Blues Playground

Created by Jan Goolsbey

https://learn.adafruit.com/blues-playground

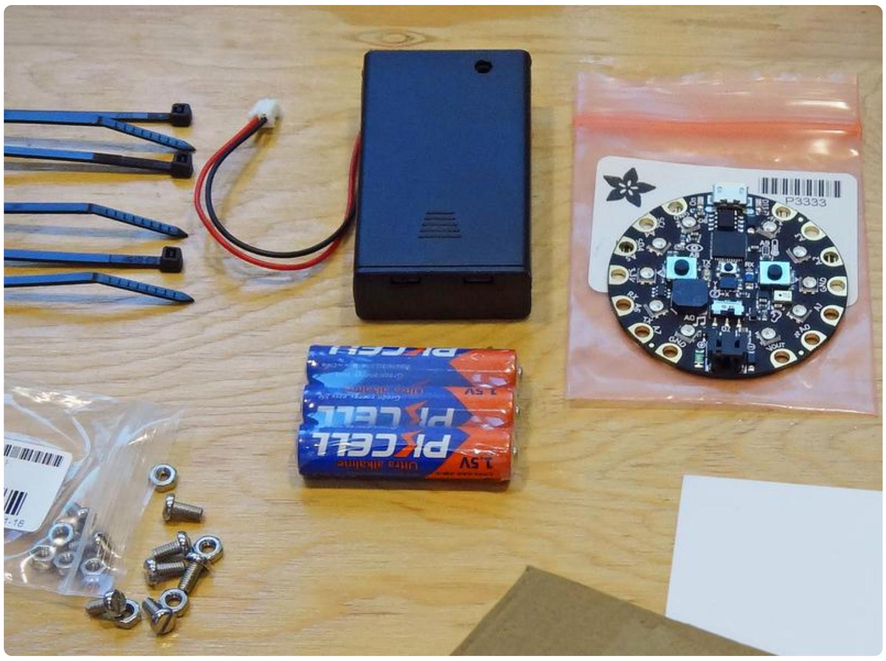
Last updated on 2023-08-29 03:45:04 PM EDT

Table of Contents

Overview

Are there times when you just feel like listening to some Keb' Mo' or Eric Clapton? Do you wish you could play along? Circuit Playground Express and CircuitPython may be all you need to add some tambourine or cowbell to the mix. And what is a vibraslap () anyway?

The Blues Playground is two blues musical instruments in one. First, it's a collection of common percussion instruments played using a hand-held paddle that resembles the shape of a tambourine. You can play with the beat or add accents to spice up the song. The Blues Playground has a second mode that steps through a typical blues chord sequence one beat at a time. The Chords Mode can be used as the musical foundation of your own blues song -- just add lyrics. If you can tap to a beat, you can play the blues!



Here's what you'll need from Adafruit to build the Blues Playground musical instrument:

1 x Circuit Playground Express An amazing microcontroller with lots of built-in features	https://www.adafruit.com/product/3333
1 x 3 x AAA Battery Holder with On/Off Switch A battery holder with a power switch and compatible JST connector	https://www.adafruit.com/product/727
1 x Alkaline AAA Batteries (3 pack) 4.5 volts of electron-filled goodness	https://www.adafruit.com/product/3520

Materials and Supplies

- Two M3 7mm screws with hex nuts (optional for guitar amp connection)
- Two 8-inch x 9-inch (20cm x 23cm) sheets of corrugated cardboard
- Six 4-inch (10cm) cable ties
- Glue stick
- Cardboard-compatible adhesive such as E6000
- Gaffer's tape (or equivalent)
- Cutting template printed on paper stock (printed from downloaded PDF file)

Tools

- Pencil
- Metal ruler

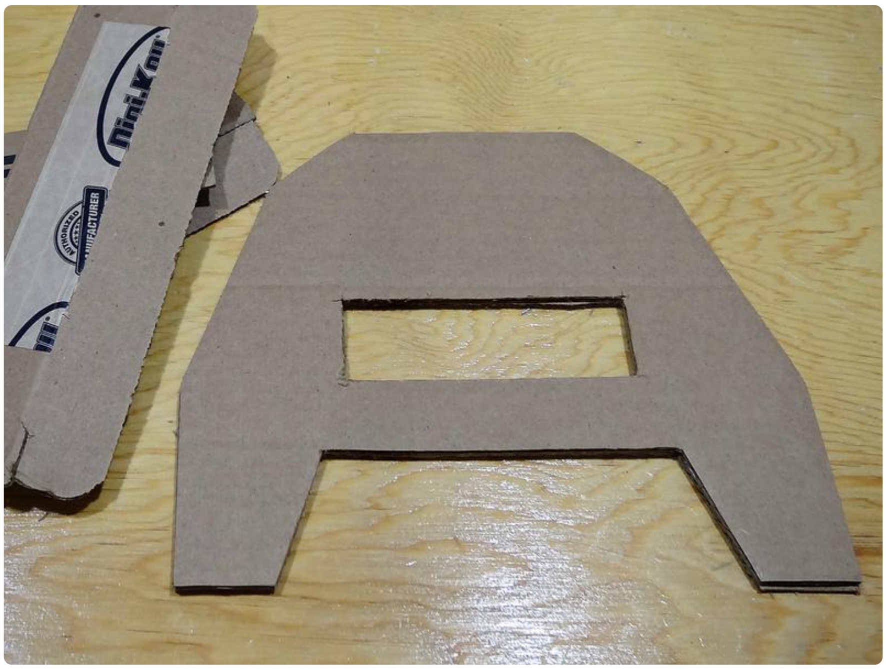
- Scissors, craft knife, or box cutter knife
- Diagonal wire cutter
- For decoration: crayons, felt pens, poster paint, spray paint, stickers

Be Safe!

This Project Uses Sharp Tools and Stinky Adhesive

This project involves cutting and gluing cardboard with sharp tools and stinky glue. If you are inexperienced with either, ask for help before starting this project!

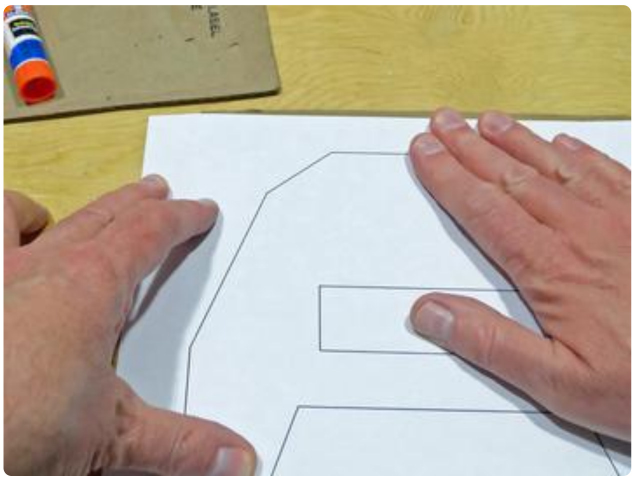
Make the Instrument

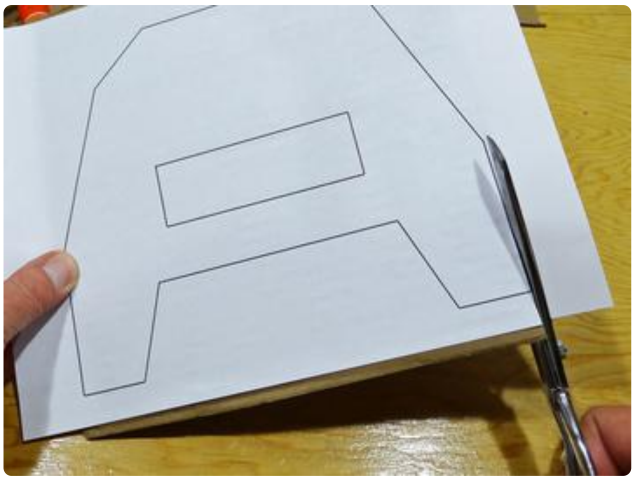


The first order of business is to make the tambourine-like paddle that supports the Circuit Playground Express and battery pack. Almost any sturdy material will work. In this case, we'll be using two layers of corrugated cardboard, laminated for strength and to make the paddle thicker and easier to hold. The microcontroller board and battery pack will be attached with cable ties and a square of gaffer's tape.

Attach the Template and Cut the First Layer

Download the Cutting Template (.pdf)

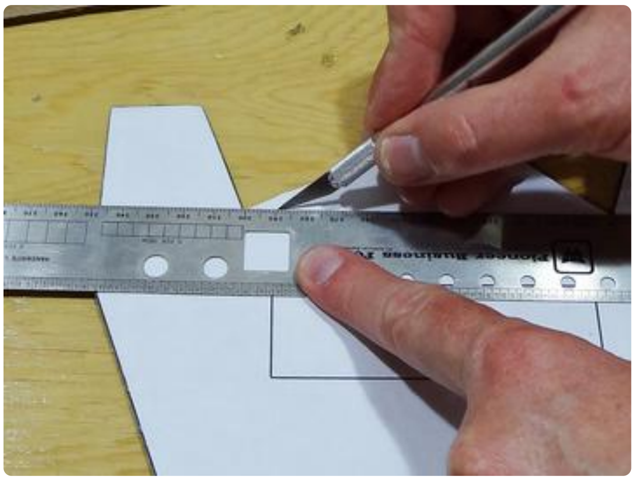
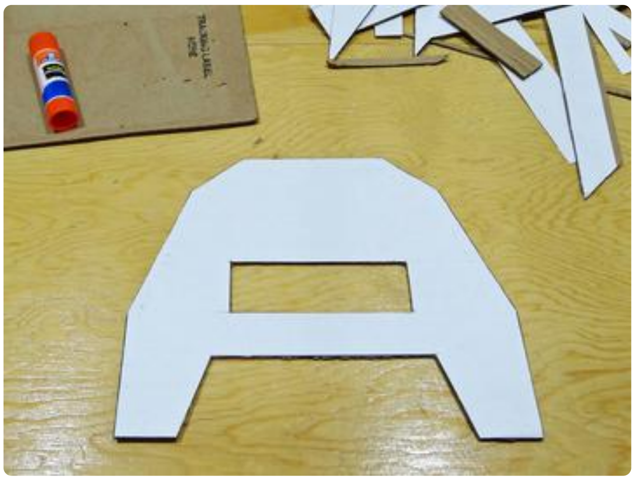




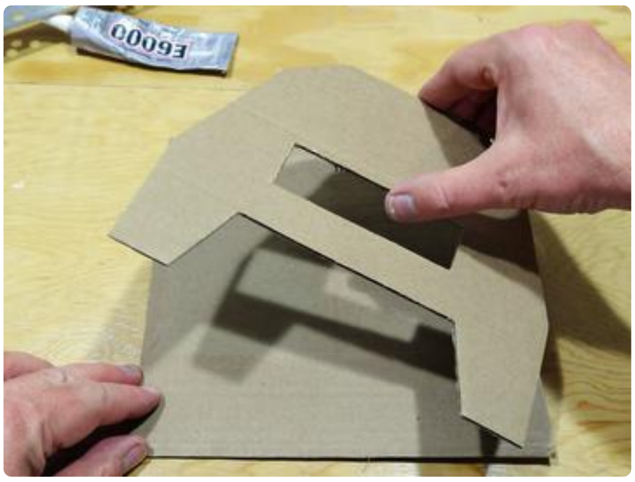
Since we'll be laminating this with another sheet of cardboard, place this piece with printing or blemishes up -- if you don't want the printing to show later.

Use a glue stick to temporarily attach the template to a piece of cardboard. After cutting, the template will be removed, so don't use too much glue; just a few dots will do.

After the glue sets for a few minutes, cut out the template using scissors or a sharp craft knife. A metal ruler can be used to keep the cut lines straight.

Laminate the Layers and Trim to Shape



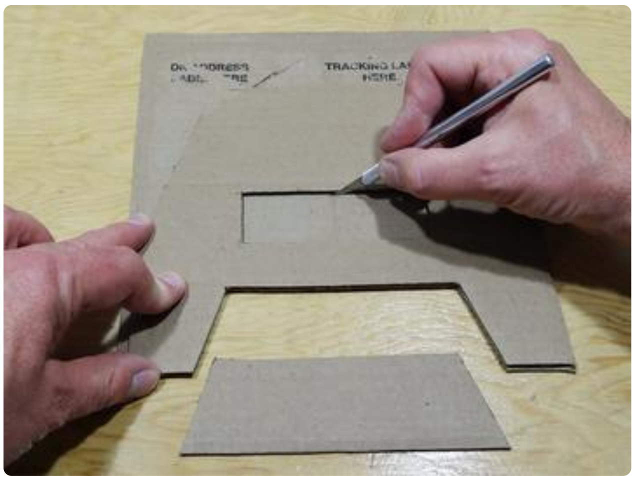
Peel the paper template off of the cut shape. We'll attach the cut cardboard to the remaining sheet using permanent adhesive.

In this case, E6000 was used to stick the layers together. If that's too aromatic for you, use wood glue or gel-type superglue. Be sure to get the adhesive close to the cut edges to avoid fraying and separation after the layers are laminated.

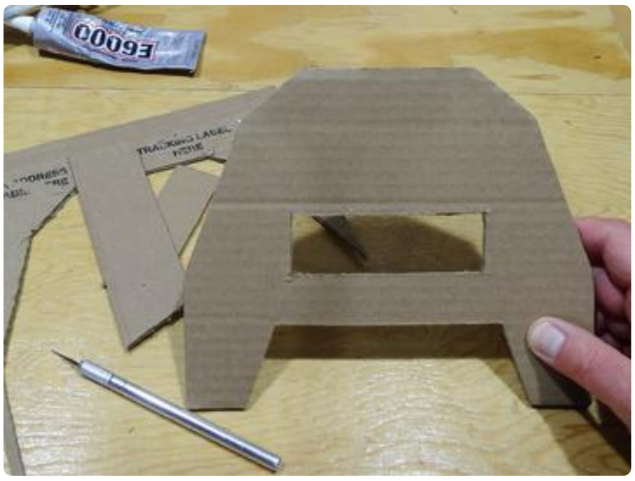
Lay the sticky side of the cut shape onto the remaining cardboard sheet and press down firmly.

After the adhesive sets, trace the cut shape with scissors or a craft knife to trim off the excess and finish the paddle.

At this point, you can decorate your Blues Playground instrument any way you like. Colored duct tape, felt-tipped markers, crayons, poster paint, or spray paint work well. Be creative!

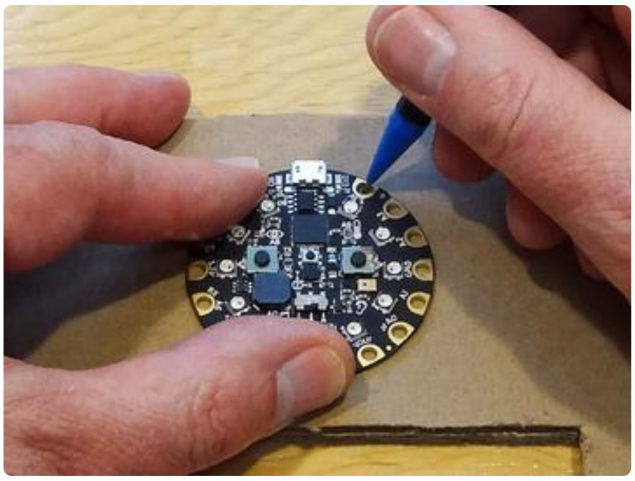


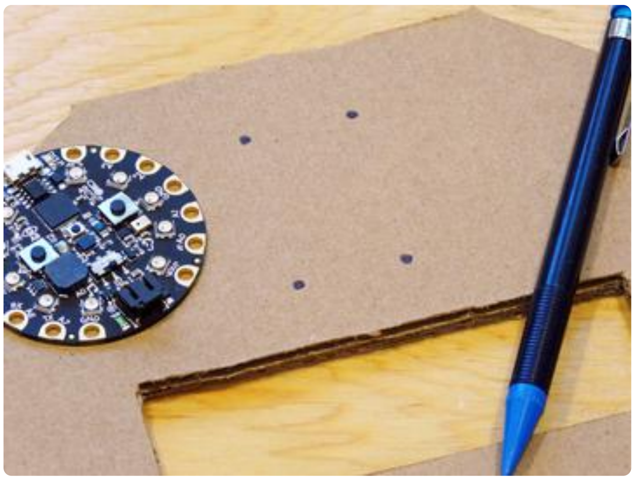
TRACKING LABEL HERE



E6000
TRACKING LABEL HERE

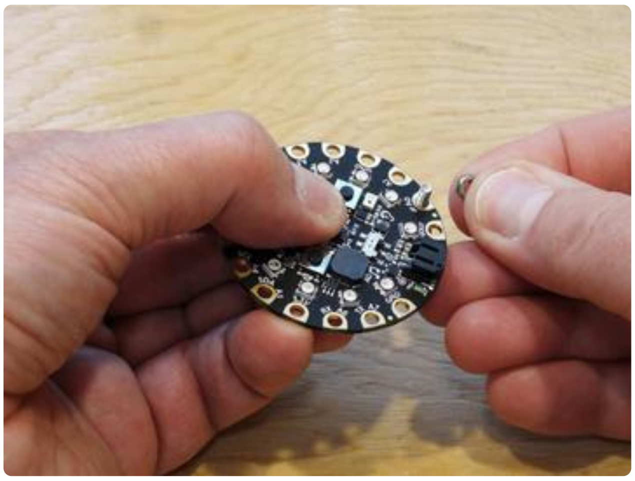
Attach the Circuit Playground Express





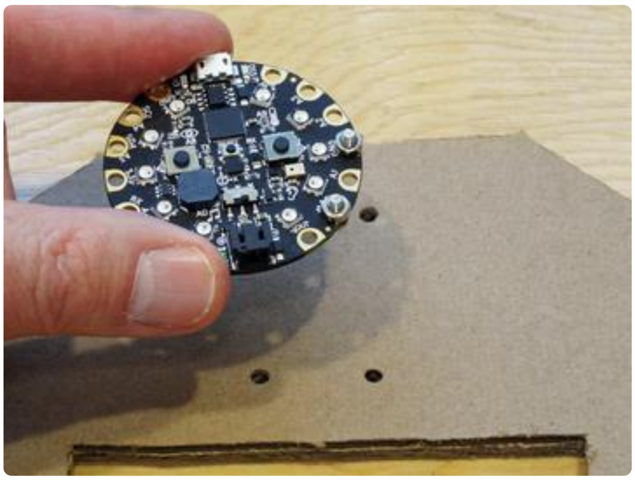
Place the CircuitPlayground Express (CPX) in the center of the top section of the paddle. Using a pencil, mark four mounting spots. Place the top two dots at the 3.3v and GND holes nearest the USB connector. For the bottom two holes, mark the GND and VOUT holes nearest the battery connector.

Using an awl or center punch, poke the cardboard at the dots to make the mounting holes. We'll be using some cable ties to attach the CPX to the paddle soon.

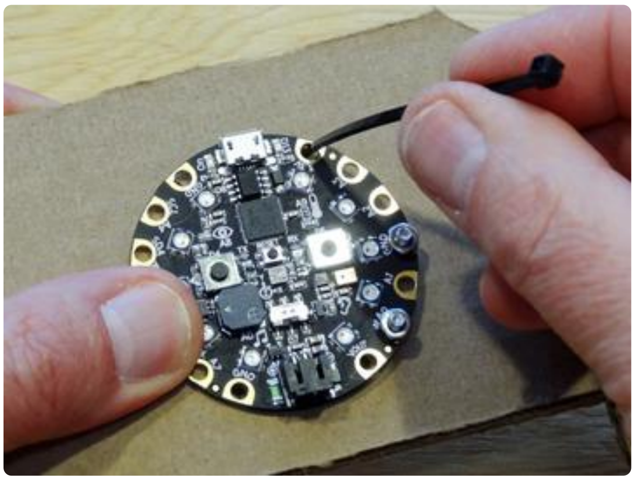


If you're planning on playing the Blues Playground through a guitar amplifier, now's a good time to add signal connectors to the CPX. We'll use alligator clips to connect to a guitar cable, so two screws will make that easier.

Before attaching the CPX to the paddle, insert and tighten two M3 screws and hex nuts through the ~A0 and nearby GND holes, threads facing upwards.

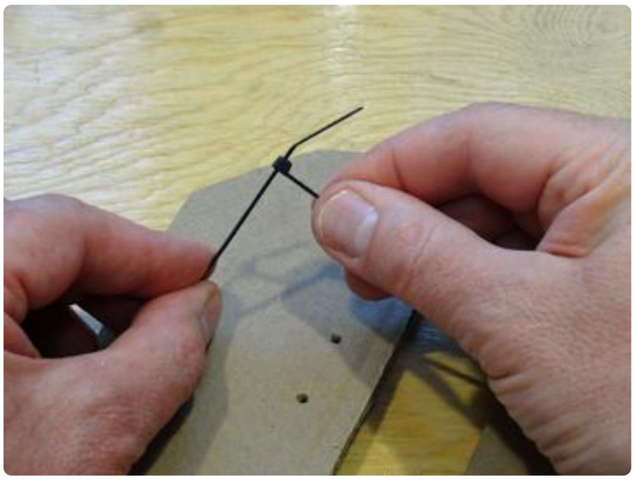


Place the CPX over the paddle's four mounting holes. Thread a cable tie through one of the top holes.

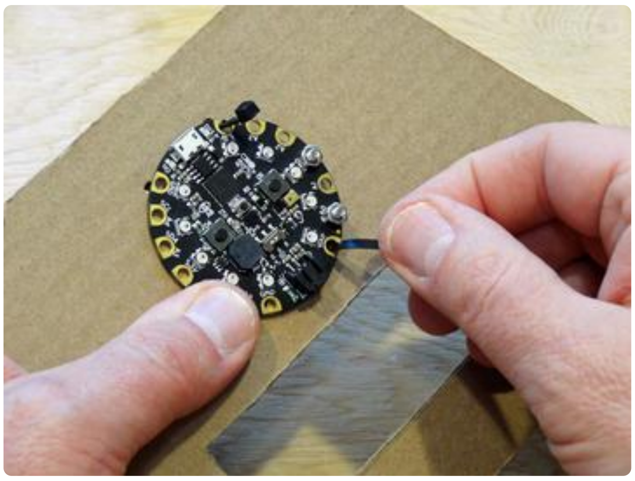
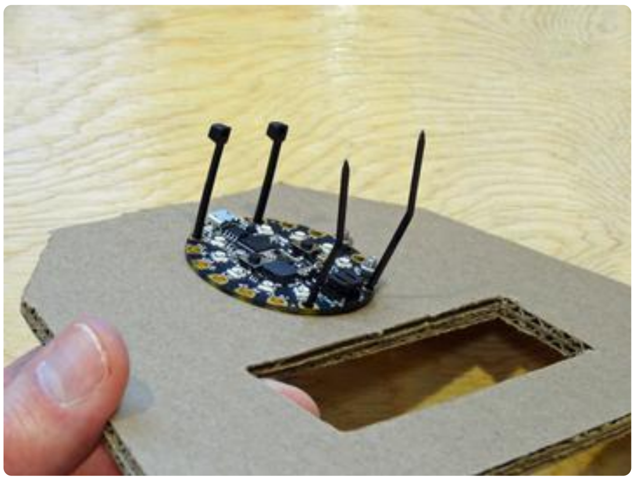
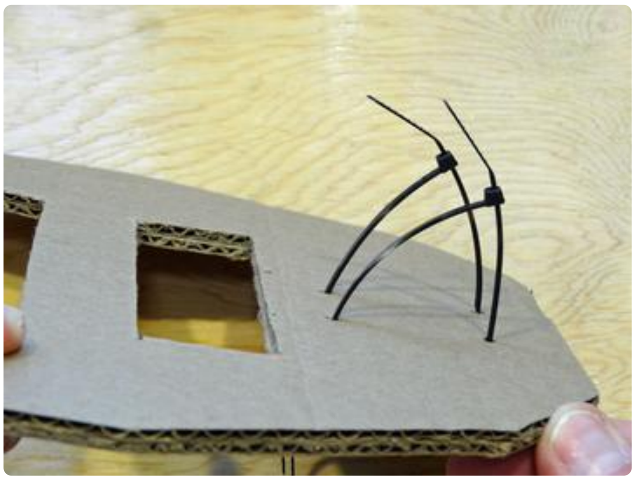
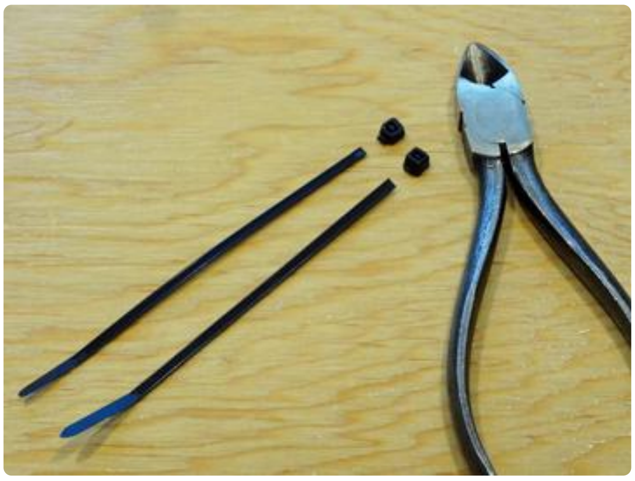


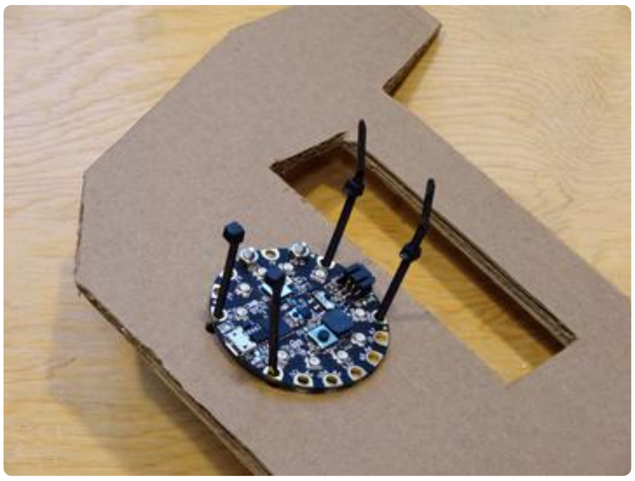
Flip the paddle over and chain a second cable tie to the first to extend the length to accommodate the battery pack. Don't slide it on too far, just enough for the ratchet section of the second to engage with the first.

Poke the end of the second cable tie through the lower mounting hole and pull it through the front of the paddle. Repeat the process for the other two mounting holes.

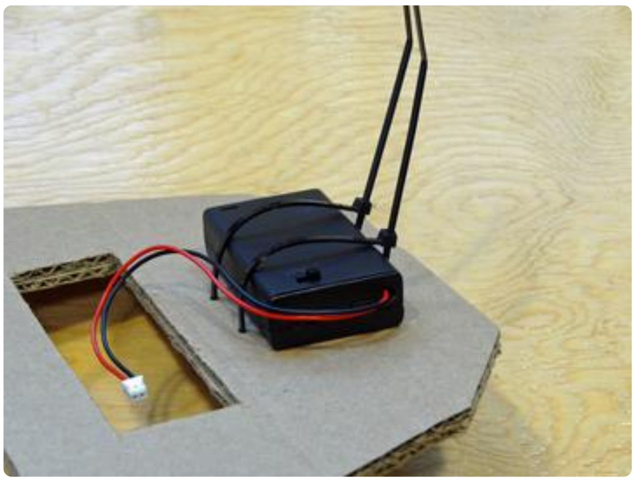
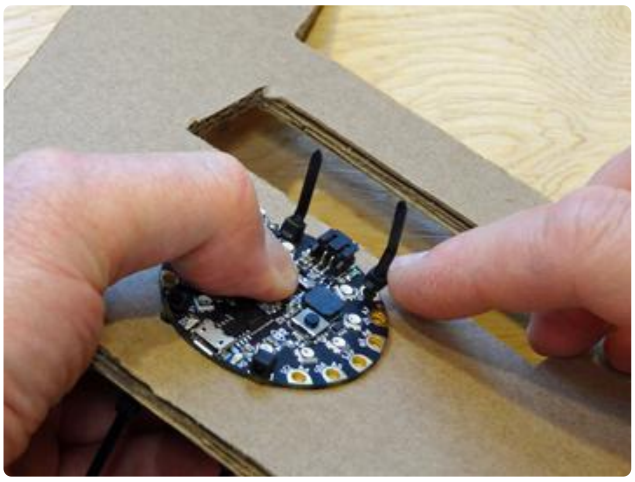
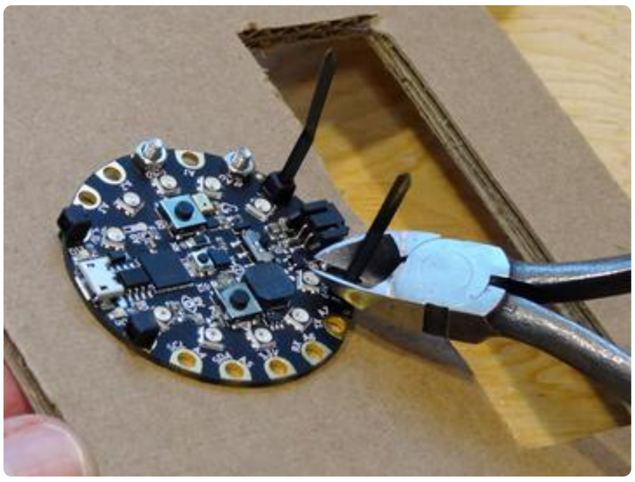


Using wire cutters, cut the ratchet ends of the two unused cable ties. Slip the ratchet ends onto the exposed ends of the cable ties as shown. As before, just slide them on until the ratchet engages. We'll tighten these after the battery pack is in place.



Attach the Battery Holder

Put the batteries into the holder and slide the cover closed. Place the battery holder on the back side of the paddle as shown. Check to see that the on-off switch is easily accessible.

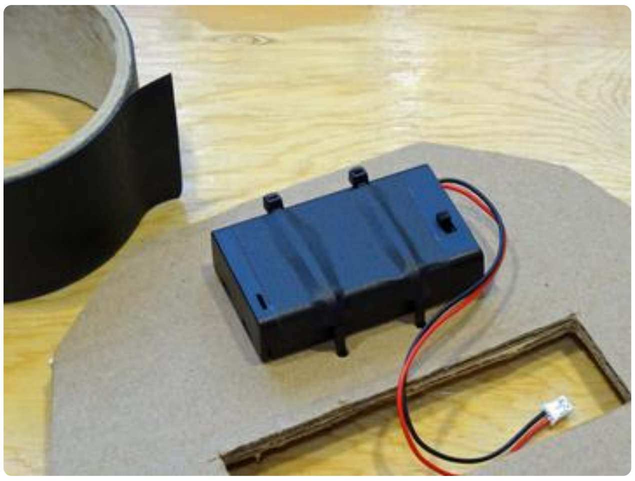
Tighten the top two cable ties somewhat by sliding the ratchets down to approximately the height of the battery pack (as shown in the first photo).

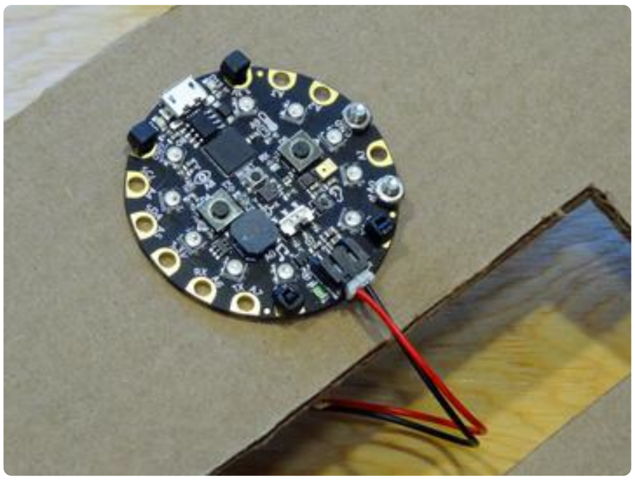
Flip the paddle over and slide the two ratchets pieces to tighten. Finger tight is good enough to hold everything in place.nstall the batteries then slip into the opening on the back of the paddle.

Trim off the excess cable tie lengths from the front and the back of the paddle.

Place a square of gaffer's tape onto the battery pack and over the cable ties to secure for playing the instrument.

Finally, plug the battery pack cable into the CPX battery connector. The cable can be passed through the paddle's hand hole. Once the CircuitPython code is loaded, we'll be ready to play the blues!





Code with CircuitPython

Are you new to using CircuitPython? No worries, there is a full getting started guide here ().

Keep Up with the Latest Version

The code will work with either CircuitPython 2.3.1 or the 3.0.0 release candidate version. Using the latest CircuitPython and library release assures that your project will function from the get-go. Visit the Circuit Playground Express learning guide () for how to install the latest version.

Download the Sound Samples

Download the sound samples folder and copy the sound file contents to the Circuit Playground Express (CPX). The code will use these files to play the different percussion and chord sounds.

Download the BP_Sound_Files.zip from GitHub

CircuitPython Code

Here's the CircuitPython code that will power the Blues Playground instrument. It's recommended that you use the Mu editor and it's REPL to edit and test your code. You can learn about installing and using Mu in this tutorial ().

You can copy the full code into Mu and save it to the CPX as file with the name code.py to start playing the blues. A breakdown of how each section of the code works follows the full code listing.

```
# SPDX-FileCopyrightText: 2018 Anne Barela for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Blues Playground Instrument
# 2018-06-19 v03

import time
from adafruit_circuitplayground.express import cpx

# lists
drums = [  # startup and drum sound files
    "chime.wav",
    "kick.wav",
    "snare.wav",
    "hat.wav",
    "cymbal.wav",
    "tamb.wav",
    "cowbell.wav",
    "vibra_slap.wav"
]

chords = ["cp_A.wav", "cp_D.wav", "cp_E.wav"]

chord_changes = [  # chord_file, beats to play, beats played
    [0, 4, 0],
    [1, 4, 0],
    [0, 2, 0], [1, 2, 0],
    [0, 4, 0],
    [1, 4, 0],
    [1, 4, 0],
    [0, 2, 0], [1, 2, 0],
    [0, 4, 0],
    [2, 4, 0],
    [1, 4, 0],
    [0, 2, 0], [1, 2, 0],
    [0, 2, 0], [2, 2, 0]
]

cpx.pixels.fill((0, 0, 0))  # clear all pixels
cpx.play_file(drums[0])  # play startup

# select instrument mode with slide switch
if cpx.switch:
    instrument = "Chords"  # remember mode for later
    for i in range(9, -1, -1):  # fill chord range background with blue
```

```
        cpx.pixels[i] = (0, 0, 1)
        time.sleep(0.05)
else:
    instrument = "Drums"  # remember mode for later
    for i in range(9, 2, -1):  # fill voice range background with white
        cpx.pixels[i] = (1, 1, 1)
        time.sleep(0.05)

    voice = 1  # first drum voice
    cpx.pixels[10 - voice] = (0, 5, 0)  # green for first voice
    cpx.play_file(drums[voice])  # play first drum voice

    # choose drum voice with button A, lock-in selection with button B
    while not cpx.button_b:
        if cpx.button_a:  # loop through voices
            cpx.pixels[10 - voice] = (1, 1, 1)  # replace background color
            voice = voice + 1
            if voice > 7:
                voice = 1
            cpx.pixels[10 - voice] = (0, 5, 0)  # green for voice selection
            cpx.play_file(drums[voice])  # play current voice
        time.sleep(0.3)
    for i in range(3, 10):
        if i != (10 - voice):
            cpx.pixels[i] = (0, 0, 0)
            time.sleep(.1)
time.sleep(0.3)

# let's play the blues!
c_idx = 0  # start with the first chord change
while True:
    cpx.detect_taps = 1
    if cpx.tapped:  # wait for single tap
        if instrument == "Drums":
            cpx.play_file(drums[voice])  # play the selected drum sound

        if instrument == "Chords":  # play chords in sequence
            cpx.pixels[9 - (c_idx % 16)] = (3, 3, 0)  # yellow foreground
            cpx.play_file(chords[chord_changes[c_idx][0]])
            chord_changes[c_idx][2] = chord_changes[c_idx][2] + 1
            cpx.pixels[9 - (c_idx % 16)] = (0, 0, 1)  # replace background
            if chord_changes[c_idx][2] >= chord_changes[c_idx][1]:
                chord_changes[c_idx][2] = 0
                c_idx = c_idx + 1
                if c_idx > 15:
                    c_idx = 0
```

Let's break down the code into sections to understand what it does.

Import Libraries

The code will need some libraries to provide support for reading the CPX's switch, buttons, and accelerometer, operating the on-board NeoPixels, and playing sound over the speaker. We will also use a library for creating time delays.

`time` provides the delays we'll need for NeoPixel animation. The most important library, `adafruit_circuitplayground.express`, is the one that supports the myriad of CPX features, of which we'll be using the slide switch, push buttons, accelerometer, speaker, and NeoPixels.

You can learn more about installing the `time` and `adafruit_circuitplayground.express` libraries in the [CircuitPython Essentials Guide on Libraries ()]().

```
import time
from adafruit_circuitplayground.express import cpx
```

Make Some Lists

The Blues Playground code frequently refers to the recorded percussion sound and chord file names. The percussion sound file names are elements of the `drums` list. The three musical chords used in the blues sequence are elements of the `chords` list.

The sequence of chords played for the 12-bar blues is contained in the `chord_changes` list. Each element of the list contains a tuple (like a sub-list) for each chord that refers to the chord file by number, lists the number of beats for playing that chord, and has a third element that keeps track of how many times that chord was played. For example, the first entry of the list, `[0, 4, 0]`, indicates that first file in the `chords` list, cp_A.wav (the chord in the key of A), should be played for four beats and that no beats have been played so far.

A "bar" in the 12-bar blues sequence is a musical measure consisting of four beats. In this sequence, one of the three chords (A, D, E) will be played 16 times during the 12 measures, so there are a few instances in the `chord_changes` list where a chord will only be played for two beats. Can you spot them?

```
# lists
drums = [  # startup and drum sound files
    "chime.wav",
    "kick.wav",
    "snare.wav",
    "hat.wav",
    "cymbal.wav",
    "tamb.wav",
    "cowbell.wav",
    "vibra_slap.wav"
]

chords = ["cp_A.wav", "cp_D.wav", "cp_E.wav"]

chord_changes = [  # chord_file, beats to play, beats played
    [0, 4, 0],
    [1, 4, 0],
    [0, 2, 0], [1, 2, 0],
    [0, 4, 0],
    [1, 4, 0],
    [1, 4, 0],
    [0, 2, 0], [1, 2, 0],
    [0, 4, 0],
    [2, 4, 0],
    [1, 4, 0],
    [0, 2, 0], [1, 2, 0],
    [0, 2, 0], [2, 2, 0]
]
```

Initialize the NeoPixels and Play the Startup Chime Sound

The next section of code turns off all the NeoPixels and plays a startup sound over the CPX speaker.

```
cpx.pixels.fill((0, 0, 0))  # clear all pixels
cpx.play_file(drums[0])  # play startup
```

Choosing a Percussion Sound

The slide switch and push buttons are used to select which percussion instrument sound to play. If the slide switch is positioned to the right, then the Blues Playground will play percussion. To the left selects the blues chords sequence mode.

When the slide switch is in the Chords position, all the NeoPixels turn blue and the instrument goes immediately into play mode, bypassing the remaining code in this section.

```
# select instrument mode with slide switch
if cpx.switch:
    instrument = "Chords"  # remember mode for later
    for i in range(9, -1, -1):  # fill chord range background with blue
        cpx.pixels[i] = (0, 0, 1)
        time.sleep(0.05)
```

When in the percussion (Drums) mode, the first seven NeoPixels (#9 through #3) turn white. The code starts with the first drum sound as indicated by the `voice` variable, turns NeoPixel #9 to green, plays the first drum sound, and waits for button A or B to be pushed. Pushing button A will turn the current NeoPixel white, step to the next sound file, light the next NeoPixel in sequence and play a sample of the next drum sound. This will repeat until button B is pressed to lock in the selection. Once a voice is selected, the Blues Playground moves on from this section of code.

```
else:
    instrument = "Drums"  # remember mode for later
    for i in range(9, 2, -1):  # fill voice range background with white
        cpx.pixels[i] = (1, 1, 1)
        time.sleep(0.05)

    voice = 1  # first drum voice
    cpx.pixels[10 - voice] = (0, 5, 0)  # green for first voice
    cpx.play_file(drums[voice])  # play first drum voice

    # choose drum voice with button A, lock-in selection with button B
    while not cpx.button_b:
        if cpx.button_a:  # loop through voices
            cpx.pixels[10 - voice] = (1, 1, 1)  # replace background color
            voice = voice + 1
            if voice &gt; 7:  voice = 1
            cpx.pixels[10 - voice] = (0, 5, 0)  # green for voice selection
            cpx.play_file(drums[voice])  # play current voice
        time.sleep(0.3)
    for i in range(3, 10):
        if i != (10 - voice):
            cpx.pixels[i] = (0, 0, 0)
            time.sleep(.1)
time.sleep(0.3)
```

Time to Play the Blues

In this section, we start by initializing the variable `c_idx` (chord index) to zero. This variable is used by the Chords mode to keep track of which chord is playing -- 0 through 15 for all the chords needed to play the 12-bar blues sequence. `c_idx` points to the chord attributes contained in the `chord_changes` list.

```
# let's play the blues!
c_idx = 0  # start with the first chord change
```

The CPX accelerometer can detect when the paddle is tapped or struck by using the `cpx.tapped` status. When a tap is detected, the code uses the slide switch selection made earlier (as indicated by the `instrument` variable) and steers the code to either play the selected drum instrument or play through the sequence of chords. Playing the selected drum sound is pretty simple: the sound file is played by the `cpx.play_file(filename)` function.

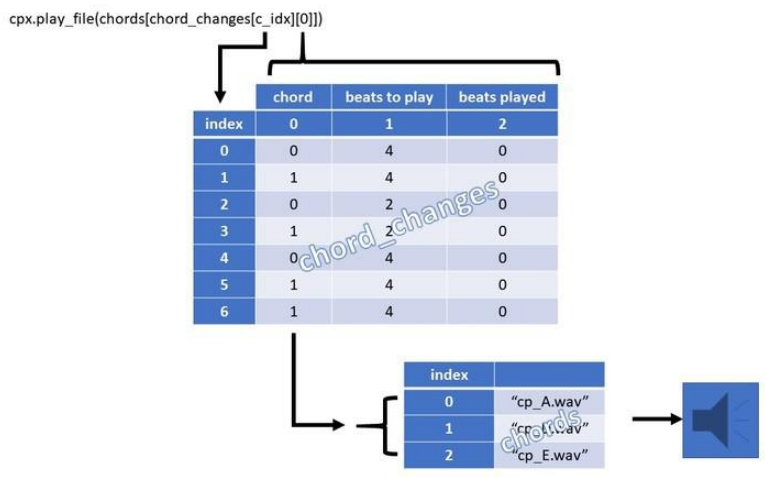
```
while True:
    cpx.detect_taps = 1
    if cpx.tapped:  # wait for single tap
        if instrument == "Drums":
            cpx.play_file(drums[voice])  # play the selected drum sound
```

Playing through the chord sequence is a bit more complicated. The code will turn a NeoPixel yellow depending on which chord is being played. NeoPixel #9 is the first chord, #8 is the second, etc. After the chord is played (`cpx.play_file(chords[chord_changes[c_idx][0]])`), the beats played portion of the tuple in the `chord_changes` list is incremented, the NeoPixel is reset to blue, then beats played is compared to the number of beats to play. When the last beat of that chord is played, the beats played portion of the tuple is reset to zero and the chord index `c_idx` moves to the next chord in sequence. After the last chord of the 16 chords is played, `c_idx` is reset to zero so that the sequence starts over from the top. Whew.

```
        if instrument == "Chords":  # play chords in sequence
            cpx.pixels[9 - (c_idx % 16)] = (3, 3, 0)  # yellow foreground
            cpx.play_file(chords[chord_changes[c_idx][0]])
            chord_changes[c_idx][2] = chord_changes[c_idx][2] + 1
            cpx.pixels[9 - (c_idx % 16)] = (0, 0, 1)  # replace background
            if chord_changes[c_idx][2] &gt;= chord_changes[c_idx][1]:
                chord_changes[c_idx][2] = 0
                c_idx = c_idx + 1
                if c_idx &gt; 15:  c_idx = 0
```

The `cpx.play_file(chords[chord_changes[c_idx][0]])` function may be a bit difficult to understand, so let's break it down a bit more. We're using the two lists, `chords` and `chord_changes`, to tell CircuitPython which chord file to play. First, we look up the first item (item 0) of the tuple in the `chord_changes` list that is pointed to by `c_idx`. That gives us a number 0 through 2 that specifies which stored chord file to play. To get the name of the chord file, we use that number to point to the corresponding item in the `chords` list so that we can get the string of characters that the `cpx.play_file` function needs to locate and play the file stored in the CPX directory.

Using multiple lists is a way of "normalizing" the information used to keep track of the 16 chord changes and the three sound file names. In this case, normalizing the information eliminates the need to duplicate the sound file names in the `chord_changes` list, reducing the overall size of the code so that it'll fit in the limited memory of the CPX.



	chord	beats to play	beats played
index	0	1	2
0	0	4	0
1	1	4	0
2	0	2	0
3	1	2	0
4	0	4	0
5	1	4	0
6	1	4	0

index	
0	“cp_A.wav”
1	“cp_[illegible].wav”
2	“cp_E.wav”

Jammin'

Playing the Blues Playground

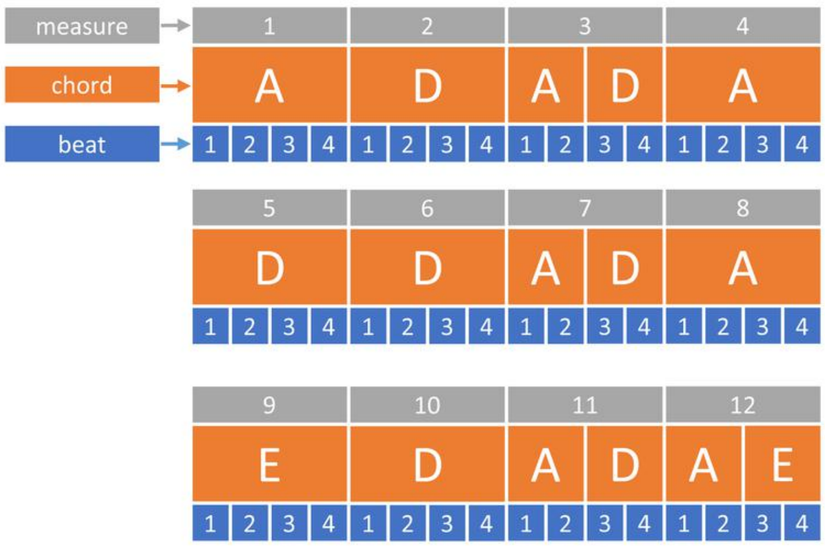
To play along using a percussion instrument, move the CPX slide switch to the right and turn on the power switch. Select the instrument you'll be playing by pressing the A pushbutton until you hear the sound you want. Press the B pushbutton to select that sound and start playing!

To play the blues chords, move the CPX slide switch to the left and turn on the power switch (or press the Reset button if power is already on). The NeoPixels will all turn a light blue indicating that the chords are ready to play. Tap the Blues Playground to play a chord along with the beat. Playing slow or fast is your choice. Slow blues can be very expressive; fast blues are more light-hearted. What's your mood?

The 12-Bar Blues in the Key of A

The blues is a very expressive form of music that also contributes to many other genres such as Jazz, Rock 'n Roll, and Country. The format and sequence for blues chords aren't completely fixed, but they usually form familiar patterns. The chord pattern, or "chord changes" as they're sometimes called, are the musical foundation for blues and jazz improvisation and can lead to the creation of your own song -- just add lyrics to tell your story.

The Blues Playground's code contains a pattern for the 12-bar blues in the key of A. That's a popular sequence that only involves just 3 different chords played 16 times. In this case, we're using the chords of A, D, and E. Here's the sequence grouped into twelve 4-beat measures:



When playing the blues, the chord pattern is repeated throughout the song. Play it as long as you'd like 'cause the Blues Playground repeats the sequence for as long as you tap with the beat.

Jammin' with Friends

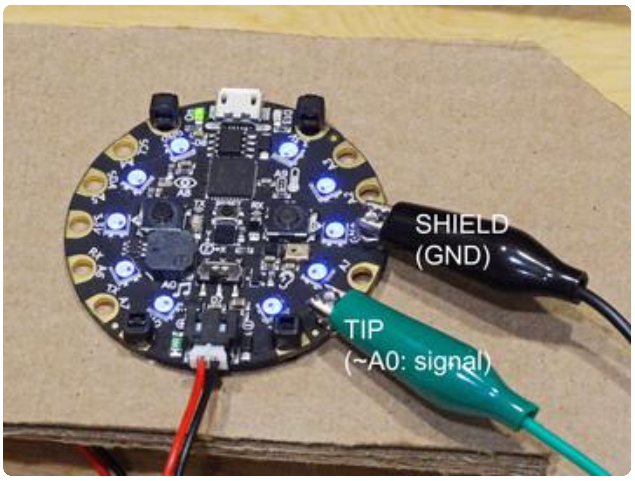
The chords around the key of A were selected for the Blues Playground because many guitar players prefer to play in the key of A and it fits the vocal range of most singers. Also, this key works well for a diatonic harmonica especially when played in the cross-harp mode. Cross-harp is when blues harmonica players use a harp tuned to the key of D when guitar players are jamming in the key of A. The harp player will start around hole #4 and draw air more than blowing to find the blues melody notes.

(Note: The author is a harmonica player and loves to play jazz and blues, so it's expected that he would mention the harp somewhere in this guide. He couldn't resist.)

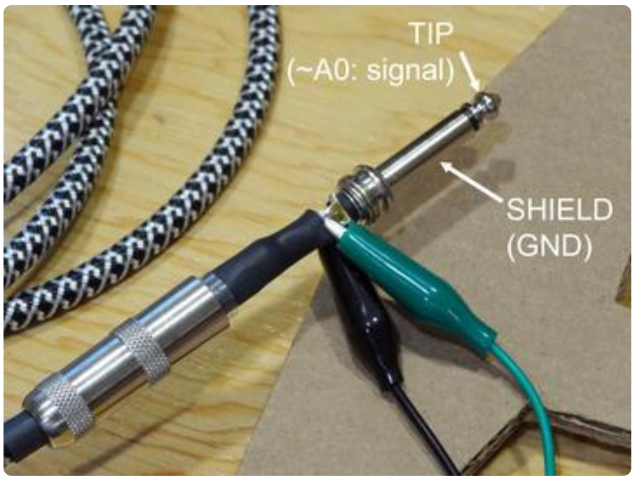
Hacking the Blues

Let's Crank up the Volume!

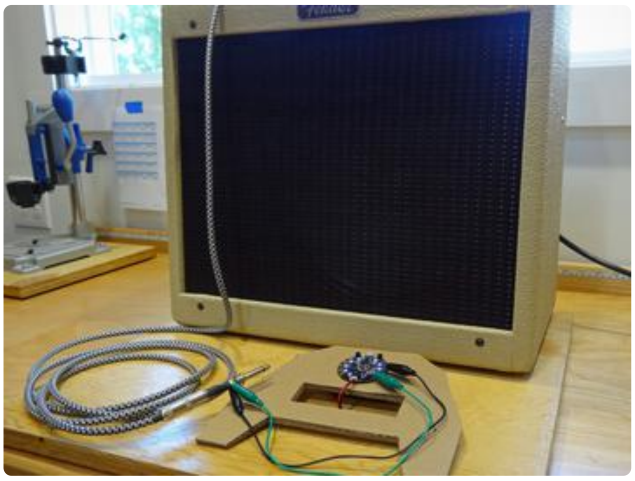
If you really need to be heard above the crowd, connect the Blues Playground to a guitar amplifier. The simplest way to connect the two is with a pair of alligator clips and a 1/4" guitar signal cable.



The guitar amplifier needs two connections, one as a common ground and the other for the musical signal. Clip a black alligator wire lead to the GND connection of the CPX. An alligator wire of a different color is clipped to the ~A0 (analog signal output) pin. Remember the screws we attached earlier? The clips will hold well to those.



The other ends of the alligator clip wires attach to the Tip and Shield of the guitar cable plug. If your alligator clips are large enough, you can attach directly to the plug. If not, unscrew the plug handle and attach directly to the pins of the plug where the guitar cable wires are soldered.



Turn the amplifier's volume all the way down before turning on its power. The CPX puts out a loud signal!

If you encounter hum when no sound is playing, you may not have a good connection or your alligator clip wires are too long. Short lengths will reduce the antenna effect of the unshielded wires.

Recording Your Own Sound Samples

Do you want to play the blues in a minor key? If you're handy at recording your own chord samples, put the new sound files in a compatible format, store them on the CPX, and update the CircuitPython code to show the new chord file names. Mike Barela wrote an excellent guide that can help with the sound file conversions: Microcontroller Compatible Audio File Conversion ().

More Information

For more information about the origin of the blues musical genre, try https://en.wikipedia.org/wiki/Blues (). Examples of blues songs can easily be found on YouTube and most recorded music catalogs.