiver isn't blocked by the battery and has a clear view out of the jar (as long as the filler material is clear it shouldn't block the IR receiver).

Now that the jar is assembled the last piece is to place it inside a tiki torch.

Pull the fuel cannister out of your tiki torch.

Push the jar lid side first into the torch. If you're using the torch suggested in this guide the mason jar should pop in and be held snugly by the wire in the top of the torch.

If the mason jar doesn't fit the torch you might need to cut the wire at the top of the torch and bend it into a smaller or larger opening to accomodate the mason jar.

Congratulations, you've built the techno-tiki torch! Decorate your backyard, a patio/ balcony, a campsite, or even your room with these beautiful glowing lights.