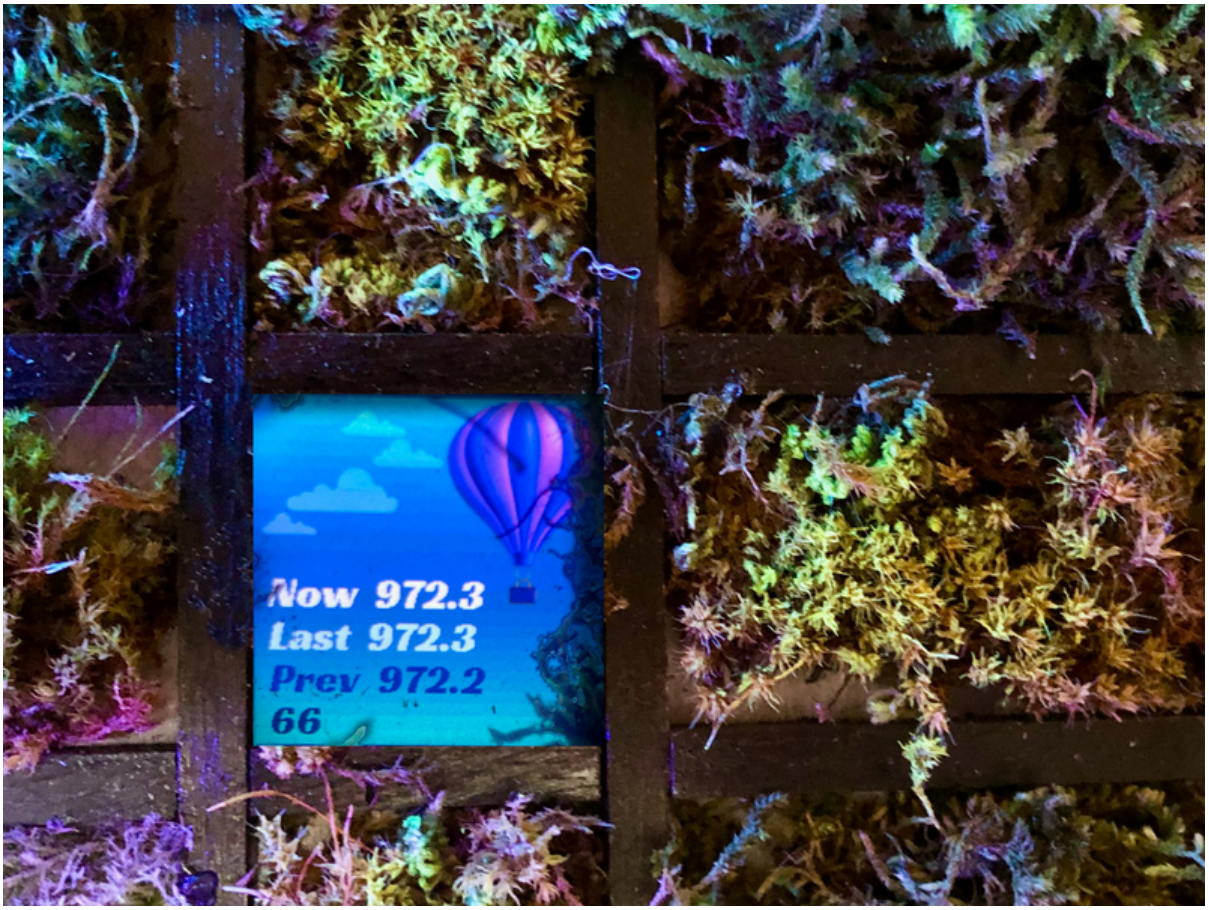
CLUE Vertical Garden Weather Visualizer

Created by Erin St Blaine



https://learn.adafruit.com/clue-vertical-garden-weather-visualizer

Last updated on 2024-06-03 03:06:13 PM EDT

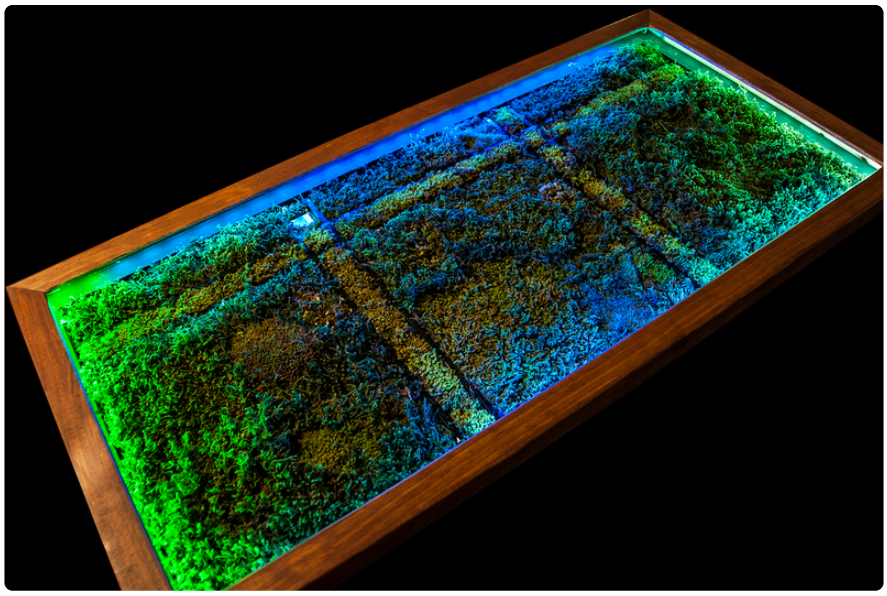
Table of Contents

Overview

The CLUE microcontroller is a powerful little board. In this project, we've used it to power a NeoPixel strip installed in a frame around a vertical moss garden.

The CLUE does a great job of lighting the plants artfully, and we've taken it a step further by incorporating data from the CLUE's onboard weather sensors. This vertical garden functions as a barometer or thermometer, showing colors based on the current and recent air pressure or temperature. Five different light zones show you at a glance whether the air pressure is rising or falling, turning the art piece into a functional barometer.

This lovely vertical moss garden predicts the weather. It's art with a superpower! That's pretty neat.



We used fresh moss from the backyard, preserved in glycerine and dyed to maintain its vibrant green color. That means there's no watering needed -- the moss will stay fresh and soft indefinitely on its own.

We installed the moss into a pre-existing frame cut down from an old, broken shoji screen we found in the basement. Then we added a frame for the lights made from corner trim from the hardware store.

The electronics build is fairly simple, with no soldering required. This is not really a beginner project, but you don't have to be an electronics wizard to hook everything up. The trickiest part for me was the woodworking -- building a frame from corner trim is harder than it looks!

The rest of the magic happens with CircuitPython code and your imagination. Customize the type of data shown: we've included code to visualize temperature or air pressure, and it wouldn't be too hard to add your own code that would visualize humidity, ambient light, sound or other data collected by the CLUE.

You can also customize colors, data thresholds, and pixel mapping (how the colors are laid out) in the CircuitPython code, so it's fun to use this guide as a starting point to create something unique and beautiful.

Parts

[Adafruit CLUE - nRF52840 Express with Bluetooth LE]

Do you feel like you just don't have a CLUE? Well, we can help with that - get a CLUE here at Adafruit by picking up this sensor-packed development board. We wanted to build some...

https://www.adafruit.com/product/4500

Adafruit NeoPixel Digital RGB LED Strip - White 30 LED

You thought it couldn't get better than our world-famous 32-LED-per-meter Digital LED strip but we will prove you wrong! These...

https://www.adafruit.com/product/1376

5V 2A (2000mA) switching power supply - UL Listed

This is an FCC/CE certified and UL listed power supply. Need a lot of 5V power? This switching supply gives a clean regulated 5V output at up to 2000mA. 110 or 240 input, so it works...

https://www.adafruit.com/product/276

JST PH 2-Pin Cable - Female Connector 100mm

Red and black tinned wires with a 2-pin JST PH connector on the end. 4" / 100mm long. Matches up nicely with our Lipoly chargers!

https://www.adafruit.com/product/261

Female DC Power adapter - 2.1mm jack to screw terminal block

If you need to connect a DC power wall wart to a board that doesn't have a DC jack - this adapter will come in very handy! There is a 2.1mm DC jack on one end, and a screw terminal...

https://www.adafruit.com/product/368

Bolts to attach LEDs to the CLUE with no soldering

1 x Bolt-On Kit https://www.adafruit.com/product/4103

Bolts to attach LEDs to the CLUE with no soldering

---

3 x Wire https://www.adafruit.com/product/1877

Silicone Stranded Wire in red, black, and white

---

2 x UV Pigment https://www.adafruit.com/product/4126

Fluorescent Pigment in Yellow & Green

Moss Prep & Preservation

- 1 qt of glycerine
- Warm water dyes in your favorite greens
- White vinegar
- Fluorescent pigments in yellow (http://adafru.it/4126) and green (http://adafru.it/4125)

Frame Materials

- Picture frame or screen with backing
- 2" corner trim
- Stain or paint as desired
- Silicone Adhesive (https://adafru.it/DeD) to secure the LEDs to the frame
- Picture hanging hardware

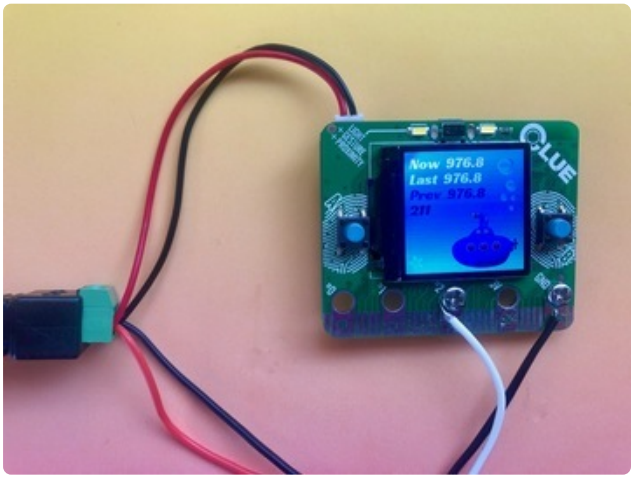
Tools & Other Stuff We Used

- Table saw or miter saw
- Drill
- Screw driver
- Heat gun
- Solder Seal Wire Connectors (https://adafru.it/KSa)
- Bolt-on kit (http://adafru.it/4103) to attach the wires to the CLUE
- Silicone stranded wire (http://adafru.it/4103) in various colors

A soldering iron is NOT required for this project, though if you've got one, you can use it in place of the solder seal wire connectors and the bolt-on kit.

How It Works

User Interface



The CLUE screen displays the current barometric pressure as well as the last two readings, so you can see whether the pressure is rising or falling. Rising pressure will display a hot air balloon image on the screen, and sinking pressure will display a submarine.

It also shows a counter, counting up to the next reading.

The CLUE has two onboard buttons. I've used button A as an on/off switch, turning off both the NeoPixel lights and the screen. Button B turns just the screen on and off, while leaving the LED lights on.

NeoPixel Visualizer

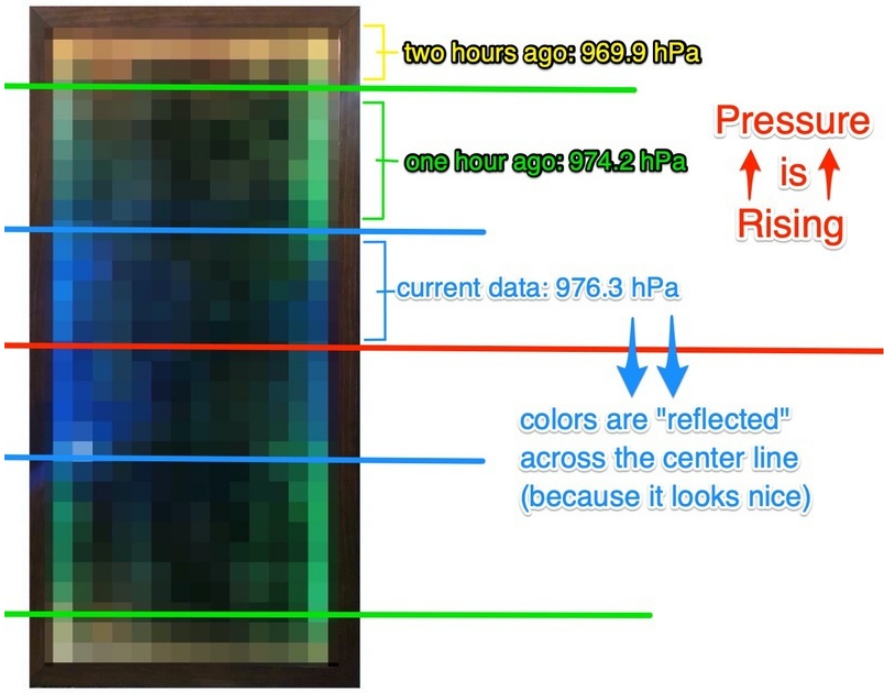
The NeoPixels will show one of five color palettes based on the pressure reading:

- Below 960: Red
- 960-965: Yellow
- 965-970: Green
- 970-975: Blue
- Above 975: Purple

That's nice and simple, but not very meaningful. It's great to know what the air pressure is, but to predict the weather with a barometer, we need to take several readings over the course of a few hours to see whether the pressure is rising or dropping.

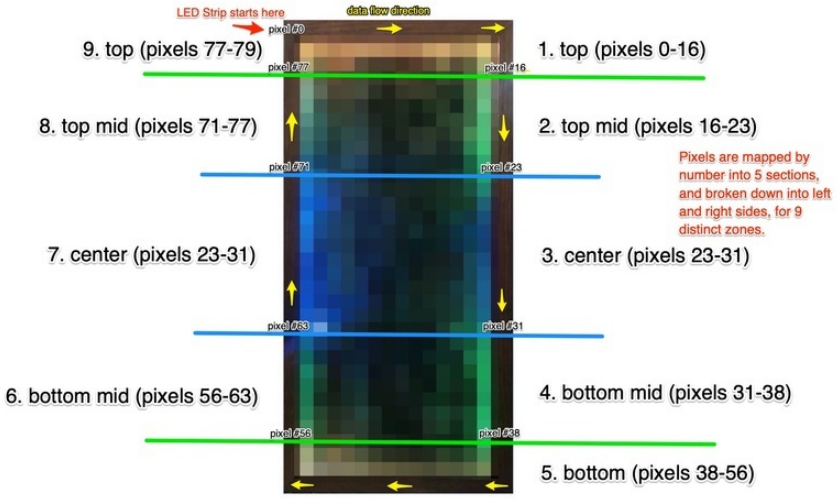
All you ever wanted to know about barometers and how they work, from the Art of Manliness website (https://adafru.it/KD4).

Rising pressure generally means good weather is on the way, and rapidly dropping pressure usually means a storm's a'comin'. (Your results may vary based on altitude, temperature, humidity, and whether the moon is in Aquarius -- meteorology is a complicated science).



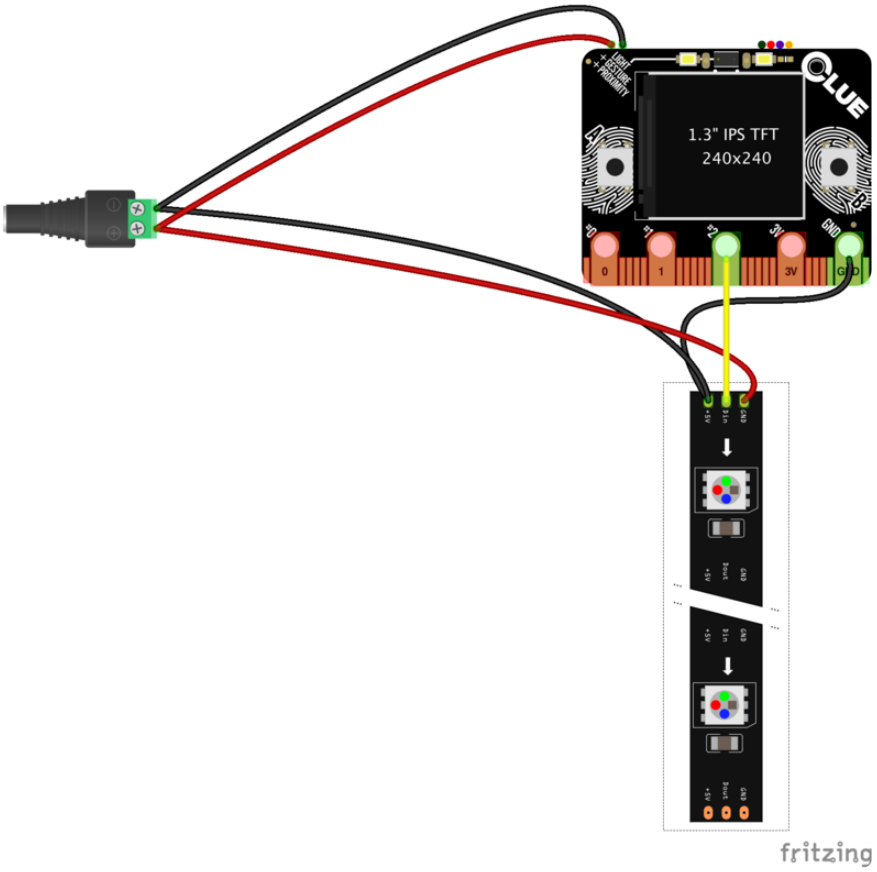
We've put current data in the center of the piece, then as the data gets older it moves outwards towards the top and bottom edges. It's laid out like a reflected gradient, so no matter what the pressure does our artwork will look beautiful and symmetrical.

So, now that we know what we want to show and where we want to show it, we need to map out our pixel numbers so the colors correspond to the correct area on the frame.



As you're laying your frame out, decide where your first pixel will be placed and count out your zones starting from that pixel. Write down which pixel numbers will be in which zone -- you'll need to know this to customize the code later on.

Wiring Diagram



Wire Connections

- NeoPixel 5v --> Power terminal +
- NeoPixel **GND** --> CLUE **GND** and Power terminal **-**
- NeoPixel **IN** --> CLUE **#2**

We're also using the JST power port on the CLUE:

- Power terminal + --> JST +
- Power terminal - --> JST -

Don't use the 3v pin to power the CLUE. The power supply we're using is a 5v supply, and it could damage the board. This wiring method ensures that the NeoPixels get the full 5v they desire, while keeping the CLUE powered safely through its power port so it gets the 3v it wants.

Moss Preparation

You can buy dried and dyed moss at many larger craft stores, but it's more of an adventure to go and collect it yourself, if you are lucky enough to live near a river or forest.

It's best to go the day after a rain. Wet, living moss is a joy to hold and touch, and peels off the rock in a large, cohesive sheet, if you're careful. There are so many textures and varieties right in my own backyard!

Get more than you think you'll need. Using moss growing on rocks works a bit better than moss from a tree or the ground, since there's less bark or dirt to remove.

Soak and Clean

Soak your moss in water overnight, then lay it out to dry. I sprayed mine with the hose to help get rid of any remaining dirt or leaves or insect forest friends.

Preserve and Dye

Fill a large pot with:

1 part glycerine
2 parts hot water
A bit of green fabric dye
A dash of white vinegar to set the dye
Submerge the moss completely, and leave it in there for an hour or more, until the water cools down. Wring the preserved moss out -- you can wring it into your pan and reheat and reuse the liquid for the next batch. Lay it out and let it dry completely.

Wear gloves to keep the dye off your hands!

More About Dyeing & Coloration

I used a forest green and a deep yellow dye from Dharma.

You could also use RIT dye (usually used for tie-dye) since that's pretty easy to find -- they have it at any craft store, or at most grocery stores in the craft / school supply section.

I wasn't all that scientific about how much dye I used. I wanted a range of greens from light to dark for my finished piece, so I just added more whenever I felt it was needed.

The glycerine solution preserves the softness of the moss, but not the color. As it dries out, it will turn brown if you don't use dye. Here's an image showing fresh moss, preserved moss, and preserved and dried moss.

As you can see, the fresh moss is still much more vibrant and appealing than the dyed moss. I may have had better results using a lighter green dye, but the forest green color was what I had on-hand. It was a little darker than what I wanted, so I had to get a little creative.

Ultraviolet Pigments

The reason bright living moss looks so very vibrant in the spring time is that it's putting out ultraviolet light. Apparently, moss can send out signals in UV/A light (https://adafru.it/KC-), though it doesn't seem to be a very well understood phenomenon yet.

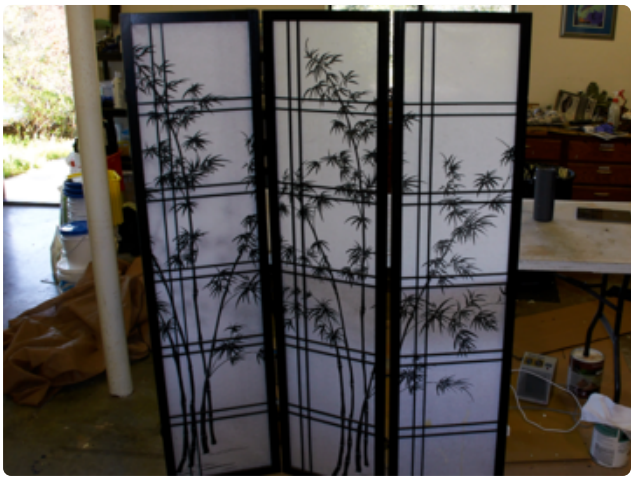
So, to recreate the look of vibrant, living moss, I added some UV pigment (http://adafru.it/4125) in green and yellow. I just painted it over the top of the moss with a wet paintbrush after the moss was mostly dry from the dyeing.

This worked like a dream. My moss is vibrant and glowing and looks almost alive. It's Zombie moss! Also, this step had the added bonus of giving me control of more of a range of colors for my final artwork.

And if I shine blue/purple NeoPixel lights on it, I get a bit of fluorescent play on the moss tips! Yes, it sounds a bit weird, like a velvet Elvis painting, but it sure looks lovely.

Build the Frame

We used repurposed materials and things we had lying around the shop to build our frame. Since your project will likely be different, I'll just go over general details here. There are lots of great frame-building tutorials online, so take a look around to figure out exactly what will work for you.



We used an old, broken shoji screen we found in the basement. The paper is torn in places, but the wood and frames are still in great shape, and it has a really interesting look to it with the additional wood trim inside. Perfect!

You could also repurpose an old picture frame, find an old mirror or painting at a junk shop, or buy something new that suits your vision.

We unscrewed the hinges holding the panels together, then cut the screen down with a chop saw and moved the bottom of the black frame up, to get it to the size we wanted.

The shoji screens were painted a shiny, glossy black. While this looks great on a screen, the gloss really reflected the LED lights in a way we weren't too crazy about. We wanted the lights to highlight the moss, not the frame. So we repainted the front of the frames with the blackest black we could find -- BLK3.0 (https://adafru.it/KEs). This paint is really really black! It completely killed the NeoPixel reflections.

This paint has a fascinating backstory. Here's a delightful episode of the 99% Invisible podcast (https://adafru.it/KEt) about VANTA black and BLK3.0 and Anush Kapoor -- settle in for a good listen while you're waiting for your paint to dry.

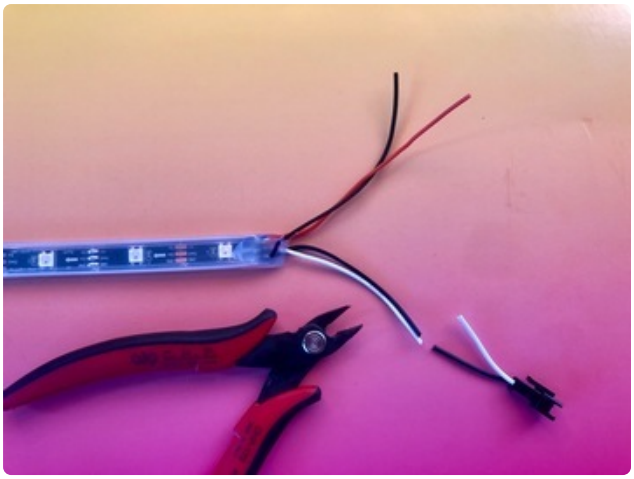
After some experimentation, we decided the lights looked best when they were set above the frame by an inch or so. This way the lights would illuminate the tips of the moss instead of getting lost in the depths. We built a secondary frame from 2" corner moulding and stained it to match our existing wainscot. We used corner clamps to glue the corners together, and reinforced them inside as well, making sure to leave a channel for the LEDs.

We notched the edge of the shoji screen so the outer frame would fit snugly on top. The frame sits up an inch or so above the screen, leaving room for the NeoPixel strip inside.

NeoPixel Assembly

We'll wire up the CLUE and NeoPixel strip up using the bolt-on kit and a screw terminal first, so we can get the code working and calibrated. Nothing is attached permanently at the moment -- we'll want to adjust wire lengths and thread the wires through the frame before we do that.

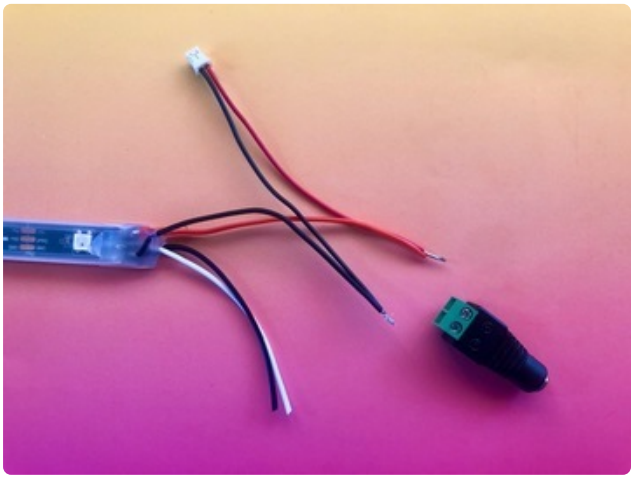
But we do need to test and calibrate our code, so don't skip this part. You've got plenty of time while your moss is drying.



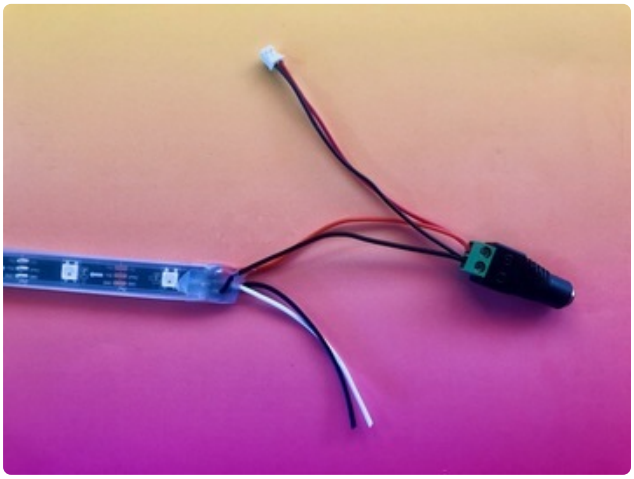
Find the IN end of your NeoPixel strip. If you look closely, you'll see arrows pointing down the length of the strip, and if they're pointing **away** from you, you know you've found it.

Cut off the 2-pin connector. You should be left with two black wires, a red wire, and a white wire.

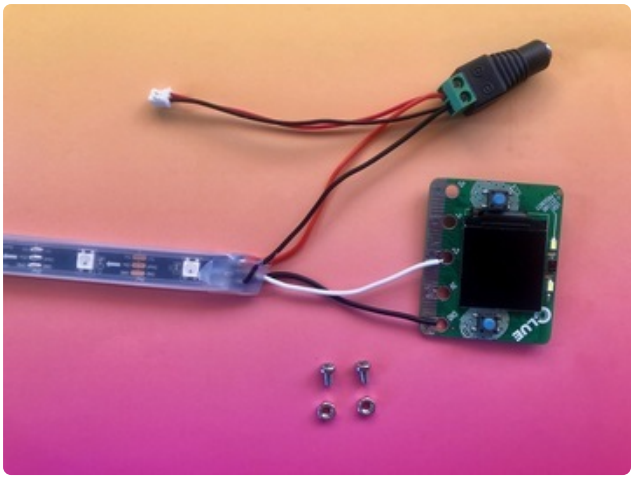
If your strip does not have pre-soldered wires that match this photo, or if you're not starting at the beginning of a brand-new strip, solder and/or splice your wires until you've got this configuration.

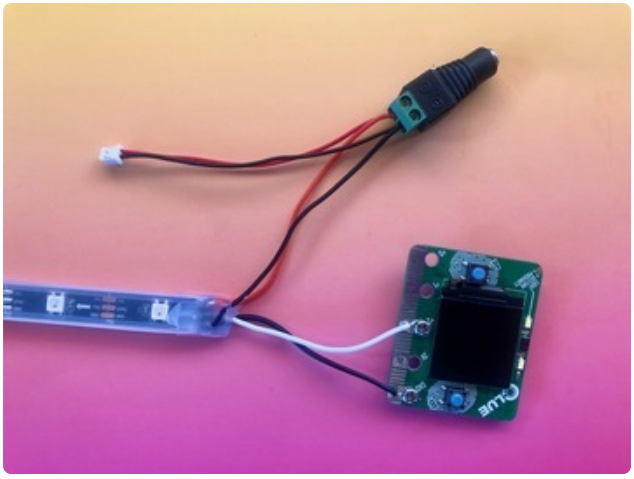


Strip about 1/4 inch of shielding off all the wires and also from your 2-pin JST power connector. Twist the red wire from the JST connector together with the red wire from the NeoPixel. Repeat with the black wire. It doesn't matter which black wire you use from the NeoPixel strip.

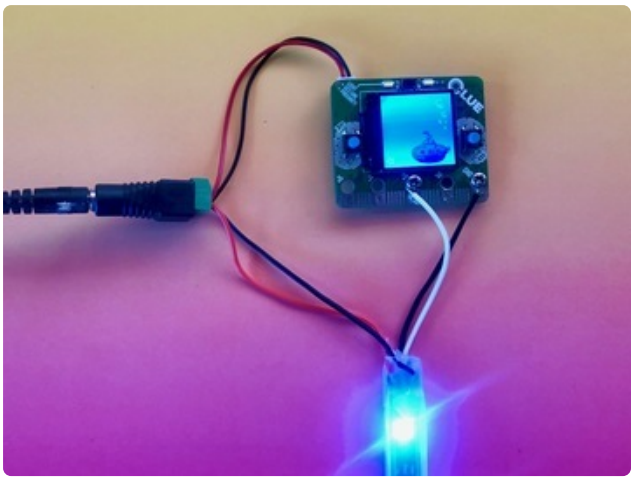


Use a screwdriver to open the screw terminal ports all the way. Slide the red wire into the + side and the black wire into the - side, and tighten the wires down. Give them a tug to be sure the connection is tight. Sometimes it takes a couple tries to get them firmly connected.





Next we'll attach the white wire and the other black wire to the CLUE. The white wire goes to #2, and the black wire goes to GND. If you're using the bolt-on kit to secure the wires, thread the wire through the hole, then insert the screw and tighten the bolt on the back. This is a bit more secure than wrapping the wire around the screw head.



Plug the JST connector into the CLUE and plug the screw terminal into your power supply. Now you're all set up for testing and customizing your code.

Lay out your NeoPixel strip around your frame and figure out exactly how many pixels you'll be using. We'll need to adjust some numbers in the code to map the colors out correctly. My strip has 79 pixels. Count yours out and write this number down somewhere.

Cut your strip carefully after the last pixel right through the copper pads, and seal up the cut end with a little hot glue.

CircuitPython on CLUE

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** flash drive to iterate.

The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

Set up CircuitPython Quick Start!

Follow this quick step-by-step for super-fast Python power :)

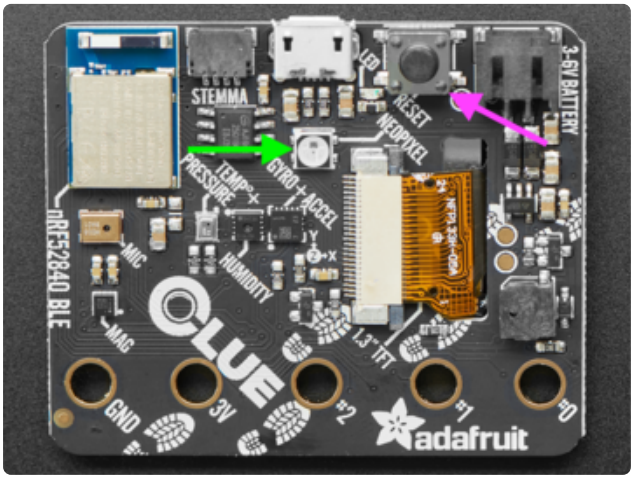
Download the latest version of CircuitPython for CLUE from circuitpython.org

https://adafru.it/IHF

Click the link above to download the latest version of CircuitPython for the CLUE.

Download and save it to your desktop (or wherever is handy).

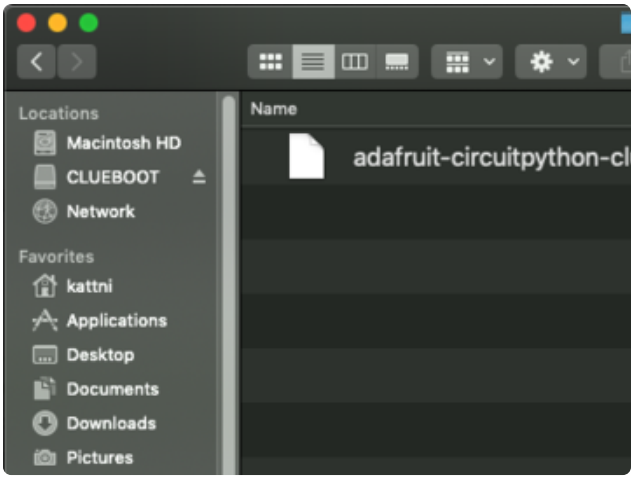
Plug your CLUE into your computer using a known-good USB cable.



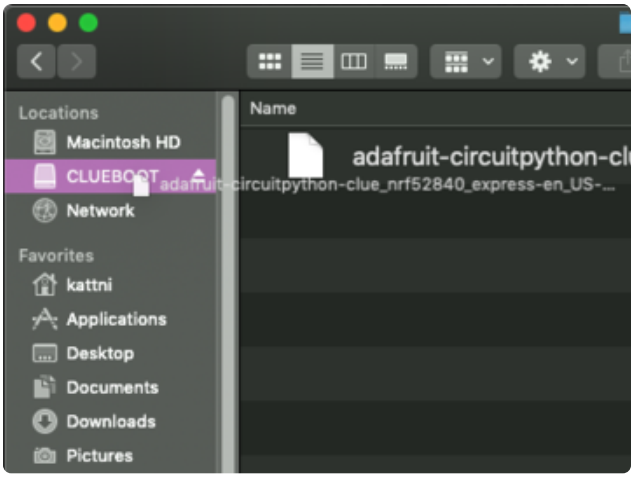
A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

Double-click the **Reset** button on the top (magenta arrow) on your board, and you will see the NeoPixel RGB LED (green arrow) turn green. If it turns red, check the USB cable, try another USB port, etc. **Note:** The little red LED next to the USB connector will pulse red. That's ok!

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

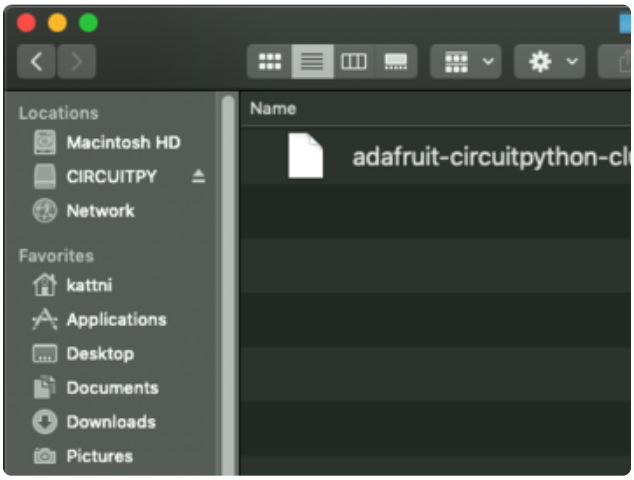


You will see a new disk drive appear called **CLUEBOOT**.



Drag the **adafruit-circuitpython-clue-etc.uf2** file to **CLUEBOOT.**

The LED will flash. Then, the **CLUEBOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.



If this is the first time you're installing CircuitPython or you're doing a completely fresh install after erasing the filesystem, you will have two files - **boot_out.txt**, and **code.py**, and one folder - **lib** on your **CIRCUITPY** drive.

If CircuitPython was already installed, the files present before reloading CircuitPython should still be present on your **CIRCUITPY** drive. Loading CircuitPython will not create new files if there was already a CircuitPython filesystem present.

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

CLUE CircuitPython Libraries

The CLUE is packed full of features like a display and a ton of sensors. Now that you have CircuitPython installed on your CLUE, you'll need to install a base set of CircuitPython libraries to use the features of the board with CircuitPython.

Follow these steps to get the necessary libraries installed.

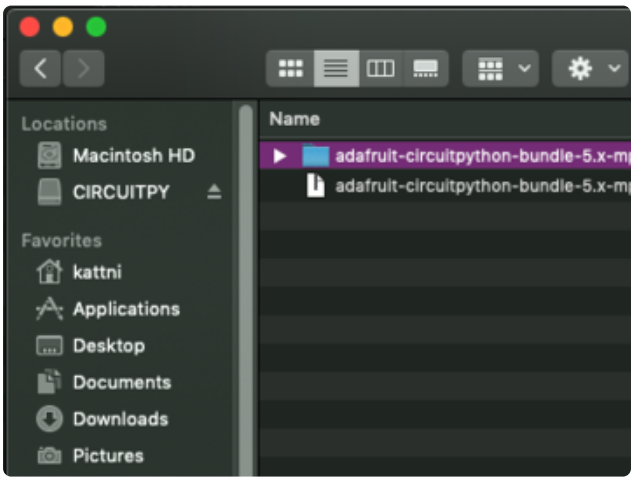
Installing CircuitPython Libraries on your CLUE

If you do not already have a **lib** folder on your **CIRCUITPY** drive, create one now.

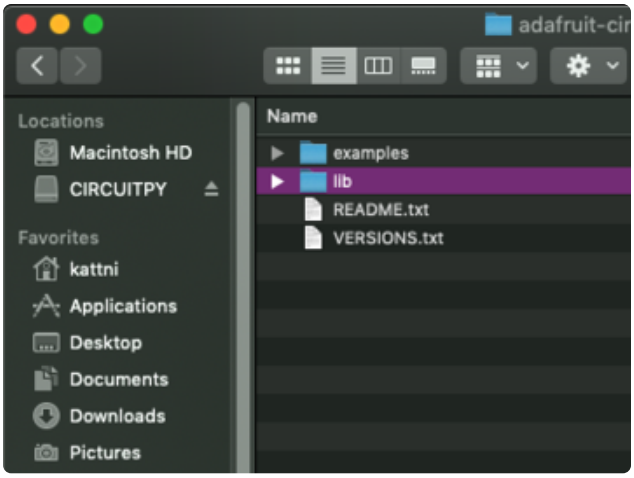
Then, download the CircuitPython library bundle that matches your version of CircuitPython from CircuitPython.org.

Download the latest library bundle from circuitpython.org

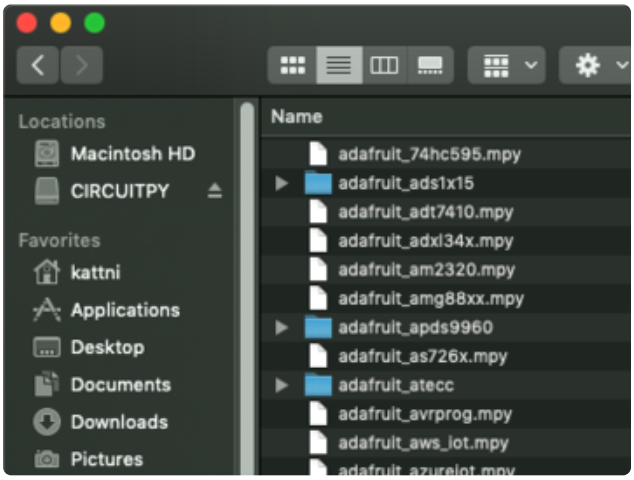
https://adafru.it/ENC



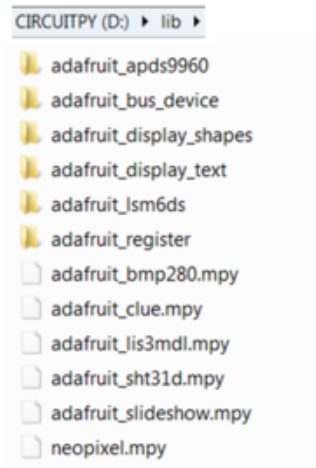
The bundle downloads as a .zip file. Extract the file. Open the resulting folder.



Open the **lib** folder found within.



Once inside, you'll find a lengthy list of folders and .mpy files. To install a CircuitPython library, you drag the file or folder from the **bundle lib folder** to **the lib folder on your CIRCUITPY drive**.



Copy the following folders and files **from the bundle lib folder** to **the lib folder on your CIRCUITPY drive**:

adafruit_apds9960
adafruit_bmp280.mpy
adafruit_bus_device
adafruit_clue.mpy
adafruit_display_shapes
adafruit_display_text
adafruit_lis3mdl.mpy
adafruit_lsm6ds
adafruit_register
adafruit_sht31d.mpy
adafruit_slideshow.mpy
neopixel.mpy

Your lib folder should look like the image on the left. These libraries will let you run the demos in the CLUE guide.

Software

Once you've gotten the board set up, click **Download: Project Zip** below in the code guide. Expand the .zip file and then drag these files to your **CIRCUITPY** Drive:

- rising.bmp
- sinking.bmp
- font (folder)

fonts.zip

https://adafru.it/LKc

Libraries Needed:

- adafruit_apds9960
- adafruit_bitmap_font
- adafruit_bmp280.mpy
- adafruit_bus_device
- adafruit_clue.mpy
- adafruit_display_shapes
- adafruit_display_text
- adafruit_fancyled
- adafruit_imageload
- adafruit_lis3dh.mpy
- adafruit_lis3mdl.mpy
- adafruit_lsm6ds.mpy
- adafruit_register
- adaafruit_sht31d.mpy
- neopixel.mpy

When you're done, your CLUE's **CIRCUITPY** drive should look like this.

Code

Copy the code from the code-block above and paste it into the Mu editor and save it to your CLUE as **code.py** (or copy **code.py** from the zip file and place on the **CIRCUITPY** drive).

Don't make any changes yet! It's tempting to go ahead and add your `num_leds` variable or start customizing right away, but in this case, that may break the code. At first, just upload the code as-is to be sure you've got it everything installed as it should be.

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

"""
Read the barometric reading in the air
Visualize air reading changes over time as a color animation on a NeoPixel strip
Display a "sinking" or "rising" graphic on the screen along with recent reading data

Code by Erin St Blaine for Adafruit Industries :)
"""
import time
import board
import neopixel
from adafruit_clue import clue
import adafruit_fancyled.adafruit_fancyled as fancy
import displayio
from adafruit_display_text import label
from adafruit_bitmap_font import bitmap_font

num_leds = 79 #number of LEDs in your strip
timeToCheck = 23400 # set the amount of time between sensor checks. 7800 is approx.
1 hour

# Barometer or Thermometer? Uncomment the section you want to use

#BAROMETER RANGES (hPa)
```

```
#set desired reading range -- the NeoPixel palette choice will be determined by
these thresholds
deviceType = 0
min_reading = 960
med_reading = 965
high_reading= 970
max_reading = 975

"""
# THERMOMETER RANGES (C)
# set desired temperature range - NeoPixel palette choice determined by these
thresholds
deviceType = 1
min_reading = 25
med_reading = 26
high_reading= 27
max_reading = 28
"""

# get an initial sensor reading
if deviceType ==0:
    reading = clue.pressure
else:
    reading = clue.temperature

#set up variables for "remembering" past readings
reading1 = reading
reading2 = reading1
reading3 = reading2
counter = 0
toggle = 1  # for on/off switch on button A
displayOn = 1  # to turn the display on and off with button B
button_b_pressed = False
button_a_pressed = False

clue.display.brightness = 0.8
clue_display = displayio.Group()

# draw the rising image
# CircuitPython 6 & 7 compatible
rising_file = open("rising.bmp", "rb")
rising_bmp = displayio.OnDiskBitmap(rising_file)
rising_sprite = displayio.TileGrid(rising_bmp, pixel_shader=getattr(rising_bmp,
'pixel_shader', displayio.ColorConverter()))

# # CircuitPython 7+ compatible
# rising_bmp = displayio.OnDiskBitmap("rising.bmp")
# rising_sprite = displayio.TileGrid(rising_bmp,
pixel_shader=rising_bmp.pixel_shader)

clue_display.append(rising_sprite)

# draw the sinking image
# CircuitPython 6 & 7 compatible
sinking_file = open("sinking.bmp", "rb")
sinking_bmp = displayio.OnDiskBitmap(sinking_file)
sinking_sprite = displayio.TileGrid(sinking_bmp, pixel_shader=getattr(sinking_bmp,
'pixel_shader', displayio.ColorConverter()))

# # CircuitPython 7+ compatible
# sinking_bmp = displayio.OnDiskBitmap("sinking.bmp")
# sinking_sprite = displayio.TileGrid(sinking_bmp,
pixel_shader=sinking_bmp.pixel_shader)

clue_display.append(sinking_sprite)

# Create text
# first create the group
text_group = displayio.Group()
```

```
# Make a label
reading_font = bitmap_font.load_font("/font/RacingSansOne-Regular-29.bdf")
reading_font.load_glyphs("0123456789ADSWabcdefghijklmnopqrstuvwxyz:!".encode('utf-8'))
reading_label = label.Label(reading_font, color=0xffffff)
reading_label.x = 10
reading_label.y = 24
text_group.append(reading_label)

reading2_label = label.Label(reading_font, color=0xdaf5f4)
reading2_label.x = 10
reading2_label.y = 54
text_group.append(reading2_label)

reading3_label = label.Label(reading_font, color=0x4f3ab1)
reading3_label.x = 10
reading3_label.y = 84
text_group.append(reading3_label)

timer_label = label.Label(reading_font, color=0x072170)
timer_label.x = 10
timer_label.y = 114
text_group.append(timer_label)

clue_display.append(text_group)
clue.display.root_group = clue_display

# Define color Palettes
waterPalette = [0x00d9ff, 0x006f82, 0x43bfb9, 0x0066ff]
icePalette = [0x8080FF, 0x8080FF, 0x8080FF, 0x0000FF, 0xC88AFF]
sunPalette = [0xffaa00, 0xffdd00, 0x7d5b06, 0xfffca8]
firePalette = [0xff0000, 0xff5500, 0x8a3104, 0xffaa00 ]
forestPalette = [0x76DB00, 0x69f505, 0x05f551, 0x3B6D00]

# set up default initial palettes, just for startup
palette = forestPalette
palette2 = waterPalette
palette3 = icePalette

# Declare a NeoPixel object on pin A4 with num_leds pixels, no auto-write.
# Set brightness to max because we'll be using FancyLED's brightness control.
pixels = neopixel.NeoPixel(board.A4, num_leds, brightness=1.0,
                           auto_write=False)

offset = 0  # Positional offset into color palette to get it to 'spin'

while True:
    # use button A to toggle the NeoPixels on or off by changing brightness
    if clue.button_a and not button_a_pressed:  # If button A pressed...
        print("Button A pressed.")
        if toggle == 1:
            toggle = 0
            pixels.brightness = 0
            clue.display.brightness = 0
        elif toggle == 0:
            toggle = 1
            pixels.brightness = 1.0
            clue.display.brightness = 0.8
        button_a_pressed = True  # Set to True.
        time.sleep(0.03)  # Debounce.
    if not clue.button_a and button_a_pressed:  # On button release...
        button_a_pressed = False  # Set to False.
        time.sleep(0.03)  # Debounce.
    if clue.button_b and not button_b_pressed:  # If button B pressed...
        print("Button B pressed.")
        # Toggle only the display on and off
        if displayOn == 0:
            clue.display.brightness = 0.8
            displayOn = 1
        else:
```

```
            clue.display.brightness = 0
            displayOn = 0
        button_b_pressed = True  # Set to True.
        time.sleep(0.03)  # Debounce.
    if not clue.button_b and button_b_pressed:  # On button release...
        button_b_pressed = False  # Set to False.
        time.sleep(0.03)  # Debounce.

    # assign color palette to NeoPixel section 1 based on the current reading
reading
    if reading1 < min_reading:
        palette = firePalette
    elif min_reading > reading1 > med_reading:
        palette = sunPalette
    elif med_reading > reading1 > high_reading:
        palette = forestPalette
    elif high_reading > reading1 > max_reading:
        palette = waterPalette
    else:
        palette = icePalette
    # Map colors to pixels. Adjust range numbers to light up specific pixels. This
configuration
    # maps to a reflected gradient, with pixel 0 in the upper left corner
    # Load each pixel's color from the palette using an offset, run it
    # through the gamma function, pack RGB value and assign to pixel.
    for i in range(23, 31):  #center right -- present moment
        color = fancy.palette_lookup(palette, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(63, 71): #center left -- present moment
        color = fancy.palette_lookup(palette, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(16, 23): #top mid right  -- 1 cycle ago
        color = fancy.palette_lookup(palette2, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(71, 78): #top mid left  -- 1 cycle ago
        color = fancy.palette_lookup(palette2, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(31, 38): #bottom mid right  -- 1 cycle ago
        color = fancy.palette_lookup(palette2, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(56, 63): #bottom mid left  -- 1 cycle ago
        color = fancy.palette_lookup(palette2, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(0, 16): #top right  -- 2 cycles ago
        color = fancy.palette_lookup(palette3, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(77, 79): #top left  -- 2 cycles ago
        color = fancy.palette_lookup(palette3, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(38, 56): #bottom  -- 2 cycles ago
        color = fancy.palette_lookup(palette3, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()
```

```
    pixels.show()
    offset += 0.01  # Bigger number = faster spin

    reading_label.text = "Now  {:.1f}".format(reading1)
    reading2_label.text = "Last  {:.1f}".format(reading2)
    reading3_label.text = "Prev  {:.1f}".format(reading3)
    timer_label.text = "{}".format(counter)
    clue.display.root_group = clue_display

    # Is it time to update?
    if counter > timeToCheck:
        # This moves the current data to the "1 hour old" section of pixels and the
"1 hour old"
        # data to the "2 hours old" section of pixels
        palette3 = palette2
        palette2 = palette
        reading3 = reading2
        reading2 = reading1
        reading1 = reading
        # take a new sensor reading and reset the counter
        if deviceType == 0:
            reading = clue.pressure
        else:
            reading = clue.temperature
        counter = 0
        # if reading is rising, show rising image and position text at the bottom
        if reading1 > reading2:
            sinking_sprite.x = 300
            reading_label.y = 134
            reading2_label.y = 164
            reading3_label.y = 194
            timer_label.y = 224
        # if reading is falling, show sinking image and position text at the top
        elif reading1 < reading2:  #reading is falling
            sinking_sprite.x = 0
            reading_label.y = 24
            reading2_label.y = 54
            reading3_label.y = 84
            timer_label.y = 114
    # otherwise keep counting up
    else:
        counter = counter + 1
```

Troubleshooting

If all goes well, your NeoPixels have come on in various colors and your screen is showing either a submarine image or a hot air balloon image, as well as a pressure readout. Hooray!

If that didn't happen, here are a few things to try:

- Double check that you have all the libraries shown above installed in your board's **lib** folder
- Make sure the fonts are installed in a folder called **fonts** at the root of your **CIRCUITPY** drive
- Check to be sure your .bmp files are both sitting at the root of your **CIRCUITPY** drive

- Try copying and saving **code.py** once more. Don't change anything in the code yet! Let's get it working as-is first.

If you're using the mu editor to load your code, click the Serial button to open the serial monitor, then press **ctrl-D** to enter the REPL. This will run your code line-by-line and give you a clue as to what is going wrong. More about this lovely debugging tool here (https://adafru.it/KSb)

If your screen is showing images and text but your LEDs aren't lighting up:

- Double check your wiring. Is the white / data wire going to pin #2 and the black / ground wire going to G?
- Check to be sure you've connected to the **IN** end of your NeoPixel strip instead of the **OUT** end. It won't work if you hook them up backwards.

If you've checked all of this and it's still not working, try updating and reloading CircuitPython on the CLUE. Here are instructions about how to do that (https://adafru.it/Jab).

Customizing Your Code

Let's take a look at the code to see what everything is doing, and so that you can calibrate your artwork to work in your location.

There are several important changes to make, so don't skip this section!

What follows are small sections of code from the full program. To download the whole thing (and associated fonts, libraries, etc.), use the “Download Project Bundle” button on the prior page. Then return here to see what different parts of the code do.

First, we import all the necessary libraries.

```
import time
import board
import neopixel
from adafruit_clue import clue
import adafruit_fancyled.adafruit_fancyled as fancy
import displayio
from adafruit_display_text import label
from adafruit_bitmap_font import bitmap_font
```

The next few sections are where we'll do most of our customization.

```
num_leds = 79 # number of LEDs in your strip
timeToCheck = 100 # set the amount of time between sensor checks. 7800 is approx. 1
hour
```

Change `num_leds` to reflect the number of LEDs in your strip. Mine has 79 lights.

If you have fewer than 79 lights this will break the code, temporarily!

Don't worry! We'll fix it in the **Mapping Colors to Pixels** section below.

The `timeToCheck` variable allows you to set the length of time between sensor checks. This isn't counting seconds, it's counting up each time the code runs. I used a stopwatch to determine that my CLUE board counts to around 7800 every hour.

In general, barometers should be checked every hour to three hours to read a meaningful change in pressure. For testing purposes you can set it to around 20 - 30 to read the sensor and update the strip every few seconds. For actual usage, I am only checking data every 3 hours so I've set this to 23400 (7800 x 3).

Barometer or Thermometer?

The CLUE's onboard barometric pressure sensor (http://adafru.it/2651) also has a temperature sensor built in. That means it's easy peezy to change this project into a thermometer instead of a barometer just by changing a couple lines of code.

I've included sample code for both types of project. Comment out the Barometer section and uncomment the Thermometer section if you'd prefer to build a temperature visualizer.

Setting Threshold Numbers

This section is also where you customize the thresholds for color visualization. Since barometers vary so much by where they're located, the best thing to do is to take a few readings at your location to find low and high pressure, and adjust these numbers as needed. My house is at around 1300 feet of elevation in a moderate weather zone in Northern California. I took a reading on a few nice sunny days to get a high threshold (around 972), and on a day when a thunderstorm was predicted to get a low threshold (around 945), then divided up my numbers into an even spread.

A low desert in Arizona or an island in the middle of the ocean would have pretty different thresholds. Barometers get less reliable as elevation goes up (https://

adafru.it/KDf), and above 5000 feet they're pretty useless, so if you live in the mountains, make a thermometer instead.

The units for pressure are in hectopascals (hPa) or for temperature are in C.

```
# Barometer or Thermometer? Uncomment the section you want to use

# BAROMETER RANGES (hPa)
# set desired reading range -- the NeoPixel palette choice will be determined by
these thresholds
deviceType = 0
min_reading = 965
med_reading = 965
high_reading= 970
max_reading = 975

"""
# THERMOMETER RANGES (C)
# set desired temperature range -- the NeoPixel palette choice will be determined
by these thresholds
deviceType = 1
min_reading = 25
med_reading = 27
high_reading= 31
max_reading = 33
"""

#get an initial sensor reading
if deviceType ==0:
    reading = clue.pressure
else:
    reading = clue.temperature
```

Screen Images

Next we set up and draw the .bmp images. We've included a submarine image for when the pressure is sinking, and a hot air balloon image for when the pressure is rising. You can add your own custom images if you'd like. Make the images 240x240 pixels, and save as .bmp with 16 bits. It's probably easiest to name them **rising.bmp** and **sinking.bmp** -- that way you don't need to update the code with different filenames, you can just replace our files with yours and it should work fine.

```
clue.display.brightness = 0.8
clue_display = displayio.Group()

# draw the rising image
# CircuitPython 6 &amp;amp; 7 compatible
rising_file = open("rising.bmp", "rb")
rising_bmp = displayio.OnDiskBitmap(rising_file)
rising_sprite = displayio.TileGrid(rising_bmp, pixel_shader=getattr(rising_bmp,
'pixel_shader', displayio.ColorConverter()))

# # CircuitPython 7+ compatible
# rising_bmp = displayio.OnDiskBitmap("rising.bmp")
# rising_sprite = displayio.TileGrid(rising_bmp,
pixel_shader=rising_bmp.pixel_shader)

clue_display.append(rising_sprite)
```

```
# draw the sinking image
# CircuitPython 6 &amp;amp; 7 compatible
sinking_file = open("sinking.bmp", "rb")
sinking_bmp = displayio.OnDiskBitmap(sinking_file)
sinking_sprite = displayio.TileGrid(sinking_bmp, pixel_shader=getattr(sinking_bmp,
'pixel_shader', displayio.ColorConverter()))

# # CircuitPython 7+ compatible
# sinking_bmp = displayio.OnDiskBitmap("sinking.bmp")
# sinking_sprite = displayio.TileGrid(sinking_bmp,
pixel_shader=sinking_bmp.pixel_shader)
```

Further down, we set up the images and the text using the display.io library. This is where we can customize the color and position of the text readouts that tell us the current data.

Change the position of the text by changing `reading_label.x` and `reading_label.y` in each text section.

```
# Create text
# first create the group
text_group = displayio.Group()
# Make a label
reading_font = bitmap_font.load_font("/font/RacingSansOne-Regular-29.bdf")
reading_font.load_glyphs("0123456789ADSWabcdefghijklmnopqrstuvwxyz:!".encode('utf-8'))
reading_label = label.Label(reading_font, color=0xffffff)
reading_label.x = 10
reading_label.y = 24
text_group.append(reading_label)

reading2_label = label.Label(reading_font, color=0xdaf5f4)
reading2_label.x = 10
reading2_label.y = 54
text_group.append(reading2_label)

reading3_label = label.Label(reading_font, color=0x4f3ab1)
reading3_label.x = 10
reading3_label.y = 84
text_group.append(reading3_label)

timer_label = label.Label(reading_font, color=0x072170)
timer_label.x = 10
timer_label.y = 114
text_group.append(timer_label)
```

Customizing Color Palettes

The next section is where we set up our color palettes. I'm using hex codes for each color, but you can also use CRGB values or CHSV, or a variety of other options. Check out the FancyLED guide (https://adafru.it/KD5) for more info about how this works.

If you don't care how it works and just want to change the colors, that's okay too! Use an online hex code picker like this one (https://adafru.it/KD6). You can put as few or as many different colors in each palette as you'd like. Just select your color, copy the 6-

digit hex code at the top of the screen, and place it into the code, copying the current format: i.e. for pure green, the hex code is #00FF00, so you'd use `0x00ff00` to express green.

These palettes are set up to give a few different shades of each color in the rainbow. The ice palette is mainly deep blues and purples -- I was trying to get the strip to express a color as close to blacklight / uv as possible to make the uv pigments in the moss really pop.

```
# Define color Palettes
waterPalette = [
            0x00d9ff,
            0x006f82,
            0x43bfb9,
            0x0066ff]
icePalette = [
            0x8080FF,
            0x8080FF,
            0x8080FF,
            0x0000FF,
            0xC88AFF]
sunPalette = [
            0xffaa00,
            0xffdd00,
            0x7d5b06,
            0xfffca8]
firePalette = [
            0xff0000,
            0xff5500,
            0x8a3104,
            0xffaa00 ]
forestPalette = [
            0xccffa8,
            0x69f505,
            0x05f551,
            0x2c8247]

#set up default initial palettes, just for startup
palette = forestPalette
palette2 = waterPalette
palette3 = icePalette
```

The next section sets up the LED strip on pin **A4**, which corresponds to **#2** on the CLUE board. We set brightness to 1.0 which is fully bright, since we will be controlling the brightness through FancyLED.

If you want to make the whole thing dimmer overall, you can lower the brightness here. Otherwise, just choose darker colors for your palettes (closer to the black side of the color layout) to make each individual color darker.

```
# Declare a NeoPixel object on pin A4 with num_leds pixels, no auto-write.
# Set brightness to max because we'll be using FancyLED's brightness control.
pixels = neopixel.NeoPixel(board.A4, num_leds, brightness=1.0,
                           auto_write=False)
```

Main Code Loop

That's it for setup. The main code loop starts next, with `while true:`

Buttons

The first thing I've done is set up toggles for the two buttons. Button A will turn the NeoPixels and the CLUE screen on and off each time it's pressed. Button B turns only the screen on and off, leaving the pixels on.

I've done this by setting the brightness of the pixels and / or the screen to 0, so they don't "forget" what they were showing when you turn them back up again.

Assigning Color Palettes

The next section correlates the palettes we set up with the pressure or temperature thresholds we made way back at the beginning of the code. Choose which palette you'd like to associate with which pressure reading in this section.

```
#assign color palette to NeoPixel section 1 based on the current reading reading
    if reading1 &lt; min_reading:
        palette = firePalette
    elif reading1 &gt; min_reading and reading1 &lt; med_reading:
        palette = sunPalette
    elif reading1 &gt; med_reading and reading1 &lt; high_reading:
        palette = forestPalette
    elif reading1 &gt; high_reading and reading1 &lt; max_reading:
        palette = waterPalette
    else:
        palette = icePalette
```

Mapping Colors to Pixels

Here is where we assign specific colors to show in the area of the artwork we want. I've set up 9 different "zones" around my frame, so that I can create a reflected gradient layout to visualize the colors. This is maybe a leeeetle bit unnecessarily complex, but, that's how you can tell it's Art.

Go find the numbers you wrote down while you were reading the How it Works page, and plug them into the `range()` numbers in the code.

Use the ending number from one zone as the starting number for the next zone, i.e.:

`for i in range (0, 16): ...`

`for i in range (16, 23): ...`

`for i in range (23, 31): ...`

```
# Map colors to pixels. Adjust range numbers to light up specific pixels. This
configuration
#  maps to a reflected gradient, with pixel 0 in the upper left corner
#  Load each pixel's color from the palette using an offset, run it
#  through the gamma function, pack RGB value and assign to pixel.
    for i in range(23, 31):  #center right -- present moment
        color = fancy.palette_lookup(palette, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(63, 71): #center left -- present moment
        color = fancy.palette_lookup(palette, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(16, 23): #top mid right  -- 1 cycle ago
        color = fancy.palette_lookup(palette2, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(71, 78): #top mid left  -- 1 cycle ago
        color = fancy.palette_lookup(palette2, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(31, 38): #bottom mid right  -- 1 cycle ago
        color = fancy.palette_lookup(palette2, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(56, 63): #bottom mid left  -- 1 cycle ago
        color = fancy.palette_lookup(palette2, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(0, 16): #top right  -- 2 cycles ago
        color = fancy.palette_lookup(palette3, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(77, 79): #top left  -- 2 cycles ago
        color = fancy.palette_lookup(palette3, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()

    for i in range(38, 56): #bottom  -- 2 cycles ago
        color = fancy.palette_lookup(palette3, offset + i / num_leds)
        color = fancy.gamma_adjust(color, brightness=0.25)
        pixels[i] = color.pack()
```

Reading the Sensor & Showing the Data

Finally, we write the sensor readings to the screen, then check to see if it's time to read the sensor again. If it is, we do so, and update all the reading variables: `reading2` (data from the last check) becomes `reading3` (data from two checks

ago), reading (current data) becomes `reading2`, and a new sensor reading is taken to replace the current data.

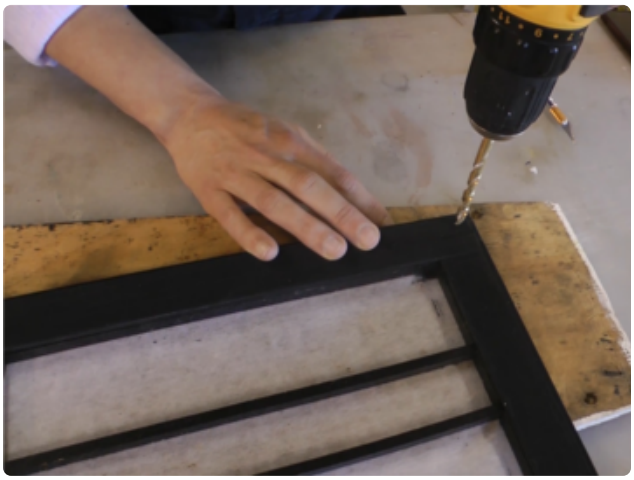
```
reading_label.text = "Now  {:.1f}".format(reading1)
    reading2_label.text = "Last  {:.1f}".format(reading2)
    reading3_label.text = "Prev  {:.1f}".format(reading3)
    timer_label.text = "{}".format(counter)
    clue.display.root_group = clue_display

    # Is it time to update?
    if counter &gt; timeToCheck:
        #This moves the current data to the "1 hour old" section of pixels and the
"1 hour old" data
        #to the "2 hours old" section of pixels
        palette3 = palette2
        palette2 = palette
        reading3 = reading2
        reading2 = reading1
        reading1 = reading
        # take a new sensor reading and reset the counter
        if deviceType == 0:
            reading = clue.pressure
        else:
            reading = clue.temperature
        counter = 0
        # if reading is rising, show rising image and position text at the bottom
        if reading1 &gt; reading2:
            sinking_sprite.x = 300
            reading_label.y = 134
            reading2_label.y = 164
            reading3_label.y = 194
            timer_label.y = 224
        # if reading is falling, show sinking image and position text at the top
        elif reading2 &lt; reading3:  #reading is falling
            sinking_sprite.x = 0
            reading_label.y = 24
            reading2_label.y = 54
            reading3_label.y = 84
            timer_label.y = 114
    # otherwise keep counting up
    else:
        counter = counter + 1
```

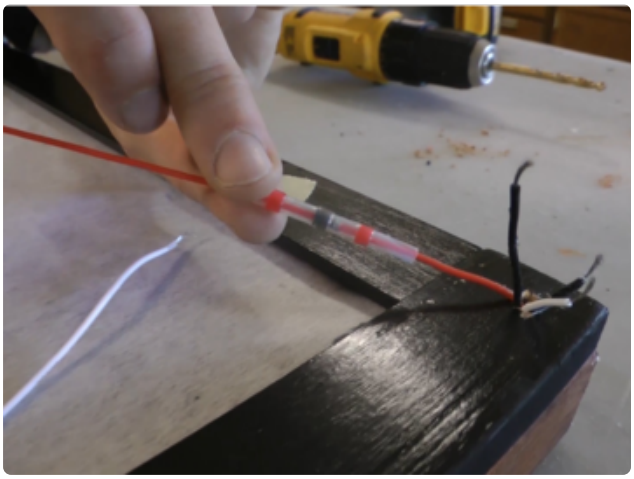
Final Assembly

I lined up the strip inside the frame, with the first pixel placed according to my layout plan. Remember, we're looking at it from the inside, so I had to flip it over in my head to be sure I had the correct corner.

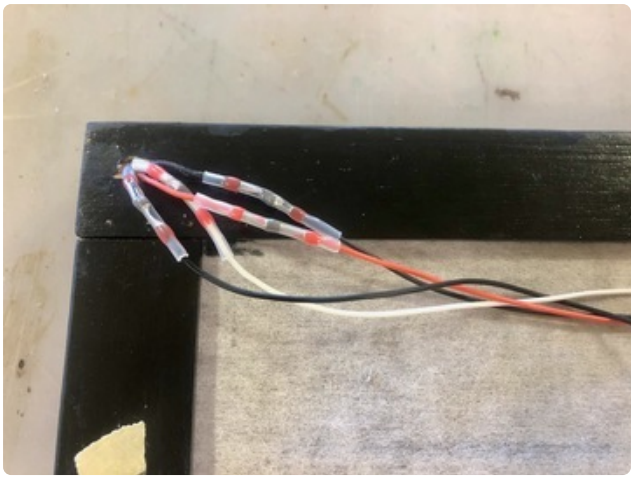
Attach the LED strip to the inside corner of the outer frame using silicone adhesive. This kind from Devcon (https://adafru.it/DeD) is my favorite silicone glue. You can't use E6000 or hot glue or just about any other kind of glue on the silicone LED shielding -- hardly anything sticks to silicone, so shell out for the right kind of glue.



I drilled a hole through the corner of my shoji screen so I could feed the wires through to the back of the project, to keep them out of sight.



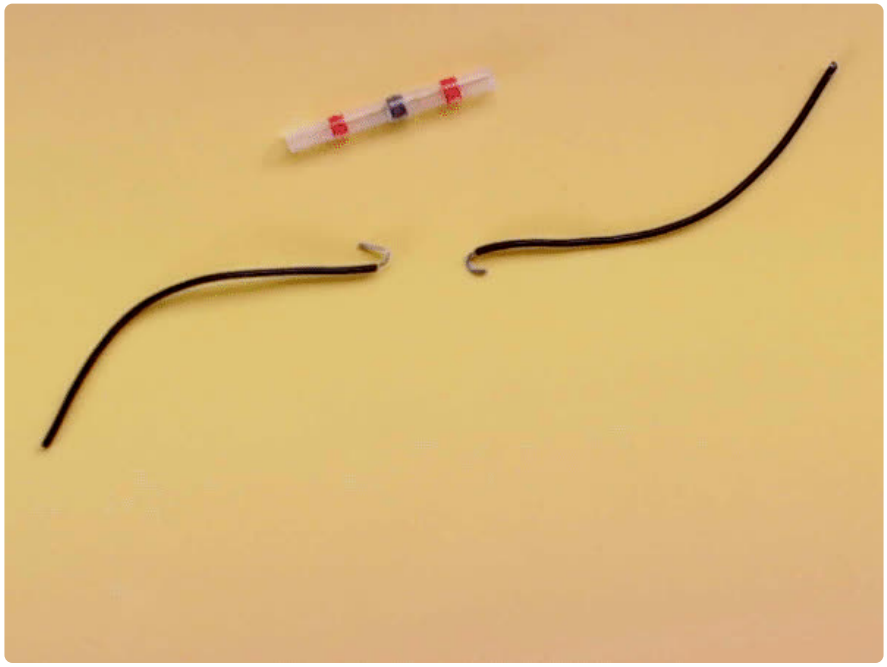
I placed the outer frame onto the shoji screen and poked the NeoPixel wires through the hole. They were too short to reach the CLUE in the spot I'd planned to mount it, so I used solder seal connectors to lengthen all four wires, and made them long enough to reach the CLUE.



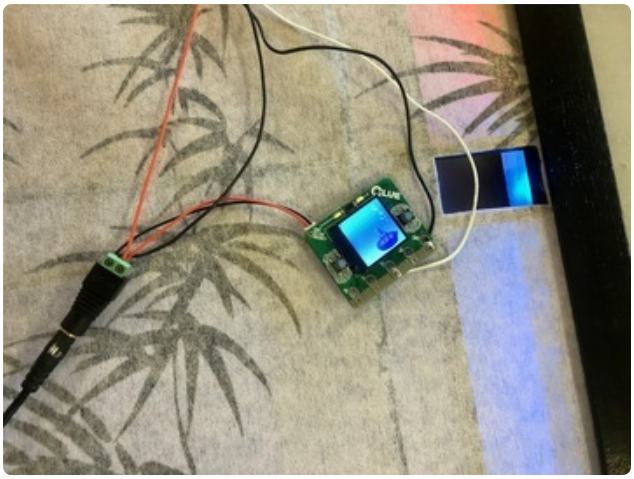
These connectors are really easy to use. Strip about 1/4 inch of shielding from the end of each wire. Twist the wires together firmly, to make a strong physical connection. They shouldn't come apart when you tug on them gently.

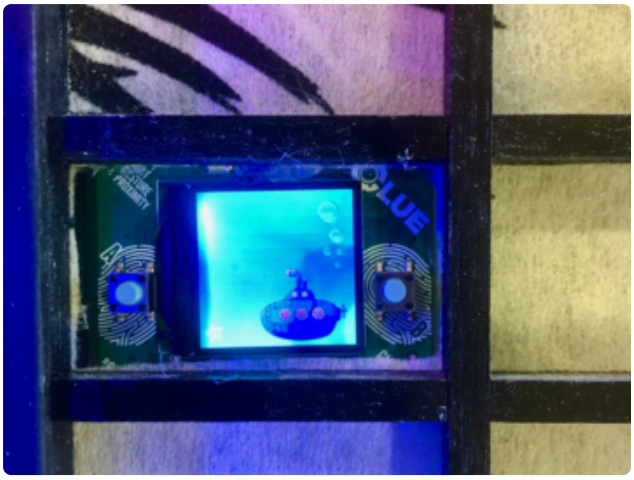
Then, slide the heat shrink connector on so the middle gray ring is positioned right on top of your twisted bare wires. This bit has some metal solder in it -- the other two rings (red, in my case) have hot melt glue.

Use a heat gun to shrink the connector. Hold it on there for a good 20-30 seconds until you see the solder ring melt and spread out a bit. Now you have a soldered, sealed, water-tight wire splice. Awesome!



Please be careful with craft knives and hot glue to prevent injury.





I decided where I wanted to mount the CLUE, and carefully cut through the paper from the back with a very sharp utility knife. Then, I mounted the CLUE in the hole using hot glue. Hot glue is a great choice for mounting electronics since it's really easy to remove -- 99% alcohol on a cotton swab will dissolve the sticking power of the glue completely without damaging the CLUE. This was lucky since I didn't get it centered in the hole the first time, and I was able to reposition it fairly easily.

Next, it was time to attach the frame to the shoji screen. Since there are electronics inside, I didn't want to glue it with wood glue or anything too permanent. If my LED strip fails at some point in the future I want to be able to fix it. Hot glue to the rescue again. I glued all four corners and pressed the frame into place, and it's staying very well.

Finally, it was time to add the moss! This was the most fun part. I arranged all the moss to get a lovely mix of textures and colors in each section of the frame. Once I was happy with the layout, I used hot glue on the back of each piece of moss to secure it to the paper. I packed the moss in fairly tightly so no white paper shows through, and trimmed along the small black separators to make the whole thing look tidy.

Finally, I added picture hanging hardware on the back of the frame, and hung it on the wall! Button A on the CLUE board turns the LEDs on and off, and button B turns just the screen on and off.

The moss has a lovely smell, and a delightful touchable texture. I'm delighted with my weather-predicting vertical garden.