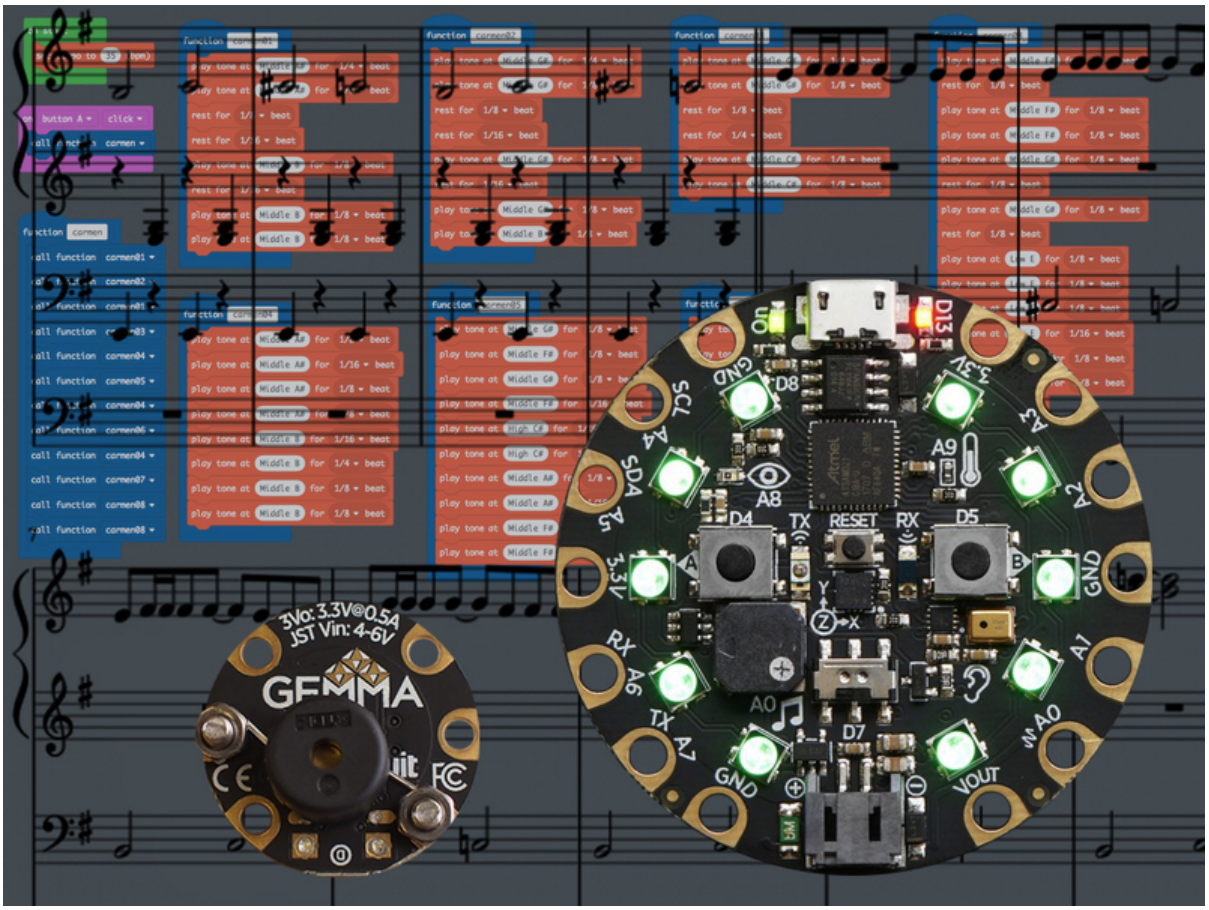
Spy Theme Playback Device

Created by John Park

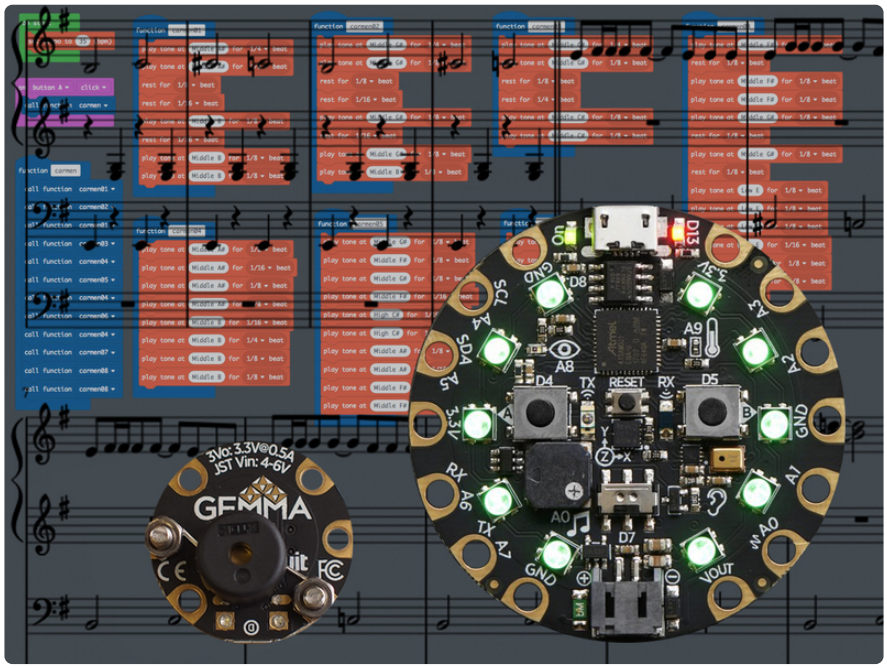


https://learn.adafruit.com/spy-theme-playback-device

Last updated on 2025-06-29 10:26:38 AM EDT

Table of Contents

Overview



When planning and gearing up for a secret op, every good agent needs some motivational spy music to set the tone! (Get it? Tone? Sorry, not sorry!)

With a Circuit Playground Express and its built-in speaker you can make your spy themes shine. Programming songs using MakeCode is fun! You can write your own tunes from scratch, or transcribe music.

On Gemma M0 with an added piezo buzzer, we'll look at using CircuitPython to write music as well!

1 x Circuit Playground Express
https://www.adafruit.com/product/3333
The microcontroller with the works

---

1 x Gemma M0
https://www.adafruit.com/product/3501
Tiny microcontroller with a kick

---

1 x Piezo Buzzer
https://www.adafruit.com/product/160
Petite but loude

---

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

---

1 x 3 x AAA Battery Holder
https://www.adafruit.com/product/727
w on/off switch and 2-pin JST

---

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

You'll also use two M3 x 8mm screws and nuts to connect the piezo to the Gemma M0.

Code Music with MakeCode for Circuit Playground Express

First, make sure you're familiar with setting up the Circuit Playground Express for use with MakeCode by following this guide. https://learn.adafruit.com/makecode (https://adafru.it/wWd)

Once you can successfully create a program with MakeCode and upload it to your board, return here.

You can create songs very easily in MakeCode, by stringing together a series of play (note) for (time) blocks and rest for (time) blocks. For a simple, short melody this works very well, for example, in the embedded MakeCode session here, press the left button on the simulator to hear it play.

Notice how there's a phrase that's repeated twice in that example? I can get pretty tedious to manually enter those same notes and durations over and over again -- so we can instead, turn that one phrase into a reusable mini program within a program! That's what a function is -- a chunk of code that can be referenced easily elsewhere in our program.

In this example, one set of the notes that make up the phrase have been moved into a **function** block (**functions** are found in the A**dvanced** section). You can click **Make a Function** to create a **function** block -- give it a good, memorable name, in this case **'phrase01'.**

Then, you can use the **call function phrase01** block wherever you want to repeat the phrase.

Here's what it looks like when we add a couple more phrases and then start to sequence them in the **on button A click** block.

Notice how we've been building the song inside the **on button A click** block. If we wanted to also play the song when right button (button B) is pressed, for example, we would have to copy the entire set of blocks and repeat it in two places. This works, but there's a more efficient way -- that's to build another **function** block that itself is filled with **function** blocks!

The new function could be named **song01** and it is full of calls to the other functions that are the phrases that make up the roadmap for the song.

Here's the full song -- notice how the song function contains many repetitions of the phrases to compose the entire song.

Here's another favorite -- the theme song from **Where in the World is Carmen Sandiego**.

Here's the beginning of the **Mission: Impossible** theme. Can you add to it? Maybe make it loop a few more times on button A click?

You can have fun creating other favorite spy themes, such as:

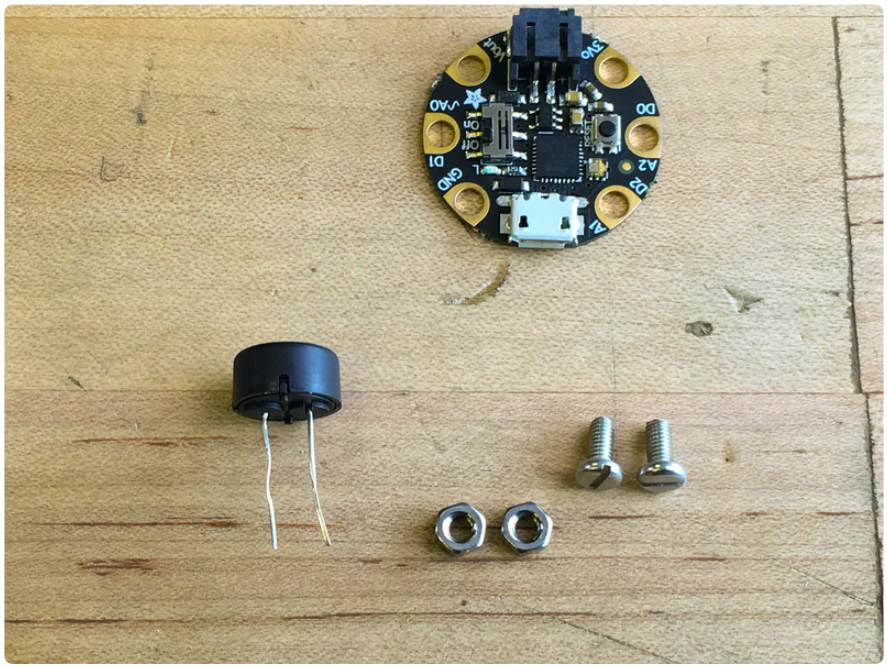
- Get Smart
- Bloodhound Gang
- Pink Panther

Or, really personalize it and write your own cool theme!

Building the Gemma M0 Spy Theme Player

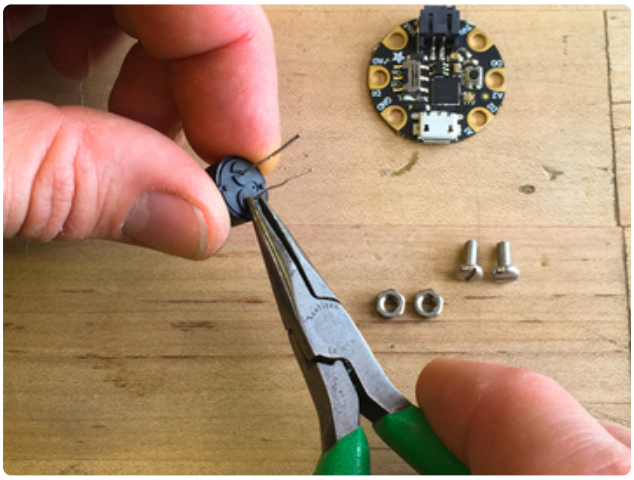
Assembly

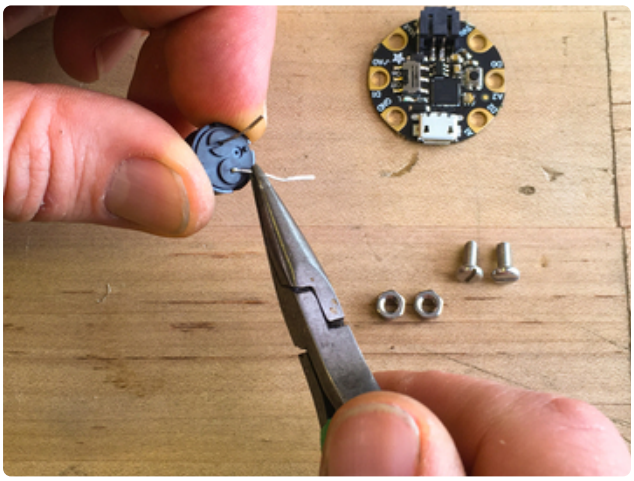
The Gemma M0 can't make any sounds on it's own, so let's connect the piezo buzzer to it so we can hear it!



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.



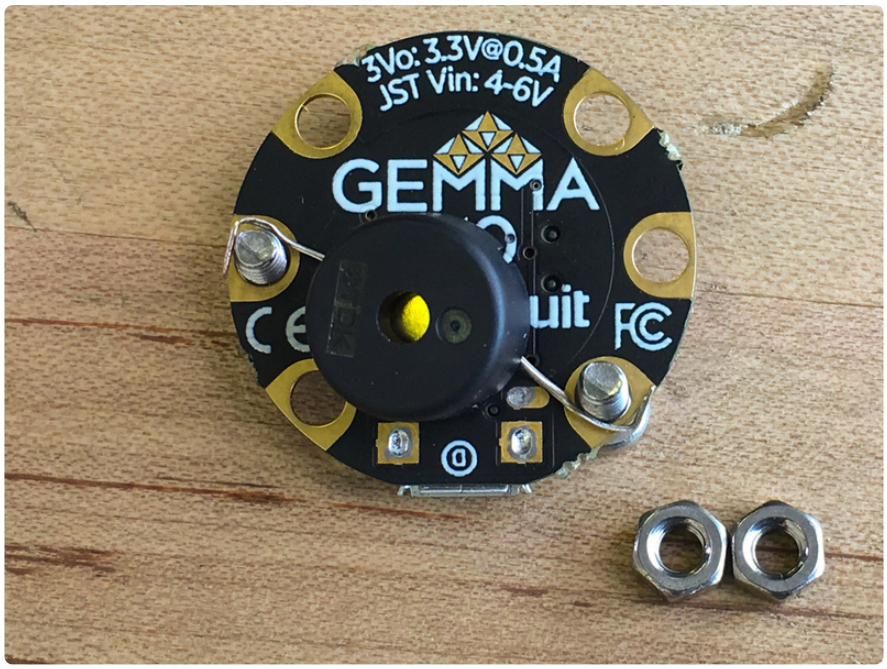


Use pliers (or your fingers) to bend the legs down, carefully, making sure not to break them off.

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.

Next, we'll code the Gemma M0 to play spy themes using CircuitPython.

Coding the Gemma M0 with CircuitPython

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 code in Mu and upload to the board, return here.

Ringtones to the Rescue

If you want to write short little ditties, there's an established format we can use: RTTL (Ring Tone Text Transfer Language) that was originally developed by Nokia for cellphone ringtones.

We've written a library to make it easy to use RTTTL in CircuitPython. You can download the library as part the CircuitPython Bundle, and then drag a copy of the **adafruit_rtttl** into your Gemma M0 **lib** folder. (Check this page (https://adafru.it/BGO) if you have questions about CircuitPython libraries.)

Click for the Latest Adafruit CircuitPython Library Bundle Release

https://adafru.it/y8E

Then, to test it out and hear a familiar spy theme, copy the code below, paste it into Mu, and then save it to your **Gemma M0** as **main.py** (if you want to run this code on a **Circuit Playground Express**, see below for alternate code.)

```
import adafruit_rtttl
import board

adafruit_rtttl.play(board.A2, "Bond:d=4,o=5,b=320:c,8d,8d,d,2d,c,c,c,c,8d#,8d#,
2d#,d,d,d,c,8d,8d,d,2d,c,c,c,c,8d#,8d#,d#,2d#,d,c#,c,c6,1b.,g,f,1g.")
```

You can see that there isn't too much to it -- we import the rtttl library, and then we can use the `adafruit_rtttl.play()` command. This command takes three elements: name, settings, and notes, all separated by a colon `:`

The song name element in this case is `Bond`.

The settings elements specify:

- `d` for duration of a default note, in this case a `4` means a quarter note. This is a convenience that allows you to not specify the duration of all the quarter notes.
- `o` for the default octave of the song, the range is `4` to `7`. Here it is set to `5`
- `b` is the tempo ("beat") of the song in BPM (beats per minute) in this case `320`

Then we get to the final element, the notes. So, for the first few notes of the Bond theme intro we have `c,8d,8d,d,2d,c,c,c,c` which is a C quarter note, two D eighth notes, a D quarter note, a D half note and four C quarter notes. Note, you can also use # for sharps.

Try writing your own tunes, or search online for ringtones published in the RTTTL format.

Here's another favorite:

```
import adafruit_rtttl
import board

adafruit_rtttl.play(board.D0, "The A Team:d=8,o=5,b=132:4d#6,a#,2d#6,16p,g#,4a#,
4d#.,p,16g,16a#,d#6,a#,f6,2d#6,16p,c#.6,16c6,16a#,g#.,2a#.")
```

With some small changes, you can run this code on a Circuit Playground Express using the on-board amp and speaker! It'll use a lovely sine wave instead of the PWM square wave. Try the code below to do so.

```
# rtttl example for Circuit Playground Express
import board
from digitalio import DigitalInOut, Direction
import adafruit_rtttl
spkrenable = DigitalInOut(board.SPEAKER_ENABLE)
spkrenable.direction = Direction.OUTPUT
spkrenable.value = True
adafruit_rtttl.play(board.A0, "Bond:d=4,o=5,b=320:c,8d,8d,d,2d,c,c,c,c,8d#,8d#,
2d#,d,d,d,c,8d,8d,d,2d,c,c,c,c,8d#,8d#,d#,2d#,d,c#,c,c6,1b.,g,f,1g.")
```

Longer Compositions

The RTTTL format works great for short songs -- after all, that's why it was originally designed. Longer songs, however, can get a bit messy if we simply string together note after note after note after note! Even traditional sheet music notation uses repeat signs and codas to avoid writing out one long, linear song!

So, we'll create a number of lists that each contain phrases or measures of the song that are repeated throughout, then we can reference those lists in a larger roadmap for the whole song.

tone Test

First, we'll use the built-in `pwmio` function to play some test tones over the piezo buzzer connected to **D0** and **GND**.

Copy the code below, paste it into Mu and then save it to your Gemma M0 to test out the tone example.

```
# SPDX-FileCopyrightText: 2018 Kattni Rembor for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""CircuitPython Essentials PWM with variable frequency piezo example"""
import time
import board
import pwmio

# For the M0 boards:
piezo = pwmio.PWMOut(board.A2, duty_cycle=0, frequency=440, variable_frequency=True)

# For the M4 boards:
# piezo = pwmio.PWMOut(board.A1, 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 = 65535 // 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)
```

In the first example, we have the James Bond 007 Theme. Note how we create lists for each section -- Bond01, Bond02, and so on -- and then call them at the bottom of the code in the `while True:` loop. See how the Bond01 section repeats twice, then the Bond02 and Bond03 pairings repeat twice more before, calling in Bond04. The opening phrase is again repeated a number of times right before the songs final passage.

First, we set up the `pwmio` output as before.

Then, we create a variable called `tempo` to define the length of a whole note, in this case 2 seconds. You can adjust that to increase or decrease the tempo. All note lengths are derived from this one variable, e.g. `whole_note` is equal to `tempo`, `half_note` is a `whole_note * 0.5` and so on.

Similarly, we create a series of variables to define the pitches different notes, starting from A2 (110Hz) up to B6 (1974Hz). This way, we can call the pitches with a note name instead of a frequency value. This makes it much easier to transcribe from standard music notation!

Here's the James Bond 007 Theme. Copy it and paste it into Mu, then save it to your Gemma M0 as **main.py**

It will play through once -- if you want to repeat it, simply press reset, or turn the board off and then on.

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

"""
Plays the 007 theme song
Gemma M0 with Piezo on D0 and GND
"""

import time

import board
import pwmio

piezo = pwmio.PWMOut(board.D0, duty_cycle=0, frequency=440,
                     variable_frequency=True)

tempo = 2

# 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

rst = 24000  # rest is just a tone out of normal hearing range

Bond01 = [[B3, half_note],
          [C4, half_note],
          [Cs4, half_note],
          [C4, half_note]]

Bond02 = [[E3, eighth_note],
          [Fs3, sixteenth_note],
          [Fs3, sixteenth_note],
          [Fs3, eighth_note],
          [Fs3, eighth_note],
          [Fs3, eighth_note],
          [E3, eighth_note],
          [E3, eighth_note],
          [E3, eighth_note]]

Bond03 = [[E3, eighth_note],
          [G3, sixteenth_note],
          [G3, sixteenth_note],
          [G3, eighth_note],
          [G3, eighth_note],
          [G3, eighth_note],
          [Fs3, eighth_note],
          [Fs3, eighth_note],
          [Fs3, eighth_note]]
```

```
Bond04 = [[E3, eighth_note],
          [G3, sixteenth_note],
          [G3, sixteenth_note],
          [G3, eighth_note],
          [G3, eighth_note],
          [G3, eighth_note],
          [Fs3, eighth_note],
          [Fs3, eighth_note],
          [E3, eighth_note]]

Bond05 = [[Ds4, eighth_note],
          [D4, eighth_note],
          [D4, half_note],
          [B3, eighth_note],
          [A3, eighth_note],
          [B3, whole_note]]

Bond06 = [[E4, eighth_note],
          [G4, quarter_note],
          [Ds5, eighth_note],
          [D5, quarter_note],
          [D5, eighth_note],
          [G4, eighth_note],
          [As4, eighth_note],
          [B4, eighth_note],
          [B4, half_note],
          [B4, quarter_note]]

Bond07 = [[G4, quarter_note],
          [A4, sixteenth_note],
          [G4, sixteenth_note],
          [Fs4, quarter_note],
          [Fs4, eighth_note],
          [B3, eighth_note],
          [E4, eighth_note],
          [Cs4, eighth_note],
          [Cs4, whole_note]]

Bond08 = [[G4, quarter_note],
          [A4, sixteenth_note],
          [G4, sixteenth_note],
          [Fs4, quarter_note],
          [Fs4, eighth_note],
          [B3, eighth_note],
          [Ds4, eighth_note],
          [E4, eighth_note],
          [E4, whole_note]]

Bond09 = [[E4, eighth_note],
          [E4, quarter_note],
          [E4, eighth_note],
          [Fs4, eighth_note],
          [Fs4, sixteenth_note],
          [E4, eighth_note],
          [Fs4, quarter_note]]

Bond10 = [
    [G4, eighth_note],
    [G4, quarter_note],
    [G4, eighth_note],
    [Fs4, eighth_note],
    [Fs4, sixteenth_note],
    [G4, eighth_note],
    [Fs4, quarter_note]]

Bond11 = [[B4, eighth_note],
          [B4, eighth_note],
          [rst, eighth_note],
```

```
           [B3, eighth_note],
           [B3, quarter_note],
           [B4, eighth_note],
           [B4, eighth_note],
           [rst, eighth_note],
           [B3, eighth_note],
           [B3, quarter_note],
           [B4, sixteenth_note],
           [B4, eighth_note],
           [B4, sixteenth_note],
           [B4, eighth_note],
           [B4, eighth_note]]

Bond12 = [[E3, eighth_note],
           [G3, quarter_note],
           [Ds4, eighth_note],
           [D4, quarter_note],
           [G3, eighth_note],
           [B3, quarter_note],
           [Fs4, eighth_note],
           [F4, quarter_note],
           [B3, eighth_note],
           [D4, quarter_note],
           [As4, eighth_note],
           [A4, quarter_note],
           [F4, eighth_note],
           [A4, quarter_note],
           [Ds5, eighth_note],
           [D5, quarter_note],
           [rst, eighth_note],
           [rst, quarter_note],
           [Fs4, whole_note]]


def song_playback(song):
    for note in song:
        piezo.frequency = (note[0])
        piezo.duty_cycle = 65536 // 2  # on 50%
        time.sleep(note[1])  # note duration
        piezo.duty_cycle = 0  # off
        time.sleep(0.01)


# this plays the full song roadmap
song_playback(Bond01)
song_playback(Bond01)
song_playback(Bond02)
song_playback(Bond03)
song_playback(Bond02)
song_playback(Bond03)
song_playback(Bond02)
song_playback(Bond04)
song_playback(Bond05)
song_playback(Bond06)
song_playback(Bond07)
song_playback(Bond06)
song_playback(Bond08)
song_playback(Bond09)
song_playback(Bond10)
song_playback(Bond09)
song_playback(Bond10)
song_playback(Bond11)
song_playback(Bond01)
song_playback(Bond01)
song_playback(Bond01)
song_playback(Bond01)
song_playback(Bond05)
song_playback(Bond12)
```

Here's another fun one -- Where in the World is Carmen Sandiego:

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

"""
Plays the Carmen Sandiego theme song
Gemma M0 with Piezo on D0 and GND
"""

import time

import board
import pwmio

piezo = pwmio.PWMOut(board.D0, duty_cycle=0, frequency=440,
                     variable_frequency=True)

tempo = 1.6
# 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

rst = 24000  # rest is just a tone out of normal hearing range

carmen01 = [[As4, quarter_note],
            [As4, eighth_note],
            [rst, eighth_note],
            [rst, sixteenth_note],
            [B4, eighth_note],
            [rst, sixteenth_note],
            [B4, eighth_note],
            [B4, eighth_note]]

carmen02 = [[Gs4, quarter_note],
            [Gs4, eighth_note],
            [rst, eighth_note],
            [rst, sixteenth_note],
            [Gs4, eighth_note],
            [rst, sixteenth_note],
            [Gs4, eighth_note],
            [B4, eighth_note]]

carmen03 = [[Gs4, quarter_note],
            [Gs4, eighth_note],
            [rst, eighth_note],
            [rst, quarter_note],
            [Cs4, eighth_note],
```

```
            [Cs4, eighth_note]]

carmen04 = [[As4, eighth_note],
            [As4, sixteenth_note],
            [As4, eighth_note],
            [As4, eighth_note],
            [B4, sixteenth_note],
            [B4, quarter_note],
            [B4, eighth_note],
            [B4, eighth_note]]

carmen05 = [[Gs4, eighth_note],
            [Fs4, eighth_note],
            [Gs4, eighth_note],
            [Fs4, sixteenth_note],
            [Cs5, sixteenth_note],
            [Cs5, sixteenth_note],
            [As4, eighth_note],
            [As4, sixteenth_note],
            [Fs4, eighth_note],
            [Fs4, eighth_note]]

carmen06 = [[Gs4, eighth_note],
            [Fs4, eighth_note],
            [Gs4, eighth_note],
            [Fs4, sixteenth_note],
            [Gs4, sixteenth_note],
            [Gs4, sixteenth_note],
            [Gs4, eighth_note],
            [rst, eighth_note],
            [rst, eighth_note],
            [Fs4, eighth_note]]

carmen07 = [[Gs4, eighth_note],
            [Fs4, eighth_note],
            [Gs4, eighth_note],
            [Fs4, sixteenth_note],
            [Cs5, sixteenth_note],
            [Cs5, sixteenth_note],
            [As4, eighth_note],
            [As4, sixteenth_note],
            [Gs4, eighth_note],
            [Fs4, eighth_note]]

carmen08 = [[Fs4, eighth_note],
            [rst, eighth_note],
            [Fs4, eighth_note],
            [Fs4, eighth_note],
            [Gs4, eighth_note],
            [rst, eighth_note],
            [Gs4, eighth_note],
            [rst, eighth_note],
            [E3, eighth_note],
            [E3, eighth_note],
            [E3, eighth_note],
            [E3, sixteenth_note],
            [Fs3, eighth_note],
            [Fs3, eighth_note],
            [rst, quarter_note]]


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

```
song_playback(carmen01)
song_playback(carmen02)
song_playback(carmen01)
song_playback(carmen03)
song_playback(carmen04)
song_playback(carmen05)
song_playback(carmen04)
song_playback(carmen06)
song_playback(carmen04)
song_playback(carmen07)
song_playback(carmen08)
song_playback(carmen08)
```