Android GBoard Morse Code Control with Circuit Playground Express

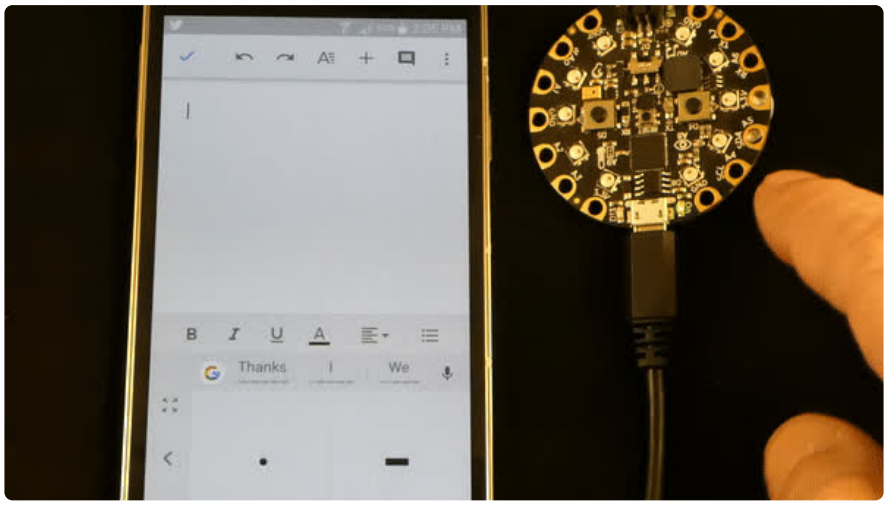
Created by Dave Astels

https://learn.adafruit.com/android-gboard-morse-code-at-with-circuitplayground-express

Last updated on 2024-06-03 02:25:50 PM EDT

Table of Contents

Overview



GBoard is an alternate keyboard for Android Devices (https://adafru.it/BVG) that lets you type using Morse code. This has seen use for people with limited mobility, but it can also be useful for practicing your morse code skills.

The goal of this project is to build a simple input device for GBoard that doesn't require soldering or elaborate construction techniques.

We'll be using a Circuit Playground Express to build this project. We have a few different versions we'll be demonstrating - from the simplest using the two onboard buttons, to using capacitive touch inputs, to connecting up some alligator clips to big-and-easy-to-press arcade buttons.

Using alligator clip pads lets us avoid avoid soldering altogether and makes this a quick, easy project to build by anyone!

If you're not familiar with the Circuit Playground Express and all it has to offer, see our introductory guide (https://adafru.it/adafruit-cpx). This guide will be using CircuitPython.

Parts

1 x Circuit Playground Express
https://www.adafruit.com/product/3333
CircuitPython ATSAMD21 based educational microcontrolelr board with an abundance of sensors and output devices.

15" cables with alligator clip on each end, color coded.

1 x Small alligator clip leads https://www.adafruit.com/product/1008

15" cables with alligator clip on each end, color coded.

---

2 x 30mm Arcade Button https://www.adafruit.com/product/476

1.5" deep, translucent, snap in arcade button. Available in various colors.

---

1 x Micro USB to Micro USB OTG Cable https://www.adafruit.com/product/3610

10" / 25mm, for connecting a board like CPX to a phone with USB OTG capability

---

Materials for the box

1 x Makedo Toolkit for Cardboard Construction https://www.adafruit.com/product/3285

The Makedo Toolkit is a starter kit of cardboard construction tools that's a great introduction to Makedo.

You'll also need some corrugated cardboard from a typical shipping box. You'll need a smooth, flat piece big enough for the box you want to make.

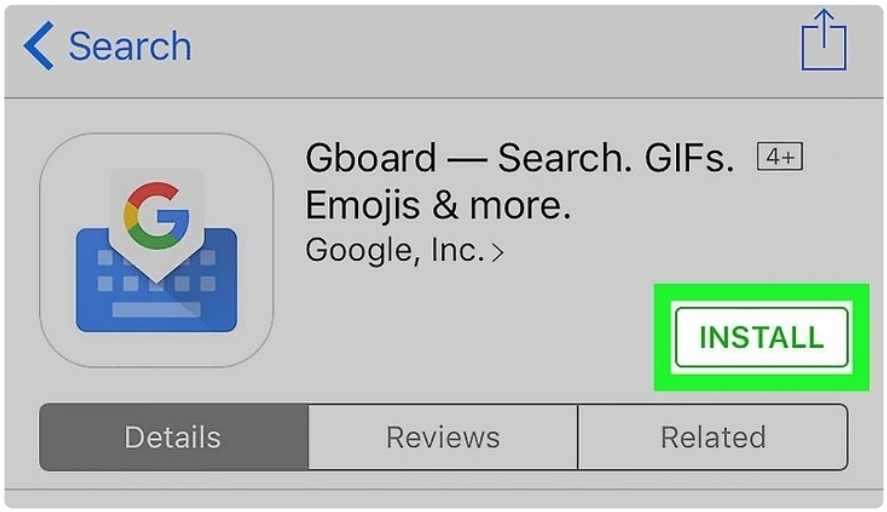
Installing GBoard

GBoard is an alternate keyboard for Android Devices. It has flexibility not found in the keyboard that came with your phone as most Android devices come with keyboard software by the phone manufacturer unless, perhaps, if you are on a Google brand phone.

You can install GBoard by going to the Google Play Store. The icon in your phone should be similar to the one below.

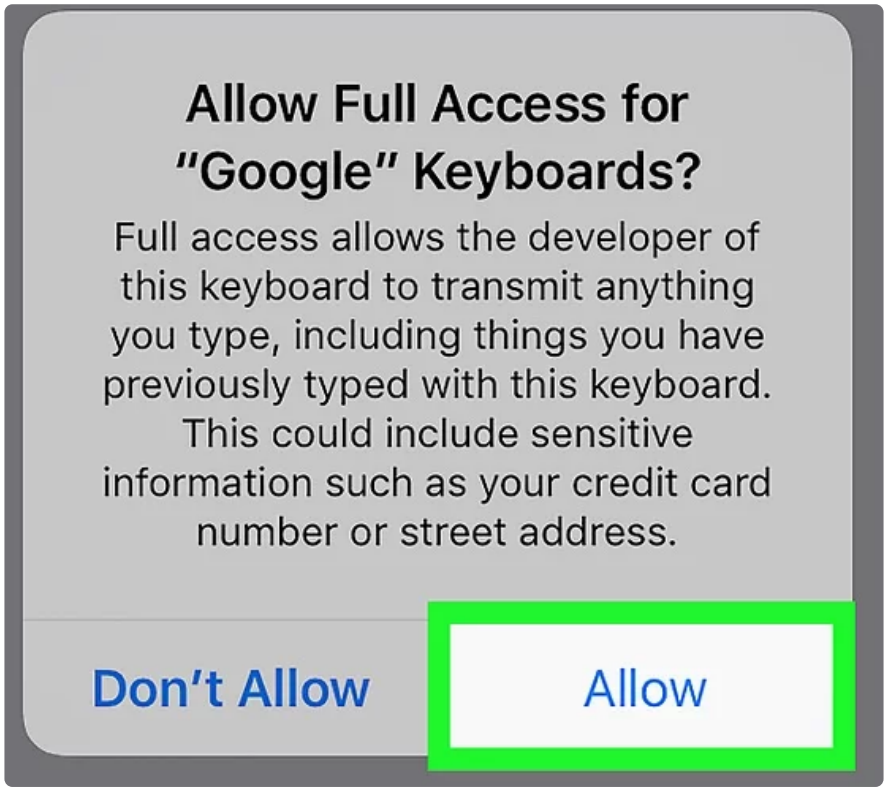
Search for GBoard. Make sure it is the app by Google and not some third party app. Select Get.

When the app is downloaded, click Install.



Launch/Run the app after it is installed. It will go through some configuration settings. Allow GBoard to be your keyboard and make it the default keyboard.

It will go to settings, you can set it up for your preferences.



Changing Keyboards to add GBoard and the Morse Option

Google has a standard guide on how to configure your Android device to accept both a regular keyboard in your language and a Morse keyboard.

Read the Google Configuration Guide for GBoard and Morse

https://adafru.it/BVG

When you have your regular keyboard up and you wish to switch to the Morse keyboard, press and hold the Globe icon on the GBoard keyboard. That is Google's method of allowing to switch back & forth.

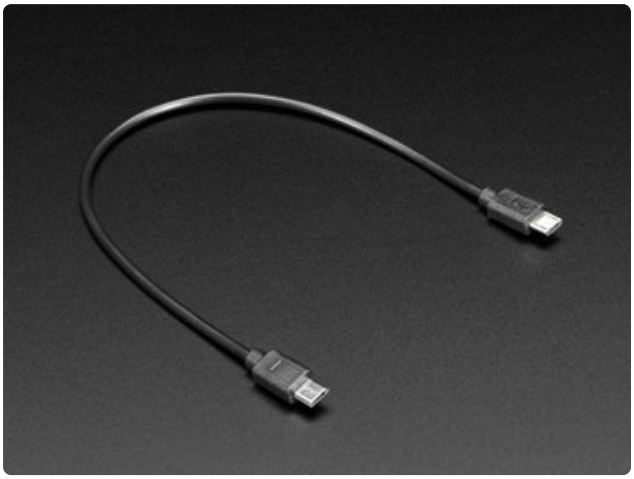
Connecting Your Phone to the Board

Android phones typically come with a micro-B USB connection. This USB connection follows the USB on-the-go (OTG) specification so peripherals can be plugged in to the port and be used by the phone.

This project is going to use the capabilities of Circuit Playground Express to act as a human interface device (HID), namely a keyboard. When the user wants the Express to send a key, they'll use a button, the board will translate this to some keyboard character, and the phone will believe a keyboard was used to enter the character.

This type of behavior is useful for a great many projects. Control of devices using alternative interfaces is the most popular. This can be in manufacturing, at home, or in assistive technology (AT) situations where traditional keyboards cannot be conveniently used.

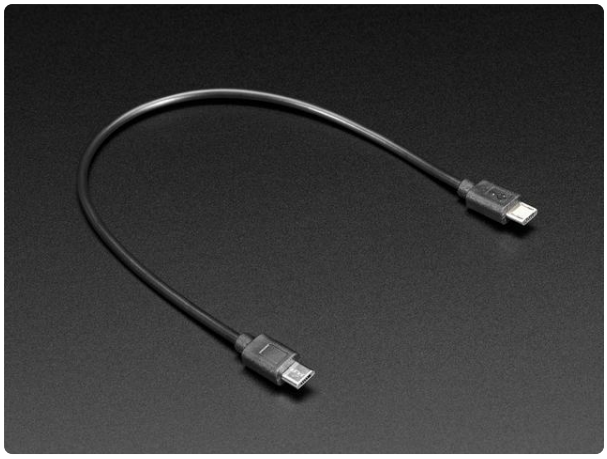
Making the Connection



For this project you will need a micro-B male to micro-B male connection that conforms to the OTG specification. Both the phone and the Circuit Playground Express have a micro-B female connector.

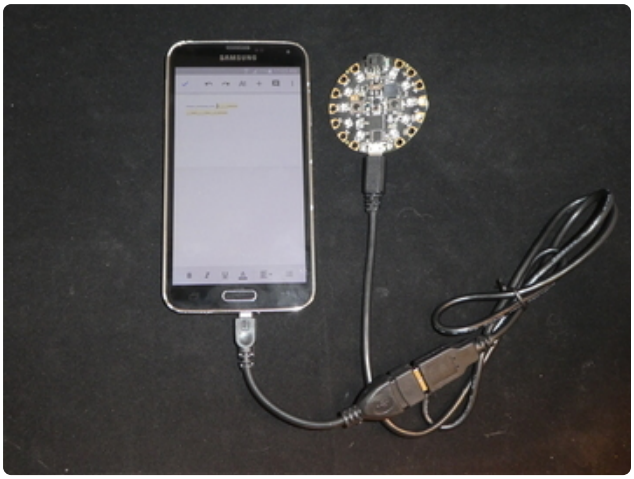
The cable at left is the most direct connection. Some electronics stores carry these - Adafruit sells them as product #3610 (http://adafru.it/3610).

There are also a number of other OTG to regular USB connector devices including Adafruit Tiny OTG Adapter (http://adafru.it/2910) and the short USB OTG Host Cable - MicroB OTG male to A female (http://adafru.it/1099).



Micro USB to Micro USB OTG Cable - 10-12" / 25-30cm long
This cable is a little unusual, rather than having a USB A plug on one end, it has two Micro B USB connections! What is this for? It's for when you have a "USB...
https://www.adafruit.com/product/3610

The Final Phone Setup



Here is a picture of the connections with a USB OTG adapter rather than the micro-micro cable as I did not have the cable at hand. The connections would be identical - connect the phone micro-B USB to the Circuit Playground Express micro-USB connector.

Buttons and Buzzers

We'll be using CircuitPython for this project. Are you new to using CircuitPython? No worries, there is a full getting started guide here (https://adafru.it/cpy-welcome).

Adafruit suggests using the Mu editor to edit your code and have an interactive REPL in CircuitPython. You can learn about Mu and its installation in this tutorial (https://adafru.it/ANO).

In this first iteration, all we need is the Circuit Playground Express. We'll use the A and B buttons on it's face to trigger events, and in response we'll play a short (dot) or long (dash) tone on the onboard speaker.

The code is pretty simple. It continuously checks the two buttons and if one has been pressed it plays a tone for the appropriate length of time. After either tone plays, there's a short delay so the tones don't blur together. After that, the code makes sure the button has been released, or waits until it has. This avoids accidentally generating multiple triggers.

Save the code below as `code.py` on your Circuit Playground Express.

```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Circuit Playground Express GBoard: onboard buttons generating tones

Adafruit invests time and resources providing this open source code.
Please support Adafruit and open source hardware by purchasing
products from Adafruit!

Written by Dave Astels for Adafruit Industries
Copyright (c) 2018 Adafruit Industries
Licensed under the MIT license.

All text above must be included in any redistribution.
"""

import time
from adafruit_circuitplayground.express import cpx

DOT_DURATION = 0.20
DASH_DURATION = 0.5

while True:
    if cpx.button_a:
        cpx.play_tone(4000, DOT_DURATION)
        time.sleep(0.1)
        while cpx.button_a:
            pass
    elif cpx.button_b:
        cpx.play_tone(4000, DASH_DURATION)
        time.sleep(0.1)
        while cpx.button_b:
            pass
```

Now, when you press button A a short tone will sound, and when you press button B a longer tone will sound. The `while cpx.button_X: pass` code causes it to wait until you release the button before playing the tone again.

Adding Touch

In this next step, we'll replace the buttons with capacitive touch. That way we don't have to click the button, just tap the pads with our fingertips

All that's needed is to replace the button checks with checks for the capacitive touch inputs. The code uses the top touch pad on each side as a replacement for the button on that side.

Touch sensitivity can be adjusted to suit your need. See the code for the use of adjust_touch_threshold

As before, save the code below as `code.py` on your Circuit Playground Express.

```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Circuit Playground Express GBoard: capacitive touch generating tones

Adafruit invests time and resources providing this open source code.
Please support Adafruit and open source hardware by purchasing
products from Adafruit!

Written by Dave Astels for Adafruit Industries
Copyright (c) 2018 Adafruit Industries
Licensed under the MIT license.

All text above must be included in any redistribution.
"""

import time
from adafruit_circuitplayground.express import cpx

DOT_DURATION = 0.20
DASH_DURATION = 0.5

# You can adjust this to get the level of sensitivity you want.
cpx.adjust_touch_threshold(100)

while True:
    if cpx.touch_A4:
        cpx.play_tone(4000, DOT_DURATION)
        time.sleep(0.1)
        while cpx.touch_A4:
            pass
    elif cpx.touch_A3:
        cpx.play_tone(4000, DASH_DURATION)
        time.sleep(0.1)
        while cpx.touch_A3:
            pass
```

Now, when you touch the **A4** pad a short tone will sound, and when you touch the **A3** pad a longer tone will sound. As before, the `while cpx.touch_X: pass` code causes it to wait until you release the button before playing the tone again. This prevents you triggering it multiple times accidentally.

Be careful how you hold the board so that you don't accidentally touch the **A3** and **A4** touch pads.

Sending Keys

This example has been updated for version 4+ of the CircuitPython HID library. On the CircuitPlayground Express this library is built into CircuitPython. So, please use the latest version of CircuitPython as well. (At least 5.0.0-beta.3)

Our next step is to pull in the HID library and send keys instead of beeping. The GBoard app accepts '.' (period) and '-' (minus sign), for dots and dashes, respectively, so that's what we'll send.

The changes are that we import the `Keyboard` and `Keycode` modules from the HID library, and initialize a keyboard object that is then used to send keys to the connected device.

For setting up the keyboard we need to add:

```
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode
import usb_hid
kbd = Keyboard(usb_hid.devices)
```

To actually send keys, we replace the dot and dash tone generation with `kbd.send(Keycode.PERIOD)` and `kbd.send(Keycode.MINUS)`

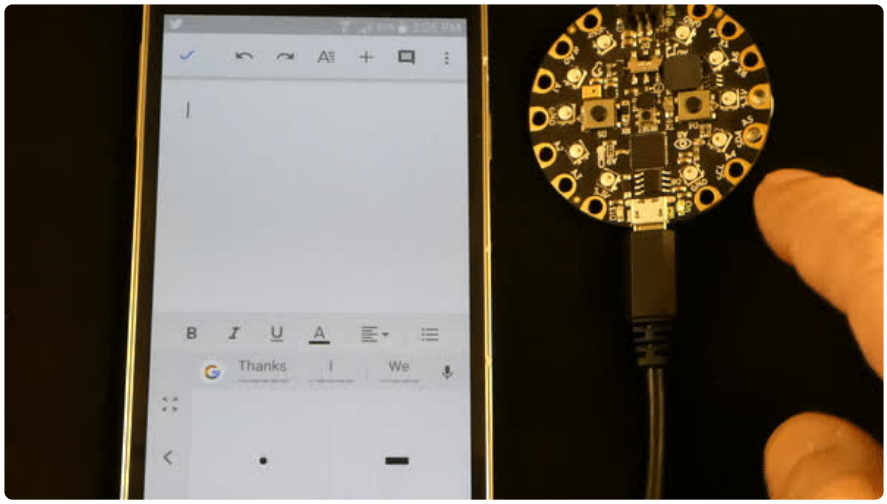
The code is below, copy it to `code.py` on your Circuit Playground Express.

```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Circuit Playground Express GBoard: capacitive touch generating keycodes

Adafruit invests time and resources providing this open source code.
Please support Adafruit and open source hardware by purchasing
products from Adafruit!
```

```
Written by Dave Astels for Adafruit Industries
Copyright (c) 2018 Adafruit Industries
Licensed under the MIT license.

All text above must be included in any redistribution.
"""

import usb_hid
from adafruit_circuitplayground.express import cpx
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode

DOT_DURATION = 0.25
DASH_DURATION = 0.5

kbd = Keyboard(usb_hid.devices)

# You can adjust this to get the level of sensitivity you want.
cpx.adjust_touch_threshold(100)


while True:
    if cpx.touch_A4:
        kbd.send(Keycode.PERIOD)
        while cpx.touch_A4:
            pass
    elif cpx.touch_A3:
        kbd.send(Keycode.MINUS)
        while cpx.touch_A3:
            pass
```

Open up an app that accepts text. It can be a word processor like Google Docs or just be a text input box like the SMS/phone message app or a Twitter message input box.

Now, when you touch the **A4** pad a **dot** will be sent, and when you touch the **A3** pad a **dash** will be. The GBoard keyboard replacement will convert those dots and dashes into conventional characters and send them to the app.



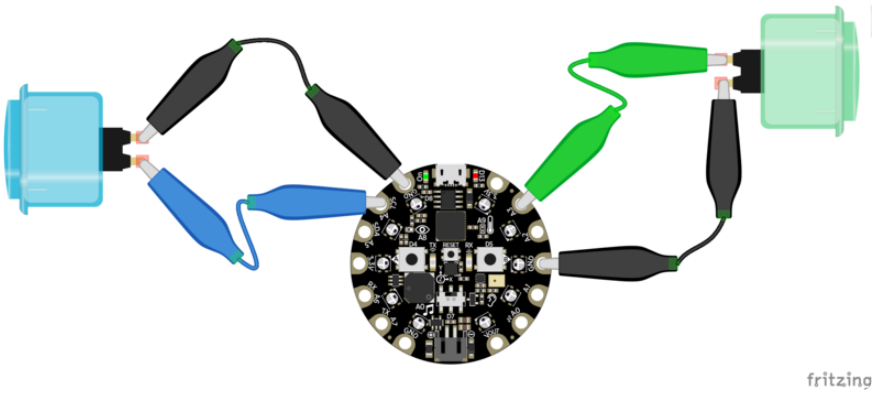
You can refer to the table below for International and American Morse Code (via Wikipedia (https://adafru.it/BW3)).

American (Railroad) vs. International Morse (similar codes highighted)

Letter	International Code	American Morse	Letter	International Code	American Morse	Digit	International Code	American Morse
A	·−	·–	N	−·	–·	0	–––––	——
B	−···	–···	O	–––	··	1	·––––	·–––
C	−·−·	···	P	·––·	·····	2	··–––	··––·
D	−··	···	Q	––·–	····	3	···––	···–·
E	·	·	R	·–·	···	4	····–	····–
F	··–·	·–·	S	···	···	5	·····	···
G	––·	––·	T	–	–	6	–····	·····
H	····	····	U	··–	··–	7	––···	····
I	··	··	V	···–	···–	8	–––··	····
J	·–––	·––·	W	·––	·––	9	––––·	····
K	−·–	·––	X	−··–	–···			
L	·–··	–·	Y	−·––	–·–·			
M	––	––	Z	––··	····			

Using External Buttons

One final iteration before we build something physical will replace touch input with large external buttons. This is where we step beyond just the Circuit Playground Express and add those alligator clip wires and arcade buttons that were listed in the parts list. The diagram below shows how to connect the buttons. Use the alligator clip wires to do this.



The code is below. Copy it to `code.py` on your Circuit Playground Express. A few things change due to using external buttons. The first is the import from the `digitalio` module:

```
from digitalio import DigitalInOut, Direction, Pull
```

the next is initialization of the input connections:

```
button_a = DigitalInOut(board.A4)
button_a.direction = Direction.INPUT
button_a.pull = Pull.UP
```

```
button_b = DigitalInOut(board.A3)
button_b.direction = Direction.INPUT
button_b.pull = Pull.UP
```

The three lines for each button:

1. create the interface object for a specific I/O pad on the Circuit Playground Express
2. set it to input (as opposed to using it for output), and
3. enable the internal pullup resistor. The pullup keeps the input HIGH when the button isn't pressed.

Not all microcontrollers have an internal pulldown resistor so the standard has become to use a pullup and have the button connect the input to ground when it's pushed. You can see this in the wiring diagram: one side of each button is connected to ground.

Notice that this change is limited to changing the two touch functions to use digital inputs in place of touch inputs; the main loop didn't change in this iteration.

```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Circuit Playground Express GBoard: arcade buttons generating keycodes

Adafruit invests time and resources providing this open source code.
Please support Adafruit and open source hardware by purchasing
products from Adafruit!

Written by Dave Astels for Adafruit Industries
Copyright (c) 2018 Adafruit Industries
Licensed under the MIT license.

All text above must be included in any redistribution.
"""

import board
import usb_hid
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode
from digitalio import DigitalInOut, Direction, Pull

DOT_DURATION = 0.25
DASH_DURATION = 0.5

button_a = DigitalInOut(board.A4)
button_a.direction = Direction.INPUT
button_a.pull = Pull.UP

button_b = DigitalInOut(board.A3)
button_b.direction = Direction.INPUT
button_b.pull = Pull.UP

kbd = Keyboard(usb_hid.devices)
```

```
def touch_a():
    return not button_a.value


def touch_b():
    return not button_b.value


while True:
    if touch_a():
        kbd.send(Keycode.PERIOD)
        while touch_a():
            pass
    elif touch_b():
        kbd.send(Keycode.MINUS)
        while touch_b():
            pass
```

As before, open up an app that accepts text. It can be a word processor like Google Docs or just be a text input box like the SMS/phone message app or a Twitter message input box.

Now, when you press the button connected to **A4** a dot will be sent to the app, and when you press the button connected to **A3** a dash will be.

Final Code

Here's the final version of the code. By uncommenting specific lines as indicated by the comments you can customize it for the various combinations of input and output methods we've explored.

The input and output functionality have been moved to separate functions so that the core code in the loop doesn't have to be changed when the input and output options change.

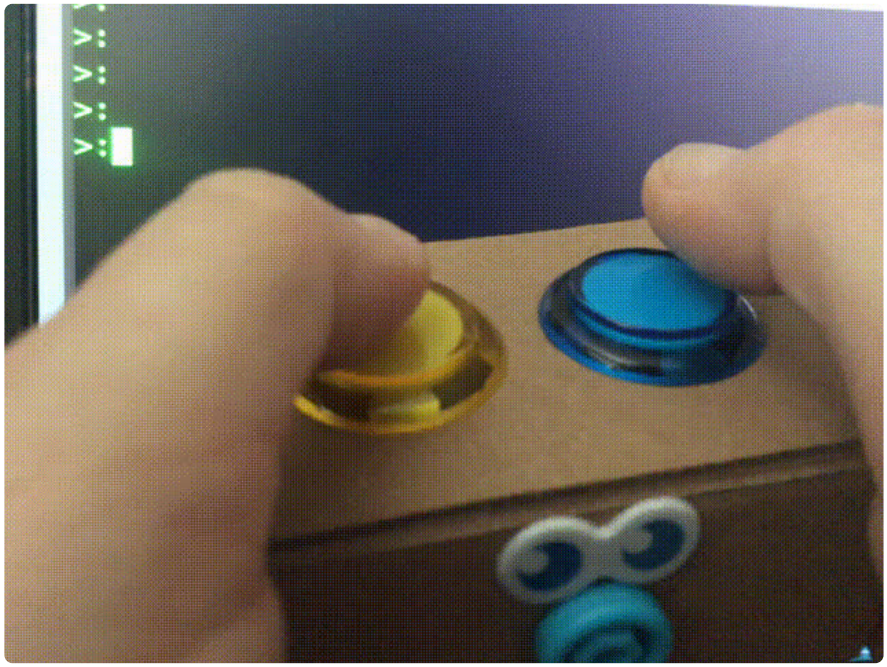
This is a common, and recommended approach in program design! We put things that can change in their own functions. This allows you to avoid changing the core algorithm. That means there's less opportunity to make a mistake and break it.

```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Circuit Playground Express GBoard: universal/customizable version

Adafruit invests time and resources providing this open source code.
Please support Adafruit and open source hardware by purchasing
products from Adafruit!

Written by Dave Astels for Adafruit Industries
Copyright (c) 2018 Adafruit Industries
```

```
Licensed under the MIT license.

All text above must be included in any redistribution.
"""
# pylint: disable=unused-import

import time
from adafruit_circuitplayground.express import cpx

# Uncomment the next 2 lines if you want to use external buttons
# from digitalio import DigitalInOut, Direction, Pull
# import board

# Uncomment the next 3 lines if you want to use HID output
# from adafruit_hid.keyboard import Keyboard
# from adafruit_hid.keycode import Keycode
# import usb_hid

DOT_DURATION = 0.20
DASH_DURATION = 0.5

# You can adjust this to get the level of sensitivity you want.
# Uncomment the next line if you want to use capacitive touch.
# cpx.adjust_touch_threshold(100)

# Uncomment the next 6 lines if you want to use external buttons
# button_a = DigitalInOut(board.A4)
# button_a.direction = Direction.INPUT
# button_a.pull = Pull.UP
# button_b = DigitalInOut(board.A3)
# button_b.direction = Direction.INPUT
# button_b.pull = Pull.UP

# Uncomment the next  line if you want to use HID output
# kbd = Keyboard(usb_hid.devices)



def touch_a():
    # Uncomment the next line if you want to use the on-board buttons
    # return cpx.button_a

    # Uncomment the next line if you want to use capacitive touch
    # return cpx.touch_A4

    # Uncomment the next line if you want to use external buttons
    # return not button_a.value

    return False   # a fail-safe to keep python happy


def touch_b():
    # Uncomment the next line if you want to use the on-board buttons
    # return cpx.button_b

    # Uncomment the next line if you want to use capacitive touch
    # return cpx.touch_A3

    # Uncomment the next line if you want to use external buttons
    # return not button_b.value

    return False   # a fail-safe to keep python happy


def dot():
    # Uncomment the next 2 lines if you want tones played
    # cpx.play_tone(4000, DOT_DURATION)
    # time.sleep(0.1)
```

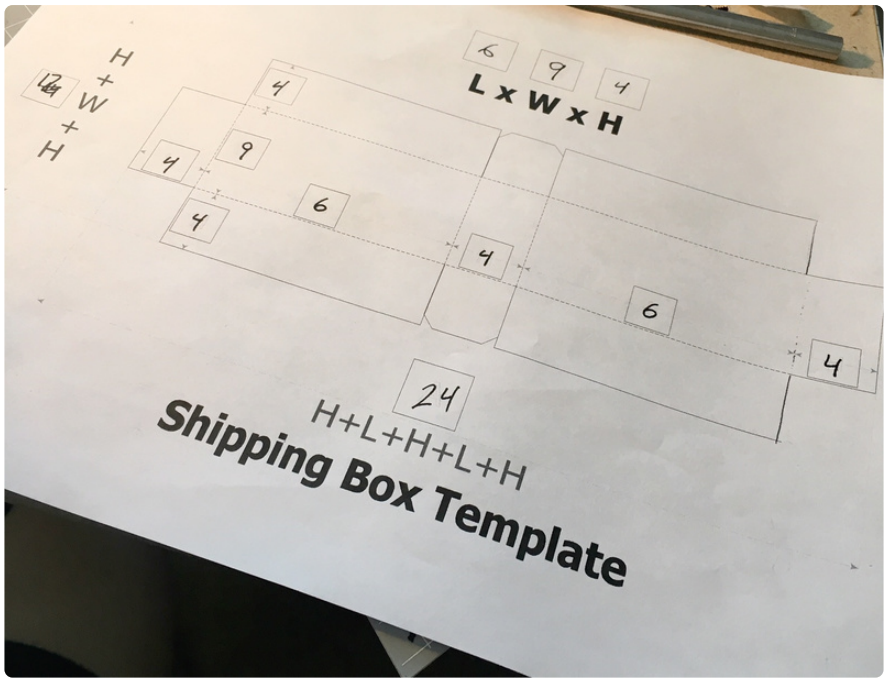
```
    # Uncomment the next line if you want to use HID output
    # kbd.send(Keycode.PERIOD)

    pass   # a fail-safe to keep python happy


def dash():
    # Uncomment the next 2 lines if you want tones played
    # cpx.play_tone(4000, DASH_DURATION)
    # time.sleep(0.1)

    # Uncomment the next line if you want to use HID output
    # kbd.send(Keycode.MINUS)

    pass   # a fail-safe to keep python happy


while True:
    if touch_a():
        dot()
        while touch_a():
            pass
    elif touch_b():
        dash()
        while touch_b():
            pass
```

Physical Build



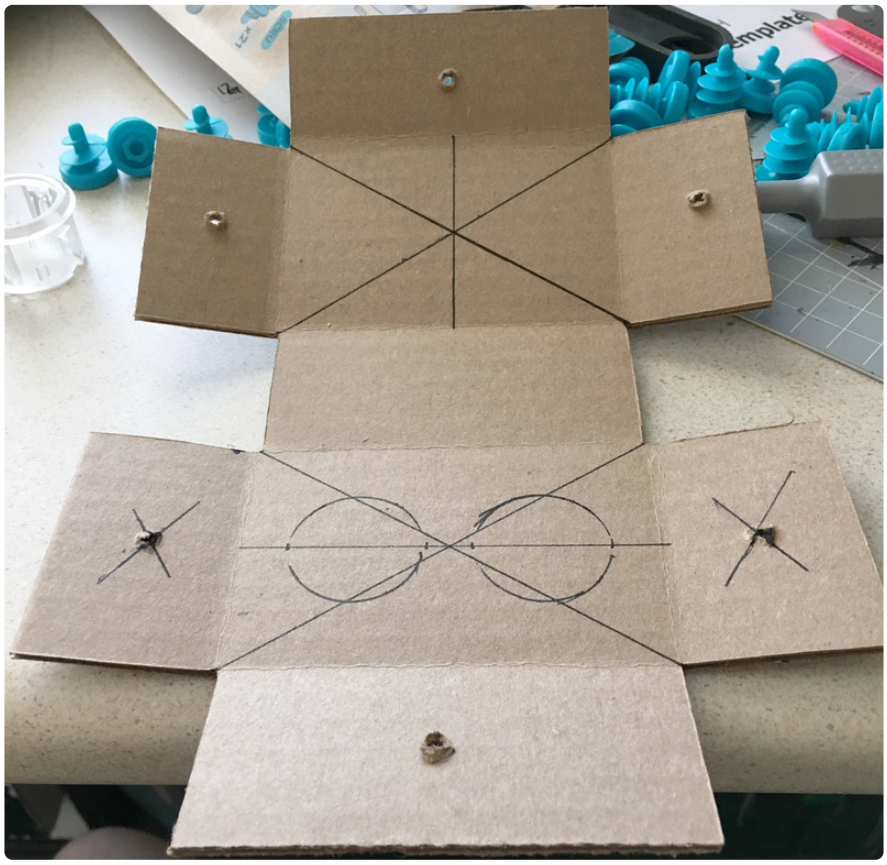
Now that we have the electronics and code figured out, let's built a box to put it all in. Inspired by John Park's recent guides (https://adafru.it/BTZ), we'll use cardboard.

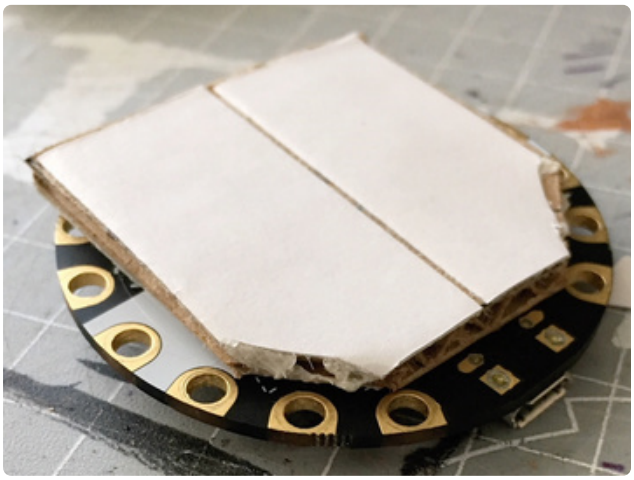
We can start with this Instructable (https://adafru.it/BT-) on making custom cardboard shipping boxes. We need a much smaller box, but it works well enough. After some quick measurements of the arcade switches and Circuit Playground Express we can see that a 6cm x 9cm x 4cm box will fit everything, snuggly but well enough.



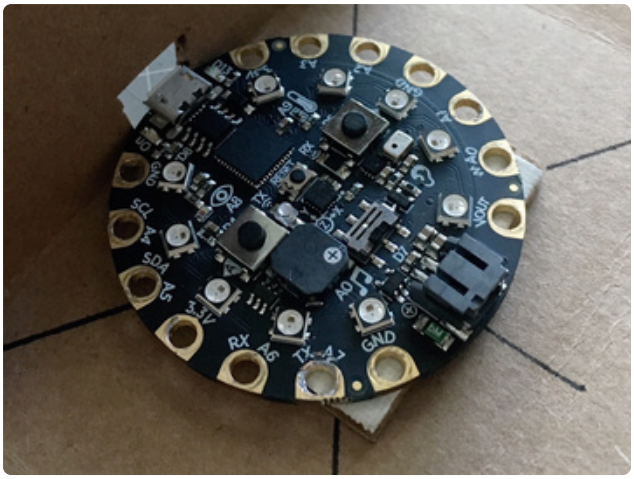
For such a small box, the small flaps in the middle are overkill, and actually get in the way. They can be removed. Once measured, cut, and the bend lines (shown as dashed lines in the template) are partially cut, we're ready to add the required openings. You can see the holes marked and poked through for the Makedo Scru connectors.

Flipping it over, you can mark center lines on the top and bottom sections that will be used to position the Circuit Playground Express and the arcade buttons. 30mm dia. circles are marked for the buttons.

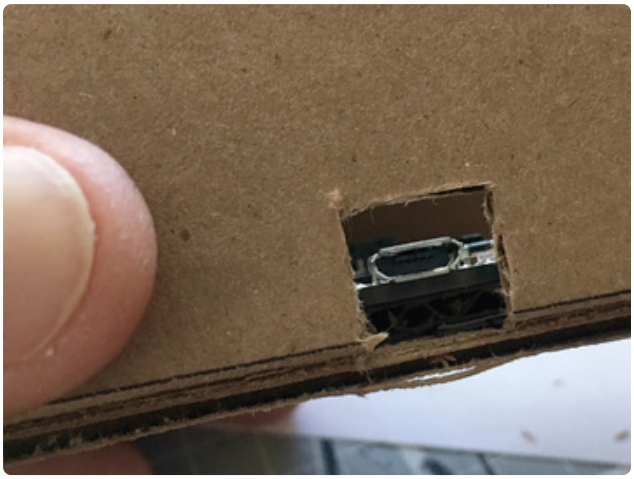




You can mount the Circuit Playground Express using double sided tape and a scrap piece of cardboard to raise it so as to give some room for the alligator clips.

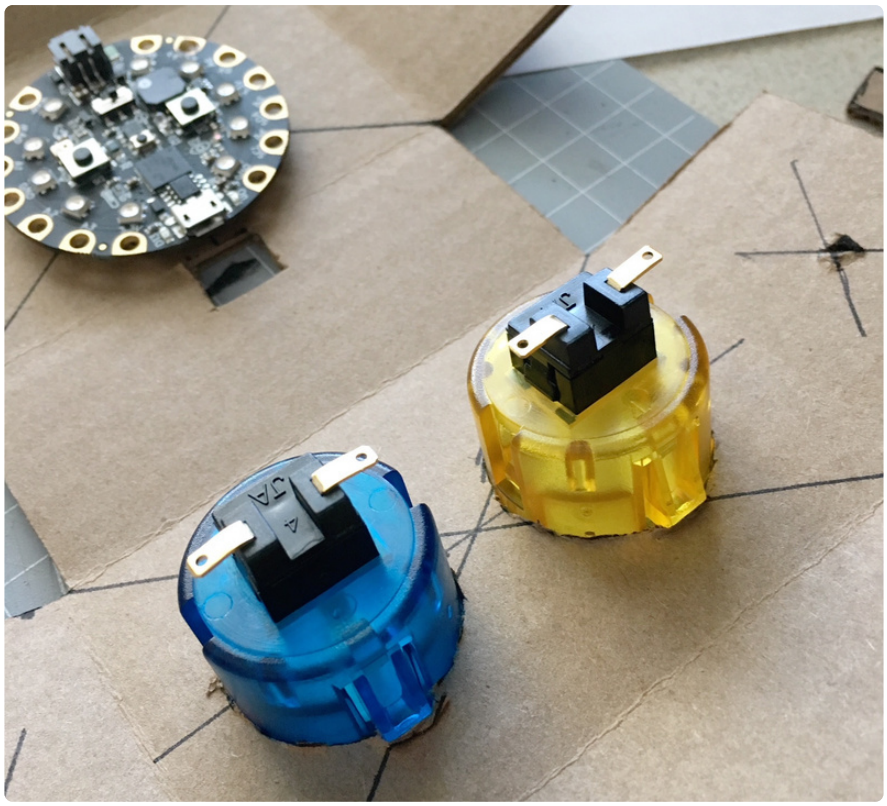


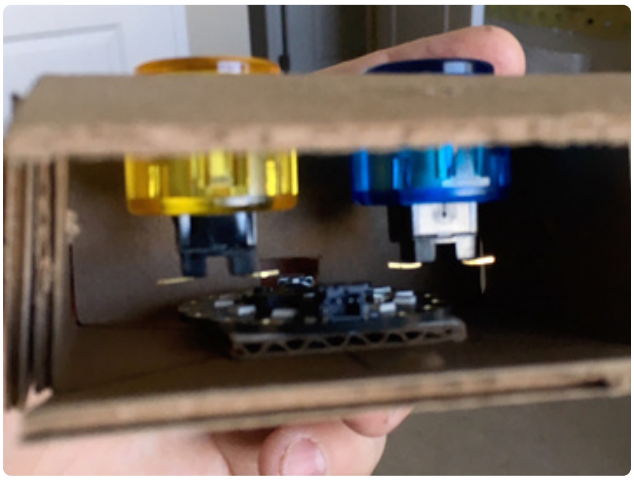
Using a spacer also provides a bit of space around the USB connector.



Make sure you have clearance for your USB cable or OTG adapter

The buttons get pushed through their holes carefully; their snap-in wings should pop out and secure them in place.

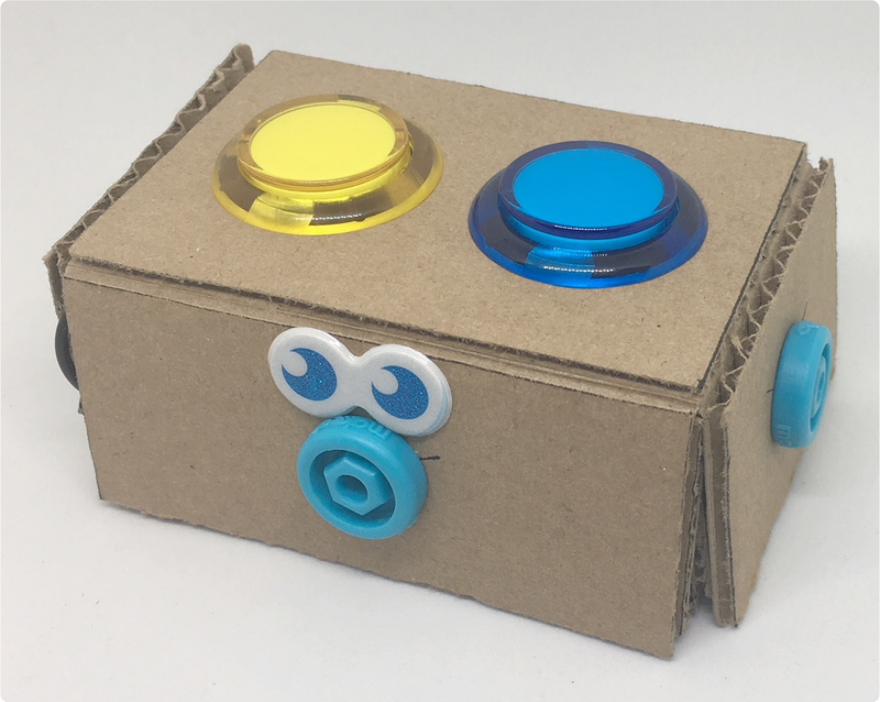




The button connections have been bend to give a bit more clearance between the buttons and the Circuit Playground.

Now it's time to connect the alligator clips and carefully fold the box together. Put the Scrus in the two ends and check that it works.

If it doesn't work, open it again and check/fix the connections. Once it works put in the final Scru.



A set of plastic eyes completes it. It won't win any engineering awards, but it works.

If you want to use it with sound rather than the USB keycodes, you'll need to make the box a bit bigger so you can fit a LiPo battery inside. Be sure to place the batter in a way that you can easily disconnect the cable from the Circuit Playground Express so that it can be recharged.