

Bluefruit Playground Hide and Seek

Created by John Park



Last updated on 2020-09-11 03:53:05 PM EDT

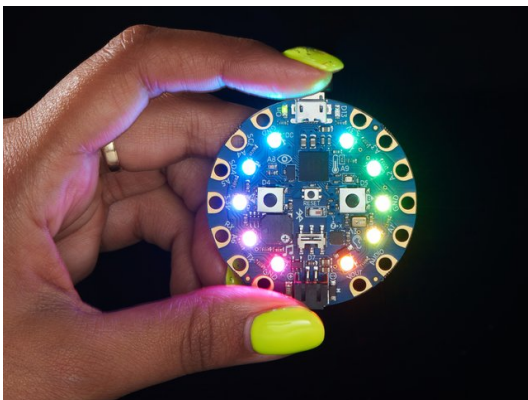
Overview

Stage a creative gift hunt all around your home with Bluetooth LE tracking ornaments! Circuit Playground Bluefruit boards can hone in on the proximity of other boards broadcasting their own signal including color coding! The stronger the signal, the more NeoPixels light up. All coded with CircuitPython.



Parts

The more the merrier here! Get together with some friends or a whole classroom full of Circuit Playground Bluefruits. You'll need at least two CPBs to test the signal strength measurement aspect of this project, and three or more to grab the hidden colors.



[Circuit Playground Bluefruit - Bluetooth Low Energy](#)

OUT OF STOCK

Out Of Stock

3 x [Circuit Playground Bluefruit BLE](#)

Bluetooth Low Energy

Out Of Stock

3 x [Lithium Ion Polymer Battery with Short Cable](#)

3.7V 350mAh

Add To Cart

3 x [DIY Ornament Kit](#)

6cm Diameter - Perfect for Circuit Playground

Add To Cart

3 x [Circuit Playground Enclosure](#)

Clear snap fit case

Add To Cart

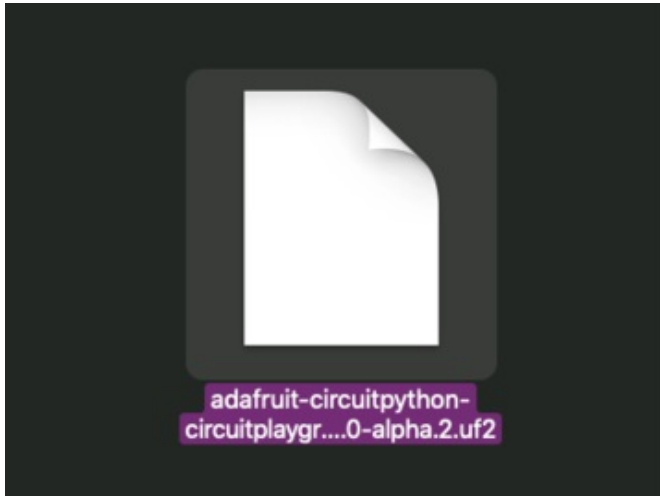
CircuitPython on Circuit Playground Bluefruit

Install or Update CircuitPython

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

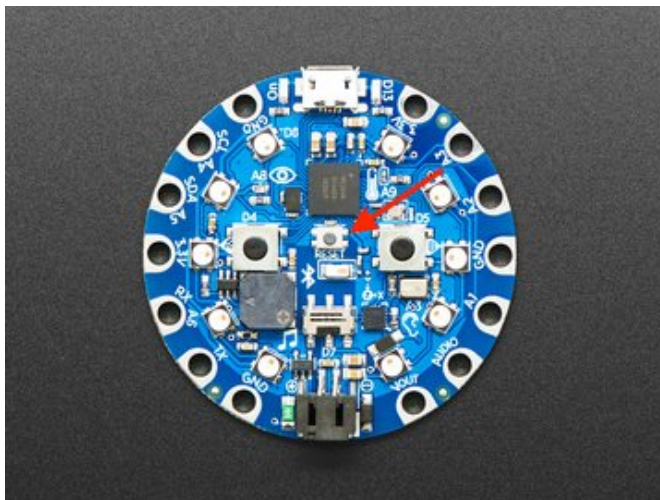
<https://adafru.it/FNK>

<https://adafru.it/FNK>



Click the link above and download the latest UF2 file

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



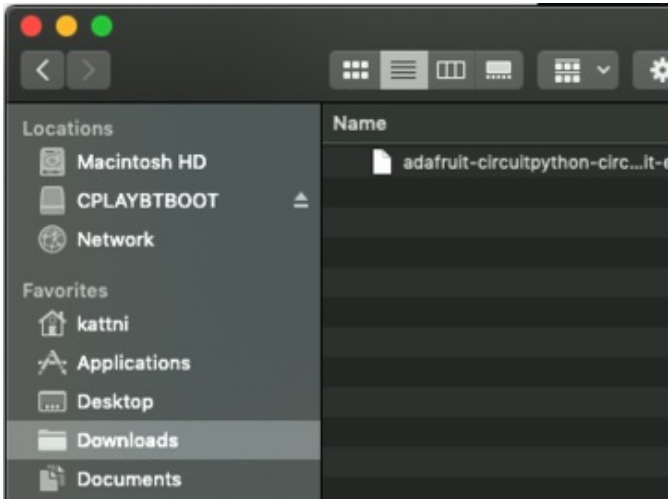
Plug your Circuit Playground Bluefruit into your computer using a known-good data-capable USB cable.

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!

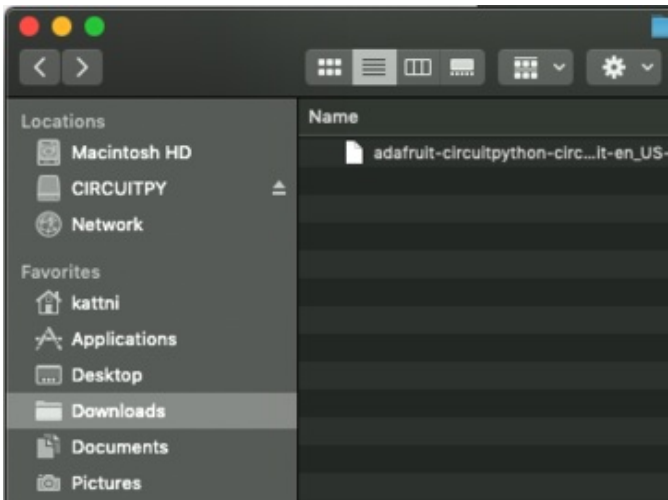
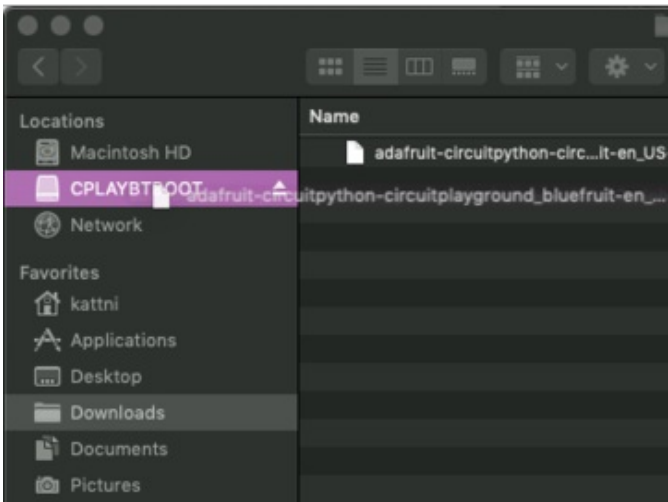
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

(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! :)

Circuit Playground Bluefruit CircuitPython Libraries

The Circuit Playground Bluefruit is packed full of features like Bluetooth and NeoPixel LEDs. Now that you have CircuitPython installed on your Circuit Playground Bluefruit, 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.

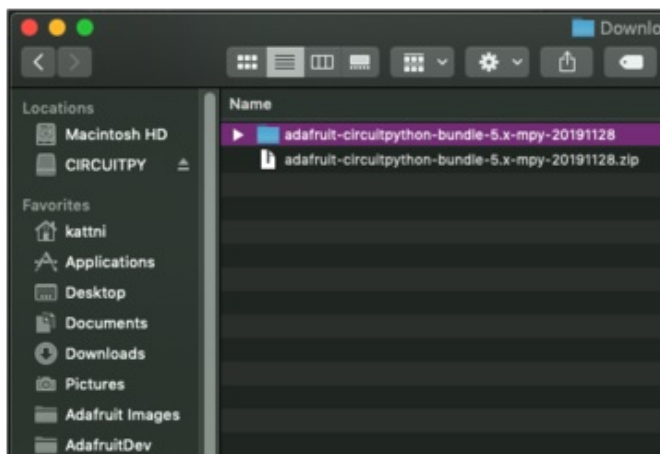
Installing CircuitPython Libraries on Circuit Playground Bluefruit

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

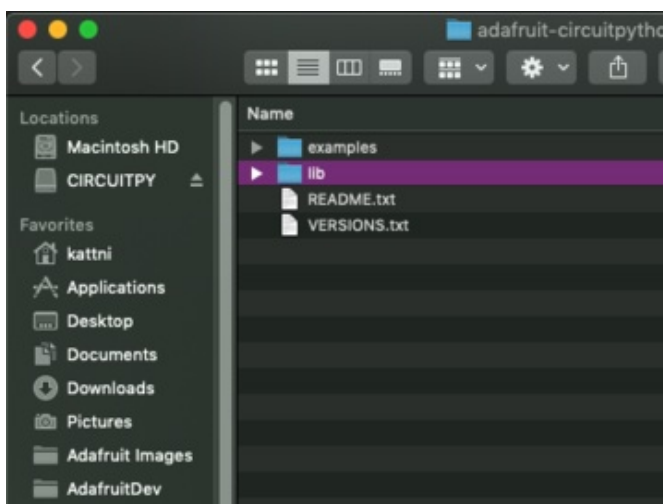
Then, download the CircuitPython library bundle that matches your version of CircuitPython from CircuitPython.org.

<https://adafru.it/ENC>

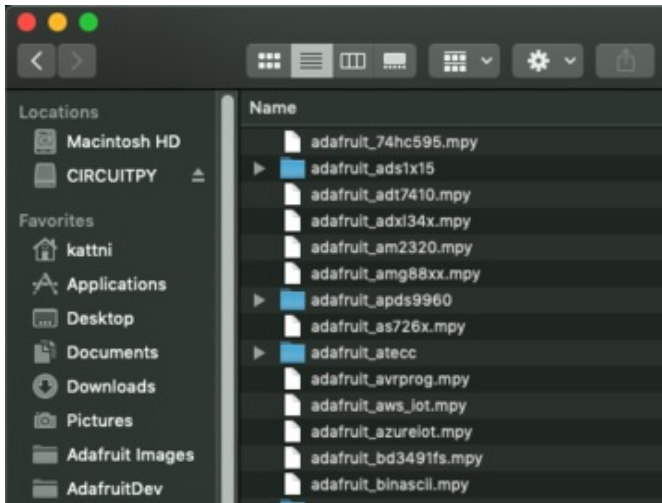
<https://adafru.it/ENC>



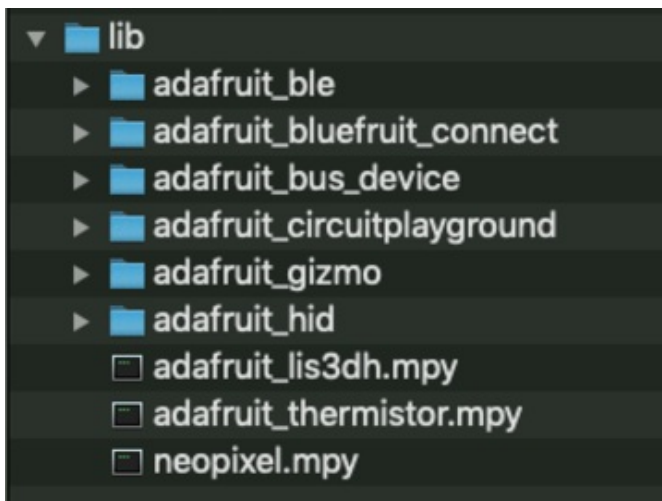
The bundle download as a .zip file. Extract the file. Open the resulting folder.



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.



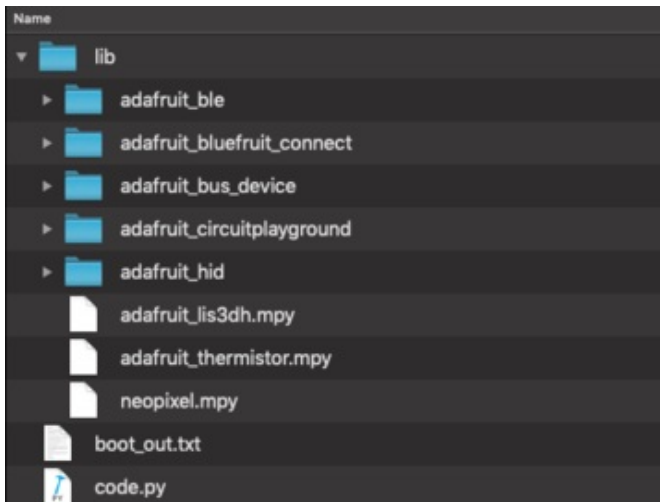
Copy the following folders and files from the **bundle lib** folder to the **lib** folder on your **CIRCUITPY** drive:

- adafruit_ble
- adafruit_bluefruit_connect
- adafruit_bus_device
- adafruit_circuitplayground
- adafruit_gizmo
- adafruit_hid
- adafruit_lis3dh.mpy
- adafruit_thermistor.mpy
- neopixel.mpy

Your lib folder should look like the image on the left.

Now you're all set to use CircuitPython with the features of the Circuit Playground Bluefruit!

Code with CircuitPython



The `adafruit_circuitplayground` library allows us to use essential commands and features of the board without a lot of setup in our code! Since we're using the `adafruit_circuitplayground` library in this project, be sure you have all of the libraries installed as specified on the previous page of the guide.

Your root directory on the **CIRCUITPY** drive should look like the one pictured here.

The Mu Editor

Adafruit recommends using the free program Mu to edit your CircuitPython programs and save them on your Circuit Playground Bluefruit. You can use any text editor, but Mu has some handy features.

See this page on the [Circuit Playground Bluefruit guide \(https://adafru.it/GDt\)](https://adafru.it/GDt) on the steps used to install Mu.

Ornament Code

Here is the code that we'll run on the Circuit Playground Bluefruit.

Copy this code and then paste it into a new document in Mu, then save it to your **CIRCUITPY** drive as `code.py`

You'll run the same code on all of the CPB boards, since you can just flip the switch to determine if a give board is broadcasting or detecting the signal.

```
"""
Circuit Playground Bluefruit Ornament Proximity
This demo uses advertising to set the color of scanning devices depending on the strongest broadcast
signal received. Circuit Playgrounds can be switched between advertising and scanning using the
slide switch. The buttons change the color when advertising.
"""

import time
from adafruit_circuitplayground.bluefruit import cpb

from adafruit_ble import BLERadio
from adafruit_ble.advertising.adafruit import AdafruitColor

# The color pickers will cycle through this list with buttons A and B.
color_options = [0x110000,
                 0x111100,
                 0x001100,
                 0x001111,
                 0x000011,
```



```

        0x110011,
        0x111111,
        0x221111,
        0x112211,
        0x111122]

ble = BLERadio()

i = 0
advertisement = AdafruitColor()
advertisement.color = color_options[i]
cpb.pixels.auto_write = False
cpb.pixels.fill(color_options[i])
while True:
    # The first mode is the color selector which broadcasts it's current color to other devices.
    if cpb.switch:
        print("Broadcasting color")
        ble.start_advertising(advertisement)
        while cpb.switch:
            last_i = i
            if cpb.button_a:
                i += 1
            if cpb.button_b:
                i -= 1
            i %= len(color_options)
            if last_i != i:
                color = color_options[i]
                cpb.pixels.fill(color)
                cpb.pixels.show()
                print("New color {:06x}".format(color))
                advertisement.color = color
                ble.stop_advertising()
                ble.start_advertising(advertisement)
                time.sleep(0.5)
        ble.stop_advertising()
    # The second mode listens for color broadcasts and shows the color of the strongest signal.
    else:
        closest = None
        closest_rssi = -80
        closest_last_time = 0
        print("Scanning for colors")
        while not cpb.switch:
            for entry in ble.start_scan(AdafruitColor, minimum_rssi=-100, timeout=1):
                if cpb.switch:
                    break
                now = time.monotonic()
                new = False
                if entry.address == closest:
                    pass
                elif entry.rssi > closest_rssi or now - closest_last_time > 0.4:
                    closest = entry.address
                else:
                    continue
                closest_rssi = entry.rssi
                closest_last_time = now
                discrete_strength = min((100 + entry.rssi) // 5, 10)
                cpb.pixels.fill(0x000000)
                for i in range(0, discrete_strength):
                    cpb.pixels[i] = entry.color
                cpb.pixels.show()

```

```
        # Clear the pixels if we haven't heard from anything recently.
        now = time.monotonic()
        if now - closest_last_time > 1:
            cpb.pixels.fill(0x000000)
            cpb.pixels.show()
    ble.stop_scan()
```

Here's how the code works.

Library Import

First, we import the libraries we'll be using:

```
import time
from adafruit_circuitplayground.bluefruit import cpb

from adafruit_ble import BLERadio
from adafruit_ble.advertising.adafruit import AdafruitColor
```

We'll be able to call on Circuit Playground Bluefruit board functions with the `cpb` command, including simplified ways to access the buttons and switch, as well as the on board red LED and the NeoPixels.

We're also setting up the Bluetooth LE radio and the package needed to advertise color values.

Color List

Next, we create a list of color values (in hex):

```
# The color pickers will cycle through this list with buttons A and B.
color_options = [0x110000,
                 0x111100,
                 0x001100,
                 0x001111,
                 0x000011,
                 0x110011,
                 0x111111,
                 0x221111,
                 0x112211,
                 0x111122]
```

Bluetooth and NeoPixel Setup

The BLE radio is instantiated next, as well as the `AdafruitColor()` object to advertise the color value of the board.

Also, we'll use the `cpb.pixels.auto_write` command to set the pixel auto write to `False` this helps avoid flickering NeoPixels (thanks Roy!) and the NeoPixel fill color.

```
ble = BLERadio()

i = 0
advertisement = AdafruitColor()
advertisement.color = color_options[i]
cpb.pixels.auto_write = False
cpb.pixels.fill(color_options[i])
```

Main Loop

The main loop of the code happens in the `while True:` section.

Switch Left to Broadcast

Here, we check the position of the switch by asking `if cpb.switch:` and if it is `True`, this means the switch is positioned to the left and the board will be in broadcast mode (the "hidden" ornaments are going to be the ones in broadcast mode).

Then, button presses are used to increment or decrement through the color list, and set the NeoPixels to that color.

Every half second, the board will advertise this color information over Bluetooth with the `ble.start_advertising(advertisement)` command.

```
if cpb.switch:
    print("Broadcasting color")
    ble.start_advertising(advertisement)
    while cpb.switch:
        last_i = i
        if cpb.button_a:
            i += 1
        if cpb.button_b:
            i -= 1
        i %= len(color_options)
        if last_i != i:
            color = color_options[i]
            cpb.pixels.fill(color)
            cpb.pixels.show()
            print("New color {:06x}".format(color))
            advertisement.color = color
            ble.stop_advertising()
            ble.start_advertising(advertisement)
            time.sleep(0.5)
    ble.stop_advertising()
```

Switch Right to Detect

In the case that the `cpb.switch` value is `False` (in the code, `not True`), this means it's flipped to the right and the board will be in listening/detect mode.

Now, we'll use the Bluetooth LE radio to scan the airwaves and use its received signal strength indicator (RSSI) capability to single out the board with the strongest signal.

The color entry being broadcast by the strongest (usually nearest) board is used in setting the NeoPixel color, while

the number of pixels being lit is based on the signal strength.

```
else:
    closest = None
    closest_rssi = -80
    closest_last_time = 0
    print("Scanning for colors")
    while not cpb.switch:
        for entry in ble.start_scan(AdafruitColor, minimum_rssi=-100, timeout=1):
            if cpb.switch:
                break
            now = time.monotonic()
            new = False
            if entry.address == closest:
                pass
            elif entry.rssi > closest_rssi or now - closest_last_time > 0.4:
                closest = entry.address
            else:
                continue
            closest_rssi = entry.rssi
            closest_last_time = now
            discrete_strength = min((100 + entry.rssi) // 5, 10)
            cpb.pixels.fill(0x000000)
            for i in range(0, discrete_strength):
                cpb.pixels[i] = entry.color
            cpb.pixels.show()

            # Clear the pixels if we haven't heard from anything recently.
            now = time.monotonic()
            if now - closest_last_time > 1:
                cpb.pixels.fill(0x000000)
                cpb.pixels.show()
        ble.stop_scan()
```

Next, we'll put together the ornaments and use them in a scavenger hunt!

Build and Use the Ornaments



Assembly

This is the simplest build ever!

First place the CPB boards into their enclosures.

Plug in the batteries into the JST battery ports.

Once they turn on, move the selector switch to the left on three of them (these are the ones you'll hide) and to the right on one (this is the detector).

On the ones that will broadcast their colors, press the **A** button to cycle through and pick a unique color for each of the ornaments. (You can press **B** to go backwards through the color list).



The detector will indicate the color and signal strength of the ornament with the strongest signal.

Fold the battery over gently and place each board/case/battery bundle into an ornament.

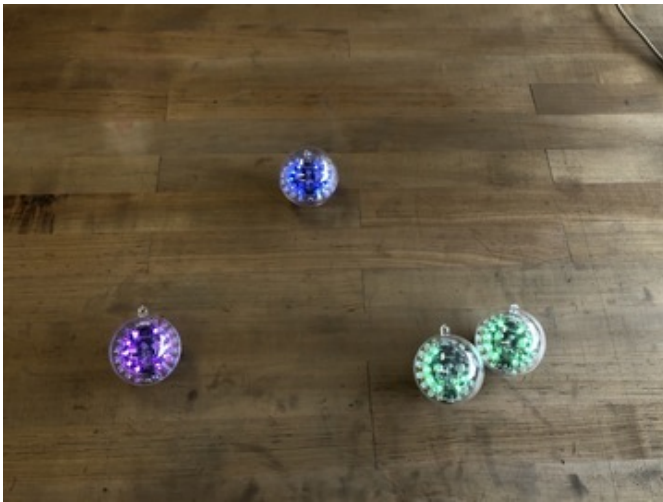




Usage

Here's how to use the ornaments. First, place the three ornaments to hide a foot or two away from each other, the farther the better!

Then, move the detector around the space. You'll see the color change as well as the number of NeoPixels that light up indicating the nearest ornament.



Now, you can hide the ornaments throughout your home, perhaps next to wrapped gifts. Then, send your scavenger hunter out on a mission to find them based on signal strength and color.

Hint, you can have multiple detector CPBs, just by flipping the switches to the right, in case you have more than one present hunter who wants to play. Maybe they are each assigned a color!



