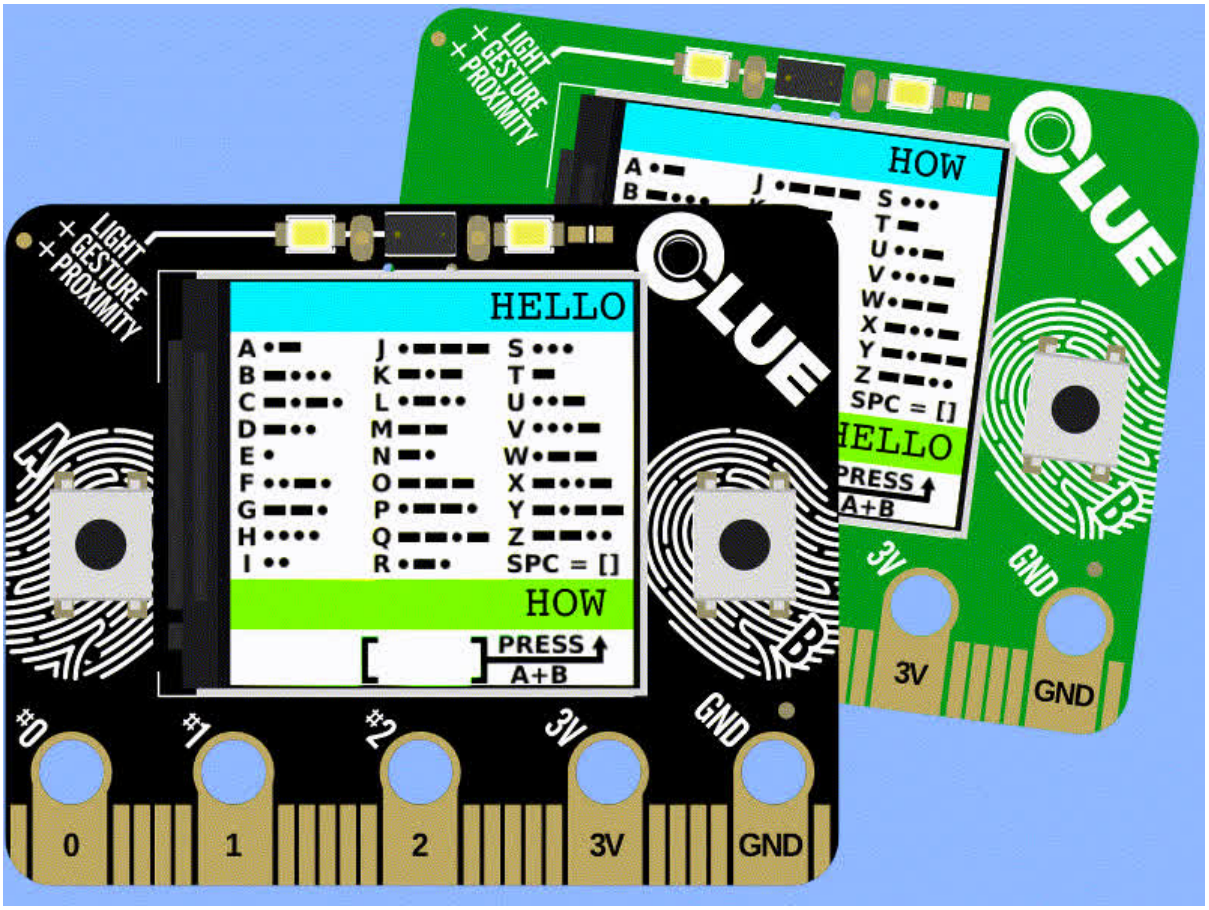
CircuitPython BLE Morse Code Chat

Created by Carter Nelson



https://learn.adafruit.com/circuitpython-ble-wireless-morse-code-chat

Last updated on 2024-06-03 03:04:08 PM EDT

Table of Contents

Overview

This is a modern take on tying two cups together to create a secret two-way intercom. But without the string! In this guide we'll show you how you can use two CLUE (http://adafru.it/4500) boards to talk to each other wirelessly over Bluetooth Low Energy (BLE).

Once the boards find each other, you can send messages back and forth one letter at a time. Morse Code is used as a way to enter the letters using only the two buttons. There's even a helpful Morse Code cheat sheet on the display!

The code is written in CircuitPython (https://adafru.it/cpy-welcome) using newly available BLE features. It's a great way to learn or practice your Morse code, and a great way to send secrets around the classroom or camp.

NOTE: You'll need at least two CLUE boards for this guide.

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

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!

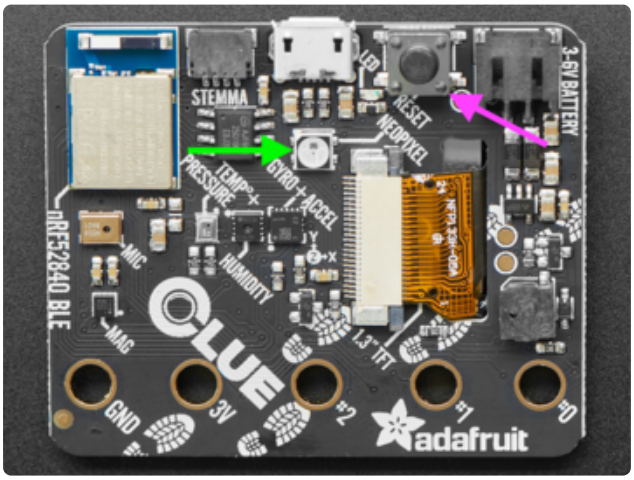
Follow this quick step-by-step for super-fast Python power :)

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).

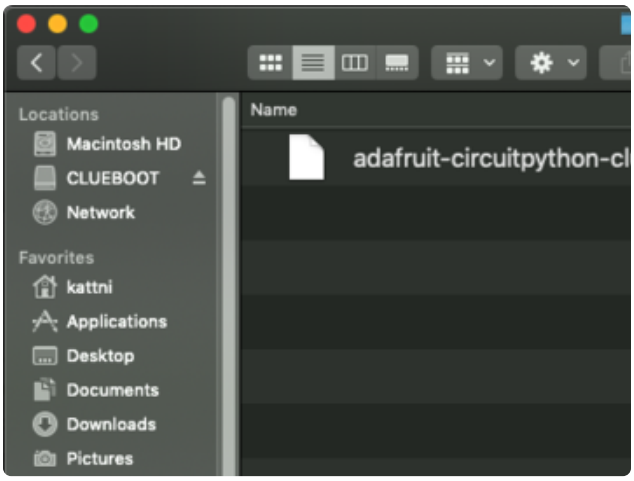


Plug your CLUE into your computer using a known-good USB cable.

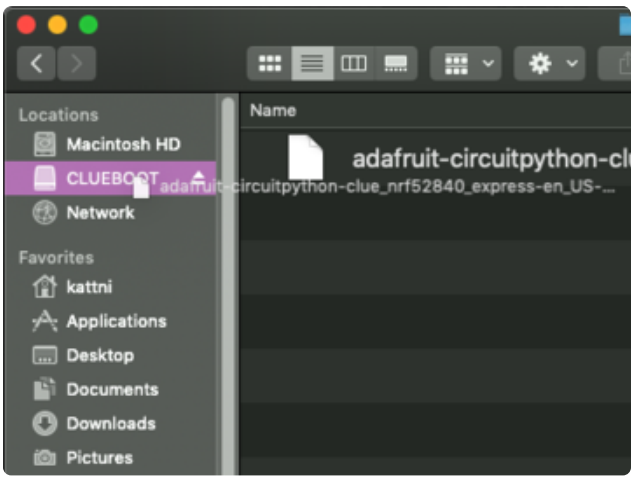
A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

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!

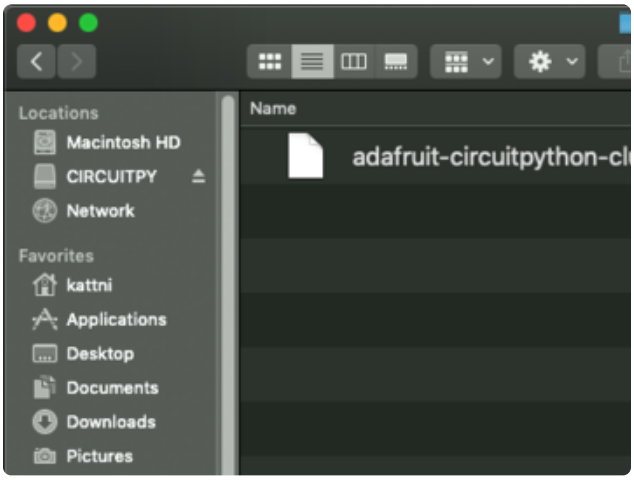
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



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! :)

Code

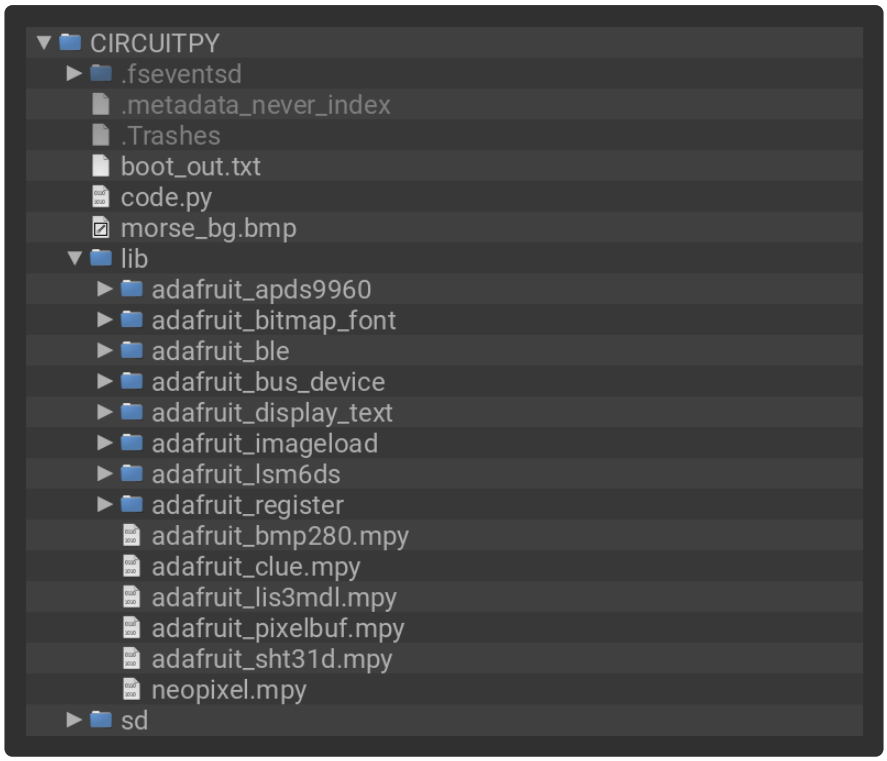
See the next page of this guide for info on some minor edits that are **required** to the code so that the CLUEs have unique names and can find each other.

Installing Project Code

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **CLUE_BLE_Morse_Code/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:




```
# SPDX-FileCopyrightText: 2020 Carter Nelson for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import displayio
import terminalio
from adafruit_clue import clue
from adafruit_display_text import label
import adafruit_imageload
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.nordic import UARTService

# --| User Config |------------------------------------------------
# Set to either A or B. The other CLUE should be set to opposite mode.
BLE_MODE = "A"
# --| User Config |------------------------------------------------

BLE_MODE = BLE_MODE.upper().strip()
if BLE_MODE not in ("A", "B"):
    raise ValueError("BLE_MODE must be set to either A or B.")

WAIT_FOR_DOUBLE = 0.05
DEBOUNCE = 0.25

# Define Morse Code dictionary
morse_code = {
    ".-": "A",
    "-...": "B",
    "-.-.": "C",
    "-..": "D",
    ".": "E",
    "..-.": "F",
    "--.": "G",
    "....": "H",
    "..": "I",
    ".---": "J",
    "-.-": "K",
    ".-..": "L",
```

```
    "--": "M",
    "-.": "N",
    "---": "O",
    ".--.": "P",
    "--.-": "Q",
    ".-.": "R",
    "...": "S",
    "-": "T",
    "..-": "U",
    "...-": "V",
    ".--": "W",
    "-..-": "X",
    "-.--": "Y",
    "--..": "Z",
}

# BLE Radio Stuff
if BLE_MODE == "A":
    MY_NAME = "CENTRAL"
    FRIENDS_NAME = "PERIPHERAL"
else:
    MY_NAME = "PERIPHERAL"
    FRIENDS_NAME = "CENTRAL"
ble = BLERadio()
uart_service = UARTService()
advertisement = ProvideServicesAdvertisement(uart_service)
ble._adapter.name = MY_NAME  # pylint: disable=protected-access

# Display Stuff
display = clue.display
disp_group = displayio.Group()
display.root_group = disp_group

# Background BMP with the Morse Code cheat sheet
bmp, pal = adafruit_imageload.load(
    "morse_bg.bmp", bitmap=displayio.Bitmap, palette=displayio.Palette
)
disp_group.append(displayio.TileGrid(bmp, pixel_shader=pal))

# Incoming messages show up here
in_label = label.Label(terminalio.FONT, text="A" * 18, scale=2, color=0x000000)
in_label.anchor_point = (1.0, 0)
in_label.anchored_position = (235, 4)
disp_group.append(in_label)

# Outging messages show up here
out_label = label.Label(terminalio.FONT, text="B" * 18, scale=2, color=0x000000)
out_label.anchor_point = (1.0, 0)
out_label.anchored_position = (235, 180)
disp_group.append(out_label)

# Morse Code entry happens here
edit_label = label.Label(terminalio.FONT, text="----", scale=2, color=0x000000)
edit_label.anchor_point = (0.5, 0)
edit_label.anchored_position = (115, 212)
disp_group.append(edit_label)


def scan_and_connect():
    """
    Handles initial connection between the two CLUES.

    The CLUE set to BLE_MODE="A" will act as Central.
    The CLUE set to BLE_MODE="B" will act as Peripheral.

    Return is a UART object that can be used for read/write.
    """

    print("Connecting...")
```

```
    in_label.text = out_label.text = "Connecting..."

    if MY_NAME == "CENTRAL":
        keep_scanning = True
        print("Scanning...")

        while keep_scanning:
            for adv in ble.start_scan():
                if adv.complete_name == FRIENDS_NAME:
                    ble.stop_scan()
                    ble.connect(adv)
                    keep_scanning = False

        print("Connected. Done scanning.")
        return uart_service

    else:
        print("Advertising...")
        ble.start_advertising(advertisement)

        while not ble.connected:
            if ble.connected:
                break

        print("Connected. Stop advertising.")
        ble.stop_advertising()

        print("Connecting to Central UART service.")
        for connection in ble.connections:
            if UARTService not in connection:
                continue
            return connection[UARTService]

    return None


# --------------------------
# The main application loop
# --------------------------
while True:
    # Establish initial connection
    uart = scan_and_connect()

    print("Connected.")

    code = ""
    in_label.text = out_label.text = " " * 18
    edit_label.text = " " * 4
    done = False

    # Run the chat while connected
    while ble.connected:
        # Check for incoming message
        incoming_bytes = uart.in_waiting
        if incoming_bytes:
            bytes_in = uart.read(incoming_bytes)
            print("Received: ", bytes_in)
            in_label.text = in_label.text[incoming_bytes:] + bytes_in.decode()

        # DOT (or done)
        if clue.button_a:
            start = time.monotonic()
            while time.monotonic() - start < WAIT_FOR_DOUBLE:
                if clue.button_b:
                    done = True
            if not done and len(code) < 4:
                print(".", end="")
                code += "."
                edit_label.text = "{:4s}".format(code)
```

```
                time.sleep(DEBOUNCE)

        # DASH (or done)
        if clue.button_b:
            start = time.monotonic()
            while time.monotonic() - start < WAIT_FOR_DOUBLE:
                if clue.button_a:
                    done = True
            if not done and len(code) < 4:
                print("-", end="")
                code += "-"
                edit_label.text = "{:4s}".format(code)
                time.sleep(DEBOUNCE)

        # Turn Morse Code into letter and send
        if done:
            letter = morse_code.get(code, " ")
            print(" >", letter)
            out_label.text = out_label.text[1:] + letter
            uart.write(str.encode(letter))
            code = ""
            edit_label.text = " " * 4
            done = False
            time.sleep(DEBOUNCE)

    print("Disconnected.")
```

How To Use

Customizing

You'll run the same code on both CLUE boards. **However, you first need to edit each one to setup the names you'll use.** You set the name you want your board to broadcast to the world via `MY_NAME`. You also need to set the name of the board for the friend you are looking for in `FRIENDS_NAME`.

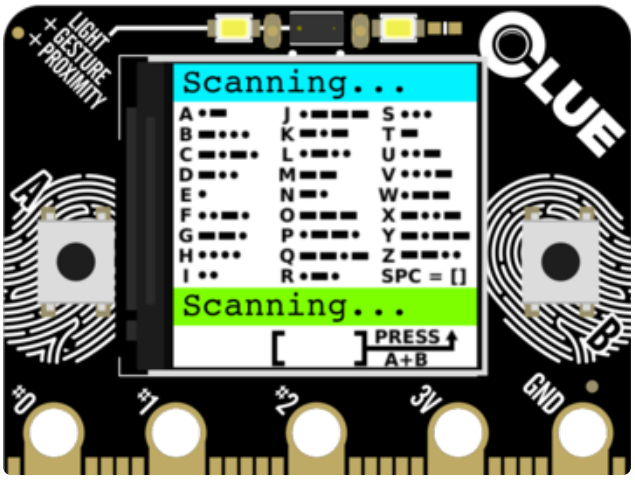
Look for these lines at the top of the code:

```
#--| User Config |------------------------------------------------
MY_NAME = "ME"
FRIENDS_NAME = "FRIEND"
#--| User Config |------------------------------------------------
```

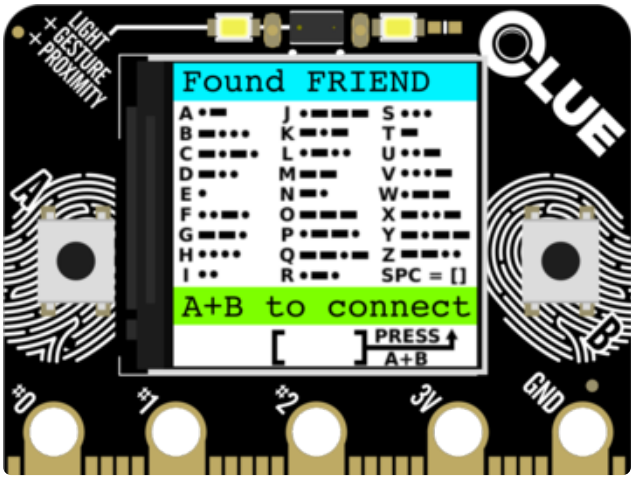
and change them as you want. **Just make sure the naming is symmetrical**. That is, your friend should set `FRIEND_NAME` to your `MY_NAME` and vice versa.

Scanning and Connecting

Once you have the code running on each CLUE, you connect them as follows.

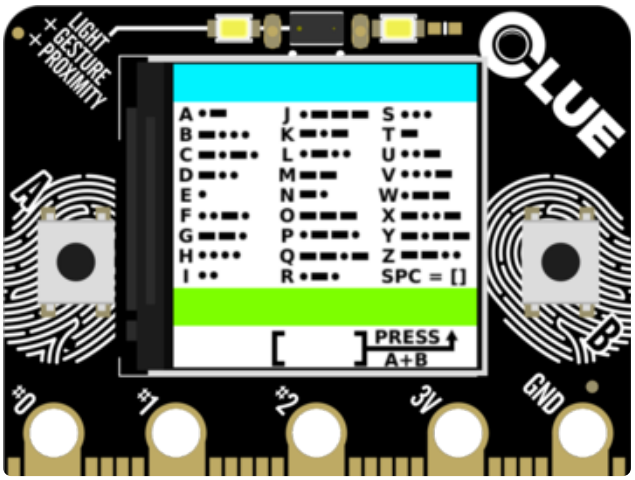


While one CLUE is scanning for the other CLUE, the message areas will display **Scanning...**



When the other CLUE has been found, the message areas will change to indicate the name of the other CLUE (ex: **FRIEND**).

Now, **on one of the CLUEs**, press the A and B buttons together to connect.



You are now connected and ready to send messages.

The **incoming** message will appear in the upper area - the **blue bar**.

The **outgoing** message will appear in the lower area - the **green bar**.

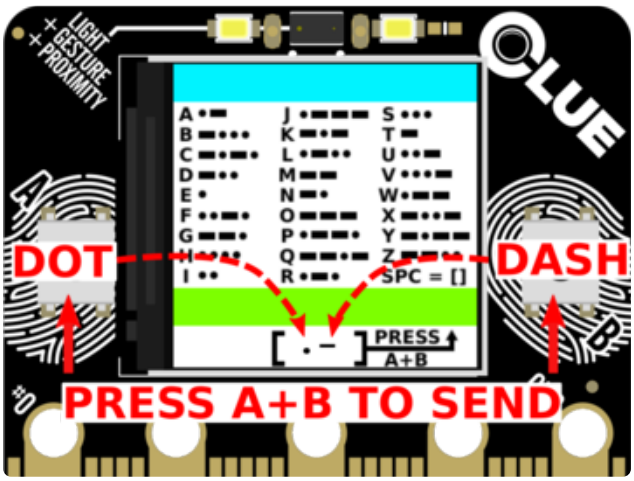
Chatting

You chat one letter at a time.

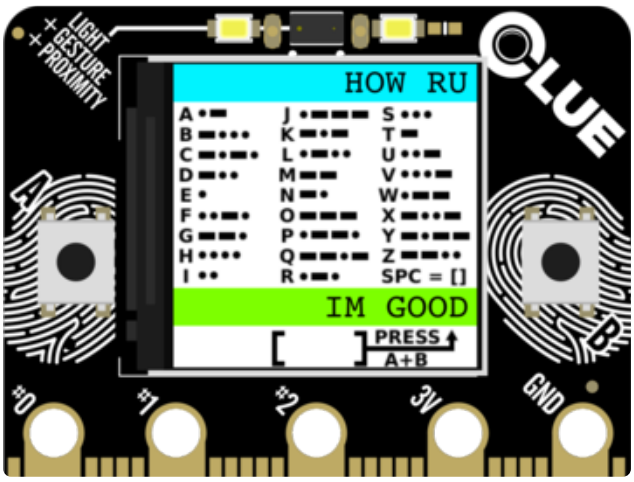
Each letter is formed by using Morse Code, using Button A for **DOT** and Button B for **DASH**. Don't know Morse Code? Don't worry, that's why the majority of the screen is filled with a cheat sheet.

Once all the dots and dashes for a letter have been entered, press **BOTH** buttons to send the letter.

NOTE: a space is generated by just sending a blank letter.



A = **DOT**
B = **DASH**
A + **B** = **SEND** letter
Each DOT/DASH will appear in the code entry area [ ] as you press the buttons.



In this example the other CLUE(**FRIEND**) has sent the message **HOW RU** which has been received and displayed in the upper area.

This CLUE (yours) has sent **IM GOOD** which is displayed in the lower area.

Type carefully - there is no backspace.

Disconnecting

If either CLUE disconnects for some reason, the other CLUE should detect this and revert back to **Scanning...** mode. Then, if the disconnected CLUE comes back, it will be seen and you can reconnect and start over.

To talk to a different CLUE, you will need to re-edit the code on each to set the names.

Pro Mode Tweaks

As you get better at typing in the Morse Code, you may find that the buttons are reacting a little too slow for your liking. This is due to the delay that is used for debouncing the button presses. You can try speeding things up by tweaking these delay values. Look for these lines of code at the top:

```
WAIT_FOR_DOUBLE = 0.05
DEBOUNCE = 0.25
```

You can try reducing them down a bit. Start with reducing the main `DEBOUNCE` setting value. The `WAIT_FOR_DOUBLE` value determines how quickly you must press A+B for it to register as a double button press vs. two single presses. Unless you're a ninja robot, you'll probably need some of this.

Running On Other Hardware

This code could be adapted to work on other hardware configurations. The main items used are:

- BLE
- A display
- Two buttons

The code and artifacts were all written assuming a 240x240 pixel display. The TFT Gizmo (http://adafru.it/4367) has that same size, so this code will run with only minor modifications on a Circuit Playground Bluefruit (http://adafru.it/4333) with a TFT Gizmo (http://adafru.it/4367). For other display sizes, you'll need to rework the text label locations. You'll also need a different sized BMP for the backgorund. You could simply scale the provided BMP, but here's the SVG used in case you want to hack further:

morse_cheat_sheet.svg

https://adafru.it/JsE