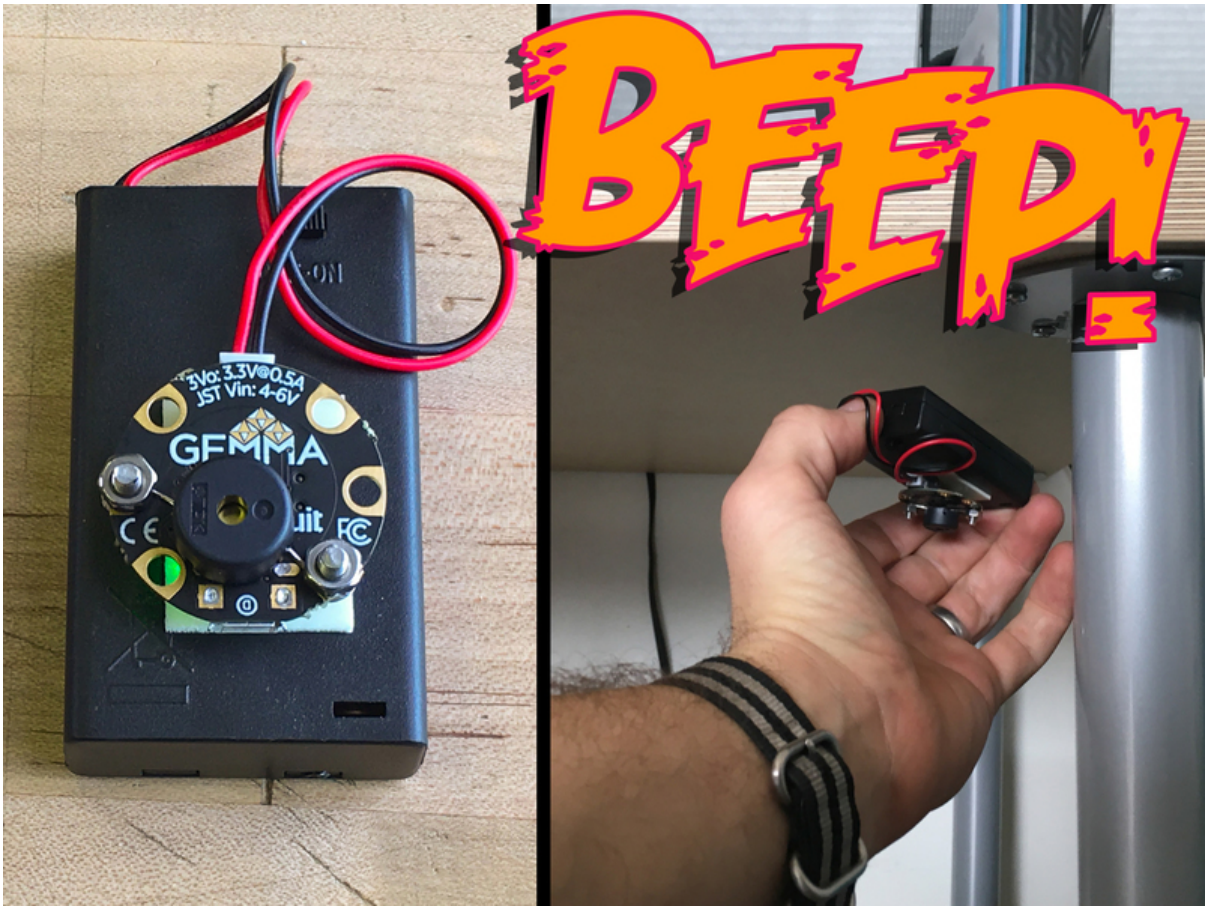
Annoy-O-Matic Sound Prank Device

Created by John Park

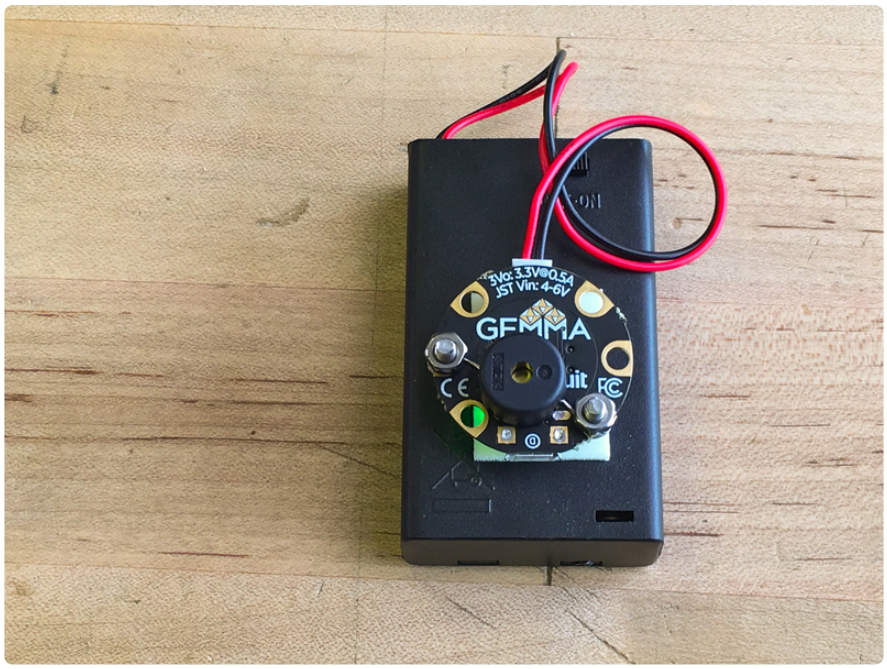


https://learn.adafruit.com/annoy-o-matic-sound-prank-device

Last updated on 2024-03-08 02:58:20 PM EST

Table of Contents

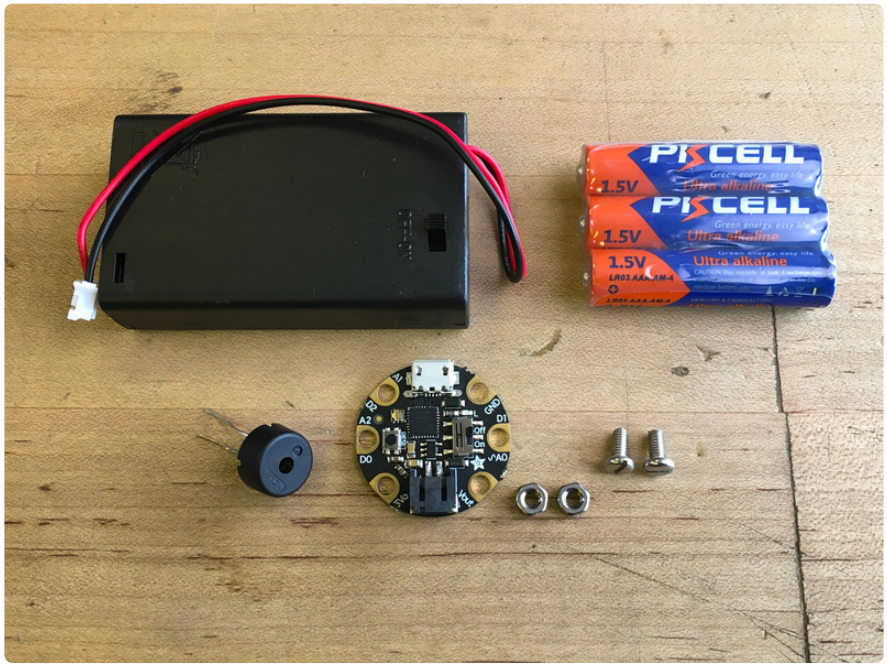
Overview



There are few things quite as maddening as beeping sounds emanating from an hidden source! Sometimes, a spy needs to engage in a bit of prank warfare, and this is the perfect tool for those battles. Enter: The Annoy-O-Matic!

The tiny Gemma M0 equipped with a piezo buzzer and battery pack is small enough to hide under a desk, inside a vase, or perhaps just above your victim in a ceiling tile...

By setting the Annoy-O-Matic to bleep out sounds, ringtones, doorbells, or crickets every half hour, your mark will be driven to distraction trying to locate it!



Parts & Materials

1 x Gemma M0 https://www.adafruit.com/product/3501
Tiny, round microcontroller

---

1 x Battery Holder with On/Off Switch and JST https://www.adafruit.com/product/727
Provides the power of 3 AAA batteries

---

1 x AAA Batteries https://www.adafruit.com/product/3520
3 pack

---

1 x Piezo Buzzer https://www.adafruit.com/product/160
Small part with a big sound

---

1 x USB Cable https://www.adafruit.com/product/898
A/MicroB cable - 6"

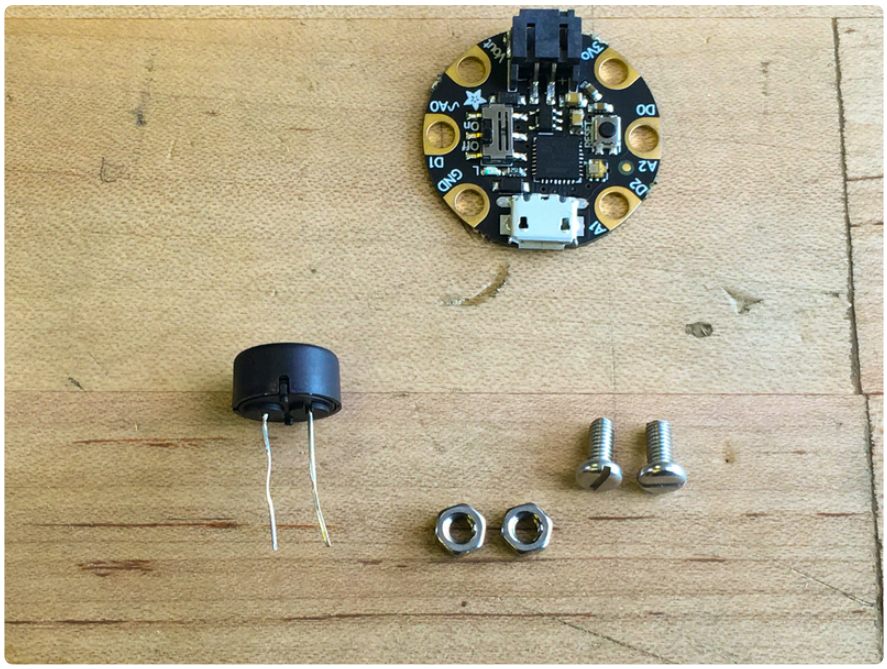
Additionally, you'll want two M3 x 8mm screws and nuts to connect the piezo buzzer to the Gemma. (You can use alligator clips if you don't have the screws and nuts.)

Optionally, if you want to stick your Annoy-O-Matic to metal objects, you can affix a rare earth magnet to the battery box, such as this one:

1 x Rare Earth magnet https://www.adafruit.com/product/9
Very strong!

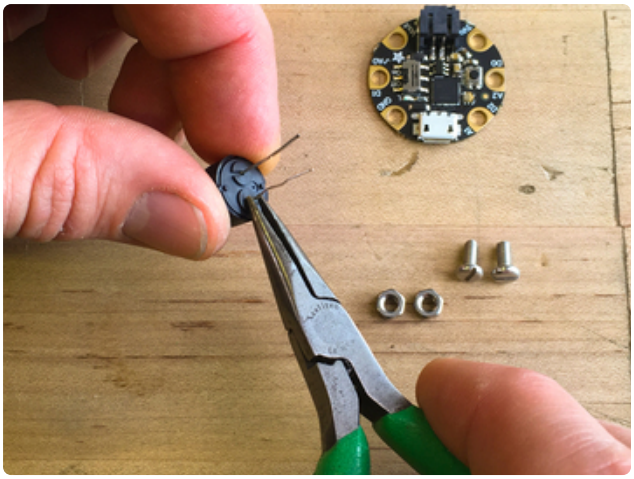
Prepare the Piezo

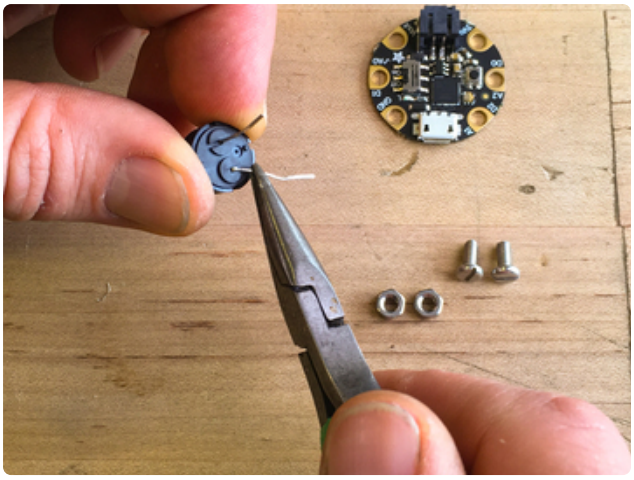
The Gemma M0 can't make any sounds on it's own, so let's connect the piezo buzzer to it so we can hear it!

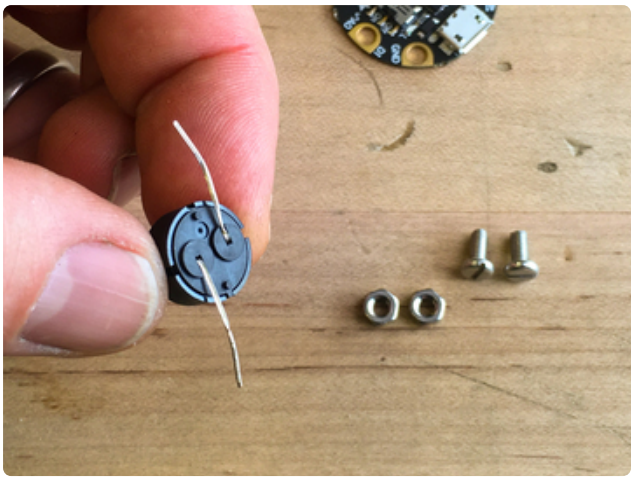


Bending Legs

We'll use the two M3 screws and nuts to connect the buzzer's legs to the **D0** and **GND** pads of the Gemma M0.







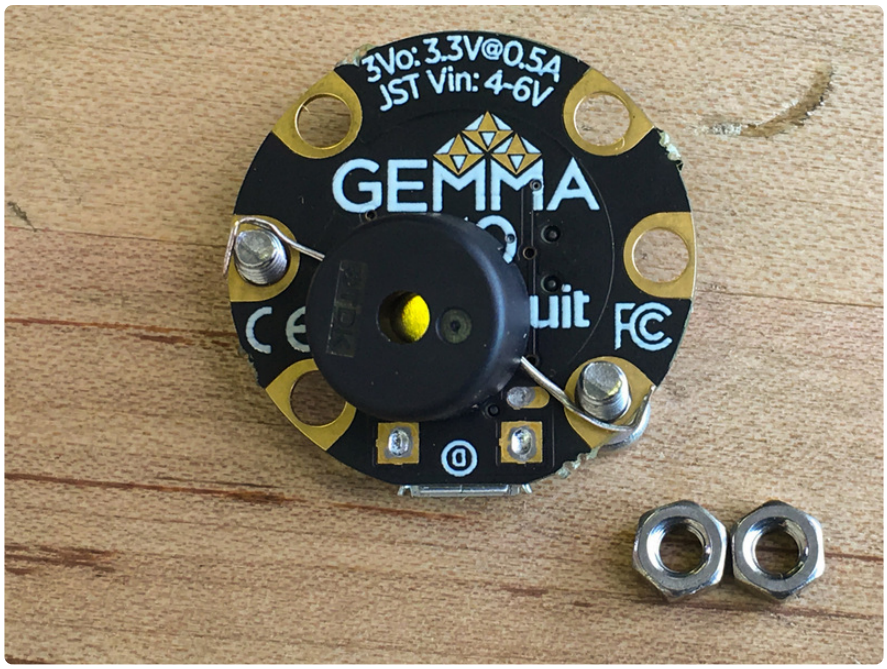
Use pliers (or your fingers) to bend the legs down, carefully, making sure not to break them off.

While the legs will bend easily to their new position, trying to move them back to their original position may be more stress than they can take.

Each leg can fit nicely into a small groove molded into the plastic case.

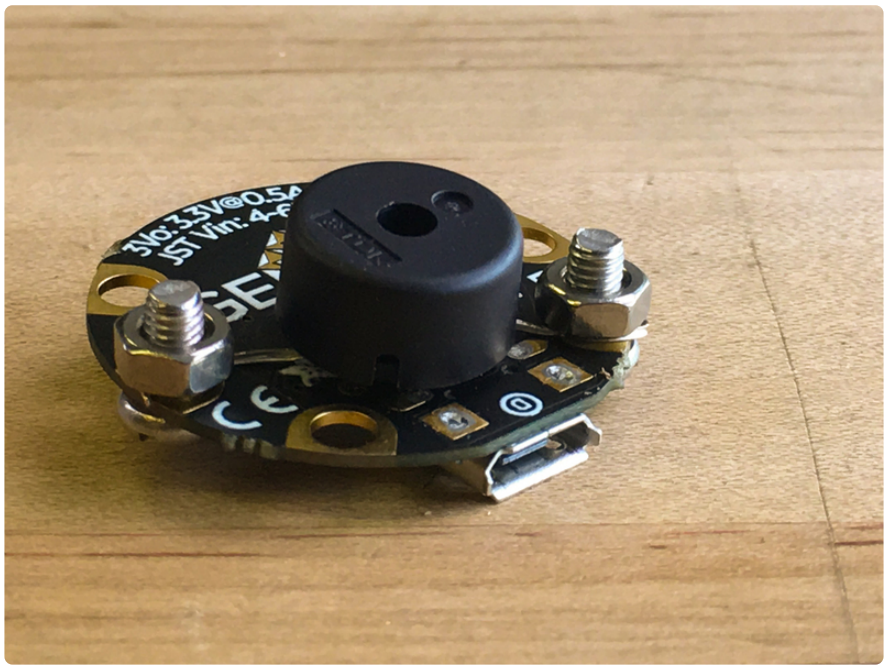
Make Feet

Use the pliers or the screws themselves to form small hook-like feet at the end of each leg, as shown here, so they will connect to **D0** and **GND.** (You can check the other side of the board to see the pad names on the silkscreen.)



It doesn’t matter which leg of the buzzer goes to which pad, it is not polarized.

Thread the nuts on and tighten them, being careful that the legs don't touch any copper pads on the board other than their respective pin assignments.



OK, now that the piezo is connected, it's time to start programming the Gemma M0 in CircuitPython!

Code the Annoy-O-Matic

CircuitPython Setup

We'll use CircuitPython (https://adafru.it/zB0) to code the Annoy-O-Matic.

First, follow this guide https://learn.adafruit.com/adafruit-gemma-m0/circuitpython (https://adafru.it/zAl) to get started with coding the Gemma M0 in CircuitPython. Install the latest release version of CircuitPython on the board. You may also want to install the Mu editor https://learn.adafruit.com/adafruit-gemma-m0/installing-mu-editor (https://adafru.it/BGP) for your coding needs.

Once you can successfully upload code from Mu to your Gemma M0, return here.

Annoy-O-Matic Code

We will only need to do a few simple things in the code, and then repeat them over and over.

Pulsio

First, we'll need to be able to create sequences of sounds of various pre-defined frequencies and durations, such as three quick, high pitched tones for a cricket sound, or the thirteen notes of the famous Nokia phone ringtone.

These tones can be played using the CircuitPython **pulsio** library, which can send PWM (pulse width modulation) signals of specific frequencies to the piezo buzzer.

Here's a great primer (https://adafru.it/Ckq) on using PWM with pulsio in CircuitPython to create tones. This is what the code looks like to play a series of notes:

```
# SPDX-FileCopyrightText: 2017 Limor Fried for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time

import board
import pwmio

piezo = pwmio.PWMOut(board.D2, duty_cycle=0,
                     frequency=440, variable_frequency=True)

while True:
    for f in (262, 294, 330, 349, 392, 440, 494, 523):
```

```
        piezo.frequency = f
        piezo.duty_cycle = 65536 // 2  # on 50%
        time.sleep(0.25)  # on for 1/4 second
        piezo.duty_cycle = 0  # off
        time.sleep(0.05)  # pause between notes
    time.sleep(0.5)
```

Functions

We'll create a number of functions in the code so that we can easily choose between different operational modes. So, **beep**, **doorbell**, **ringtone**, **crickets**, and **teen tone** will all be different pre-defined functions that can then be selected at the top of the code.

Below is the full code we'll use. You can read in the comments how the different variables at the top work. For example, by changing `annoy_mode = 3` to `annoy_mode = 2`, you'll get a doorbell sound instead of a ringtone.

The way this works, is that at the very bottom of the code the `while True:` loop runs and checks the value of `annoy_mode`. Depending on that value, one of the different functions, such as `def annoy_doorbell()` is called.

You can look at the different functions to see how they work, and play around with the values to change pitch frequencies and delay timings.

```
# SPDX-FileCopyrightText: 2018 John Edgar Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Annoy-O-Matic Sound Prank Device
# choose from a variety of sounds and timings to drive your victim bonkers
import time

import board
import pwmio

# pylint: disable=unused-variable,consider-using-enumerate,redefined-outer-name,too-
many-locals
piezo = pwmio.PWMOut(board.D0, duty_cycle=0, frequency=440,
                     variable_frequency=True)

# pick the mode here:
#  1 = beep
#  2 = doorbell
#  3 = ringtone
#  4 = crickets
#  5 = teen tone
#  6 = demo mode
annoy_mode = 3

# set general parameters here
interval = 300  # seconds before next annoyance. Default 300 (5 minutes)

# set beep details here
frequency = 5000  # pitch of the beep in Hz.              Default 5000
length = 0.2  # length (duration) of the beep in seconds. Default 0.2
rest = 0.05  # seconds between beep repeats.              Default 0.05
```

```
# set beep/cricket details here
repeat = 3  # number of times to repeat the beep/cricket. Default 3

# set ringtone details here
ringtone = 3  # song choices: 1 = Nokia, 2 = iPhone, 3 = Rickroll
ringtone_tempo = 1.6  # suggested Nokia 0.9 , iPhone 1.3 , Rickroll 2.0


def annoy_beep(frequency, length, repeat, rest, interval):
    for _ in range(repeat):
        piezo.frequency = frequency  # 2600 is a nice choice
        piezo.duty_cycle = 65536 // 2  # on 50%
        time.sleep(length)  # sound on
        piezo.duty_cycle = 0  # off
        time.sleep(rest)
    time.sleep(interval)  # wait time until next beep


def annoy_doorbell(interval):
    piezo.frequency = 740
    piezo.duty_cycle = 65536 // 2
    time.sleep(0.85)
    piezo.duty_cycle = 0
    time.sleep(0.05)
    piezo.frequency = 588
    piezo.duty_cycle = 65536 // 2
    time.sleep(1.25)
    piezo.duty_cycle = 0
    time.sleep(interval)


# pylint: disable=too-many-statements
def annoy_ringtone(ringtone, tempo, interval):
    # ringtone 1: Nokia
    # ringtone 2: Apple iPhone
    # ringtone 3: Rick Astley Never Gonna Give You Up

    # tempo is length of whole note in seconds, e.g. 1.5
    # set up time signature
    whole_note = tempo  # adjust this to change tempo of everything
    dotted_whole_note = whole_note * 1.5
    # these notes are fractions of the whole note
    half_note = whole_note / 2
    dotted_half_note = half_note * 1.5
    quarter_note = whole_note / 4
    dotted_quarter_note = quarter_note * 1.5
    eighth_note = whole_note / 8
    dotted_eighth_note = eighth_note * 1.5
    sixteenth_note = whole_note / 16

    # set up note values
    A2 = 110
    As2 = 117  # 's' stands for sharp: A#2
    Bb2 = 117
    B2 = 123

    C3 = 131
    Cs3 = 139
    Db3 = 139
    D3 = 147
    Ds3 = 156
    Eb3 = 156
    E3 = 165
    F3 = 175
    Fs3 = 185
    Gb3 = 185
    G3 = 196
    Gs3 = 208
```

```
Ab3 = 208
A3 = 220
As3 = 233
Bb3 = 233
B3 = 247

# C4 = 262
Cs4 = 277
# Db4 = 277
D4 = 294
# Ds4 = 311
# Eb4 = 311
E4 = 330
# F4 = 349
Fs4 = 370
# Gb4 = 370
G4 = 392
# Gs4 = 415
# Ab4 = 415
# A4 = 440
# As4 = 466
# Bb4 = 466
B4 = 494

C5 = 523
Cs5 = 554
Db5 = 554
D5 = 587
Ds5 = 622
Eb5 = 622
E5 = 659
F5 = 698
Fs5 = 740
Gb5 = 740
G5 = 784
Gs5 = 831
Ab5 = 831
A5 = 880
As5 = 932
Bb5 = 932
B5 = 987

# here's another way to express the note pitch, double the previous octave
C6 = C5 * 2
Cs6 = Cs5 * 2
Db6 = Db5 * 2
D6 = D5 * 2
Ds6 = Ds5 * 2
Eb6 = Eb5 * 2
E6 = E5 * 2
F6 = F5 * 2
Fs6 = Fs5 * 2
Gb6 = Gb5 * 2
G6 = G5 * 2
Gs6 = Gs5 * 2
Ab6 = Ab5 * 2
A6 = A5 * 2
As6 = As5 * 2
Bb6 = Bb5 * 2
B6 = B5 * 2

if ringtone == 1:
    # Nokia
    nokia_ringtone = [[E6, eighth_note], [D6, eighth_note],
                      [Fs5, quarter_note], [Gs5, quarter_note],
                      [Cs6, eighth_note], [B5, eighth_note],
                      [D5, quarter_note], [E5, quarter_note],
                      [B5, eighth_note], [A5, eighth_note],
                      [Cs5, quarter_note], [E5, quarter_note],
```

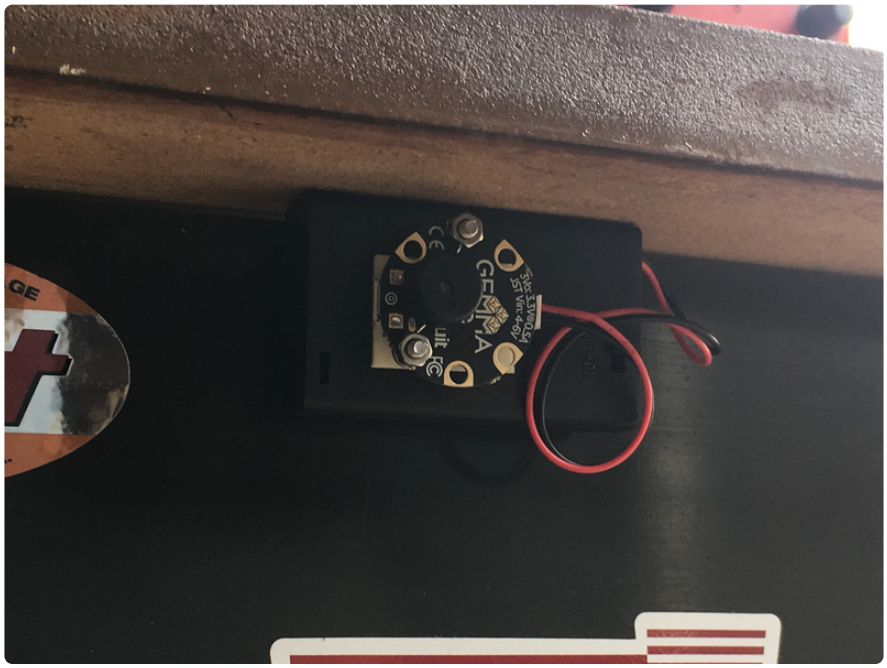
```
                          [A5, whole_note]]

        for n in range(len(nokia_ringtone)):
            piezo.frequency = (nokia_ringtone[n][0])
            piezo.duty_cycle = 65536 // 2  # on 50%
            time.sleep(nokia_ringtone[n][1])  # note duration
            piezo.duty_cycle = 0  # off
            time.sleep(0.01)

    if ringtone == 2:
        # iPhone Marimba
        iPhone_ringtone = [[B4, eighth_note], [G4, eighth_note],
                           [D5, eighth_note], [G4, eighth_note],
                           [D5, eighth_note], [E5, eighth_note],
                           [D5, eighth_note], [G4, eighth_note],
                           [E5, eighth_note], [D5, eighth_note],
                           [G4, eighth_note], [D5, eighth_note]]

        for n in range(len(iPhone_ringtone)):
            piezo.frequency = (iPhone_ringtone[n][0])
            piezo.duty_cycle = 65536 // 2  # on 50%
            time.sleep(iPhone_ringtone[n][1])  # note duration
            piezo.duty_cycle = 0  # off
            time.sleep(0.01)

    if ringtone == 3:
        # Rickroll
        rick_ringtone = [[A3, sixteenth_note], [B3, sixteenth_note],
                         [D4, sixteenth_note], [B3, sixteenth_note],
                         [Fs4, dotted_eighth_note], [Fs4, sixteenth_note],
                         [Fs4, eighth_note], [E4, eighth_note],
                         [E4, quarter_note],
                         [A3, sixteenth_note], [B3, sixteenth_note],
                         [Cs4, sixteenth_note], [A3, sixteenth_note],
                         [E4, dotted_eighth_note], [E4, sixteenth_note],
                         [E4, eighth_note], [D4, eighth_note],
                         [D4, sixteenth_note], [Cs4, sixteenth_note],
                         [B3, eighth_note]]

        for n in range(len(rick_ringtone)):
            piezo.frequency = (rick_ringtone[n][0])
            piezo.duty_cycle = 65536 // 2  # on 50%
            time.sleep(rick_ringtone[n][1])  # note duration
            piezo.duty_cycle = 0  # off
            time.sleep(0.035)

    time.sleep(interval)


def annoy_crickets(repeat, interval):
    for _ in range(repeat):
        for _ in range(6):
            piezo.frequency = 8000  # 2600 is a nice choice
            piezo.duty_cycle = 65536 // 2  # on 50%
            time.sleep(0.02)  # sound on
            piezo.duty_cycle = 0  # off
            time.sleep(0.05)
        time.sleep(0.2)
    time.sleep(interval)  # wait time until next beep


def annoy_teen_tone(interval):
    piezo.frequency = 17400
    piezo.duty_cycle = 65536 // 2  # on 50%
    time.sleep(10)
    piezo.duty_cycle = 0
    time.sleep(interval)
```

```
while True:
    if annoy_mode == 1:
        annoy_beep(frequency, length, repeat, rest, interval)
    elif annoy_mode == 2:
        annoy_doorbell(interval)
    elif annoy_mode == 3:
        annoy_ringtone(ringtone, ringtone_tempo, interval)
    elif annoy_mode == 4:
        annoy_crickets(repeat, interval)
    elif annoy_mode == 5:
        annoy_teen_tone(interval)
    elif annoy_mode == 6:
        annoy_beep(5000, 0.2, 2, 0.05, 3)
        annoy_doorbell(3)
        annoy_ringtone(1, 0.9, 3)
        annoy_ringtone(2, 1.3, 3)
        annoy_ringtone(3, 2.0, 3)
        annoy_crickets(3, 3)
        annoy_teen_tone(6)
```

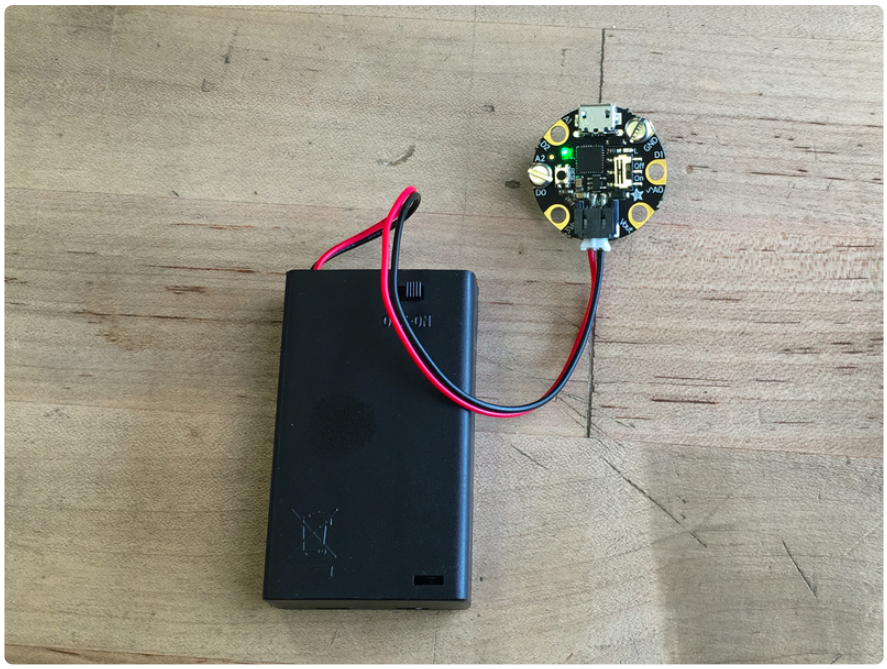
Copy the full code, and paste it into Mu. Then, save the code as **main.py** onto your Gemma M0. The board will reboot and you'll hear your Annoy-O-Matic play its first sound! After that first instance, the device will wait five minutes before playing again.

Of course, you can change the delay by adjusting the value of the `interval` variable at the top. The `interval` value is in seconds, so if you want to have it wait for half an hour, the value would be 30 minutes * 60 seconds, or 1800 seconds.

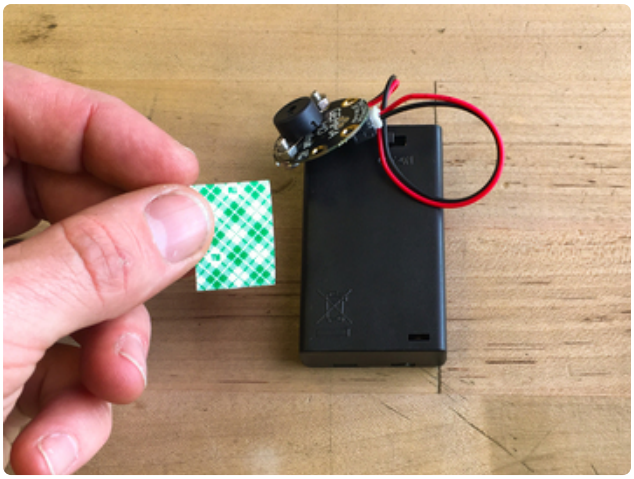
Deploy the Annoy-O-Matic

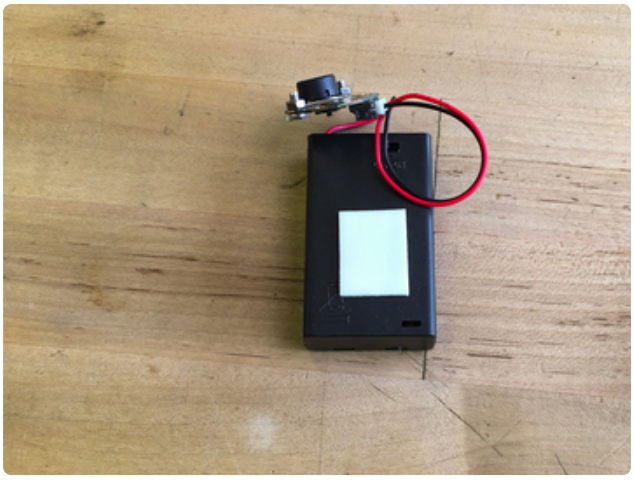


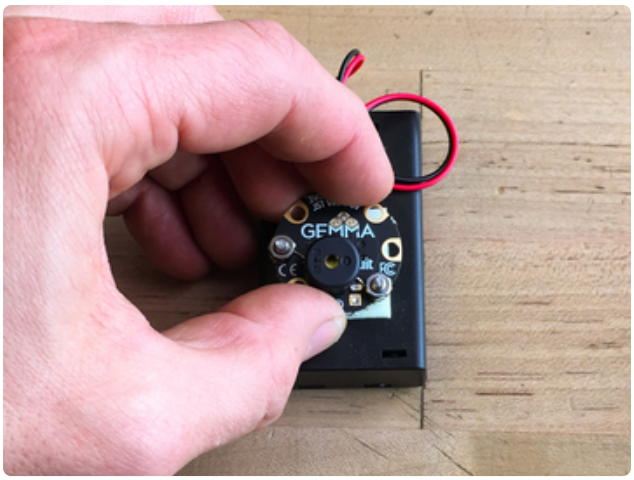
Your Annoy-O-Matic is coded and ready to be deployed! Plug in the AAA battery pack. Test it out by turning on the battery pack's ON/OFF switch. Your first chosen sound or song will play.



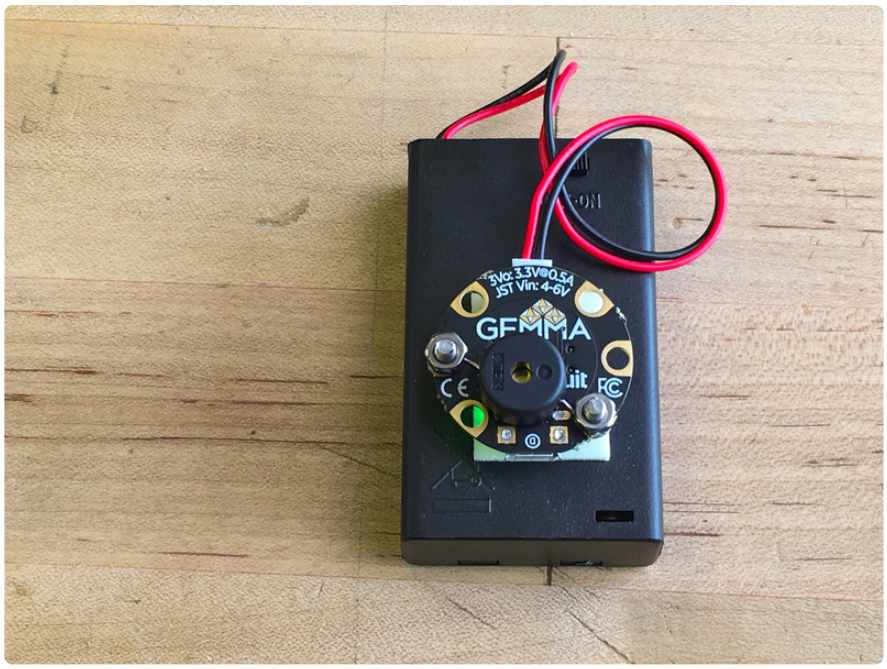
In order to neaten up the package a bit, you can use double stick foam tape to attach the Gemma M0 to your battery box.





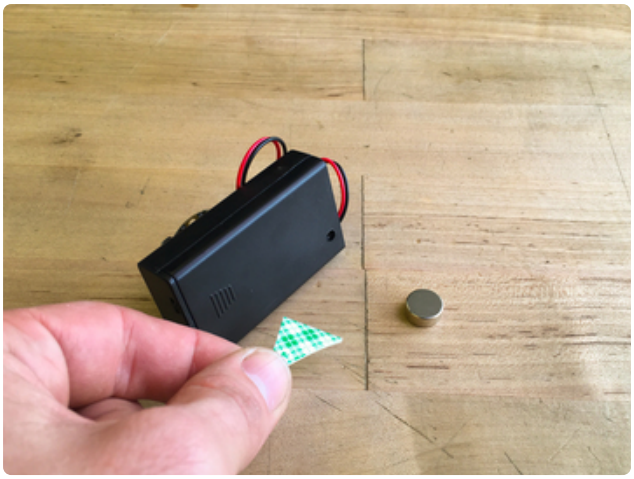


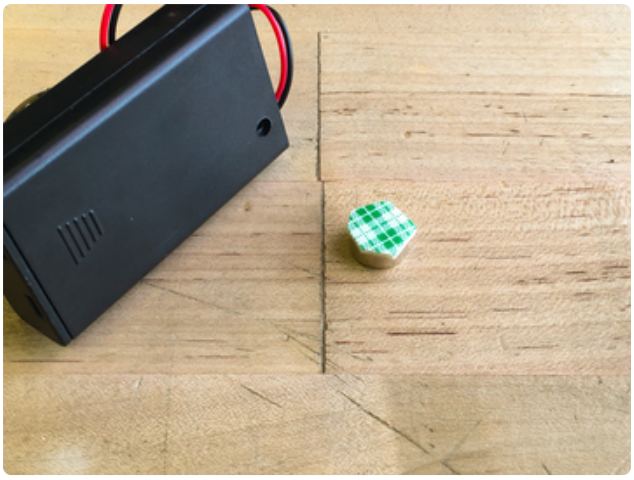
In order to neaten up the package a bit, you can use double stick foam tape to attach the Gemma M0 to your battery box Make sure to keep the On/Off switch accessible

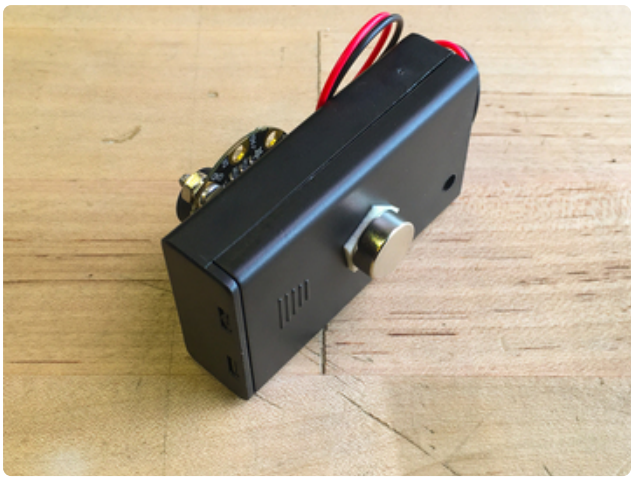


Magnetic Attraction

You can hide the Annoy-O-Matic in a drawer or box as it is -- but if you want to get even more sneaky, add a magnet so it can be surreptitiously attached to metal objects, such as the underside of a desk...







Use another piece of double-stick foam tape to adhere the magnet to the battery box
You can place the tape on the magnet and then trim the tape to fit
Press the other side of the tape firmly to the battery case

Deploy the Annoy-O-Matic!

Turn on the power switch and listen to the first instance of the sound play -- it is now armed and ready to go off again in five minutes, or whatever you set for your interval time. The longer the interval, the harder it will be for them to figure out the source of their annoyance!!

Hurry up and hide the Annoy-O-Matic somewhere within earshot of your victim and wait for the fun to begin!