NeoTrellis Tabletop RPG Soundboard

Created by Dave Astels

https://learn.adafruit.com/neotrellis-dungeon-crawl-soundboard

Last updated on 2024-06-03 02:33:36 PM EDT

Table of Contents

Overview 3

- This Project
- Parts

Sound Files and Customization 5

- The soundboard.txt Customization File
- Getting Sounds
- No Mixing & Matching Mono and Stereo Files, Please
- Double Check: File Locations

Code and Use 8

- Navigating the NeoTrellis
- Download Code
- Use

Code Walkthrough 13

- Code Walkthrough

Troubleshooting 16

- Troubleshooting

Overview

Creating the right atmosphere is incredibly important when running a tabletop role playing game (RPG). A rich, immersive atmosphere can take a game from one of tactics and statistics to a lively adventure.

There are many aspects to creating that atmosphere: good maps, storytelling, improve, and a little voice acting all contribute. Sound, both ambient and triggered (e.g. the wind whistling through a twisty mine tunnel and the roar of a dragon as it attacks) can go a long way to making the experience immersive for players.

A sound board is an easy way to have such sounds at your fingertips.

The NeoTrellis M4 is a great base on which to build a soundboard. We've seen a few projects along these lines.

This Project

This projects is a bit different in that it takes advantage of some of the new work on the audio module that gives CircuitPython the ability to use a mixer to play multiple sounds files at the same time.

This is important for a tabletop RPG soundboard. We typically want to have a background loop to set the mood. Something with echoes and drips for exploring an abandoned mine, or a jolly tavern full of happy patrons to relax (or not) in, after that mine.

On top of the background, it is good to have assorted ambient sounds (a gurgling stream cutting across the floor or coins being spilled on the table to pay for food or information), events (a trap being sprung or the door being thrown open to admit a group of ruffians), and combat sounds (an attacking swarm of rats or the clashing of swords).

This project differs from previous ones in that it doesn't use a naming scheme to define what files there are and what button they correspond to, and what color to use. That information is stored in a file. This lets you tweak the soundboard arrangement by simply editing a single file, and give the sound files meaningful names.

Parts

Adafruit NeoTrellis M4 with Enclosure and Buttons Kit Pack

So you've got a cool/witty name for your band, a Soundcloud account, a 3D-printed Daft Punk...

https://www.adafruit.com/product/4020

USB Powered Speakers

Add some extra boom to your audio project with these powered loudspeakers. We sampled half a dozen different models to find ones with a good frequency response, so you'll get...

https://www.adafruit.com/product/1363

You will also need power for the NeoTrellisM4 as well as the speakers if you use them. Here are some options. The large battery pack will power both the NeoTrellis and the speakers for portability.

5V 2.5A Switching Power Supply with 20AWG MicroUSB Cable

Our all-in-one 5V 2.5 Amp + MicroUSB cable power adapter is the perfect choice for powering single-board computers like Raspberry Pi, BeagleBone, or anything else that's...

https://www.adafruit.com/product/1995

USB cable - USB A to Micro-B

This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...

https://www.adafruit.com/product/592

USB Battery Pack for Raspberry Pi - 10000mAh - 2 x 5V outputs

A large-sized rechargeable battery pack for your Raspberry Pi (or Arduino, or

https://www.adafruit.com/product/1566

Sound Files and Customization

The **soundboard.txt** Customization File

This file should be placed in the root directory of the **CIRCUITPY** flash drive that appears when you plug your NeoTrellis into your computer via a USB cable. The file defines what sounds are available: which button on the NeoTrellis is used and the chosen color of that button.

Each line is made up of a filename and a color, separated by a comma. Color names correspond to those in the **color_names.py** file. Alternatively, you can use a hex color constant, e.g. `0x00FFFF`, or a color tuple, e.g. `(128, 255, 0)`. Notice the three ways to specify the color blue at the bottom of the example file below. Blank lines and comments (lines starting with `#` are ignored. Spaces around the separating commas are also ignored. If a button has no sound associated with it, it's line is simply `pass`.

Below is an example **soundboard.txt** file. The corresponding wav files are in the **samples** directory when you unzip the project download (see next page).

```
Temporarily unable to load content:
```

Sound clips have been trimmed and edited from the following free-to-use audio files:

From tabletopaudio.com (https://adafru.it/D4J)

- 65_Dungeon_I.mp3 (https://adafru.it/Div)
- 37_Catacombs.mp3 (https://adafru.it/Diw)
- drips1_lp.ogg (https://adafru.it/Dix)
- stream.ogg (https://adafru.it/Diy)
- bats.ogg (https://adafru.it/Diz)
- flies.ogg (https://adafru.it/DiA)
- screams.ogg (https://adafru.it/DiB)
- rumble.ogg (https://adafru.it/DiC)
- roar.ogg (https://adafru.it/DiD)
- creature1.ogg (https://adafru.it/DiE)
- humanoid_attack_x4.ogg (https://adafru.it/DiF)
- goblin_attack_x4.ogg (https://adafru.it/DiG)
- large_beast_attack.ogg (https://adafru.it/DiH)
- elemental_attack_x3.ogg (https://adafru.it/DiI)
- construct_attack_x3.ogg (https://adafru.it/DiJ)
- dragon_attack_x3.ogg (https://adafru.it/DiK)
- door_open.ogg (https://adafru.it/DiL)
- door_close.ogg (https://adafru.it/DiM)
- trap_arrow.ogg (https://adafru.it/DiN)
- trap_gate.ogg (https://adafru.it/DiO)
- magic_trap.ogg (https://adafru.it/DiP)
- trap_pit.ogg (https://adafru.it/DiQ)
- trap_poison.ogg (https://adafru.it/DiR)
- trap_spike.ogg (https://adafru.it/DiS)

From freesound.org (https://adafru.it/D0A)

- 43707__digifishmusic__unlocking-security-door.wav (https://adafru.it/DiT)
- 423146__ogsoundfx__secret-door-fantasy-wav.wav (https://adafru.it/DiU)

Getting Sounds

There are many places online to get free to use sounds for Tabletop RPG gaming.

Tabletop Audio (https://adafru.it/D4J) has some incredibly well done sounds. The background sound files are 10 minutes long and all files are very high quality so they will have to be edited/converted to fit in the NeoTrellis M4's flash. If you use their sounds, consider throwing some support their way on Patreon or their own donate link.

Freesound (https://adafru.it/D0A) is another source of free to use audio for various purposes. These are generally uploaded by community members and have varying quality.

Regardless where you get your sounds, they will likely have to be converted to work/fit on the NeoTrellis M4. See the guide on sound file conversion (https://adafru.it/BvU).

No Mixing & Matching Mono and Stereo Files, Please

Make sure your audio files are exported as **16-bit PCM WAV** at **22,050 Hz or lower** and they are **all Stereo** or **all Mono** -no mix and match!

Double Check: File Locations

Create a subdirectory on your **CIRCUITPY** flash drive named **samples,** and place all your WAV files in that directory. Your **code.py** and **soundboard.txt** files will go in the main directory.

Key image of the dragon mini (https://adafru.it/D4K) is from FreeImages.com (https://adafru.it/D4L)/Allan Browne (https://adafru.it/D4M)

Code and Use

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).

There's a guide to get you up and running with Circuit Python specifically for the NeoTrellis M4 (https://adafru.it/C-O). You should read it before starting to get the most recent CircuitPython build for the NeoTrellis M4 installed and running along with the required libraries.

Be sure to use at least CircuitPython 4.0 alpha 3 for the NeoTrellis M4, as it uses the latest audioio capabilities.

Navigating the NeoTrellis

To get your NeoTrellis M4 set up to run this project's code, first follow these steps:

1) Update the bootloader for NeoTrellis (https://adafru.it/C-N)from the NeoTrellis M4 guide

2) Install the latest CircuitPython for NeoTrellis (https://adafru.it/C-O)from the NeoTrellis M4 guide

For this project you will need the following libraries:

- **adafruit_trellism4.mpy**
- **neopixel.mpy**
- **adafruit_matrixkeypad.mpy**

Download Code

Time to install the software, here's the **code.py** listing, click on the **Download Project Bundle** link in the top left to grab all the code.

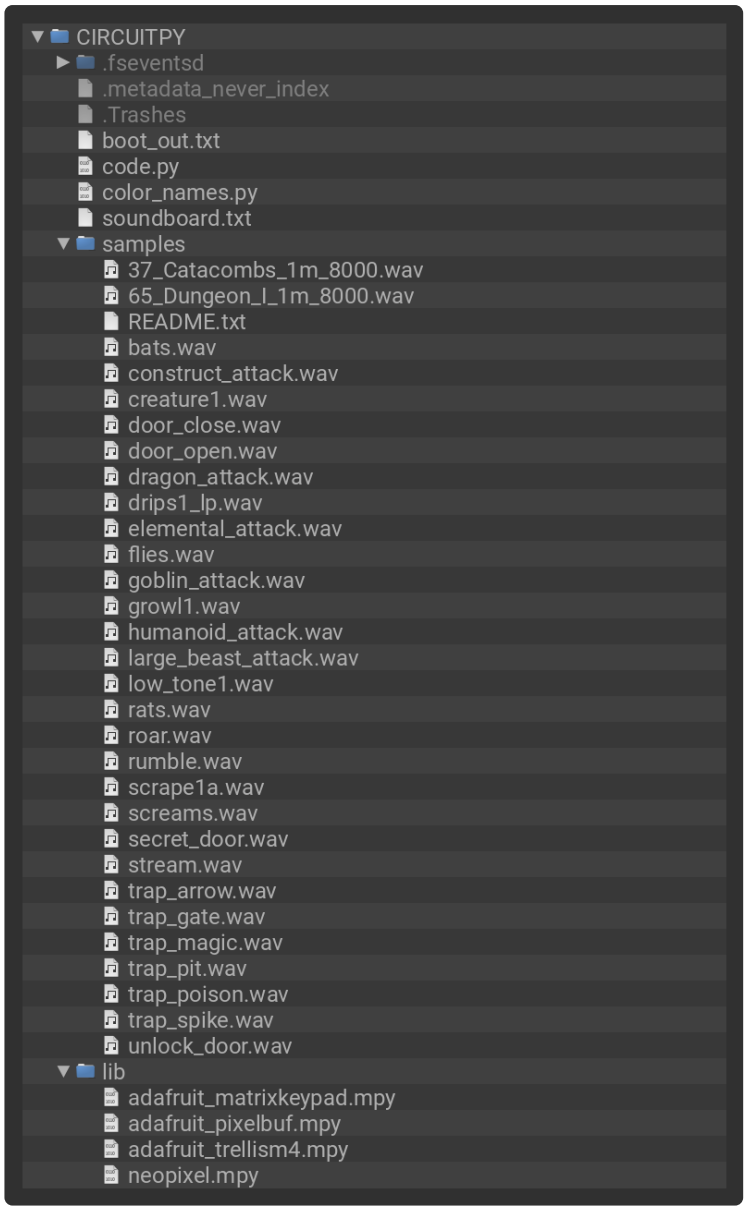
```
# SPDX-FileCopyrightText: 2018 Dave Astels for Adafruit Industries
#
# SPDX-License-Identifier: MIT

"""
Tabletop RPG soundboard for the NeoTrellisM4

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=wildcard-import,unused-wildcard-import,eval-used

import time
import board
import audioio
import audiocore
import audiomixer
import adafruit_trellism4
from color_names import *

# Our keypad + neopixel driver
trellis = adafruit_trellism4.TrellisM4Express(rotation=0)


SELECTED_COLOR = WHITE            # the color for the selected sample
SAMPLE_FOLDER = '/samples/'       # the name of the folder containing the samples
SAMPLES = []
BLACK = 0x000000


# load the sound & color specifications
with open('soundboard.txt', 'r') as f:
    for line in f:
        cleaned = line.strip()
        if len(cleaned) > 0 and cleaned[0] != '#':
            if cleaned == 'pass':
                SAMPLES.append(('does_not_exist.wav', BLACK))
            else:
                f_name, color = cleaned.split(',', 1)
                SAMPLES.append((f_name.strip(), eval(color.strip())))
```

```
# Parse the first file to figure out what format its in
channel_count = None
bits_per_sample = None
sample_rate = None
with open(SAMPLE_FOLDER+SAMPLES[0][0], 'rb') as f:
    wav = audiocore.WaveFile(f)
    channel_count = wav.channel_count
    bits_per_sample = wav.bits_per_sample
    sample_rate = wav.sample_rate
    print('%d channels, %d bits per sample, %d Hz sample rate ' %
          (wav.channel_count, wav.bits_per_sample, wav.sample_rate))

    # Audio playback object - we'll go with either mono or stereo depending on
    # what we see in the first file
    if wav.channel_count == 1:
        audio = audioio.AudioOut(board.A1)
    elif wav.channel_count == 2:
        audio = audioio.AudioOut(board.A1, right_channel=board.A0)
    else:
        raise RuntimeError('Must be mono or stereo waves!')

mixer = audiomixer.Mixer(voice_count=2,
                         sample_rate=sample_rate,
                         channel_count=channel_count,
                         bits_per_sample=bits_per_sample,
                         samples_signed=True)
audio.play(mixer)

# Clear all pixels
trellis.pixels.fill(0)

# Light up button with a valid sound file attached
for i, v in enumerate(SAMPLES):
    filename = SAMPLE_FOLDER+v[0]
    try:
        with open(filename, 'rb') as f:
            wav = audiocore.WaveFile(f)
            print(filename,
                  '%d channels, %d bits per sample, %d Hz sample rate ' %
                  (wav.channel_count, wav.bits_per_sample, wav.sample_rate))
            if wav.channel_count != channel_count:
                pass
            if wav.bits_per_sample != bits_per_sample:
                pass
            if wav.sample_rate != sample_rate:
                pass
            trellis.pixels[(i % 8, i // 8)] = v[1]
    except OSError:
        # File not found! skip to next
        pass


def stop_playing_sample(details):
    print('playing: ', details)
    mixer.stop_voice(details['voice'])
    trellis.pixels[details['neopixel_location']] = details['neopixel_color']
    details['file'].close()
    details['voice'] = None


current_press = set()
current_background = {'voice' : None}
currently_playing = {'voice' : None}
while True:
    pressed = set(trellis.pressed_keys)
    just_pressed = pressed - current_press
    # just_released = current_press - pressed
```

```
    for down in just_pressed:
        sample_num = down[1]*8 + down[0]
        try:
            filename = SAMPLE_FOLDER+SAMPLES[sample_num][0]
            f = open(filename, 'rb')
            wav = audiocore.WaveFile(f)

            if down[1] == 0:              # background loop?
                if current_background['voice'] != None:
                    print('Interrupt')
                    stop_playing_sample(current_background)

                trellis.pixels[down] = WHITE
                mixer.play(wav, voice=0, loop=True)
                current_background = {
                    'voice': 0,
                    'neopixel_location': down,
                    'neopixel_color': SAMPLES[sample_num][1],
                    'sample_num': sample_num,
                    'file': f}
            else:
                if currently_playing['voice'] != None:
                    print('Interrupt')
                    stop_playing_sample(currently_playing)

                trellis.pixels[down] = WHITE
                mixer.play(wav, voice=1, loop=False)
                currently_playing = {
                    'voice': 1,
                    'neopixel_location': down,
                    'neopixel_color': SAMPLES[sample_num][1],
                    'sample_num': sample_num,
                    'file': f}
        except OSError:
            pass # File not found! skip to next

    # # check if any samples are done
    # # this currently doesn't work with the mixer until it supports per voice
"is_playing" checking
    # if not audio.playing and currently_playing['voice'] != None:
    #     stop_playing_sample(currently_playing)

    time.sleep(0.01)  # a little delay here helps avoid debounce annoyances
    current_press = pressed
```

Use

The directory/file arrangement on **CIRCUITPY** after unzipping the the project is shown below.



The **code.py** CircuitPython code should run as soon as you copy the file onto the NeoTrellis **CIRCUITPY** drive. Be sure you copy all the files shown above to your **CIRCUITPY** drive.

Now you can start pressing buttons for your sound effects!

If some sounds work but not others, you'll need to edit the **soundboard.txt** file as explained on the previous page to ensure the file names are correct and you get the color you want for the button.

If you believe you have the **soundboard.txt** file correct and you do not hear the sound:

- Check your cabling to ensure you have powered amplified speakers and that the are on and volume at a good level.

- Be sure all of your sounds have been converted to 22,050 KHz, 16-bit PCM WAV files. Use all Mono or Stereo files, don't mix Mono and Stereo. This guide (https://adafru.it/BvU) will help you with the conversions. MP3 files cannot be played in this project, sorry, they'll need conversion to WAV.

Code Walkthrough

Code Walkthrough

As usual, we start with imports, hardware setup, and initialization of global constants. OK, `SAMPLES` isn't literal constant, but it gets filled from the `soundboard.txt` file and is treated as a constant after that.

```
import time
import board
import audioio
import audiocore
import adafruit_trellism4
from color_names import *

# Our keypad + neopixel driver
trellis = adafruit_trellism4.TrellisM4Express(rotation=0)


SELECTED_COLOR = WHITE            # the color for the selected sample
SAMPLE_FOLDER = '/samples/'       # the name of the folder containing the samples
SAMPLES = []
```

The next thing is to read the contents of `SAMPLES` from the `soundboard.txt` file.

We iterate through the file, line by line. If the line is empty, or it's first character is `#` (marking it as a comment) it gets ignored. If it is only the string pass, an empty element is appended to `SAMPLES`. Otherwise it is split in two at the first comma. Each piece has any whitespace trimmed from the ends before being assembled into a tuple and appended to the `SAMPLES` list. Notice how the `eval` function is used to convert the string representing the color to the appropriate value (either number or RGB tuple depending on the form of the string).

```
with open('soundboard.txt', 'r') as f:
    for line in f:
        cleaned = line.strip()
        if len(cleaned) &gt; 0 and cleaned[0] != '#':
            if cleaned == 'pass':
                SAMPLES.append(('does_not_exist.wav', BLACK))
            else:
                f_name, color = cleaned.split(',', 1)
                SAMPLES.append((f_name.strip(), eval(color.strip())))
```

It was mentioned earlier that all files need to have the same format. That format is determined by examining the first sample.

```
channel_count = None
bits_per_sample = None
sample_rate = None
with open(SAMPLE_FOLDER+SAMPLES[0][0], 'rb') as f:
    wav = audiocore.WaveFile(f)
    channel_count = wav.channel_count
    bits_per_sample = wav.bits_per_sample
    sample_rate = wav.sample_rate
    print('%d channels, %d bits per sample, %d Hz sample rate ' %
          (wav.channel_count, wav.bits_per_sample, wav.sample_rate))

    # Audio playback object - we'll go with either mono or stereo depending on
    # what we see in the first file
    if wav.channel_count == 1:
        audio = audioio.AudioOut(board.A1)
    elif wav.channel_count == 2:
        audio = audioio.AudioOut(board.A1, right_channel=board.A0)
    else:
        raise RuntimeError('Must be mono or stereo waves!')
```

Since one of our goals is to have a background loop over which we can play shot sound clips on demand, we need to use the mixer instead of the raw audio object.

A mixer object is created with the same sample settings as the audio object, with 2 channels. The audio object is then told to play what comes out of the mixer.

```
mixer = audioio.Mixer(voice_count=2,
                      sample_rate=sample_rate,
                      channel_count=channel_count,
                      bits_per_sample=bits_per_sample,
                      samples_signed=True)
audio.play(mixer)
```

Once we have the audio taken care of, we turn to the buttons. Each audio file specified in the `soundboard.txt` file (or the non-existant `does_not_exist.wav` where `pass` was specified) is checked. If it doesn't exist or it doesn't match the parameters fetched previously from the first file, that button is unlit. Otherwise it's color is set to that specified in `soundboard.txt`.

```
trellis.pixels.fill(0)

for i, v in enumerate(SAMPLES):
    filename = SAMPLE_FOLDER+v[0]
    try:
        with open(filename, 'rb') as f:
            wav = audiocore.WaveFile(f)
            print(filename,
                  '%d channels, %d bits per sample, %d Hz sample rate ' %
                  (wav.channel_count, wav.bits_per_sample, wav.sample_rate))
            if wav.channel_count != channel_count:
                pass
            if wav.bits_per_sample != bits_per_sample:
                pass
            if wav.sample_rate != sample_rate:
```

```
                pass
            trellis.pixels[(i % 8, i // 8)] = v[1]
    except OSError:
        # File not found! skip to next
        pass
```

Whenever a sound is played, it's details are captured in a structure. That structure is used in the `stop_playing_sample` function:

```
def stop_playing_sample(details):
    print('playing: ', details)
    mixer.stop_voice(details['voice'])
    trellis.pixels[details['neopixel_location']] = details['neopixel_color']
    details['file'].close()
    details['voice'] = None
```

After some initialization we now come to the main loop. `Current_press` keeps track of what button(s) are currently pressed. This is used to determine what buttons have been pressed or released since the last time through the loop. Add a short delay at the end of the loop and this amounts to a form of budget debouncing.

We have a structure for the background loop as well as any other sound that gets played.

```
current_press = set()
current_background = {'voice' : None}
currently_playing = {'voice' : None}
while True:
```

The first thing that happens in the loop is figuring out what buttons were pressed since last time.

```
    pressed = set(trellis.pressed_keys)
    just_pressed = pressed - current_press
```

Now we loop through the buttons that have been pressed.

Based on the button's location in the grid, we figure out which sample it corresponds to. We try to open that file. If there's a problem doing so, the `except` clause ignores the attempt to play it.

If the button is in the top row, it is a background loop. This is played on **voice 0** of the mixer, and set to loop repeatedly. If it's on any other row, it plays once on mixer **voice 1**. Any file is currently open/playing on the associated voice is first stopped and closed.

At the end of the loop, we delay briefly and update `current_press` for next time.

```
for down in just_pressed:
        sample_num = down[1]*8 + down[0]
        try:
            filename = SAMPLE_FOLDER+SAMPLES[sample_num][0]
            f = open(filename, 'rb')
            wav = audiocore.WaveFile(f)

            if down[1] == 0:              # background loop?
                if current_background['voice'] != None:
                    stop_playing_sample(current_background)

                trellis.pixels[down] = WHITE
                mixer.play(wav, voice=0, loop=True)
                current_background = {
                    'voice': 0,
                    'neopixel_location': down,
                    'neopixel_color': SAMPLES[sample_num][1],
                    'sample_num': sample_num,
                    'file': f}
            else:
                if currently_playing['voice'] != None:
                    stop_playing_sample(currently_playing)

                trellis.pixels[down] = WHITE
                mixer.play(wav, voice=1, loop=False)
                currently_playing = {
                    'voice': 1,
                    'neopixel_location': down,
                    'neopixel_color': SAMPLES[sample_num][1],
                    'sample_num': sample_num,
                    'file': f}
        except OSError:
            pass # File not found! skip to next

    time.sleep(0.01)  # a little delay here helps avoid debounce annoyances
    current_press = pressed
```

Troubleshooting

Troubleshooting

Project produces error:

> AttributeError: 'WaveFile' object has no attribute 'channel_count'

You should update the NeoTrellis firmware as described in https://learn.adafruit.com/adafruit-neotrellis-m4/circuitpython (https://adafru.it/C-O).