Glowing Bottle Castle and Capacitive Touch Piano
Created by Erin St Blaine

https://learn.adafruit.com/glowing-bottle-castle-and-capacitive-touch-piano

Last updated on 2021-11-15 08:05:40 PM EST
# Table of Contents

## Overview
- Adafruit Products Needed

## Circuit Diagram
- Full Wiring Diagram
- Simple Lights-Only Wiring

## Easier Code with MakeCode
- How To Upload the Code
- How To Use It
- How to Customize Your Code
- Troubleshooting

## Advanced Code with CircuitPython
- Install CircuitPython
- Upload Files
- Customizing Your Code
- NeoPixel Setup
- Demo Mode Animations
- Audio Files

## Electronics Assembly
- NeoPixel Connections
- Power Connection
- Capacitive Touch Board Connections
- Speaker Connections
- Troubleshooting

## Build the Castle
- Build the Base
- Attach NeoPixels
- Decorate the Bottles

## Build Bottle Toppers

## Play It
Overview

There is a castle on a cloud
I like to go there in my sleep
Aren't any floors for me to sweep
Not in my castle on a cloud

- Castle On A Cloud, from Les Miserables / Claude-Michel Schönberg

Build your own castle-on-a-cloud from recycled glass bottles and make it glow with NeoPixels. Each bottle is topped with a whimsical, electrically conductive clay sculpture. We’ve wired all the sculptures to a capacitive touch pad sensor and speaker, so each bottle lights up and sounds a pretty tone when it's touched. You can play this castle like a musical instrument! And if you touch the small bottle near the entry, the castle will play a song for you, with a lovely LED animation accompanying it.

The full build is fairly complicated, with a lot of wires and soldering and fiddling, but the end result is absolutely breathtaking. Get ready to dig in and become a song castle virtuoso.

If you love the idea but aren't a musician, we've also included instructions for a simplified lights-only version that's really easy to assemble. It's not interactive, but it's still going to show off your bottle artwork beautifully.
# Adafruit Products Needed

2 x **Side-Light NeoPixels**
Adafruit NeoPixel LED Side Light Strip - Black 90 LED  
[https://www.adafruit.com/product/3635](https://www.adafruit.com/product/3635)

1 x **Power Terminal**
Female DC Power adapter - 2.1mm jack to screw terminal block  
[https://www.adafruit.com/product/368](https://www.adafruit.com/product/368)

1 x **Power Supply**
5V 4A (4000mA) switching power supply - UL Listed  
[https://www.adafruit.com/product/1466](https://www.adafruit.com/product/1466)

1 x **Bolt-on Kit (for no-solder version)**
Bolt-On Kit for Circuit Playground, micro:bit, Flora or Gemma (Optional)  
[https://www.adafruit.com/product/4103](https://www.adafruit.com/product/4103)

If you’re making the capacitive touch musical version you’ll also need:

1 x **Capacitive Touch Breakout**
Adafruit 12-Key Capacitive Touch Sensor Breakout - MPR121  

2 x **Speaker**
Adafruit STEMMA Speaker - Plug and Play Audio Amplifier  
[https://www.adafruit.com/product/3885](https://www.adafruit.com/product/3885)

1 x **Speaker Cable**
JST PH 3-pin Plug-Plug Cable - 100mm long  

3 x **Wire**
Silicone Cover Stranded-Core Wire - 25ft 26AWG, in red, black, and white  
Additional Materials

- Glass bottles in varying sizes - my castle has 11 bottles
- 1/8" or thicker acrylic or polycarbonate sheet for the base. Mine is 6.5" x 35"
- Base frame material - I used leftover wood siding
- Air-dry clay - you can find this at any craft store or online
- Powdered graphite - often available at art / craft stores, or online
- Metallic gold paint and/or metallic gold spray paint
- Mirror Spray Paint by Rustoleum
- Colored glass, shells, jewels, cellophane wrap, shiny vinyl, or other decorative findings
- Clear packing tape - 2" wide
- Zip ties
- E6000 Glue

Tools Needed

To make the lights-only version, you'll need a screw driver and wire cutters / strippers.

For the capacitive touch version, you'll also need a soldering iron and accessories.

You may also find it helpful to have a heat gun, a hot glue gun, and a saw or woodworking tools for building the frame.
Circuit Diagram

Full Wiring Diagram

NeoPixel Strip

- Circuit Playground VOUT --> NeoPixel +
- Circuit Playground G --> NeoPixel G
- Circuit Playground A1 --> NeoPixel DIN

Wire in your power terminal midway down your strip of LEDs for the most even distribution of power.

Speakers

- Circuit Playground VOUT --> Speaker POWER
- Circuit Playground G --> Speaker G
- Circuit Playground A0 --> Speaker SIGNAL

If you're using two speakers, plug the second into the first with your 3-pin STEMMA connector.

Capacitive Touch Board

- Circuit Playground 3.3V --> MPR121 POWER
- Circuit Playground G --> MPR121 G
- Circuit Playground SCL --> MPR121 SCL
- Circuit Playground SDA --> MPR121 SDA
- Pins 1-11 are connected to the capacitive touch pads on the bottles.
I also wired the additional G pin on the MPR121 directly into the power terminal G. The more grounding you can get, the better this board works.

Simple Lights-Only Wiring

If you don't care about the musical setup and just want to make a pretty light display, here's a simplified wiring diagram without all the fancy capacitive touch and sound elements.

NeoPixel Strip

- Circuit Playground VOUT --> NeoPixel +
- Circuit Playground G --> NeoPixel G
- Circuit Playground A1 --> NeoPixel DIN

Wire in your power terminal midway down your strip of LEDs for the most even distribution of power.
Easier Code with MakeCode

This is the lights-only version of the code. If you’re adding capacitive touch interactivity, head to the CircuitPython page.

MakeCode is an easy way to get up and running with the Circuit Playground. No prior coding knowledge is needed, and it’s an easy way to experiment and learn to think like a coder. You just drag and drop code blocks, like building with Lego. MakeCode makes coding fun!

Head over to this Intro to MakeCode (https://adafru.it/AEp) guide for more info on getting started with MakeCode.

How To Upload the Code

Open the project code in MakeCode

https://adafru.it/MaA

1. Open the project with the button above and click the blue DOWNLOAD button near the bottom of the project.
2. Plug your Circuit Playground into your computer via its USB port and click the "reset" button. All the lights will turn green and your Circuit Playground will appear as a drive on your computer called CPLAYBTBOOT.
3. Drag the code you just download onto this drive to program the Circuit Playground -- like putting files on a USB stick. That's all you need to do.
Note: If you plug in the board and you see a drive called CIRCUITPY appear, press the reset button again (double-click) to get to CPLAYBTBOOT.

How To Use It

On/off:

The tiny slide switch on the face of the Circuit Playground turns the light strand on or off. Slide right for on, slide left for off. (So, if you just downloaded the code and you don't see anything happening, flip this switch! You might just be in "off" mode.)

Modes

This code has four different modes. Click button A to cycle between them:

1. Rainbow: an animated LED rainbow. Who doesn't love rainbows?
2. Rainbow Fade: the lights are all the same color and fade through the rainbow as a unit
3. Solid: Pick a color, any color ya want!
4. Twinkle: Lovely glittery white twinkles

Variations

Button B is our variation button. Click it to vary the speed of the rainbow and twinkle animations, and the color (hue) of the solid animation. There are seven speed levels,
from fast to slow, and the hue will just keep changing and rotating each time you press the button.

How to Customize Your Code

You can add your own modes, give yourself more speed variations, or set it to start up on your favorite mode. The possibilities are endless! Poke around and change some stuff, and if it stops working you can always come back here and start fresh. Make it yours.

![Code blocks diagram]

The green box on the left contains two blocks: on start and forever. These are found under the LOOPS tab at the left. Anything in the on start block runs just once, when the board first powers up. Anything in the forever loop runs over and over, forever.
On Start Block

The first line sets up our light strand. If you have more than one strand daisy-chained, change the set strip to create WS2812 strip number to reflect your total number of lights.

Next, we define our variables.

If you add or remove modes, change numModes to reflect the total number of modes (including mode 0).

To select a different start-up mode, change set mode to 0 to the number of the mode you want to appear first.

You can also change the starting hue by choosing a number between 0 and 255.

Forever Block

Here is where we set up the order of our modes. If you want to add more modes you can do it here by clicking the + button at the bottom and calling more functions. Be sure to increment the mode number in each block you make, and don't forget to change numModes in the previous block if you add or subtract modes.

I really like having solid come after rainbowFade. I can choose which solid color to "land" on by the timing of when I press button B -- as the lights fade through the rainbow, press the button to "capture" that color as the solid hue.
Functions

The blue box in the middle contains our functions. These are the actual LED modes themselves, which we call in the `forever` loop. You can add as many functions as you like.

The **rainbow** function is using MakeCode's built-in rainbow animation. We're using `show frame of animation` instead of `show animation` because that allows us to change the speed with the `delay` variable, by pressing button B.

The **rainbowFade** function can be customized too. To make the lights more pastel / less saturated, change the `saturation` number: 255 is fully saturated, and 0 is pure white. To make the fade go more slowly / smoothly, decrease `change hue by` to a smaller number. Set this to 1 for a very slow, smooth fade -- experiment to see what you like.

The **solid** function is a simplified version of **rainbowFade**. You can vary the saturation here too if you'd like.

The **twinkle** function is another built-in animation provided by MakeCode. There are 6 animations in that dropdown -- experiment with the others to see if there's one you like!
Button Controls

This block is where we tell the CircuitPlayground to scroll through the modes each time button A is clicked. When it reaches the last mode (defined by the `numModes` variable from the `on start` block) it resets to 0.

Button B is where we've created our variations. I'm varying speed and color, but you could also vary brightness or saturation or any other variable you can dream up.

Try changing the numbers around to customize how many speed levels there are or how much the color changes each time the button is pressed.

There are also other triggers available -- you can trigger a mode change or variation when you shake or move the CircuitPlayground, or whenever a loud sound is heard, or a whole host of other options. Look under the INPUT tab to see them all.

This is an easy way to make your project react to its environment. Make the castle change color whenever you clap your hands!
On/Off Switch

Here's where we turn the lights on and off with the slide switch. We've done this by changing the brightness to 0 or back to 255, so if you want to change the overall strand brightness be sure to change it here as well as in on start.

Doing it this way means the Circuit Playground will "remember" the light mode and variation you were in when you turn the lights back on. Cool!

However, remember that the whole project is NOT turned off when the slide switch is in the off position. The board is still awake and using power, so if you've got a battery powered project, this mode will slowly drain your battery. Turn it off with the switch on the battery pack or unplug your battery when you're ready to leave it off long-term.

Troubleshooting

If you are having trouble, here are a few things to check:

• Is your slide switch in the "off" position? If you download the code and nothing happens, flip the switch the other way to see if that fixes things.
• Are you connected to pin A1 with your data wire? Double check your wiring to be sure it matches the diagram.
• Did you connect to the IN end of the strip? The lights should be face up when the Circuit Playground is face up, without any twist in the wires. If they're face down, you've connected to the wrong end of the strip. Switch it around and try again.
• Are you using a Circuit Playground Express instead of a Circuit Playground Bluefruit? This code will work with the Express, but you need to select it in the MakeCode window. Click the ... button next to the Download button and select Choose Hardware to switch boards.
If you still can't get it working, head over to the Circuit Playground BlueFruit Intro Guide (https://adafru.it/GYc) for more help and suggestions.

Advanced Code with CircuitPython

This is the full version of the code with capacitive touch functionality. If you're making the lights-only version, head to the MakeCode page for software instructions.

It's always a good idea to get your software loaded onto your board before the build. That way you'll be able to test your solder joints at each step of the way, and you'll get instant gratification when you plug in the lights.

Getting the software loaded is a 3-step process:

1. Install the operating system (CircuitPython) on the board
2. Copy the required libraries into the /lib folder on the board
3. Save the code.py file to the board

CircuitPython (https://adafru.it/EFq) is a fairly new OS that's changing rapidly. New features are being added and bugs are being fixed all the time, so it's always best to get a fresh version of CircuitPython and the library files before coding.

Install CircuitPython

The Circuit Playground Bluefruit ships with CircuitPython, but let's go ahead and update it to the latest version. It's super easy with the circuitpython.org website.
Follow the directions on the CircuitPlayground BlueFruit Guide (https://adafru.it/GYc), or click the button below for a quick link to the download.

Download CircuitPython for the Circuit Playground Bluefruit

https://adafru.it/FNK

Download the file, plug your Circuit Playground into your computer via the USB port, and double-click the reset button. You'll see a drive appear called CPLAYBTBOOT. Drag the .uf2 file you just downloaded onto this drive to install CircuitPython.

You'll know it worked if the CPLAYBTBOOT drive name changes to CIRCUITPY.

Adafruit Circuit Python Libraries

Download the CircuitPython library bundle per the Circuit Playground Bluefruit instructions here (https://adafru.it/I3c). Unzip the files into a folder on your computer. Create a new folder on the CIRCUITPY drive and name it lib. The following libraries are required to run the code properly.

- adafruit_bus_device (directory)
- adafruit_circuitplayground (directory)
- adafruit_led_animation (directory)
- adafruit_lis3dh.mpy
- adafruit_mpr121.mpy
- adafruit_thermistor.mpy
- neopixel.mpy

Find all these files in the library bundle and copy them into the lib file you just made on your CIRCUITPY drive.
Upload Files

Click the link below to download the project zip – This contains the code and audio files. Upload the code.py file to the CIRCUITPY drive root (main) folder.

Create a new folder on the CIRCUITPY drive and name it sounds. Upload the audio files to that folder. The code will run properly when all of the files have been uploaded.

Check out the image above to see what your CIRCUITPY drive should look like when all the files are in place.

```
import time
import digitalio
import board
import neopixel
import busio
import adafruit_mpr121
from audiocore import WaveFile
from audiopwmio import PWMAudioOut as AudioOut
from adafruit_led_animation.animation.rainbow import Rainbow
from adafruit_led_animation.animation.rainbowchase import RainbowChase
from adafruit_led_animation.animation.chase import Chase
from adafruit_led_animation.animation.rainbowcomet import RainbowComet
from adafruit_led_animation.sequence import AnimationSequence
from adafruit_led_animation.color import BLACK, RED, ORANGE, YELLOW,
```
GREEN,
BLUE,
MAGENTA,
PURPLE,
AMBER,
TEAL,
)
from adafruit_debouncer import Debouncer

# Neopixel Strip Setup -- Set your total number of pixels here -------------------
PIXEL_NUM = 200
pixel_pin = board.A1
pixels = neopixel.NeoPixel(pixel_pin, PIXEL_NUM, brightness=1, auto_write=False)
LAST_BUTTON = None

... Customize your light strip for individual notes. Each line represents a bottle. Change the numbers to reflect the first and last pixel in each ring. You can also change the colors assigned to each bottle here. ...

bottle_lights = (  
(0, 10, RED),  
(15, 30, ORANGE),  
(31, 52, AMBER),  
(53, 72, YELLOW),  
(77, 96, GREEN),  
(98, 119, TEAL),  
(120, 145, BLUE),  
(150, 173, PURPLE),  
(180, 200, MAGENTA),  
)

# Cap Touch Board Setup --------------------------------------------------------
i2c = busio.I2C(board.SCL, board.SDA)
mpr121 = adafruit_mpr121.MPR121(i2c)

# Demo Mode LED Animations -----------------------------------------------------
rainbow = Rainbow(pixels, speed=0, period=10, name="rainbow", step=1)
rainbow_chase = RainbowChase(pixels, speed=0, size=5, spacing=10)
chase = Chase(pixels, speed=0.1, color=RED, size=1, spacing=6)
rainbow_comet = RainbowComet(pixels, speed=0.01, tail_length=60, bounce=True)

# Animation Sequence Playlist -- rearrange to change the order of animations
animations = AnimationSequence(  
    rainbow_chase,  
    chase,  
    rainbow_comet,  
    auto_clear=True,  
    auto_reset=True,  
)

def go_dark():  
    '''set all pixels to black'''  
pixels.fill(BLACK)  
pixels.show()

# Debouncer ---------------------------------------------------------------------
buttons = [Debouncer(mpr121[i]) for i in range(12)]

# Audio Setup -------------------------------------------------------------------
spkr_enable = digitalio.DigitalInOut(board.SPEAKER_ENABLE)
spkr_enable.direction = digitalio.Direction.OUTPUT
spkr_enable.value = True

audio = AudioOut(board.SPEAKER)

tracks = (  
    WaveFile(open("sounds/F2.wav", "rb")), # 0
    WaveFile(open("sounds/G2.wav", "rb")), # 1
    WaveFile(open("sounds/A2.wav", "rb")), # 2
    WaveFile(open("sounds/Bb2.wav", "rb")), # 3
    WaveFile(open("sounds/C2.wav", "rb")), # 4
    WaveFile(open("sounds/D3.wav", "rb")), # 5
    WaveFile(open("sounds/E3.wav", "rb")), # 6
    WaveFile(open("sounds/F3.wav", "rb")), # 7
    WaveFile(open("sounds/F1.wav", "rb")), # 7
    WaveFile(open("sounds/G1.wav", "rb")), # 8
    WaveFile(open("sounds/A1.wav", "rb")), # 9
    WaveFile(open("sounds/Bb1.wav", "rb")), # 10
    WaveFile(open("sounds/C1.wav", "rb")), # 11
    WaveFile(open("sounds/D2.wav", "rb")), # 12
    WaveFile(open("sounds/E2.wav", "rb")), # 13
    WaveFile(open("sounds/F2.wav", "rb")), # 13
)

# Add or change song track names here. They will play in the order listed.
demo_tracks = (  
    WaveFile(open("sounds/undersea.wav", "rb")),
    WaveFile(open("sounds/tequila.wav", "rb")),
    WaveFile(open("sounds/lion.wav", "rb")),
)

MODE = 0  # Initial mode = OFF
SONG = 0

def light_up(bottle):
    '''light up the bottles'''
    lights = bottle_lights[bottle]
    for pixel_id in range(lights[0], lights[1]):
        pixels[pixel_id] = lights[2]

def play_bottle(bottle_id, is_octave):
    '''play audio tracks and light up bottles'''
    global MODE
    go_dark()
    light_up(bottle_id)
    if is_octave:
        audio.play(tracks[bottle_id + 7])  # Start playing sound
        light_up(8)
    else:
        audio.play(tracks[bottle_id])  # Start playing sound
        pixels.show()
    MODE = 2

def check_buttons(touched):
    '''check to see if buttons have been pressed'''
    global MODE, LAST_BUTTON
    octave = touched[11]
    off = touched[9]
    if octave:
        light_up(8)
        for pad in range(1, 9):
            if LAST_BUTTON is not None and not touched[LAST_BUTTON]:
                LAST_BUTTON = None
            if pad != LAST_BUTTON and touched[pad]:
LAST BUTTON = pad
play_bottle(pad - 1, octave)

if off:
    MODE = 9
    go_dark()
if touched[10]:
    go_dark()
    audio.play(demo_tracks[SONG])
while audio.playing:
    animations.animate()

MODE = 3

while True:
    # Idle mode: Play a Rainbow animation when nothing's being touched
    if MODE == 0:
        pixels.brightness = 0.3  # rainbow mode is much brighter than the other
        for button in buttons:
            button.update()
        check_buttons(mpr121.touched_pins)
        time.sleep(0.1)
        for i in range(12):
            if buttons[i].fell:
                MODE = 1
        # If not idle mode
    if MODE >= 1:
        pixels.brightness = 1
        check_buttons(mpr121.touched_pins)
        time.sleep(0.1)
        if MODE == 2:  # mode 2 is individual notes
            if audio.playing:
                pixels.show()
                while audio.playing:
                    check_buttons(mpr121.touched_pins)
                    time.sleep(0.07)
            else:
                MODE = 0  # Return to idle mode
    if MODE == 3:
        SONG = SONG + 1
        animations.next()
        if SONG == 3:
            MODE = 0
            SONG = 0
        else:
            MODE = 0  # Return to idle mode
    if MODE == 9:  # MODE 9 is "off" mode, listening for a new button press to
wake up.
        for button in buttons:
            button.update()
        for i in range(12):
            if buttons[i].fell:
                MODE = 1

Customizing Your Code

The best way to edit and upload your code is with the Mu Editor, a simple Python
editor that works with Adafruit CircuitPython hardware. It's written in Python and
works on Windows, MacOS, Linux and Raspberry Pi. The serial console is built right in
so you get immediate feedback from your board's serial output. Instructions for installing Mu is here (https://adafru.it/JF4).

**NeoPixel Setup**

Look near the top of the code to find `PIXEL_NUM`. Change this number to reflect your total number of LEDs in your NeoPixel strips.

```python
# NeoPixel strip setup -- set your total number of pixels here -----------------
PIXEL_NUM = 200
pixel_pin = board.A1
pixels = neopixel.NeoPixel(pixel_pin, PIXEL_NUM, brightness=1, auto_write=False)
LAST_BUTTON = None
```

Next we'll set up the pixel mapping for the individual bottles. Starting from 0, count out the pixels in the section of LED strip below each bottle. Replace my numbers with yours -- the first number is the first pixel in the section and the second number is the last pixel in the section.

If you want to customize the colors you can do that here as well. Remember to import any additional colors you want to add in the list near the top of the code.

This code is based on the CircuitPython LED Animations (https://adafru.it/LZF) code library by Kattni Rembor. Check out that guide for a lot more info on using this library.

```python
Customize your light strip for individual notes. Each line represents a bottle. Change the numbers to reflect the first and last pixel in each ring. You can also change the colors assigned to each bottle here.

bottle_lights = (  
  (0, 10, RED),  
  (15, 30, ORANGE),  
  (31, 52, AMBER),  
  (53, 72, YELLOW),  
  (77, 96, GREEN),  
  (98, 119, TEAL),  
  (120, 145, BLUE),  
  (150, 173, PURPLE),  
  (180, 200, MAGENTA),
  )
```
Demo Mode Animations

Customize the animations that play in idle mode and demo mode here. Play with the settings until the animations look exactly the way you want. Or try out some other animations -- there are a lot more plug-and-play animations and ideas in the LED animations guide. (https://adafru.it/LZF)

The lower section is your animation playlist. Rearrange the order so they go with your songs.

```python
# Demo MODE LED Animations
rainbow = Rainbow(pixels, speed=0.1, period=10, name="rainbow", step=1)
rainbow_chase = RainbowChase(pixels, speed=0, size=5, spacing=10)
chase = Chase(pixels, speed=0.1, color=RED, size=1, spacing=5)
rainbow_comet = RainbowComet(pixels, speed=0.01, tail_length=60, bounce=True)

# Animation Sequence Playlist -- rearrange to change the order of animations
animations = AnimationSequence(
    rainbow_chase,
    chase,
    rainbow_comet,
    auto_clear=True,
    auto_reset=True,
)
```

Audio Files

We've included two octaves of bottle tones in the key of F, as well as three demo tracks. This castle can play a few bars of Under the Sea (from the Little Mermaid movie), Tequila, and The Lion Sleeps Tonight. It's not hard to add your own favorite melodies, but you'll need to keep them fairly short - there is 2MB of memory on this board and we've just about filled it up with all these files.

Adafruit CircuitPython supports 16-bit, Mono, 22.050kHz .wav audio format. See this guide (https://adafru.it/BvU) to help format any audio files you might want to use in this project besides the files provided.

```python
tracks = (
    WaveFile(open("sounds/F1.wav", "rb")), # 0
    WaveFile(open("sounds/G1.wav", "rb")), # 1
    WaveFile(open("sounds/A1.wav", "rb")), # 2
    WaveFile(open("sounds/Bb1.wav", "rb")), # 3
    WaveFile(open("sounds/C1.wav", "rb")), # 4
    WaveFile(open("sounds/D2.wav", "rb")), # 5
    WaveFile(open("sounds/E2.wav", "rb")), # 6
    WaveFile(open("sounds/F2.wav", "rb")), # 7
    WaveFile(open("sounds/G2.wav", "rb")), # 8
    WaveFile(open("sounds/A2.wav", "rb")), # 9
    WaveFile(open("sounds/Bb2.wav", "rb")), # 10
)
```python
WaveFile(open("sounds/C2.wav", "rb")), # 11
WaveFile(open("sounds/D3.wav", "rb")), # 12
WaveFile(open("sounds/E3.wav", "rb")), # 13
WaveFile(open("sounds/F3.wav", "rb")), # 13
)

# Add or change song track names here. They will play in the order listed.
demo_tracks = (
    WaveFile(open("sounds/undersea.wav", "rb")),
    WaveFile(open("sounds/tequila.wav", "rb")),
    WaveFile(open("sounds/lion.wav", "rb")),
)
```

Adding your Own Wav Files

I found that even following the very specific guidelines for .wav file setup, some files would crash the Feather when triggered. If you're having this problem, head to this guide (https://adafru.it/BvU) and re-crunch your files using Audacity.

This is a teeny tiny speaker, so sound files with more high-end frequencies will sound a lot better than deep, booming sounds. Sound files that have a lot of (or even any) low frequencies do not sound good when played back over small speakers. Too much low end will cause a small speaker to distort. To optimize your sound files for small speaker playback use a High Pass Filter (HPF) in your preferred audio editing software. Rolling off low frequencies below 250 Hz is a good starting point.

Audacity is great and simple audio editing app, and best of all, it's free!
Electronics Assembly

NeoPixel Connections

Data Flow Direction

NeoPixel strips have an "in" end and an "out" end. There are usually arrows marked on the strip showing you which is which. Power can connect from either end but data MUST flow from IN to OUT. We've got a whole guide on connecting NeoPixels - check it out here. (https://adafru.it/M9F)

For the side-light LEDs, instead of an arrow, the pads are marked with DI on one side and D0 on the other. Line up the lights so DI is closer to you as the strip is going away from you.

Find the IN end of your first strip. Cut off the connector and strip some shielding from a red, black, and white wire.

Use the bolt-on kit or solder the wires as shown:

• Red to VOUT
• White to A1
• Black to G

Power Connection

We're using a screw terminal to connect the power supply. For projects with a lot of LEDs, it's always good practice to connect the power midway through your LED strip. This way the power flows evenly to all parts of your project, and you won't get brownouts. This will also protect your Circuit Playground from drawing too much power.
Find the OUT end of your first strip. Plug the connector (female, in my case) into the IN end of the second strip to daisy-chain them together.

Strip a bit of shielding from the loose red and black wires near this connector. Twist the red wires together and push the ends into the + hole on your screw terminal.

If you're adding the capacitive touch board, cut an additional black wire and twist it together with your two NeoPixel black wires.

Insert the black wires into the - hole on the screw terminal. Tighten both screws until the wires won't pull out when you tug on them.

If you're doing the lights-only version of this project, you're done -- that's all the wiring you need to do. To add sound effects and interactivity, read on.

**Capacitive Touch Board Connections**

Solder your MPR121 board to your Circuit Playground as shown:

- 3.3V to VIN (red wire)
- G to G (black wire)
- SCL to SCL (yellow wire)
- SDA to SDA (green wire)

Find the third black wire you connected to your screw terminal and solder it into the remaining G pin on your MPR121.

That's it for now. We'll solder the wires to the individual bottles into pads 1-11 later -- it's much easier to build the bottles with loose wires and hook them up to the MPR121 board at the end.
Speaker Connections

Solder or screw the speaker to the Circuit Playground as shown:

- AUDIO to SIGNAL (yellow wire)
- 3.3V to POWER (red wire)
- G to GND (black wire)

If you're using two speakers, plug the 3-pin JST cable into the JST port on both speakers to daisy-chain them.

Once you've uploaded your code, plug your power supply into your screw terminal and test it out. Touch the pins on the MPR121 board. If all goes well, you'll hear different tones play and see different sections of your light strip come on. Woo hoo! Time to build the castle.

Troubleshooting

If it's not working, here are a few things to try.

If the lights aren't coming on:

1. Have you uploaded your code correctly? Head back to the Software page and double check you've uploaded all the files. If even one is missing the code will hang.
2. Did you solder to the IN end of the strip or get that mixed up? The markings can be confusing. If you’re hearing sounds but not seeing lights, this might be your problem.
3. Check to be sure you've got the NeoPixel strip data line wired to A1.
4. Try re-connecting to the screw terminal. These things are notoriously twitchy.

If you're not hearing sound:

1. Double check your wiring connections and be sure they match the wiring diagram.
2. Try plugging in just one speaker. Simplifying things usually helps with troubleshooting.
3. Be sure you've got the audio files uploaded along with the code.

For more troubleshooting ideas, head over to the Circuit Playground Bluefruit guide (https://adafru.it/GYc). Try uploading some test code to test your NeoPixel strip and your speakers individually.
Build the Castle

Someone gave me a lovely bottle of rosé at the beginning of the year, and the bottle was so gorgeous that I saved it for months, waiting for the perfect project. So naturally, it became the centerpiece of this lovely fairy castle. I'm also using barbecue sauce bottles, pickle jars, and some different brands of almond butter jars and spice jars. Art supplies are everywhere!

Use GooGone or another solvent to remove all the labels, and wash them all clean. Arrange and rearrange them until you're delighted with the layout.

Build the Base

The base of your castle will house your LEDs and hide all the electronics. Mine is made from a piece of 1/8" acrylic attached to a box built from leftover wood siding. I won't go into all the woodworking details here, but will give an idea of how I put it all together.
Cut a piece of 1/8" (or thicker) acrylic or polycarbonate to match the dimensions of your bottle layout. Give yourself a little extra room on all sides. My castle is going to sit on a 7" deep windowsill, so that influenced the shape and size dimensions I chose.

Spray the acrylic with 1-2 coats of Mirror Effect spray paint (https://adafru.it/Ma0). This will create a one-way mirror effect, letting light come up through the plastic but hiding all the wires and electronics beneath it.

This is one of my favorite LED diffusion tricks. Mirror Effect FTW.

Build a box to hold your acrylic about 1-2 inches up from the table, to make room for your LEDs and electronics. I made a shallow dado cut on the inside of my wood trim and slid the acrylic in so it's supported on all sides, but it would work just as well to attach the acrylic to the top of the frame. Be sure it's very well supported -- the bottles will weigh a fair bit when you're done. I added some hot glue and wood glue along the underside seams to be sure my bottles had plenty of support.
Cut a hole for the power cord to come through. I ended up placing the speakers on the outside of my box for better volume, so if you want to do that, make a hole for those wires too. And if you think you will want to change up your code and add different songs or tones later on, you could also make a port for the USB cable to make reprogramming easier.

Attach NeoPixels

We'll attach the NeoPixel strip to the underside of the mirrored acrylic so there's a ring of pixels underneath each bottle.
We're using side-light NeoPixels because we can bend them into these tight rings without damaging the LEDs. This is much easier, more flexible and more affordable than wiring 11 actual NeoPixel rings would be. Yay, side-light NeoPixels!

Create your light rings and hold them in place with zip ties. Don't tighten the zip ties too much - you don't want to damage the LED strip, so just make them tight enough to hold the rings in place.

Attach the rings to the underside of your acrylic with clear packing tape. This allows you to move and adjust them as needed until they're all exactly where you want them. Once they're perfect, slap on some more packing tape until they feel really secure.

If you're adding capacitive touch, drill a small hole behind each NeoPixel ring for the capacitive touch wire to come through.
Decorate the Bottles

Start decorating your bottles. My first layer was iridescent cellophane gift wrap (from the craft store) wrapped around the bottle and heated with a heat gun to shrink it to fit. I used a couple different colors for variety.

I used hot glue to affix glass beads or glass shards, gems, pearls, and sea shells to the bottles. On a few, I designed vinyl "windows" on my Cricut vinyl cutter and stuck them on. I topped a few bottles with dried sea urchins I found at the craft store. I topped this with another layer made from air-dry clay to create the columns and pediments.

This is the fun part. Create the castle of your dreams.
My castle's design was influenced by the architecture of Antoni Gaudi (https://adafru.it/Ma1), a Spanish architect who lived in the early 20th century.

Gaudi's work was heavily inspired by nature and nature's forms. His buildings have a minimum of straight lines or corners. Walking through them, you get the feeling that they just grew up out of the ground instead of being planned and built. Gaudi didn't draw plans for his buildings. He would work almost exclusively from models, letting gravity and natural forces inform the structure of his arches and hallways. His use of light and color inside his masterpiece, the church La Sagrada Familia, left me a quivering mess of emotion. I am in awe of his connection to nature, spirit, and form and in love with his whimsical designs.
Place your bottles onto the base as you work, to test how they're catching the light.

If you're making the lights-only version of this project, you're finished! You can glue the bottles down to the base if you like, or keep them loose and rearrangeable if you desire.

To add the capacitive touch bottle-toppers and make this a musical instrument and interactive castle of wonder, read on.
Build Bottle Toppers

The bottle toppers are made from air-dry clay that's been mixed with powdered graphite. Graphite is a very conductive molecule and is easy to obtain and work with, and it won't get in the way of your clay setting up.

We'll embed a long wire into the clay of each bottle topper. The wires should be long enough to reach through the holes you drilled in the acrylic and comfortably reach your MPR121 board. Once all the bottles are assembled and the clay is dry, we'll thread the wires through the base and solder them to the MPR121.

Warning - Water Is Conductive

While the clay is wet, it will carry more of a capacitive charge than dry clay will carry. For my prototype castle, I tested all the toppers as I built them, and they all worked great for the first day or two. Once the clay had time to fully dry, however, they became less conductive and I had to remake or re-coat about half of them. I don't want that to happen to you. Be sure to let them dry fully, and test, test, test before final assembly!

Get out your powdered graphite. This stuff is extremely messy and not good to breathe, so wear a particle mask and gloves! (I promise I went and put on gloves right after taking these first two photos.. what a mess. It cleans up easily with soap and water though).

Mix the graphite into your clay. Use a LOT. Add some water if it starts to feel too dry. Keep mixing until the clay is black through and through. You need a solid, unbroken chain of graphite atoms, and the best way to be sure you've got that is if you can't see any white clay at all. Pack as much graphite as you can into that clay.
I sculpted some of my bottle toppers entirely from graphite-impregnated clay. For others, I rolled out the clay and wrapped it around an object -- a small bottle, a shell, or a ball of aluminum foil.

The clay in this photo does not have enough graphite. When this bottle topper dried, the capacitive touch stopped working. That's very sad!

To fix this, I made a slurry of graphite and water, mixed together until it was a paintable consistency. I applied this over the top of the clay to achieve that unbroken graphite atom-chain, so that anywhere I touch on this sculpture will set off the capacitive touch pad.
To carry the capacitive touch signal to the Circuit Playground we'll need to embed our long wire into the wet clay. Strip a few inches of wire and wrap it around your topper to achieve maximum electrical connection. Embedding the wire in a ball of wet graphite clay and pressing it onto your sculpture works well. Make sure there's a lot of contact between the wire and the graphite.

You can hide the wires inside the bottle decorations or just let them run down the backs of the bottles.

Once the clay and graphite slurry are completely dry, mask your bottle - aluminum foil works great for this - and paint the toppers. Metallic paint will not block the capacitive touch powers of your sculpture as long as you keep it to a thin coat. I had success with both spray paint and metallic acrylic paint. I didn't try a non-metallic color, so I don't know whether the metal in the paint helps with the conductivity or not (or whether this paint even has real metal?). I experimented and used what worked! Feel free to do the same. It's your castle.
Once your bottles are painted, it's time for some serious testing. Temporarily attach the other end of each bottle's wire to your MPR121 capacitive touch breakout and touch the heck out of it. Make sure it's reliable and fires the capacitive touch every time, and in all the places you touch. Get someone else to touch it too, to double check your responsiveness.

Test again the next day to be sure that your bottles still work when the clay is fully dry.

My octopus sculptures worked for days, then finally dried out to the point that they stopped working. Instead of starting over, I added a copper tape sticker to the tops of their heads with a line of copper tape attached that wraps around the wire. Copper tape is very conductive, and will not disappoint me by drying out.

It's not cheating to add copper tape if you're having trouble!

Once you're sure your toppers are rock-solid conductors, place the bottles onto your base and thread the wires through the holes. I found it helpful to place the base up on blocks to give myself room to work underneath.

Solder the wires into the holes on the MPR121. Here's how they're laid out in the code we've provided:

- Pin 0: not used
- Pins 1-8: Tone-playing bottles, from low to high
- Pin 9: Octave-shifting bottle
- Pin 10: Off switch bottle
- Pin 11: Demo mode bottle
Once everything is working, glue the bottles down to the acrylic base with strong glue. I used E6000. 2-part epoxy would work as well. Make sure they're solid, since they'll get tapped and touched a lot and you don't want them falling over.

Finish up by using hot glue to secure the electronics in place inside the base. I glued my speakers on the outside back center of the wooden frame, for maximum volume and resonance.

Play It

We've included two octaves of .wav files in the key of F.

The leftmost bottle (as you look from the front) is the tonic, and the next 7 bottles go up through the scale. Bottle #8 plays F again, one octave up.

Bottle 9 (the rightmost bottle) acts just like a thumb hole on a recorder. When you touch this bottle, all the other bottles will play a note that's one octave higher. This gives you two full octaves plus a high F -- plenty of notes to play melodies in a major or minor key.
Bottle 10 is my off switch. Touching this bottle will make all the bottles go dark.

Bottle 11 starts "demo mode". We've included 3 .wav files that play short melodies. Touching this bottle will cycle through the three melodies, and play an accompanying LED animation too.

Customize your castle using the instructions on the Software page. Enjoy your beautiful glowing musical fairy castle!