Neopixel Crystal Chandelier with CircuitPython Animations and Speed Control

Created by Erin St Blaine


Last updated on 2023-09-30 12:40:49 PM EDT
# Table of Contents

## Overview
- Parts & Materials
- Additional Materials

## Wiring Diagram
- Rotary Encoder
- NeoPixels

## CircuitPython on Circuit Playground Bluefruit
- Install or Update CircuitPython

## Software
- Install Libraries
- Simplified Code
- Full Code
- Troubleshooting

## 3D Printed Rings

## Rotary Encoder Wiring

## Prepare Rings and Pixels
- Circuit Playground Wiring
- 24 Pixel Ring
- 60 Pixel Ring
- Individual Pixels

## Make the Crystals
- Download the Crystal Shape
- Laminate Your Cellophane
- Cutting on Cricut Vinyl Cutter
- Insert the Individual Pixels

## Final Assembly
- Center Tiers
- Crystal Gems
Overview

Floating crystals and glowing lights are a match made in heaven. This project combines a wide variety of skills and tools into one lovely project. Make a gorgeous hanging lamp with sparkly beads, glowing crystals, live edge wood and of course, lots of NeoPixels.

My chandelier is unique, and designed to show my personal style. Since you, dear reader, have your very own unique style, this tutorial will focus on giving you the tools to design and create your own one-of-a-kind bespoke hanging lamp. This tutorial will provide source files and ideas, and give guidance on how the electronics fit together.

This tutorial will also get you started with customizing your own software animations. The sample code uses CircuitPython and the delightfully easy to use LED Animations Library (https://github.com/kattni/circuitpython_ledAnimations) by Kattni Rembor. This code gives you a framework that allows speed and brightness control using a rotary encoder knob, so you can adjust the lighting to suit any environment or mood.

Parts & Materials

The Circuit Playground Bluefruit is a great choice for this project. It has the speed and memory necessary to run complex animations, and also has 10 onboard pixels we can use, without buying more rings and doing more soldering. The code also makes use of the Circuit Playground's onboard toggle switch to toggle between controlling brightness or controlling speed with the rotary encoder knob.

The rotary encoder also has a built-in button (when you push down on the knob) which we can use as an on/off switch for our lights.

Circuit Playground Bluefruit - Bluetooth Low Energy
Circuit Playground Bluefruit is our third board in the Circuit Playground series, another step towards a perfect introduction to electronics and programming. We've...
https://www.adafruit.com/product/4333
Rotary Encoder + Extras
This rotary encoder is the best of the best, it's a high-quality 24-pulse encoder, with detents and a nice feel. It is panel mountable for placement in a box, or you can plug it...
https://www.adafruit.com/product/377

My chandelier uses two NeoPixel rings in addition to the Circuit Playground's pixels, for a three-tiered effect. I'm using a 24 pixel ring and a 60 pixel ring, since the size variation pleases me - but be sure to poke around in the shop, we've got a lot of sizes.

Take note that the 60 pixel ring comes in 4 segments, sold individually, so you'll need to order x4 of these.

NeoPixel Ring - 24 x 5050 RGB LED with Integrated Drivers
Round and round and round they go! 24 ultra bright smart LED NeoPixels are arranged in a circle with 2.6" (66mm) outer diameter. The rings are 'chainable' - connect the...
https://www.adafruit.com/product/1586

NeoPixel 1/4 60 Ring - 5050 RGB LED w/ Integrated Drivers
The biggest NeoPixel Ring yet! With four of these you can make a huge ring with 60 ultra bright smart LED NeoPixels are arranged in a circle with a 6.2" diameter. Each...
https://www.adafruit.com/product/1768
The hanging crystals on my lamp each contain one individual NeoPixel. I love the look of individual pixels when they're treated as a group in the software. It's such a delight to run a chase animation up and down the crystal spiral.

This technique requires a whole lot of fiddly soldering, and a high potential for swearing and burned fingers, but a little patience and perseverance can go a long way. Be sure to order a few more pixels than you think you'll need, since having some backups during final assembly may be really helpful.

Flora RGB Smart NeoPixel version 2 - Sheet of 20
So, you want lots and lots of NeoPixels? And you want them for less? Not a problem! Here's a sheet of Flora NeoPixels fresh from the (reflow) oven. Cut them off as you need 'em...
https://www.adafruit.com/product/1559

Adafruit sells a lot of other types of pixels as well. My design doesn't incorporate any Soft Flexible Wire NeoPixels or Side Light NeoPixels, but be sure to consider those for your design.

You'll also need some wire. High quality silicone stranded wire will make this lamp much easier to assemble, and will make it last a lot longer. If you haven't switched to this noodle-y wire yet, it's time.

Be sure to get at least 4 different colors. Having multiple colors means your design can become much more complicated without the risk of hooking things up incorrectly and frying your electronics.

4 x Silicone Stranded Wire
26 awg Silicone Stranded Wire
https://www.adafruit.com/product/2513

Finally, you'll need a power supply and a screw terminal to connect the pixels to the power supply. A high-quality one will save you a lot of heartache and head-scratching.

1 x Power Supply
Compact Switching Power Supply - Selectable Output
3-12VDC
https://www.adafruit.com/product/1448
Additional Materials

You'll need an opaque lamp base. I'm using a slab of live-edge wood with a lot of character. I'm also using an upside-down wooden bowl on top of the lamp to hide the wires and power supply on top.

The rings are supported with 3d printed ring holders. I've provided the files for three different holders. If you don't have a 3d printer, you can upload them to a 3d printing service and they'll mail them to you. Isn't the internet great?

You'll also need a big pile of beads and crystals and pearls, or other findings that will catch the light and sparkle. Get a combination of transparent and opaque beads. The transparent ones will change color with reflected light and the opaque ones can be used to hide your wire runs. Be sure to get some monofilament line to string the beads as well.

The hanging crystals are made from laminated cellophane wrap, and cut out on a vinyl cutting machine. You can also cut them by hand, or order pre-cut kits from my Etsy store.(

For hooking up the hanging crystals, I'm using a length of white telephone wire. Telephone wire is straight and smooth and pretty to look at on the outside, and it has four tiny wires on the inside, which makes it perfect for hooking up single pixels in a series. It's more difficult to work with than silicone stranded wire, since the wires are very tiny and prone to breakage, but since these wires will be showing I'm choosing form over function in this case.

I finished my chandelier off with a lamp cord and some easy-connect plugs from the hardware store, so my 5v power supply doesn't show.
Wiring Diagram

Rotary Encoder

With the 3-pin side at the top:

- Top left pin to A3
- Top center pin to G
- Top right pin to A2
- Bottom left pin to G
- Bottom right pin to A5

NeoPixels

All NeoPixels have an IN and an OUT pin. Data must flow from IN to OUT or the pixels won't work. So our data flow will start from pin A1 on the Circuit Playground.

The 24-pixel ring is first in line, followed by the 60-pixel ring, and then the individual pixels that go inside the crystals. The data wire (shown in yellow) is connected to A1, then flows to the IN pad on the first ring, then to the OUT pad on that ring and on to the IN on the larger ring. After data flows through all four sections, it comes back OUT of the large ring and then IN to OUT on each individual pixel.

Power can flow either direction - it's not one-way like the data flow is. We'll connect the power wires (shown in red and black) at the top of the chandelier and let the power flow downward until it reaches the Circuit Playground.

Our screw terminal connects to all the individual pixels at once for the power and ground wires. Then these lines flow through the large ring, the small ring, and end up at the Circuit Playground.
CircuitPython on Circuit Playground Bluefruit

Install or Update CircuitPython

Follow this quick step-by-step to install or update CircuitPython on your Circuit Playground Bluefruit.

Download the latest version of CircuitPython for this board via circuitpython.org

Click the link above and download the latest UF2 file

Download and save it to your Desktop (or wherever is handy)
Plug your Circuit Playground Bluefruit into your computer using a known-good data-capable USB cable.

A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

Double-click the small Reset button in the middle of the CPB (indicated by the red arrow in the image). The ten NeoPixel LEDs will all turn red, and then will all turn green. If they turn all red and stay red, check the USB cable, try another USB port, etc. The little red LED next to the USB connector will pulse red - this is ok!

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

(If double-clicking doesn't do it, try a single-click!)
You will see a new disk drive appear called CPLAYBTBOOT.

Drag the adafruit_circuitpython_etc.uf2 file to CPLAYBTBOOT.

The LEDs will turn red. Then, the CPLAYBTBOOT drive will disappear and a new disk drive called CIRCUITPY will appear.

That's it, you're done! :)

---

### Software

[Download CircuitPython Libraries]
Install Libraries

Download the latest version of the Circuit Python Library Repository using the button above. Be sure the library version matches the version of CircuitPython you just installed (i.e. if you installed CircuitPython 6.x, install the 6.x version of the library files).

Copy the following files into a directory called /lib on your CIRCUITPY drive:

- adafruit_bus_device (directory)
- adafruit_circuitplayground (directory)
- adafruit_led_animation (directory)
- adafruit_lis3dh.mpy
- adafruit_thermistor.mpy
- neopixel.mpy

When you’re done, your CIRCUITPY drive should look like this:

Simplified Code

This sample code demonstrates how to use a rotary encoder to control the speed and the brightness of animations from the LED Animations Library. It also uses the rotary encoder’s push button to turn the lights on and off with a software toggle switch.

This code also demonstrates how to run animations on two different LED strips or pins at the same time: in this case, the lights on pin A1 and the lights on the face of the Circuit Playground board.
I've included the full chandelier code, including pixel mapping and multiple layered animations, at the bottom of this page.

```python
# SPDX-FileCopyrightText: 2021 Erin St Blaine for Adafruit Industries
#
# SPDX-License-Identifier: MIT
#

Crystal Chandelier with Circuit Playground BlueFruit
Adafruit invests time and resources providing this open source code. Please support Adafruit and open source hardware by purchasing products from Adafruit!
Written by Erin St Blaine & Limor Fried for Adafruit Industries
Copyright (c) 2020-2021 Adafruit Industries
Licensed under the MIT license.
All text above must be included in any redistribution.

# pylint: disable=import-error
# pylint: disable=no-member
import board
import digitalio
import rotaryio
import neopixel
from adafruit_circuitplayground import cp
from adafruit_led_animation.group import AnimationGroup
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.solid import Solid
from adafruit_led_animation.color import BLACK, PURPLE,

NEOPIXEL_PIN = board.A1 # NeoPixels connected here (crystals + rings)
NUM_PIXELS = 92        # Number of pixels in the crystals and rings

MIN_BRIGHTNESS = 1   # Minimum LED brightness as a percentage (0 to 100)
MAX_BRIGHTNESS = 25  # Maximum LED brightness as a percentage (0 to 100)
MIN_FPS = 2         # Minimum animation speed in frames per second (>0)
MAX_FPS = 8        # Maximum animation speed in frames per second (>0)

# Set initial brightness and speed to center values
BRIGHTNESS = (MIN_BRIGHTNESS + MAX_BRIGHTNESS) // 2
FPS = (MIN_FPS + MAX_FPS) // 2
SPEED = 1 / FPS      # Integer frames-per-second to seconds interval
LEVEL = BRIGHTNESS * 0.01 # Integer brightness percentage to 0.0-1.0 coeff.

button = digitalio.DigitalInOut(board.A5)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP
BUTTON_STATE = None
BUTTON_VALUE = None

PIXELS = neopixel.NeoPixel(NEOPIXEL_PIN, NUM_PIXELS, brightness=LEVEL, auto_write=False)

# NeoPixels off ASAP on startup
cp.pixels.fill(0) # Onboard pixels
cp.pixels.show()  
PIXELS.fill(0) # NeoPixel strip
PIXELS.show()

ENCODER = rotaryio.IncrementalEncoder(board.A2, board.A3)

# LED ANIMATIONS  
```
COMET = Comet(PIXELS, speed=SPEED, tail_length=8, color=PURPLE, bounce=True)
CP_COMET = Comet(cp.pixels, speed=SPEED, tail_length=8, color=PURPLE, bounce=True)
DARK_RINGS = Solid(PIXELS, color=BLACK)
DARK_CPB = Solid(cp.pixels, color=BLACK)
DARK = AnimationGroup(
    DARK_RINGS,
    DARK_CPB,
)

# Animations Playlist, reorder as desired. AnimationGroups play at same time
ANIMATIONS = AnimationGroup(
    COMET,
    CP_COMET,
)

# MAIN LOOP  

LAST_POSITION = ENCODER.position
MODE = 1
while True:
    POSITION = ENCODER.position
    if POSITION != LAST_POSITION:
        MOVE = POSITION - LAST_POSITION
        if cp.switch:
            FPS = max(MIN_FPS, min(FPS + MOVE, MAX_FPS))
            SPEED = 1.0 / FPS
            COMET.speed = SPEED
            print("comet speed = ", SPEED)
        else:
            BRIGHTNESS = max(MIN_BRIGHTNESS, min(BRIGHTNESS + MOVE, MAX_BRIGHTNESS))
            LEVEL = BRIGHTNESS * 0.1
            if LEVEL > 1:
                LEVEL = 1
            cp.pixels.brightness = LEVEL
            PIXELE.s.brightness = LEVEL
            print("brightness = ", LEVEL)
    LAST_POSITION = POSITION
    if not BUTTON_VALUE and BUTTON_STATE is None:
        BUTTON_STATE = "pressed"
    if BUTTON_VALUE and BUTTON_STATE == "pressed":
        print("Button pressed.")
        if MODE == 1:
            MODE = 2
        else:
            MODE = 1
    if MODE == 1:
        ANIMATIONS.animate()
    else:
        DARK.animate()

Download the code.py file or copy/paste the code into your Mu editor window and save it as code.py at the root of your CIRCUITPY drive. You should see a purple comet animation running on the face of your Circuit Playground, and also on any lights you have hooked up to pin A1.
If you have your rotary encoder connected, test it out. When the tiny slide switch on the Circuit Playground is switched to the left, the speed of the comet will change when you turn the dial. If the switch is flipped to the right, the brightness will change. Press down on the knob to toggle the lights on and off.

Let's delve into the code a bit.

First, we'll import our libraries and the animations we want to use from the LED Animations Library. Check out that guide for a lot more info on using animations.

```python
import board
import digitalio
import rotaryio
import neopixel
from adafruit_circuitplayground import cp
from adafruit_led_animation.group import AnimationGroup
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.solid import Solid
from adafruit_led_animation.color import BLACK, PURPLE,
```

Next, we'll set up the NeoPixel strip on pin A1. We don't need to set up the Circuit Playground's onboard pixels - the Circuit Playground library does that for us.

```python
NEOPIXEL_PIN = board.A1 # NeoPixels connected here (crystals + rings)
NUM_PIXELS = 92        # Number of pixels in the crystals and rings
PIXELS = neopixel.NeoPixel(NEOPIXEL_PIN, NUM_PIXELS, brightness=LEVEL, auto_write=False)
```

Next we'll set up the parameters for our brightness and speed controls. We're controlling speed in frames-per-second (FPS). You can play around with these numbers to define your maximum and minimum speed and brightness. Open the REPL and turn the knob to get a readout of the current brightness level or speed.

Be sure to back up your code before you mess around with these numbers too much! There's a possibility of feeding your rotary encoder a reading of 0, which will hang your board. Check the troubleshooting section below if this happens to you.

```python
MIN_BRIGHTNESS = 1   # Minimum LED brightness as a percentage (0 to 100)
MAX_BRIGHTNESS = 25  # Maximum LED brightness as a percentage (0 to 100)
MIN_FPS = 2          # Minimum animation speed in frames per second (&gt;0)
MAX_FPS = 8          # Maximum animation speed in frames per second (&gt;0)

# Set initial brightness and speed to center values
BRIGHTNESS = (MIN_BRIGHTNESS + MAX_BRIGHTNESS) // 2
FPS = (MIN_FPS + MAX_FPS) // 2
```
SPEED = 1 / FPS        # Integer frames-per-second to seconds interval
LEVEL = BRIGHTNESS * 0.01 # Integer brightness percentage to 0.0-1.0 coeff.

Next we set up the push-button for our on/off toggle.

```python
button = digitalio.DigitalInOut(board.A5)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP
BUTTON_STATE = None
BUTTON_VALUE = None
```

We set the pixels to OFF initially, then set up our encoder.

```python
# NeoPixels off ASAP on startup
cp.pixels.fill(0) # Onboard pixels
cp.pixels.show()
PIXELS.fill(0) # NeoPixel strip
PIXELS.show()
ENCODER = rotaryio.IncrementalEncoder(board.A2, board.A3)
```

Next we set up our LED animations. You can add as many different animation types and varietals as you’d like. Just be sure to import the animation from the LED Animations library at the top of the code.

The LED Animations library can only address one pin per animation. Since we’re using two different pins: the pixels on A1 and the pixels on the face of the Circuit Playground, we need to use AnimationGroup to make them play at the same time.

I’ve set up the COMET animation to play on PIXELS (pin A1) and also on cp.pixels (the onboard pixels). I’ve also set up a DARK animation on both strips, for use when we press the toggle switch to turn the pixels off.

```python
# LED ANIMATIONS -----------------------------------------------
COMET = Comet(PIXELS, speed=SPEED, tail_length=8,
               color=PURPLE, bounce=True)
CP_COMET = Comet(cp.pixels, speed=SPEED, tail_length=8,
                  color=PURPLE, bounce=True)
DARK_RINGS = Solid(PIXELS, color=BLACK)
DARK_CPB = Solid(cp.pixels, color=BLACK)
DARK = AnimationGroup(
    DARK_RINGS,
    DARK_CPB,
)

# Animations Playlist, reorder as desired. AnimationGroups play at same time
ANIMATIONS = AnimationGroup(
    COMET,
    CP_COMET,
    )
```
That's it for setup. Next comes our main loop. First the code reads the position of the rotary encoder to determine if it's been moved. If the onboard switch (\texttt{cp.switch}) is set to the left, the code will adjust the speed of the \texttt{COMET} animation whenever the switch is turned.

If the switch is set to the right, the code adjusts the brightness level up or down.

\begin{verbatim}
LAST_POSITION = ENCODER.position
MODE = 1
while True:

    POSITION = ENCODER.position
    if POSITION != LAST_POSITION:
        MOVE = POSITION - LAST_POSITION
        if cp.switch:
            FPS = max(MIN_FPS, min(FPS + MOVE, MAX_FPS))
            SPEED = 1.0 / FPS
            COMET.speed = SPEED
            print("comet speed = ", SPEED)
        else:
            BRIGHTNESS = max(MIN_BRIGHTNESS, min(BRIGHTNESS + MOVE, MAX_BRIGHTNESS))
            LEVEL = BRIGHTNESS * 0.1
            if LEVEL &gt; 1:
                LEVEL = 1
            cp.pixels.brightness = LEVEL
            PIXELS.brightness = LEVEL
            print("brightness = ", LEVEL)
        LAST_POSITION = POSITION

Finally, the code looks for button presses and toggles between the \texttt{COMET} animation group and the \texttt{DARK} animation group depending on the button's state.

\begin{verbatim}
if not BUTTON_VALUE and BUTTON_STATE is None:
    BUTTON_STATE = "pressed"
if BUTTON_VALUE and BUTTON_STATE == "pressed":
    print("Button pressed.")
    if MODE == 1:
        MODE = 2
    else:
        MODE = 1
    BUTTON_STATE = None
if MODE == 1:
    ANIMATIONS.animate()
else:
    DARK.animate()
\end{verbatim}

\section*{Full Code}

Here's my full chandelier code. In this version I've added pixel mapping so I can address the crystals separately from the rings, plus I've added more animations and animation groups. You can learn more about these features in the \texttt{LED Animations Library Guide}().
This code runs a half-brightness rainbow animation on the rings and Circuit Playground face, balancing the brightness with the single-pixel crystals, while it runs a speed-controlled comet animation on the crystals themselves.

```python
# SPDX-FileCopyrightText: 2021 Erin St Blaine for Adafruit Industries
# SPDX-FileCopyrightText: 2021 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT
#
Crystal Chandelier with Circuit Playground BlueFruit
Adafruit invests time and resources providing this open source code. Please support Adafruit and open source hardware by purchasing products from Adafruit!
Written by Erin St Blaine & Limor Fried for Adafruit Industries
Copyright (c) 2020-2021 Adafruit Industries
Licensed under the MIT license.
All text above must be included in any redistribution.
#
# pylint: disable=import-error
# pylint: disable=no-member
import board
import digitalio
import rotaryio
import neopixel
from adafruit_circuitplayground import cp
from adafruit_led_animation.helper import PixelMap
from adafruit_led_animation.sequence import AnimationSequence
from adafruit_led_animation.group import AnimationGroup
from adafruit_led_animation.animation.rainbow import Rainbow
from adafruit_led_animation.animation.comet import Comet
from adafruit_led_animation.animation.solid import Solid
from adafruit_led_animation.color import BLACK, PURPLE,
)

NEOPIXEL_PIN = board.A1 # NeoPixels connected here (crystals + rings)
NUM_PIXELS = 92        # Number of pixels in the crystals and rings
MIN_BRIGHTNESS = 1   # Minimum LED brightness as a percentage (0 to 100)
MAX_BRIGHTNESS = 25 # Maximum LED brightness as a percentage (0 to 100)
MIN_FPS = 2         # Minimum animation speed in frames per second (>0)
MAX_FPS = 8        # Maximum animation speed in frames per second (>0)

# Set initial brightness and speed to center values
BRIGHTNESS = (MIN_BRIGHTNESS + MAX_BRIGHTNESS) // 2
FPS = (MIN_FPS + MAX_FPS) // 2
SPEED = 1 / FPS    # Integer frames-per-second to seconds interval
LEVEL = BRIGHTNESS * 0.01 # Integer brightness percentage to 0.0-1.0 coeff.

button = digitalio.DigitalInOut(board.A5)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP
BUTTON_STATE = None
BUTTON_VALUE = None

PIXELS = neopixel.NeoPixel(NEOPIXEL_PIN, NUM_PIXELS, brightness=LEVEL, auto_write=False)

# NeoPixels off ASAP on startup
cp.pixels.fill(0) # Onboard pixels
cp.pixels.show()
PIXELS.fill(0) # NeoPixel strip
```
©Adafruit Industries
ENCODER = rotaryio.IncrementalEncoder(board.A2, board.A3)

# PIXEL MAPS reorder pixels so animations run in different configs --------

# Crystals Only
PIXEL_MAP_CRYSTALS = PixelMap(PIXELS, [84, 85, 86, 87, 88, 89, 90, 91], individual_pixels=True)

# Rings Only
PIXEL_MAP_RINGS = PixelMap(PIXELS, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83], individual_pixels=True)

# LED ANIMATIONS -----------------------------------------------------------
RAINBOW = Rainbow(PIXEL_MAP_RINGS, speed=0.01, period=10, step=3)
CP_RAINBOW = Rainbow(cp.pixels, speed=0.01, period=10, step=3)
CRYSTAL_COMET = Comet(PIXEL_MAP_CRYSTALS, speed=SPEED, tail_length=8, color=PURPLE, bounce=True)
DARK_RINGS = Solid(PIXELS, color=BLACK)
DARK_CPB = Solid(cp.pixels, color=BLACK)
DARK = AnimationGroup(DARK_RINGS, DARK_CPB,)

# Animations Playlist, reorder as desired. AnimationGroups play at same time
ANIMATIONS = AnimationSequence(
    AnimationGroup(
        CRYSTAL_COMET, RAINBOW, CP_RAINBOW),
    auto_clear=True,
    auto_reset=True,
)

# MAIN LOOP -----------------------------------------------------------------

LAST_POSITION = ENCODER.position
MODE = 1
while True:
    POSITION = ENCODER.position
    if POSITION != LAST_POSITION:
        MOVE = POSITION - LAST_POSITION
        if cp.switch:
            FPS = max(MIN_FPS, min(FPS + MOVE, MAX_FPS))
            SPEED = 1.0 / FPS
            CRYSTAL_COMET.speed = SPEED
            #RAINBOW.speed = SPEED
            #CP_RAINBOW.speed = SPEED
            print("crystal speed = ", SPEED)
        else:
            BRIGHTNESS = max(MIN_BRIGHTNESS, min(BRIGHTNESS + MOVE, MAX_BRIGHTNESS))
            LEVEL = BRIGHTNESS * 0.1
```python
if LEVEL > 2:
    LEVEL = 2
    cp.pixels.brightness = LEVEL/2
    PIXEL_MAP_RINGS.brightness = LEVEL/2
    PIXEL_MAP_CRYSTALS.brightness = LEVEL
    print("ring brightness = ", LEVEL/2, ", crystal brightness = ", LEVEL)
LAST_POSITION = POSITION
if not BUTTON_VALUE and BUTTON_STATE is None:
    BUTTON_STATE = "pressed"
if BUTTON_VALUE and BUTTON_STATE == "pressed":
    print("Button pressed.")
    if MODE == 1:
        MODE = 2
    else:
        MODE = 1
        BUTTON_STATE = None
    if MODE == 1:
        ANIMATIONS.animate()
    else:
        DARK.animate()
```

Troubleshooting

If you're having trouble getting the code to load, head over to the Circuit Playground Bluefruit guide for more detailed instructions and things to try.

Another problem I encountered is that the rotary encoder will hang if it's fed a value of 0. If you're playing with calibrating your numbers and your board's lights turn red, your CIRCUITPY drive vanishes, and pressing the reset button has no effect, this may be what happened.

Don't worry! You can erase and recover the board but you'll need to reinstall CircuitPython and your libraries. Visit the Troubleshooting page from the CircuitPython guide and scroll way down until you find CIRCUITPY Drive Issues. Follow the instructions to wipe your board clean and start over.

3d Printed Rings

My chandelier has three 3d-printed rings to hold the NeoPixel rings, with appropriately sized holes for 26awg silicone stranded wires to poke through and a hole at the bottom for the rotary encoder knob.

The rings were designed in Tinkercad, a free online 3d Design service from Autodesk. Tinkercad is wonderfully easy to use and great for learning 3d design. Its intuitive interface lets you choose premade shapes and squish them together to create quick and simple downloadable .stl files.
Download the files below, or click the Edit button to add your own flair or adjust the sizing or styling on Tinkercad.

![Neopixel Chandelier Rings.zip](image)

![Edit Design on Tinkercad](image)

I printed these in Glow in the Dark PLA with no raft on a glass bed coated with a thin layer of PET (glue stick), and the bottoms came out beautifully smooth and glassy. No supports are needed to print.

---

**Rotary Encoder Wiring**

Find the side with three solder pads. This side controls the rotating knob. Solder a black wire to the center pad. Solder a blue wire to the leftmost pad and a green wire to the rightmost pad.

Flip the encoder over and find the side with two solder pads. This side controls the button that clicks when you press down on the knob. Solder another black wire to the pad on the left (with the encoder facing right-side up) and a yellow wire to the pad on the right.
Press the rotary encoder down inside the 3D printed holder so the knob sticks out the bottom. It should fit snugly, with none of the pads touching each other and no accidental solder bridges.

Solder both of the two black wires to the G pin on the Circuit Playground as shown.

Solder the yellow wire (from the 2-pin side) to pad A5. Solder the blue and green wires to pins A2 and A3 respectively.

Plug in your Circuit Playground and test to be sure the knob works. Pressing down on the knob should turn the lights on and off. Twisting the knob should change the brightness (if your Circuit Playground onboard switch is set to the left) or the animation speed (if it's set to the right).
Nestle your Circuit Playground gently into
the holder. If you experience flickering or
shorting, put a piece of tape on the back
of the Circuit Playground just to be extra-
sure the rotary encoder's pads aren’t
touching the pads on the back of the
board and causing confusion.

Prepare Rings and Pixels

DO NOT CONNECT THE RINGS OR PIXELS TO EACH OTHER YET

If you're using the 3d printed rings, we'll want the wires to pass through the holes in
the rings and get strung with beads or pearls before you connect both ends of the
wire. The hanging crystal wires will need to pass through the lamp base before being
connected together.

For now we'll work from the bottom up, and prep the three wires that go OUT of each
part to the next one in the series. We'll solder the three IN wires to the rings and
connect the individual pixels together during final assembly.

More about soldering NeoPixel rings can be found in the Make It Glow: How to Solder
NeoPixels Guide.

Circuit Playground Wiring

Solder a red wire to VOUT, a white or
yellow wire to A1, and a black wire to G.
These wires will eventually connect to the
smaller 24 pixel ring.
24 Pixel Ring

Solder a red wire to one of the + pads, a black wire to one of the G pads, and a yellow wire to DOUT. You can find detailed instructions on soldering to this ring on the How To Solder NeoPixels guide().

These wires will go out from this ring to the 60 pixel ring. Don't solder an IN wire just yet - we'll use the wires we just soldered to the Circuit Playground for that later on.

60 Pixel Ring

This ring comes in 4 segments that you'll need to assemble into a ring. The technique is almost identical to this technique for soldering two NeoPixel strips together().

Tape the rings to your table and match up the pads. Bridge 3 of the 4 segments with solder. If you want extra stability, you can add a small piece of solid core wire (or paperclip, or a staple, or whatever) across the segments.

For the fourth segment, tin all the pads but do NOT bridge the solder together. We'll use this fourth segment as our data input and output section.

Notice also that there are small circular pads marked G and + on the back of each segment. Solder a red wire to one of the + pads and a black wire to one of the G pads. It's helpful to spread your wires out so they're as equidistant as you can make them.
Again, don't solder the IN wires yet. We'll use the wires we just soldered to the 24 pixel ring to bring power, ground, and data to this ring.

Individual Pixels

I'm using 4 conductor stranded telephone wire for the individual pixels. I cut about 3 feet of wire for each hanging crystal, even though my crystals will mostly hang a lot closer than that. It's easy to cut the wires shorter and much more challenging to lengthen them.

This wire can be tricky to strip without breaking the inner wires. There are purpose-made tools for this, but I didn't have any of those so used my flush-cutters for the outer coating and a pair of good wire strippers for the tiny inner wires.

Strip off about an inch of the outer shielding. Strip 1/4" from each of the internal wires. Solder the yellow wire to IN, the green wire to OUT, the red wire to + and the black wire to G.

Repeat for each pixel. My chandelier has 8 crystals so I wired up 8 pixels. Then I wired a couple extras just to have as a backup since these wires are so tiny and break so easily.
Make the Crystals

Download the Crystal Shape

Download the crystal pattern. I also have a couple other shapes available for download from my Etsy store (along with pre-cut crystal kits if you’re in a hurry!). The design has two pieces that connect together with locking tabs.

Crystal Gem Download

If you don’t have a vinyl cutter you can still do this project. Just print the shapes out and use them as a template. The dotted lines are score lines and the solid lines are cut lines.

My chandelier has 8 crystals that each vary in size by about 1/4". You can resize the pattern before printing, just be sure to resize both pieces by the same amount.

Laminate Your Cellophane

Unroll your cellophane wrap and cut pieces that are the same size as or slightly smaller than your laminating pouches.

I use a rotary cutter for this and find it goes quickly and gives me nice clean straight edges.

I’m using 3 mil thickness laminating pouches. The 5 mil will also work, but the extra stiffness makes the cutting and folding a lot more difficult, so stick with 3 mil if you can.
Crumple up the cellophane and then flatten it out again to give your material some fun texture. Place the cellophane inside the laminating pouch so no edges are poking out.

Run it through your laminating machine. I find it helpful to smooth it gently as it comes out of the machine. The flatter and smoother you can get your sheets, the better they'll work with the cutting machine.

A Note about Laminating Machines

You can find these for very inexpensive prices (around $20) on Amazon or other online retailers. I started out with a slightly pricier Scotch brand machine, then replaced it with a less expensive Amazon brand one when it finally gave up the ghost (after faithfully laminating around 600 sheets, bless its heart).

I've had a lot of trouble with the new cheap machine jamming and crumpling up my materials. I came up with a fix, which I'll explain below. But if I had it to do over again, I'd spend the extra $10 and get a good quality machine. Quality tools make for far fewer headaches.
If Your Machine Keeps Jamming

My sheets kept getting stuck and not emerging cleanly from the machine. I fixed this by adding a stiffener across the very top of each sheet. I took one sheet that had laminated successfully, and cut a narrow strip off one edge, then slipped this up inside the un laminated pouch, against the sealed edge, before running it through the machine. This kept the top edge nice and stiff and really minimized the jamming.

Place your laminated cellophane sheet onto your vinyl cutter's sticky mat and press it down firmly. I use a wallpaper scraper for this -- they're almost free at the hardware store and I haven't found a tool I like better for the purpose. They also work great for cleaning the mats in between use.
Open the Cricut Design Space app and choose "New Project". Click "Upload" and upload the crystal design of your choice. Choose "Simple" on the next screen, then "Continue" on the next screen since there is no background to remove. On the third screen, select "Save as a Cut Image".

Do this for both halves of the gem.

Import the uploaded images into your project. I like to make them a color other than black, so they are easier to see and work with. You'll need to resize them as well -- I've given you high resolution 300dpi images, since the Cricut software does a much better job with those, but that does mean you'll need to size each crystal down.

Be sure to resize both images at the same time so they keep the exact same size ratio.
Recommended Sizes
My tallest gem is around 6" high when assembled, and my smallest gem is around 4" high. Each gem is around 1/4" taller than the next one. Making the gems too much smaller than that will make them difficult to assemble and the pixel won't fit as well in the top.

Click the icon in the upper left and select Custom Materials. Scroll until you find "Stencil Film 0.4mm" and select it. Make sure the settings read:

- Fine Point Blade
- Off (single pass mode)
- Pressure should be around 325

Close out of the Custom Materials menu. Once you're happy with your sizing, click "Make It". You have an opportunity to change your layout here, so make sure your crystal will fit nicely on your material.

Set your Cricut dial to "Custom" and select the Stencil Film 0.4mm. Load your sticky mat into the machine and press Go.

Crease all the fold-lines and tabs. Line up the tabs on either side of the hole segments and crease to the point of the crystal.

Insert the tabs from the outside in, so they end up on the interior of the crystal.

Here's an assembly video to help you out.
Crease all the folds on both pieces, then assemble, leaving the three tabs between the two light holes open for now. Cut and crumple a small piece of un-laminated cellophane and tuck it inside to create the illusory "occlusions" that catch the light and make this crystal so convincing.
Insert the Individual Pixels

Unfold the top tab and slip the individual pixel inside the top of the crystal. You can trim the crystal just a bit to allow the wires room to pass through and sit neatly.

Secure the pixel with a dab of hot glue.

Add weight to your crystal and cover up the small wires by stringing a few beads down the outside of the wire.

Final Assembly

Center Tiers

We'll start at the bottom with the rotary encoder, and work our way up.
Use some superglue to secure the rotary encoder in its niche, making sure not to get any glue in the mechanism.

Place the Circuit Playground on top of the rotary encoder and thread the three NeoPixel wires through three of the holes. Place a dab of hot glue on the inside of each hole around the wire, to add strain relief. It's never a good idea to rely solely on a solder joint for structural connections, and the hot glue will keep the wire from slipping around or pulling out.

For the fourth hole, we'll use a dummy wire. Tie a knot in a length of yellow wire (or any color that's different from your live wires) and thread it through the fourth hole with the knot on the inside.

String a series of beads or faux pearls along each of the four wires. I like using pearls since they are opaque and will hide the wires while still reflecting a lot of ambient light. Use a variety of sizes to add interest, but keep the sequence and number of pearls the same on all four strands.
Thread the four wires in through the four outer holes in the smaller ring. (In the photo they're threaded through the inner holes, but in my final design I ended up switching this around and threading through the outer holes for these wires, and I think it looks a lot better).

Solder these wires to the NeoPixel ring: red to +, black to G, and white/data to DIN. Plug your CircuitPlayground in to power and test to be sure the ring lights up.

Once you're sure everything's working, thread the "out" wires you soldered to the ring in the last step through the inner holes, adding a dummy wire to the fourth hole. Nestle the ring in place and glue all 8 wires in place, making sure to take up any slack from below. You can hide the "slack" in the wires underneath the ring.

This part is tricky, so take your time and be gentle. A pair of needle-nosed pliers can be really helpful.

Repeat for the large ring. Again, the photo shows the wires entering from the inside and emerging from the outside of the 3d printed ring, but I ended up reversing this in the final design.

Thread pearls or beads onto the wires emerging from the large ring.
You should now have three tiers of lights. Shake it around a little bit just to make sure all your connections and strain relief points are tight. It's easier to fix things now than it will be later. If everything looks good, you can add some hot glue to hold the rings down into their 3d printed bases, if needed.

Mark four equidistant attachment points on your lamp base for the four pearl strands coming from the upper ring. I think it looks best if they're a little wider than the ring's diameter.

This is also a great time to mark out where your crystals or bead strands will go.

Drill holes through all your marks. Thread the wires through the holes from the underside of the lamp base, and secure them on the top of the base with some hot glue or a zip tie.

Crystal Gems

Temporarily hang your lamp base up, or place it on sawhorses, so you can easily reach both the top and the bottom.
Thread your crystal gem wires through the lamp base. Adjust the height of each crystal until it pleases you, then put a zip tie around the wire on the top of the lamp base to hold it exactly where you want it.

I added some extra beads to the bottom of my crystals to weight them a bit more and make the wires hang straight. If your wires won't straighten out, you can gently heat them with a heat gun and hold them taut as the plastic cools.

Trim the crystal wires down, leaving around 8-12 inches of wire sticking up from each crystal. These wires are hard to strip and break easily, so make sure you have some extra length in case of breakage.

Strip off about 6 inches of the outer shielding. You'll find a red, black, yellow, and green wire inside. When you wire up the pixels, wire yellow to IN and green to OUT.

Connect the data wire coming from your NeoPixel rings to the IN wire (yellow) on your first crystal. Connect the green wire from that bundle to the yellow wire on the next crystal, and so on, until you have a yellow to green connection between each of the crystals.
Cut a length of red and black wire, and insert one stripped end into your screw terminal (red to +, black to -).

Then, splice ALL the red wires you can find, including the screw terminal wire and the wire coming from the NeoPixel rings, into one giant happy conjunction.

Repeat with the black wires, including the one from the screw terminal and the one from the rings.

I found it easiest to twist half the wires together into a big bundle, then do the same with the other half, slip on some heat shrink and then solder the two halves together.

Plug in your power supply and test to be sure all the lights come on. Turn your knob to test the different code features. Hooray! Now it's time to decorate with more hanging beads, jewels, swarovski crystals, or whatever catches your fancy.
Finish your lamp off with a cover to hide your huge mess of wires and your power supply.

I used a wooden bowl turned upside-down on the top of my lamp and screwed into the lamp base. I used a nice looking lamp cord and an easy-to-wire plug from the hardware store to hang my lamp and make it look finished.

It took a couple tries to find the center point so my lamp hangs straight. I eventually got it almost right, and added a small piece of steel to finish out the balance.