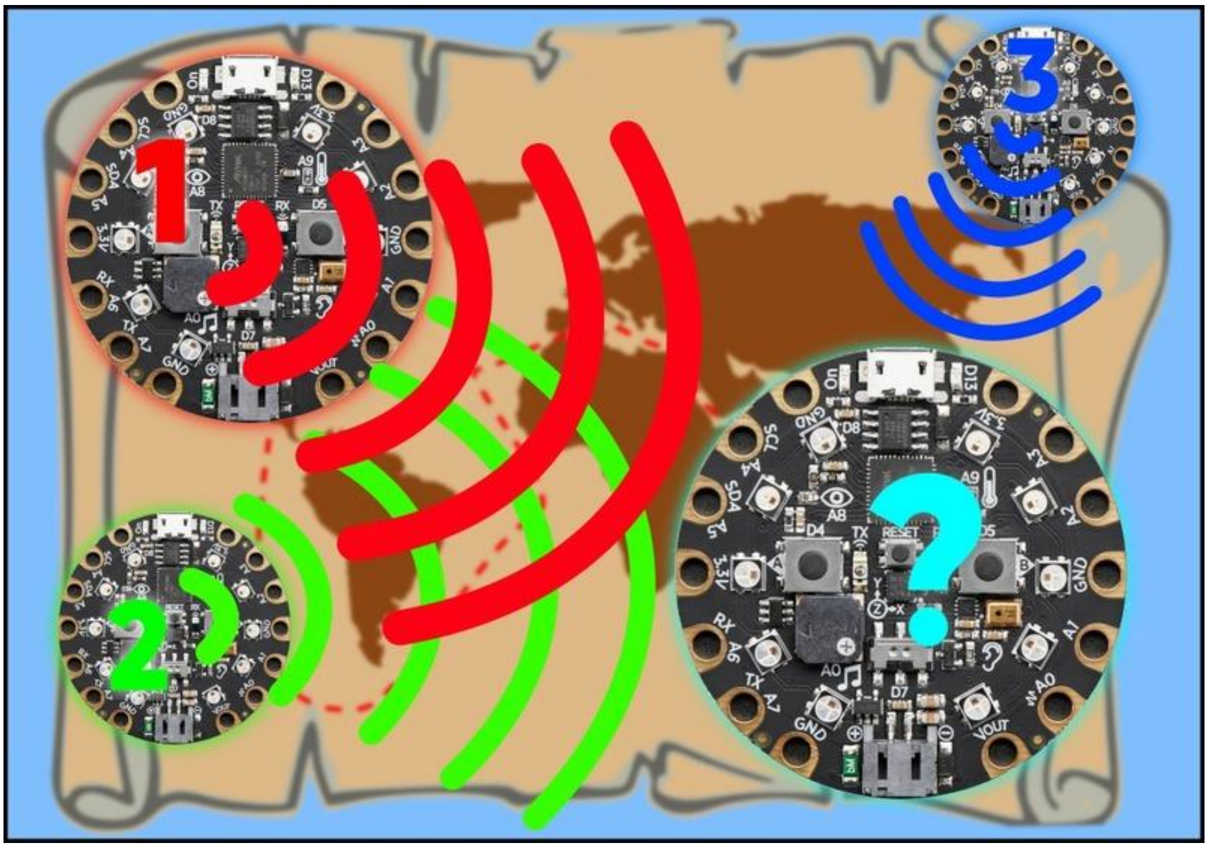
Circuit Playground Express Treasure Hunt

Created by Carter Nelson



https://learn.adafruit.com/circuit-playground-treasure-hunt

Last updated on 2021-11-15 07:17:32 PM EST

Table of Contents

Overview 3

Talking With Infrared 4

MakeCode Treasure Hunt 5

• The Treasure 5
• The Hunter 7
• on start 7
• forever 8
• on infrared received 9

CircuitPython Treasure Hunt 10

• The Treasure 11
• The Hunter 12

Playing The Game 14

• Prepare The Circuit Playground Express Boards 15

Overview

Oh no! Someone hid a bunch of treasures all over the place. How are you going to find them? Well, it just so happens those "treasures" are actually Circuit Playground Express boards that are sending out their location. So all we need to do is create a "hunter" Circuit Playground Express to sniff them out. Then the hunt is on!

In this guide we will go over how to create programs to act as the "treasure" and the "hunter". Then, using multiple Circuit Playground Express boards, we can create a fun game by loading the "treasure" program on several of these and hiding them. By loading the "hunter" program on another Circuit Playground Express, we can use that to search out and find all the treasures. The game is over when all treasures have been found.

Let the Treasure Hunt begin!

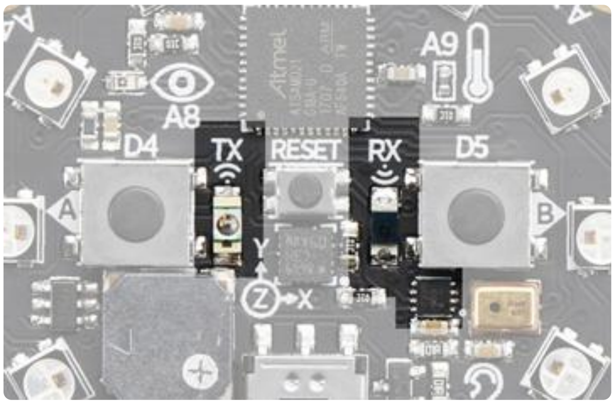
Talking With Infrared

There is already an excellent guide that goes over the basics of using the Circuit Playground Express to receive and transmit information using the built in infrared hardware. Check it out by following the link below.

Infrared Receive and Transmit

https://adafru.it/BX3

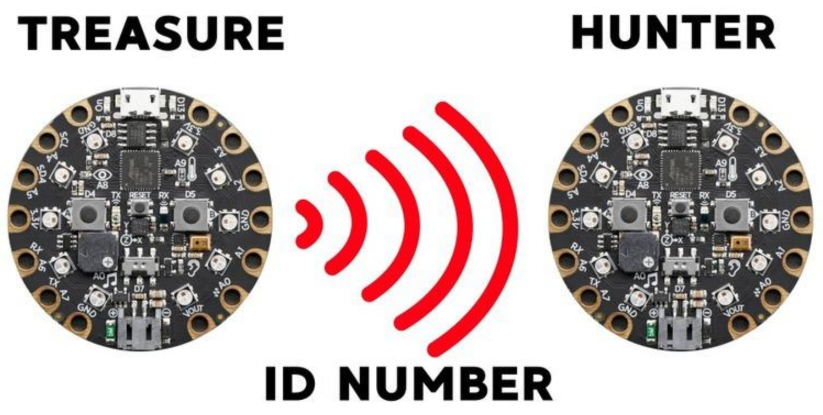
To summarize, the key Circuit Playground Express feature we will use is the infrared transmitter and receiver pair.



The IR transmitter and receiver on the Circuit Playground Express can be found near the center of the board.

The transmitter is labeled TX and is on the left side of the reset button, to the right of button A. The receiver is labeled RX and is on the right side of the reset button, to the left of button B.

For each of our "treasures", we will use the IR transmitter to send out a unique ID number. These numbers don't need to be fancy, so we will just use 1, 2, 3, etc. For the "hunter", we use the IR receiver to listen for any incoming signals. When one comes in, we check the number and see if it's one of the missing treasure ID's. So, something like this:



The Hunter can keep track of how many Treasures have been found and give an indication of progress. When all of the Treasures are found, the Hunter will do a little victory dance of some kind.

Let's see how to do this using MakeCode and CircuitPython.

MakeCode Treasure Hunt

If you haven't used MakeCode before, then check out the following guide:

MakeCode for Circuit Playground Express

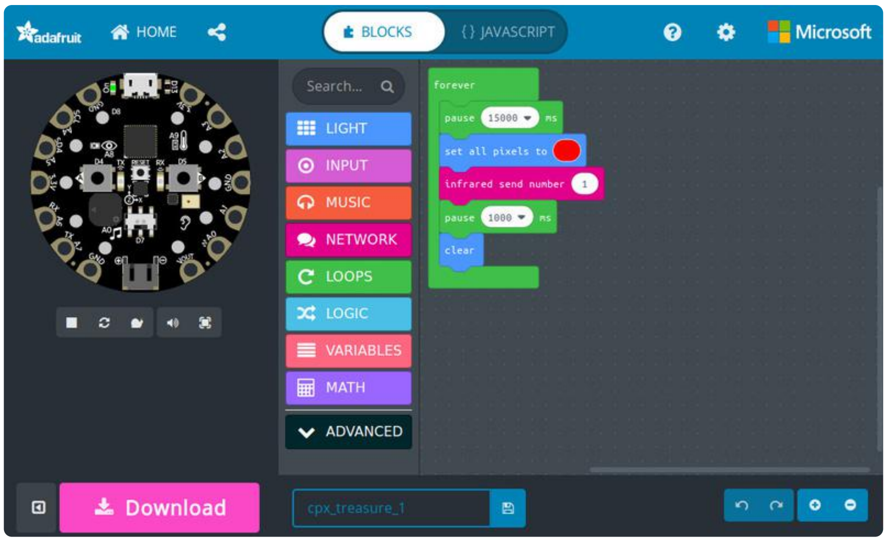
https://adafru.it/AgQ

The Treasure

First, let's make the "treasure" code. This one just sends out a number, so it's pretty easy. Here's a link to the code:

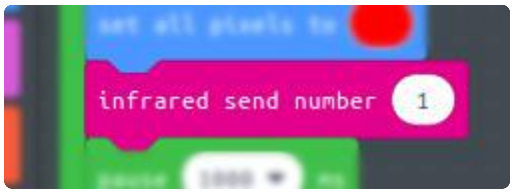
Treasure 1

https://adafru.it/BX4



```
forever
    pause 15000 ms
    set all pixels to
    infrared send number 1
    pause 1000 ms
    clear
```

The key block that matters is the `infrared send number` block found under **`NETWORK`**.



```
set all pixels to
infrared send number 1
pause 1000 ms
```

This block sends out the provided number on the infrared transmitter. Here, we've just set it to 1.

The rest of the code just waits 15 seconds so the number is not transmitted constantly (this makes the game more challenging). We also turn on the NeoPixels when we transmit the number. This is because our human eyes can not see the infrared light. So the NeoPixels give us a visible indication that the number is being sent. Otherwise it would look like the Treasure was just sitting there doing nothing.

The other Treasures are the same. The only thing that changes is the number being sent out. We're going to make 3 total Treasures. You can use the code above for Treasure 1. Here are links to code ready to go for Treasures 2 and 3:

Treasure 2

https://adafru.it/BX5

Treasure 3

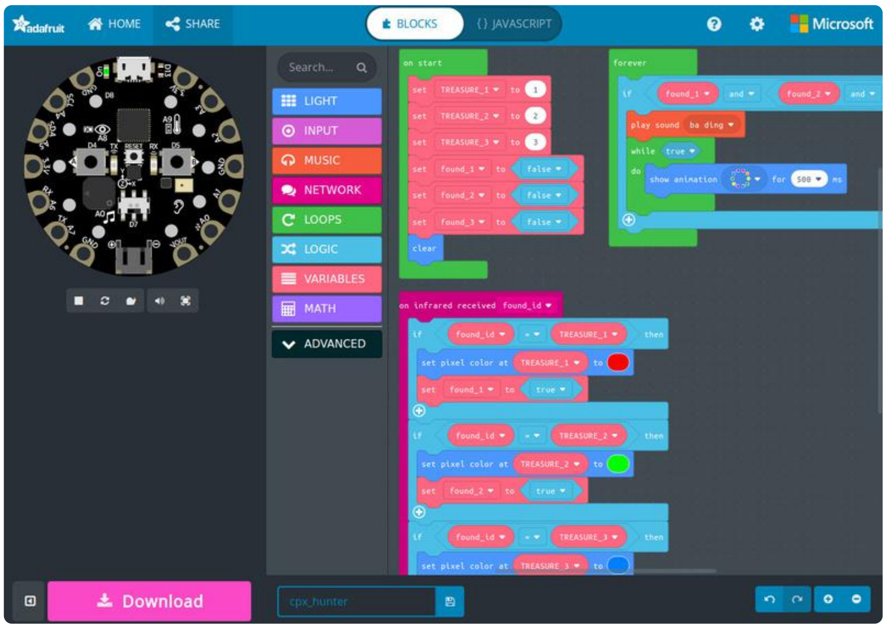
https://adafru.it/BX6

The Hunter

OK, this one is a little more complex - it has a lot to do. We'll go through it, but here's the link to the final code:

Hunter

https://adafru.it/BX7



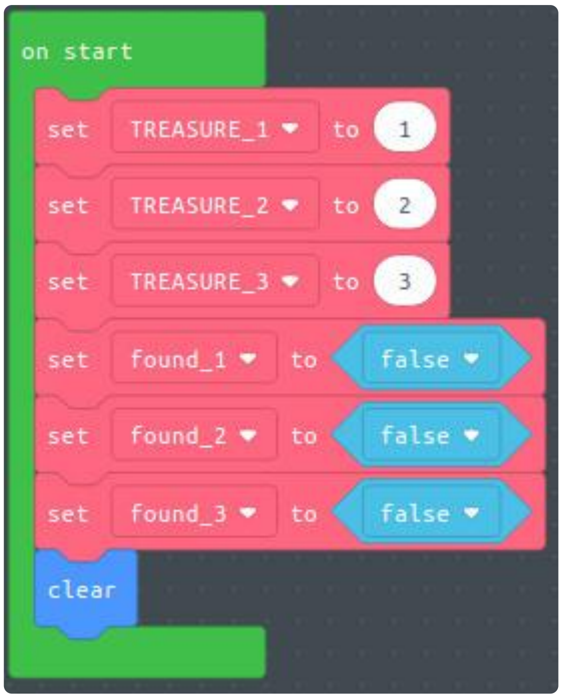
```
on start
    set TREASURE_1 to 1
    set TREASURE_2 to 2
    set TREASURE_3 to 3
    set found_1 to false
    set found_2 to false
    set found_3 to false
    clear

forever
    if found_1 and found_2 and
        play sound ba ding
        while true
        do
            show animation for 500 ms

on infrared received found_id
    if found_id = TREASURE_1 then
        set pixel color at TREASURE_1 to
        set found_1 to true
    if found_id = TREASURE_2 then
        set pixel color at TREASURE_2 to
        set found_2 to true
    if found_id = TREASURE_3 then
        set pixel color at TREASURE_3 to
```

on start

Let's go through the three main chunks. First, there's the `on start` block.

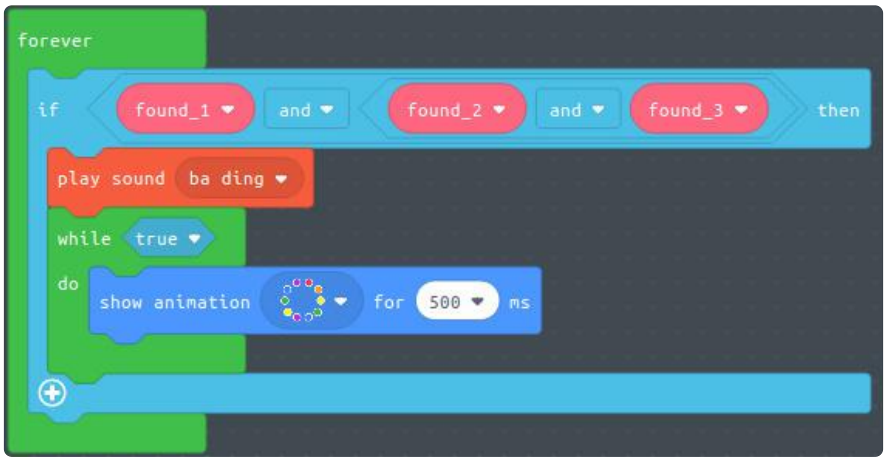


```
on start
  set TREASURE_1 to 1
  set TREASURE_2 to 2
  set TREASURE_3 to 3
  set found_1 to false
  set found_2 to false
  set found_3 to false
  clear
```

Even though there are a lot of blocks, it's basically just doing the same thing three times - one for each treasure. First, it sets variables (`TREASURE_1` , etc.) that hold the actual treasure ID numbers 1, 2, and 3. Then, to keep track of whether a treasure has been found, we set three more variables (`found_1` , etc.) to be initially false. These will become true when the treasures are found. You will see how that is done later. Finally, we just make sure all the NeoPixels are off by using the `clear` block.

forever

Next, there's the `forever` block. As the name implies, this runs forever - over and over again in a loop.



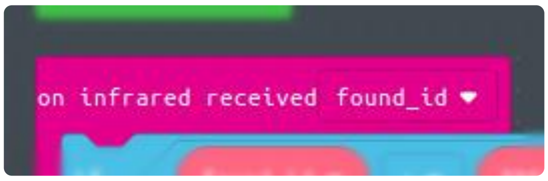
```
forever
  if found_1 and found_2 and found_3 then
    play sound ba ding
    while true
    do
      show animation for 500 ms
```

This is where we check to see if all of the treasures have been found. When that happens, all of the variables like `found_1` will be true. So the `if` statements just checks for that using `and` to tie them all together.

Once all of the variables are true, then the code inside the block runs. This plays a little "ba ding" sound and then loops a rainbow animation on the NeoPixels. The animation will run forever, so you'll need to press reset to start a new game.

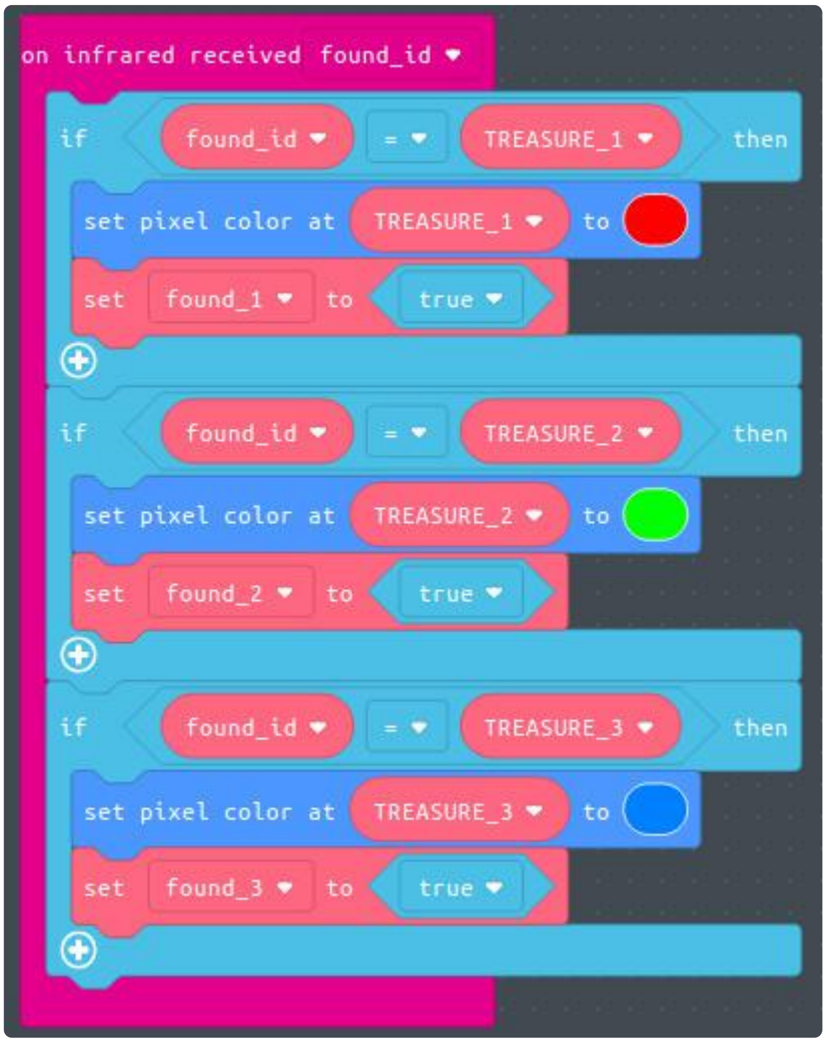
on infrared received

Now for the most important part - the part that listens for incoming signals on the IR receiver. This is handled using the `on infrared received` block which is also found under **`NETWORK`**.



```
on infrared received found_id ▾
```

Four our Hunter code, the entire block looks like this:



```
on infrared received found_id
  if found_id = TREASURE_1 then
    set pixel color at TREASURE_1 to 🔴
    set found_1 to true
  if found_id = TREASURE_2 then
    set pixel color at TREASURE_2 to 🟢
    set found_2 to true
  if found_id = TREASURE_3 then
    set pixel color at TREASURE_3 to 🔵
    set found_3 to true
```

Every time the IR receiver detects a number, this code block will run. The number that the IR receiver saw is stored in `found_id`. We just check that value to see if it's one of the treasure ID's, which are stored in the `TREASURE_1` variables we created in `on start`.

Each time we get a match, we turn on one of the NeoPixels. This gives you a visual indication of your progress in finding the treasures. More importantly, we set the `found` variable to true.

CircuitPython Treasure Hunt

If you are new to CircuitPython, be sure to check out the Welcome guide for an overview. And if you want to know even more, check out the Essentials guide.

Welcome to CircuitPython!

https://adafru.it/AIP

CircuitPython Essentials

https://adafru.it/BX8

Let's see how we can create our Hunter and Treasures using CircuitPython. This is a little more complex than using MakeCode, so you if you would rather just play the game, skip this and move on to the Playing The Game section.

The Treasure

Once again we'll start with the Treasure code, as it is the most simple. Here it is:

```
import time
import board
import pulseio
import adafruit_irremote
import neopixel

# Configure treasure information
TREASURE_ID = 1
TRANSMIT_DELAY = 15

# Create NeoPixel object to indicate status
pixels = neopixel.NeoPixel(board.NEOPIXEL, 10)

# Create a 'pulseio' output, to send infrared signals on the IR transmitter @ 38KHz
pwm = pulseio.PWMOut(board.IR_TX, frequency=38000, duty_cycle=2 ** 15)
pulseout = pulseio.PulseOut(pwm)

# Create an encoder that will take numbers and turn them into IR pulses
encoder = adafruit_irremote.GenericTransmit(header=[9500, 4500], one=[550, 550],
zero=[550, 1700], trail=0)

while True:
    pixels.fill(0xFF0000)
    encoder.transmit(pulseout, [TREASURE_ID]*4)
    time.sleep(0.25)
    pixels.fill(0)
    time.sleep(TRANSMIT_DELAY)
```

After importing all the goodies we need, we set the ID for the Treasure as well as the time to wait between transmissions.

```
# Configure treasure information
TREASURE_ID = 1
TRANSMIT_DELAY = 15
```

You'll want to change `TREASURE_ID` to a unique value for each Treasure. Just use simple numbers like 1, 2, 3, etc. The value of `TRANSMIT_DELAY` determines how often (in seconds) to send out the ID.

Then we create various items needed for using NeoPixels, the IR receiver, as well as an encoder for creating the transmitted signal.

After all that, we just loop forever sending out the ID:

```
while True:
    pixels.fill(0xFF0000)
    encoder.transmit(pulseout, [TREASURE_ID]*4)
    time.sleep(0.25)
    pixels.fill(0)
    time.sleep(TRANSMIT_DELAY)
```

Currently, the CircuitPython IR Remote library works with 4 byte NEC codes only. We create this in place with the syntax `[TREASURE_ID]*4`. If that looks too magical, you can try it out in the REPL to see what it does.

```
Adafruit CircuitPython 3.0.0 on 2018-07-09; Adafruit CircuitPlayground
Express with samd21g18
>>> [1]*4
[1, 1, 1, 1]
>>> ["hello"]*4
['hello', 'hello', 'hello', 'hello']
>>>
```

It's just a simple syntax for creating a list with all the same content. In this case, 4 of the same thing.

The Hunter

OK, now for the Hunter code. There's a bit more to it, but here it is in its entirety:

```
import time
import board
import pulseio
import adafruit_irremote
import neopixel

# Configure treasure information
#                  ID       PIXEL    COLOR
TREASURE_INFO = { (1,)*4 : (  0   , 0xFF0000) ,
                  (2,)*4 : (  1   , 0x00FF00) ,
                  (3,)*4 : (  2   , 0x0000FF) }
treasures_found = dict.fromkeys(TREASURE_INFO.keys(), False)

# Create NeoPixel object to indicate status
pixels = neopixel.NeoPixel(board.NEOPIXEL, 10)

# Sanity check setup
if len(TREASURE_INFO) &gt; pixels.n:
    raise ValueError("More treasures than pixels.")
```

```
# Create a 'pulseio' input, to listen to infrared signals on the IR receiver
pulsein = pulseio.PulseIn(board.IR_RX, maxlen=120, idle_state=True)

# Create a decoder that will take pulses and turn them into numbers
decoder = adafruit_irremote.GenericDecode()

while True:
    # Listen for incoming IR pulses
    pulses = decoder.read_pulses(pulsein)

    # Try and decode them
    try:
        # Attempt to convert received pulses into numbers
        received_code = tuple(decoder.decode_bits(pulses, debug=False))
    except adafruit_irremote.IRNECRepeatException:
        # We got an unusual short code, probably a 'repeat' signal
        # print("NEC repeat!")
        continue
    except adafruit_irremote.IRDecodeException as e:
        # Something got distorted or maybe its not an NEC-type remote?
        # print("Failed to decode: ", e.args)
        continue

    # See if received code matches any of the treasures
    if received_code in TREASURE_INFO.keys():
        treasures_found[received_code] = True
        p, c = TREASURE_INFO[received_code]
        pixels[p] = c

    # Check to see if all treasures have been found
    if False not in treasures_found.values():
        pixels.auto_write = False
        while True:
            # Round and round we go
            pixels.buf = pixels.buf[-3:] + pixels.buf[:-3]
            pixels.show()
            time.sleep(0.1)
```

After the necessary import, we configure things using a dictionary.

```
# Configure treasure information
#                  ID      PIXEL    COLOR
TREASURE_INFO = { (1,)*4: (  0  , 0xFF0000) ,
                  (2,)*4: (  1  , 0x00FF00) ,
                  (3,)*4: (  2  , 0x0000FF) }
```

The key is Treasure ID, in the form of the expected decoded IR signal - we use the same trick as above to create the necessary 4 byte value expected by the irremote library. The values of the dictionary are tuples which contain the location and color of the NeoPixel to use to indicate the Treasure has been found.

If you wanted to increase the number of Treasures in the game, just add an entry to the dictionary. Keep in mind that you'll have to modify a copy of the Treasure code to create a corresponding CPX Treasure to match the new entry.

A second dictionary is created to keep track of whether the Treasures have been found or not:

```
treasures_found = dict.fromkeys(TREASURE_INFO.keys(), False)
```

This is created using the same keys as the previous dictionary. The values are just booleans to indicate found state. Initially they are all False, since nothing has been found yet.

After some other setup, we just loop forever getting raw pulses:

```
    # Listen for incoming IR pulses
    pulses = decoder.read_pulses(pulsein)
```

Which we then try and decode:

```
        # Attempt to convert received pulses into numbers
        received_code = tuple(decoder.decode_bits(pulses, debug=False))
```

If anything happens, exceptions are thrown, which are currently just silently ignored.

Once we get a valid signal, we check the code against the ones we setup in the dictionary. If it matches an entry, then we update the dictionary of found Treasures and turn on the corresponding NeoPixel.

```
    # See if received code matches any of the treasures
    if received_code in TREASURE_INFO.keys():
        treasures_found[received_code] = True
        p, c = TREASURE_INFO[received_code]
        pixels[p] = c
```

Once all of the Treasures have been found, there will no longer be any False entries in the dictionary of found Treasures. So it's a simple check to see if they have all been found. If they have, then we just spin the NeoPixels round and round forever.

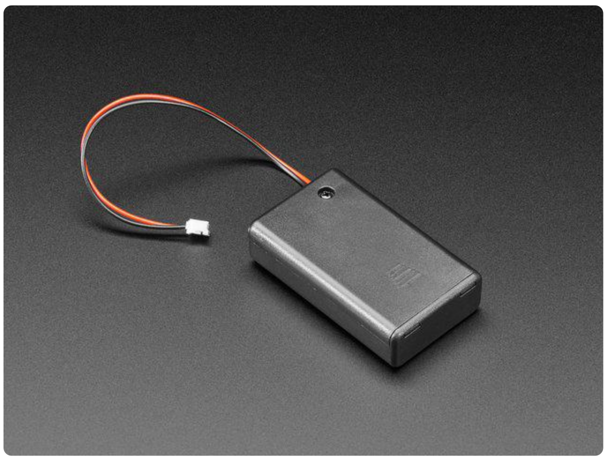
```
    # Check to see if all treasures have been found
    if False not in treasures_found.values():
        pixels.auto_write = False
        while True:
            # Round and round we go
            pixels.buf = pixels.buf[-3:] + pixels.buf[:-3]
            pixels.show()
            time.sleep(0.1)
```

Playing The Game

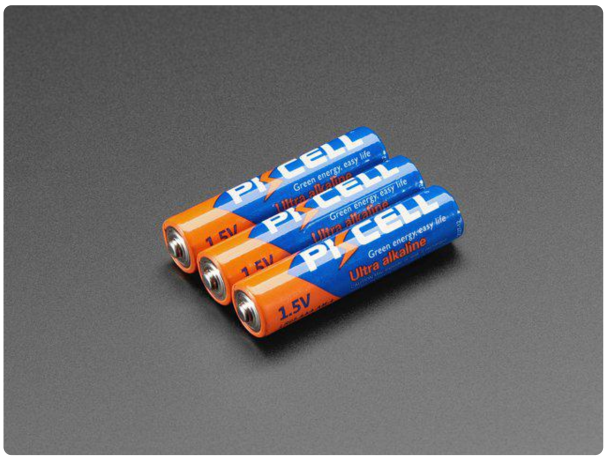
With either the MakeCode or the CircuitPython version of the program, the game play is basically the same. You'll need at least 4 Circuit Playground Express boards, 3 for the Treasures and 1 for the Hunter. So this is probably best done is a group or classroom setting.

Prepare The Circuit Playground Express Boards

This is pretty simple. Other than the software, all you need is a battery supply of some kind to power the boards. This 3xAAA holder works really well.

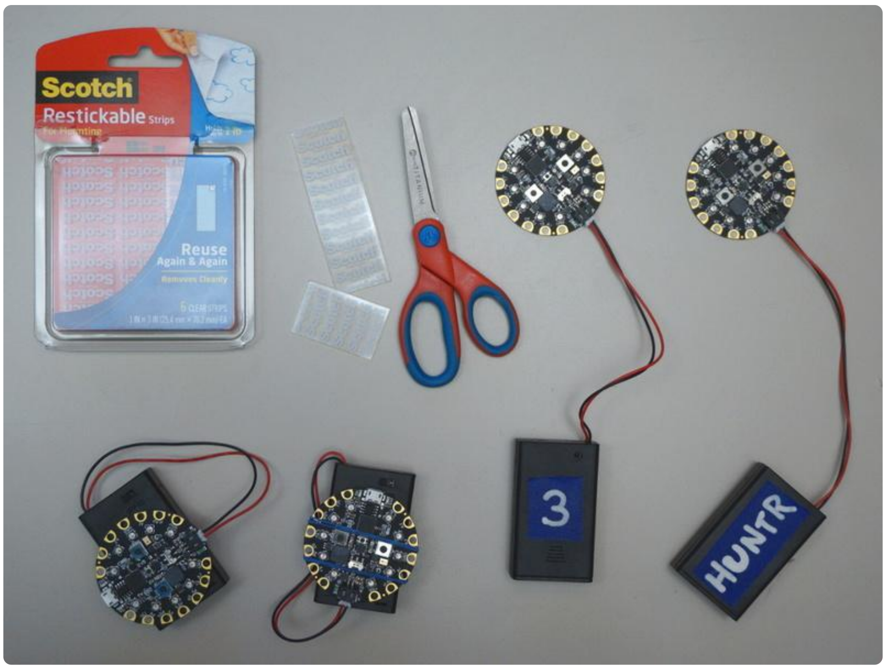


3 x AAA Battery Holder with On/Off Switch and 2-Pin JST
This battery holder connects 3 AAA batteries together in series for powering all kinds of projects. We spec'd these out because the box is slim, and 3 AAA's add up to about...
https://www.adafruit.com/product/727



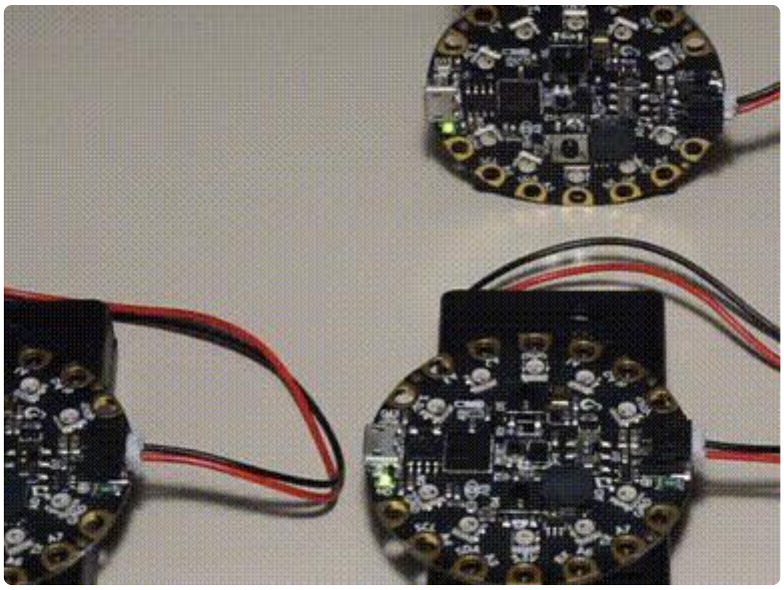
Alkaline AAA batteries - 3 pack
Battery power for your portable project! These batteries are good quality at a good price, and work fantastic with any of the kits or projects in the shop that use AAA's. This is a...
https://www.adafruit.com/product/3520

Then you can attach the Circuit Playground Express board to the battery pack. Rubber bands, double back tape, etc. Doesn't really matter. You could just leave them unattached, but this may put some rough stresses on the battery connector cable during game play.



It also helps to label the boards somehow. This way you'll know the Hunter from the Treasures, and also the ID's of the Treasures. I just used some blue tape and a white paint pen.

Then, upload the software from the previous pages to the boards. When you're done, you should have 3 Treasures and 1 Hunter. The video below has the MakeCode version loaded to all the boards. When the NeoPixels turn red on the Treasures, this is when the IR is sending out the ID number. You can see the Hunter keep track of these on its NeoPixels. After the 3rd Treasure sends out its ID, all Treasures are found and the Hunter displays a rainbow chase on the NeoPixels.



But this is too easy. Make it fun by coming up with challenging places to hide the Treasures. You can think of the IR transmitter as a bright light. It can broadcast pretty far, but can't broadcast out of closed box. But maybe that can be part of the game play - you have to open the box, or drawer, or whatever to find the Treasure.

Get creative with this and see what happens. Have fun!