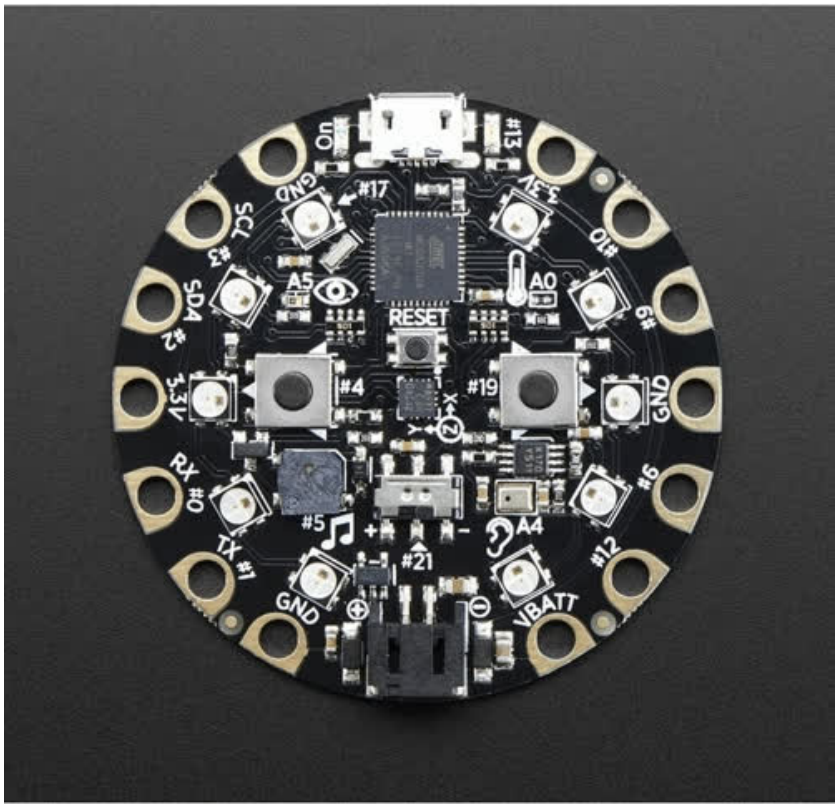
Circuit Playground Sound and Music

Created by Mike Barela



Last updated on 2016-08-17 06:23:33 PM UTC

Guide Contents

Guide Contents 2
Overview 3
What is Sound? 3
Using the Circuit Playground Speaker 5
Our First Sounds! 5
The Circuit Playground Library 7
The Sound of Music 9
Chiptunes 12
Light to Sound 15
Applications 15
Temperature to Sound 17
Motion to Sound 19
Applications 19
Simon Says Game 21

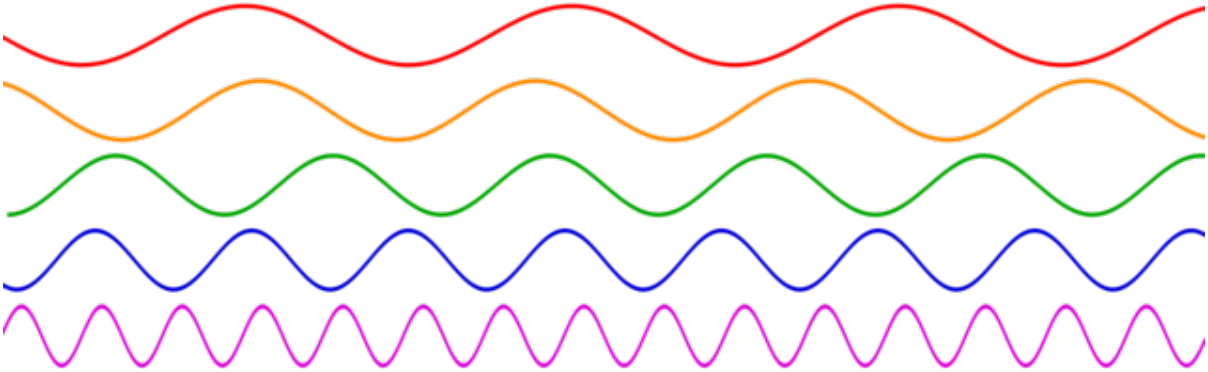
Overview

I love projects involving our senses! Sight, sound, touch, smell, taste. Adafruit's inclusion of a microphone and speaker on the Circuit Playground board provides a gazillion ways to explore our sound sense.

Let's review what sound is and build our skills from there.

What is Sound?

You can think of most phenomena around us as existing in the form of waves. From the lowest frequency waves of an earthquake to high frequency cosmic rays, the term you will hear is that waves go "from DC to Daylight". That refers to the broad *spectrum* of wave action in the energy around us. Below are some waves of differing frequencies (the top one is at a lower frequency, the bottom the highest frequency).



via Public Domain, Wikipedia (http://adafru.it/pFL)

There are so many frequencies! What we call ***sound*** is air vibrating at frequencies our ears can perceive. For a child, that would be 20 Hertz (cycles per second) to 20,000 Hertz (20 kiloHertz). For grandparents, they might not hear that upper limit, maybe 15 to 17 kiloHertz. Frequencies above 20 kilohertz may be audible by animals (that is where dog whistles fall, in the range 23 to 54 kiloHertz).

Sound can be soft and low like a thud or high and loud like an opera singer on a high note. How do we produce so many sounds? We can define how we characterize the sounds we hear:

Frequency: how fast the sound wave vibrates back and forth. One frequency would be monophonic, playing several sounds of different frequencies together gives polyphonic sound.

Duration: How long and short a sound is made.

Loudness: How strong the sound is (in the waves above, how tall the wave is shows the loudness or intensity).

Timbre: how a sound sounds, say dull like a plucked string vs. a "pure" musical note.

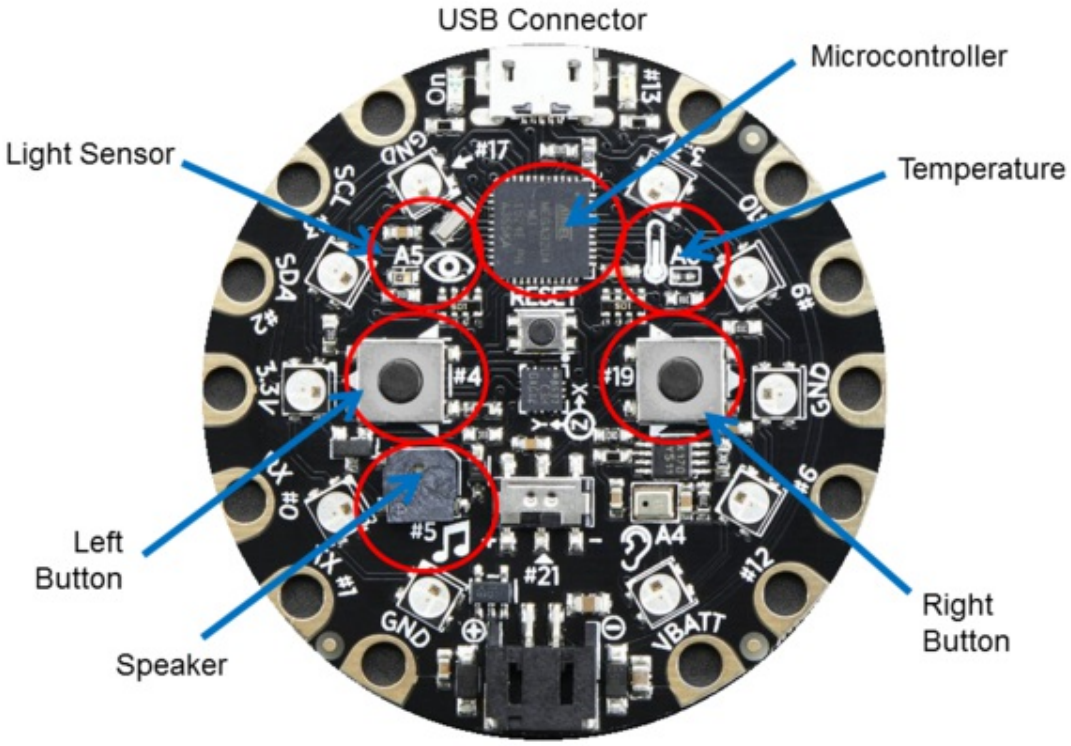
Spacial location: where your ear perceives a sound coming from. Think of a train passing you, the location changes from your perspective. If you note a sound change to your ear, this is due to more science in the form of the *Doppler Effect*.

Ok, enough talk, let's make some sounds!

Using the Circuit Playground Speaker

You will want to review Adafruit tutorials on how to use Circuit Playground and program in the Arduino integrated development environment. If you are new to using your computer to program Arduino style projects, please review the tutorial Circuit Playground Lesson #0 (http://adafru.it/pFM). That will familiarize you with the Circuit Playground board and ensure your programming environment is set up and ready to go.

Below are the components on the Circuit Playground board we will be looking at using in this tutorial:



If you prefer to go to the Circuit Playground library for using the speaker rather than peeking under the hood, you can skip to the next page without worries.

Our First Sounds!

The speaker on the Circuit Playground is connected to the microcontroller digital pin #5. The left button is on digital pin #4 and the right button is on digital pin #19 (it is handy that these pin numbers are printed in white on the circuit board!).

What we will do first is play one tone (a sound at one frequency) when the left button is pressed, and another if the right button is pressed. Let's pick two frequencies in Hertz (vibrations per second). I'll pick 440 Hertz (abbreviated Hz) and 1760 Hz (not quite random values, we'll go into that very soon!). Our code will make one of the sounds when the corresponding pushbutton is pressed on Circuit Playground:

```
// Adafruit Circuit Playground - Two tone sounds    Support Open Source, buy at Adafruit
//   2016-08-05 Version 1  by Mike Barela for Adafruit Industries

const int speaker = 5;       // The CP microcontroller pin for the speaker
const int leftButton = 4;    // The CP microcontroller pin for the left button
const int rightButton = 19;  // The CP microcontroller pin for the right button

void setup() {
  pinMode(speaker, OUTPUT);     // We will write out to the speaker
  pinMode(leftButton, INPUT);   // We'll read in from the buttons
  pinMode(rightButton,INPUT);
}

void loop() {
  if(digitalRead(leftButton)) {   // if reading the left button returns true
    makeTone(speaker,440,100);    // output a 440 Hz sound for a tenth of a second
  }
  else if(digitalRead(rightButton)) { // if reading the right button returns true
    makeTone(speaker,1760,100);   // output a 1760 Hz sound for a tenth of a second
  }
}

// the sound producing function (a brute force way to do it)
void makeTone (unsigned char speakerPin, int frequencyInHertz, long timeInMilliseconds) {
  int x;
  long delayAmount = (long)(1000000/frequencyInHertz);
  long loopTime = (long)((timeInMilliseconds*1000)/(delayAmount*2));
  for (x=0; x<loopTime; x++) {        // the wave will be symetrical (same time high & low)
     digitalWrite(speakerPin,HIGH);   // Set the pin high
     delayMicroseconds(delayAmount);  // and make the tall part of the wave
     digitalWrite(speakerPin,LOW);    // switch the pin back to low
     delayMicroseconds(delayAmount);  // and make the bottom part of the wave
  }
}
```

The program's main loop function reads the two Circuit Playground push buttons. If the left one is pressed, the speaker outputs a sound at 440 Hz (a low tone) , if the right button is pressed, it outputs a sound at 1760 Hz (a higher tone). At the bottom of the program we can peek into the code making the tone via the makeTone function. The loop turns on the speaker, waits, then turns it off. The speed of switching, slow or fast, makes the tone at the frequency at which the speaker pin is turned on and off.

Next is how to write the same code using the Circuit Playground library.

The Circuit Playground Library

The Arduino development environment provides a handy function for creating a varying square wave on a digital pin: the tone command. It is basically the same function as the makeTone function we used on the last page. You can read more on the arduino.cc website here (http://adafru.it/pFN).

For Circuit Playground, Adafruit provides a number of handy functions to use all the goodies on the board - all wrapped up into a library called CircuitPlayground.

Visit this page to learn how to install the library
http://adafru.it/pAQ

Adding the library gives us great functionality which we'll explore in the examples going forward.

Let's start by rewriting the demo program on the previous page.

We will add the following:

A call to CircuitPlayground.begin to initialize the library

Reading the buttons is accomplished via CircuitPlayground.leftButton and CircuitPlayground.rightButton

Finally, the Circuit Playground library plays tones through the speaker through the CircuitPlayground.playTone function.

Note we do not have to know anything about the underlying hardware to use the library, rather handy.

Here is the whole example:

```
// Adafruit Circuit Playground - Two tone sounds    Support Open Source, buy at Adafruit
//   2016-08-05 Version 1  by Mike Barela for Adafruit Industries
// Uses the CircuitPlayground library to easily use the full functionality of the board

#include <Adafruit_CircuitPlayground.h>

void setup() {
  CircuitPlayground.begin();
}

void loop() {
  if(CircuitPlayground.leftButton()) {   // if reading the left button returns true
    CircuitPlayground.playTone(440,100);    // output a 440 Hz sound for a tenth of a second
  }
  else if(CircuitPlayground.rightButton()) { // if reading the right button returns true
    CircuitPlayground.playTone(1760,100);   // output a 1760 Hz sound for a tenth of a second
  }
}
```

The code is alot shorter and is easy to read. It should also work on any board which supports the Circuit Playground library.

The program will play one tone when you push the left pushbutton, another tone when you push the right pushbutton.

We'll use other Circuit Playground library functions as we explore what can be done with the board's sound capabilities. You'll see why 440 and 1760 Hertz were chosen above.

The Sound of Music

One of the joys of our existence: music! But what is music? It turns out our ears are tuned to certain tone frequencies. These frequencies make up mathematically defined sets of tones called *musical notes (http://adafru.it/pFO)*. You can generate notes yourself by saying "Do-Re-Me-Fa-Sol-La-Si". You can say those notes in a low pitch (at low frequencies) or very high pitch (at higher frequencies). In musical notation, the syllables above are given the letters C-D-E-F-G-A-B. In some languages, other symbols are used but the English convention is widespread.

As you go from lower to higher frequencies, you progress through sound*octaves*. Low octaves are very low, while the highest octave would be rather high pitched. Think of a piano, one side has notes at a low frequency, and as you play the keys you hear the "Do-Re-Mi" sequence, repeating to higher and higher frequency tones.

The following is a list of musical note frequencies we can use in our Circuit Playground sketches. The definitions are adapted from the arduino.cc tutorial ToneMelody (http://adafru.it/pFP) credited to Brett Hagman. Copy and save to file pitches.h in your sketch folder.

```
/*************************************************************
 * Musical Notes via https://www.arduino.cc/en/Tutorial/ToneMelody  *
 *************************************************************/

#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
```

```
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
```

```
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```

Now you can see why I picked the two frequencies in previous examples. 440 Hertz is note A4, an A tone in the 4th octave. 1760 Hz is an A note in the 6th octave.

To play a song with the musical notes above, we need a bit more information. How long do we play each tone is important in music, things "sound right" as you play musical note at defined times, creating a *tempo (http://adafru.it/pFQ)* or a pace to which the notes are played. So each note should have both a frequency to play and a duration for how long we should play. They tempo is set in fractions or multiples of a *whole note (http://adafru.it/pFR)* for example a half note, quarter note, 8th, 16th and so on. In terms of time, music is played in beats per minute with notes at a faster or slower tempo.

So we'll play a short selection of notes (not quite a song) with several notes.

```
// Adafruit Circuit Playground - Melody    Support Open Source, buy at Adafruit
//   2016-08-06 Version 1  by Mike Barela for Adafruit Industries
//   Adapted from melody by Tom Igoe on arduino.cc
// Uses the CircuitPlayground library to easily use the full functionality of the board

#include <Adafruit_CircuitPlayground.h>
#include "pitches.h"

const int numNotes = 8;                    // number of notes we are playing
int melody[] = {                           // specific notes in the melody
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3, NOTE_G3, 0, NOTE_B3, NOTE_C4 };

int noteDurations[] = {     // note durations: 4 = quarter note, 8 = eighth note, etc.:
  4, 8, 8, 4, 4, 4, 4, 4 };

void setup() {
  CircuitPlayground.begin();  // initialize the CP library
}

void loop() {
  if(CircuitPlayground.rightButton()) {   // play when we press the right button
    for (int thisNote = 0; thisNote < numNotes; thisNote++) { // play notes of the melody
      // to calculate the note duration, take one second divided by the note type.
      //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
      int noteDuration = 1000 / noteDurations[thisNote];
      CircuitPlayground.playTone(melody[thisNote], noteDuration);

      // to distinguish the notes, set a minimum time between them.
      //   the note's duration + 30% seems to work well:
      int pauseBetweenNotes = noteDuration * 1.30;
      delay(pauseBetweenNotes);
    }
  }
}
```

You can find songs encoded with tone and duration in many places on the Internet.

Chiptunes

It is very popular in music and electronics culture to emulate the sounds/music of classic video games. The music was most often on 8 bit machines and the resulting music sounds a bit like the music we're playing. The melodies from classic gaming and remixed music are called chiptunes.

For an in-depth discussion of Chiptunes, you can catch the Adafruit video postPseudorandom #13 (http://adafru.it/pFS) which discusses the chiptunes scene.

Below is a melody from a classic video game - can you name that tune?

```
// Adafruit Circuit Playground - Theme Song    Support Open Source, buy at Adafruit
//   2016-08-12 Version 1  by Mike Barela for Adafruit Industries

#include <Adafruit_CircuitPlayground.h>
#include "pitches.h"

int melody[] = {                          // specific notes in the melody
NOTE_E7, NOTE_E7, 0, NOTE_E7,
  0, NOTE_C7, NOTE_E7, 0,
  NOTE_G7, 0, 0,  0,
  NOTE_G6, 0, 0, 0,

  NOTE_C7, 0, 0, NOTE_G6,
  0, 0, NOTE_E6, 0,
  0, NOTE_A6, 0, NOTE_B6,
  0, NOTE_AS6, NOTE_A6, 0,

  NOTE_G6, NOTE_E7, NOTE_G7,
  NOTE_A7, 0, NOTE_F7, NOTE_G7,
  0, NOTE_E7, 0, NOTE_C7,
  NOTE_D7, NOTE_B6, 0, 0,

  NOTE_C7, 0, 0, NOTE_G6,
  0, 0, NOTE_E6, 0,
  0, NOTE_A6, 0, NOTE_B6,
  0, NOTE_AS6, NOTE_A6, 0,

  NOTE_G6, NOTE_E7, NOTE_G7,
  NOTE_A7, 0, NOTE_F7, NOTE_G7,
  0, NOTE_E7, 0, NOTE_C7,
  NOTE_D7, NOTE_B6, 0, 0
 };
int numNotes; // Number of notes in the melody

int noteDurations[] = {     // note durations
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,

  12, 12, 12, 12,
```

```
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,

  9, 9, 9,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,

  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,

  9, 9, 9,
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,
};

void setup() {
  CircuitPlayground.begin();              // initialize the CP library
  numNotes = sizeof(melody)/sizeof(int);  // number of notes we are playing
}

void loop() {
  if(CircuitPlayground.rightButton()) {   // play when we press the right button
    for (int thisNote = 0; thisNote < numNotes; thisNote++) { // play notes of the melody
      // to calculate the note duration, take one second divided by the note type.
      int noteDuration = 1000 / noteDurations[thisNote];
      CircuitPlayground.playTone(melody[thisNote], noteDuration);

      // to distinguish the notes, set a minimum time between them.
      //   the note's duration + 30% seems to work well:
      int pauseBetweenNotes = noteDuration * 1.30;
      delay(pauseBetweenNotes);
    }
  }
}
```

Why doesn't our electronic music sound like a rock band?

Most music is *polyphonic* - multiple notes played together. If we used several Circuit Playground boards playing, we could achieve some of that.

But we still must deal with the physics of the speaker and the way we have been *driving*, or moving the speaker. Surround sound systems usually have larger speakers which can reproduce a wide range of sounds very well. Due to the construction and size of the Circuit Playground speaker, it is not as good as a name brand in making musical sounds.

We are also driving the speaker with a frequency which is on and off. As you can see in the first example code, we calculate the desired frequency and turn the speaker on then pause, then off, then pause and repeat. This is a square wave which in many speakers sounds pretty good but not "pure" like the sine waves we looke at in the overview. Mathematically, a square wave can be

composed of multiple sine waves added together (http://adafru.it/pFT). The wave we want is the *fundemental frequency* and smaller amplitude sine waves at higher frequencies that we might not want.

Circuit Playground allows us to easily work with sound and music. When we want to explore more band-style music, we can look at other Adafruit products to build great sounding musical circuits.

Light to Sound

Making music by varying lights is a very popular project. The effect is often called a Light Theremin. The original Theremin (http://adafru.it/pFU) instrument by Léon Theremin (http://adafru.it/pFV) used tuned resonant radio frequency circuits converting changes in resonance (http://adafru.it/pFW) to sound. Using a light changes to make different sounds is quite a bit easier.

Circuit Playground has a perfect sensor for detecting light in the upper left of the board (the part with the eye next to it). Light falling on the sensor changes the sensors resistance, changing the voltage the microcontroller reads. The value read by the sensor may be from 0 to 1023.

To make music, we will map the number received from the light sensor to the frequency ranges we looked at in The Sound of Music page (notes C3 to A6). The program will map to values that are not true musical notes though.

One more change is to use the Circuit Playground slide switch. There are times when the people around you do not want to hear the sounds (believe me). The light to sound code is only executed if the slide switch is moved to the "+" side.

```
// Adafruit Circuit Playground - Light Theremin    Support Open Source, buy at Adafruit
//   2016-08-07 Version 1  by Mike Barela for Adafruit Industries
// Uses the CircuitPlayground library to easily use the full functionality of the board

#include <Adafruit_CircuitPlayground.h>

void setup() {
  CircuitPlayground.begin();   // initialize the Circuit Playground library
}

void loop() {
   uint16_t value, sound;
   if(CircuitPlayground.slideSwitch()) {      // if the slide switch is on
     value = CircuitPlayground.lightSensor(); //   read the light sensor
     sound = map(value, 5, 1000, 131, 1760);  //   map light values to music values
     CircuitPlayground.playTone(sound, 100);  //   play sound for 100 milliseconds
   }
}
```

To ensure each output frequency is a musical note, one could create an array of musical notes and have the light values mapped to one of the musical notes in the array and output the musical note.

I chose the musical note range so sounds did not get too low (low sounds may create more of a click on the speaker) or too high (making a sound that is rather high or a whine, harsh on the ear). Feel free to adjust the last two values in the map function to change the sound values. Use the notes in pitches.h on the previous page to help you look at frequency ranges to play.

Applications

Ways in which light to sound may be used:

- Night Alarm - if you go to sleep but want to be awakened if someone enters a room.
- Box Alarm - if you want to be alerted if someone goes into your lunchbox, toolbox, etc. place Circuit playground inside and it will make a high pitched sound when someone opens the box.
- Musical Instrument - be the Leo Theremin of the 21st century

Temperature to Sound

A useful use of sensor and actuator on Circuit Playground is to use the temperature sensor and the speaker.

There are two modes you might want to consider in monitoring temperature and using sound:

- A sharp transition - for example, if the temperature hits a specific value like 79 degrees Fahrenheit / 26 degrees Celsius (a common thermostat value), a single tone is played. Or if a refridgerator/dring cooler gets above a certain temperature.
- A changing tone - changing tyemperatures make a change in sound, useful if you are monitoring temperature and also doing something else, you can hear if a temperature is going up or down, even while working on something else.

Both of these may be programmed in the same sketch - we can use the slide switch to chose one or the other:

```
// Adafruit Circuit Playground - Temperature to Sound  Support Open Source, buy Adafruit
//   2016-08-07 Version 1  by Mike Barela for Adafruit Industries
// Uses the CircuitPlayground library to easily use the full functionality of the board

#include <Adafruit_CircuitPlayground.h>

const float alertTemp = 90.0;  // temperature to alert on (use 32.0 for a freezer etc.)

void setup() {
  CircuitPlayground.begin();   // initialize the Circuit Playground library
  Serial.begin(9600);
}

void loop() {
   float temp;
   uint16_t sound;
   if(CircuitPlayground.slideSwitch()) {       // if the slide switch is at "+"
     temp = CircuitPlayground.temperatureF();  //   read the light sensor
     Serial.println(temp);
     sound = (int) map(temp, 70.0, 100.0, 131.0, 1760.0);   // map light to music values
     CircuitPlayground.playTone(sound, 1000);   //   play sound fora second
   }
   else {   // switch set to "-" for absolute temperature measurement
     temp = CircuitPlayground.temperatureF(); // read the light sensor
     Serial.println(temp);
     if( temp > alertTemp ) {   // if the read temperature is > your prepicked alartTemp
       CircuitPlayground.playTone(3520, 1000); // play sound for a second
     }
   }
}
```

If you are more comfortable with Celcius, use the CircuitPlayground.temperature function. I picked

the temperatures to be reactive around body temperature for demonstration. The temperature is printed on the Arduino serial monitor so you can read the temperature while working with the code and adjusting values.

With the slide switch at "-", you should hear no sound if the air (ambient) temperature is less than 90 degrees. Place your fingertip on the temperature sensor (where the little thermometer is) and it will heat up and the CP will make a sound when it gets above 90 degrees (the body is at 98.6 degrees F). Take your finger off and lightly blow on the sensor, the sound should stop as the temperature goes below 90 degrees.

With the slide switch on "+", behavior changes. Circuit Playground will make a continuous tone in response to the ambient temperature. You can use your finger again to heat up the sensor. The tone should get higher as it heats up. TBlow on the sensor to cool it down, and the tone will go lower.

Motion to Sound

The accelerometer at the center of the Circuit Playground board allows for some great effects when it comes to using lights and sound.

The code below reads the accelerometer and outputs sound depending on the speed of motion.

```
// Adafruit Circuit Playground - Movement to Sound    Support Open Source, buy Adafruit
//   2016-08-07 Version 1  by Mike Barela for Adafruit Industries
// Uses the CircuitPlayground library

#include <Adafruit_CircuitPlayground.h>

void setup() {
  CircuitPlayground.begin();   // initialize the Circuit Playground library
  Serial.begin(9600);
}

void loop() {
   float movementX, movementY, movementZ, movement;
   uint16_t sound;
   if(CircuitPlayground.slideSwitch()) {   // sense & play when slide whitch at "+"
      movementX = abs(CircuitPlayground.motionX());  // read the X motion (absolute value)
      movementY = abs(CircuitPlayground.motionY());  // read the Y motion (absolute value)
      movementZ = abs(CircuitPlayground.motionZ());  // read the Z motion (absolute value)
      movement = movementX + movementY + movementZ;  // aggregate the movement sensed
      Serial.println(movement);
      sound = (int) map(movement, 8.0, 60.0, 440.0, 1760.0); // map movement to music values
      CircuitPlayground.playTone(sound, 500);   //   play sound for 500 milliseconds
   }
}
```

The more rapidly you shake your Circuit Playground, the higher the pitch of sound it output. Be careful of your power cord or battery pack while swinging it around.

Any movement in the X, Y, or Z axis is detected. The abs function makes movement in either direction (negative or positive) positive as we're just looking to detect movement, not movement in a specific direction. All movement is added together and printed out on the serial monitor for inspection. The movement is mapped from values observed from 8 to 60 (you can change these if you observe different minimum and maximum values). I picked the low A and higher A notes from previous examples but you can use any frequency range you want.

Applications

- Bicycle sounds - connect to your bike spokes near the tire/rim side. Use tie wraps or other connection to ensure it does not slip of fall off or your battery pack does not fly off. Now the faster you pedal, the higher the sound.
- Skate sounds - placed on your board, it will make different sounds based on how hast you are

going.
- Drop sensor - makes a sound if Circuit Playground detects it has fallen.

Simon Says Game

One of the more fun electronic games of childhood is Simon Says. The game has 4 buttons with different color lights beside each button. The game will play a sequence and you have to repeat it. Every time you copy Simon correctly, the game makes the sequence you need to repeat one longer. Keep repeating the pattern until you get through all the levels (you win!) or miss a pattern (and you lose).

The retail Simon games have sounds in addition to lighted buttons. So will we, with Circuit Playground! The sequences get longer by one for each round - can you remember them all?

```
// Adafruit Circuit Playground - Simon Game    Support Open Source, buy Adafruit
//   2016-08-07 Version 1  by Mike Barela for Adafruit Industries
// Uses the CircuitPlayground library to easily use the full functionality of the board
// Based on Simon Sings by @TheRealDod with permission http://goo.gl/ea4VDf

#include <Adafruit_CircuitPlayground.h>  // Library for Circuit Playground functions
#include "pitches.h"                     // File for musical tones

const int NLEDS = 4;
const int LEDPINS[NLEDS] = {1,3,6,8};    // The NeoPixels used (counterclockwise from USB)
const int SWITCHPINS[NLEDS] = {2,0,6,9}; // Capacitive inputs 1-4 (match the NeoPixel positions)

const int FADESTEPS = 8;
const int FADEINDURATION = 200;
const int FADEOUTDURATION = 150;
const int SEQDELAY = 50;    // Millis between led flashes.
const int PAUSEB4SEQ = 500; // Millis before starting the sequence.
const int MINLEVEL =  2;
const int MAXLEVEL = 16;
int gameLevel;
int simonSez[MAXLEVEL]; // sequence of 0..NLEDS-1

const int16_t CAP_SAMPLES   = 20;      // number of samples for each capacitive input pad
const int16_t CAP_THRESHOLD = 300;     // Threshold for a capacitive touch (higher = less sensitive)

// -- song-array note fields --
// Tone
const int NOTETONE = 0;
const int SILENCE = 0;
const int ENDOFSONG = -1;
// Duration
const int NOTEDURATION = 1;
const int SINGLEBEAT = 125; // Note duration (millis) is multiplied by this
const float PAUSEFACTOR=0.2; // relative length of silence after playing a note
// LED
const int NOTELED = 2;
const int ALLLEDS = -1;

const int NOTES[NLEDS] = {NOTE_C4, NOTE_D4, NOTE_E4, NOTE_F4}; // Notes for each LED/Switch
```

```
int CORRECTSONG[][3] = {    // song to play if you mimiced the sequence correctly
   {SILENCE,2,ALLLEDS}
  ,{NOTE_G4,1,ALLLEDS}
  ,{NOTE_G4,1,ALLLEDS}
  ,{NOTE_A4,2,ALLLEDS}
  ,{ENDOFSONG,ENDOFSONG,ENDOFSONG}
};

int WINSONG[][3] = {       // song to play if you win the entire game
   {SILENCE,2,ALLLEDS}
  ,{NOTE_E4,1,2}
  ,{NOTE_E4,1,2}
  ,{NOTE_E4,1,2}
  ,{NOTE_F4,1,3}
  ,{NOTE_E4,1,2}
  ,{NOTE_D4,3,1}
  ,{NOTE_G4,1,ALLLEDS}
  ,{NOTE_G4,1,ALLLEDS}
  ,{NOTE_G4,1,ALLLEDS}
  ,{NOTE_A4,2,ALLLEDS}
  ,{NOTE_G4,5,ALLLEDS}
  ,{ENDOFSONG,ENDOFSONG,ENDOFSONG}
};

int LOSESONG[][3] = {   // notes to play if you don't mimic correctly and lose the game
  {NOTE_B5,2,3},{NOTE_A5,2,2},{NOTE_GS5,2,1},{NOTE_G5,8,ALLLEDS},{ENDOFSONG,ENDOFSONG,ENDOFSONG}
};

//============================================================================================

void setup() {
  CircuitPlayground.begin();           // initialize the Circuit Playground library
  CircuitPlayground.setBrightness(30); // we don't want the NeoPixels too bright
  Serial.begin(9600);                  // Serial monitor for debugging and info
  randomSeed(analogRead(4));           // random value based on sounds
  gameLevel=MINLEVEL;                  // start game at the minimum guess level
  playWinSequence();                   // Visual feedback after reset.
}

void loop() {
  int done;
  initGameSequence(gameLevel);        // Set up moves for new game
  done = 0;
  while (!done) {                     // set up to loop while playing
    while( !CircuitPlayground.leftButton() && !CircuitPlayground.rightButton() ) ; // wait to start
    delay(PAUSEB4SEQ);
    playGameSequence(gameLevel);      // Play the sequence to user
    if (playerGuess(gameLevel)) {     // See if person repeated the same sequence
      playCorrectSequence();          //   You did it right, make it harder by 1 move
      done = 1;
      if(gameLevel < MAXLEVEL) {      //   Increasing level by 1
        gameLevel++;
      }
      else {                // You played all the levels
        playWinSequence();  // You won the entire game!
        while(1) ;          // Press Circuit Playground Reset to restart
```

```
      }
    }
    else {
      playLoseSequence();    // You didn't get it right, sorry
      gameLevel = MINLEVEL;  // Reset to the starting level
    }
  }
}

void initGameSequence(int gameLevel) {    // Set the values for the values to mimic
  // assertion: gameLevel<=MAXLEVEL
  if( gameLevel == MINLEVEL ) {           // Minimum level all random
    for( int i=0; i < gameLevel; i++ ) {
      simonSez[i] = random(NLEDS);        // Min - select all new random pattern
    }
  }
  else {
    simonSez[gameLevel-1] = random(NLEDS); // add one more random value for next level
  }                                       //  which is different than some variations of game
}

void playGameSequence(int gameLevel) {     // Play the sequence to mimic
  Serial.print("Try this: ");
  for (int i=0; i < gameLevel; i++) {
    playLed(simonSez[i]);                 // Light LED and play its note
    Serial.print(SWITCHPINS[simonSez[i]]);
    if( i != gameLevel-1 )
       Serial.print(", ");
    else
       Serial.println(" ");
  }
}

void fadeLed(int theLed,int val,int duration) {  // fade NeoPixels in or out
  int fadeStep  = 256/FADESTEPS;
  int fadeDelay = duration/FADESTEPS;
  for(int i=0; i < 256; i+=fadeStep) {
    if( theLed >= 0 ) {
      lightPixel(theLed, val?i:255-i);
    }
    else { // ALLLEDS
      for(int j=0; j < NLEDS; j++) {
        lightPixel(j, val?i:255-i);
      }
    }
    delay(fadeDelay);
  }
  // force val (in case fadeStep doesn't divide 256)
  if( theLed >= 0 ) {
      lightPixel(theLed, val);
  }
  else {
    for(int j=0; j < NLEDS; j++) {
      lightPixel(j, val);
    }
  }
```

```
}

void playLed(int theLed) {                          // Fade LED and play its note
  if(CircuitPlayground.slideSwitch()) {             // Only play song if slide switch on "+"
    CircuitPlayground.playTone(NOTES[theLed],100);
  }
  fadeLed(theLed,HIGH,FADEINDURATION);              // Fade in LED
  fadeLed(theLed,LOW,FADEOUTDURATION);              // Fade out LED
}

int playerGuess(int gameLevel) {        // Get the user's guess and compare to Simon's sequence
  for (int i=0 ; i < gameLevel ; i++) {
    int guess=getSwitchStroke();
    //Serial.print(guess,DEC);
    //Serial.print(",");
    //Serial.println(simonSez[i]);
    if (guess!=simonSez[i]) {
      return 0;
    }
    else {
      playLed(guess);                   // Fade LED and play its note
    }
  }
  return 1;
}

void playSong(int song[][3]) {          // Play a predefined song sequence on CP speaker
  for (int note=0; song[note][NOTETONE]!=ENDOFSONG; note++) {
    int theDuration=SINGLEBEAT*song[note][NOTEDURATION];
    int theTone=song[note][NOTETONE];
    if ( theTone && CircuitPlayground.slideSwitch() ) {
      CircuitPlayground.playTone(theTone, 500);
    }
    int theLed=song[note][NOTELED];
    fadeLed(theLed,HIGH,theDuration);           // Fade in
    fadeLed(theLed,LOW,theDuration*PAUSEFACTOR); // Fade out + silence between note
  }
}

int playWinSequence() {     // Play the winning song and light sequence
  playSong(WINSONG);
  Serial.println("Win!");
}

int playLoseSequence() {     // Play the loosing song and light sequence
  playSong(LOSESONG);
  Serial.println("Loss");
}

int playCorrectSequence() {  // Play the song and light sequence for a correct move
  playSong(CORRECTSONG);
  Serial.println("Correct Repeat");
}

int getSwitchStroke() {     // Code to get a switch entry
  while (get1stPressedSwitch()>=0) {
```

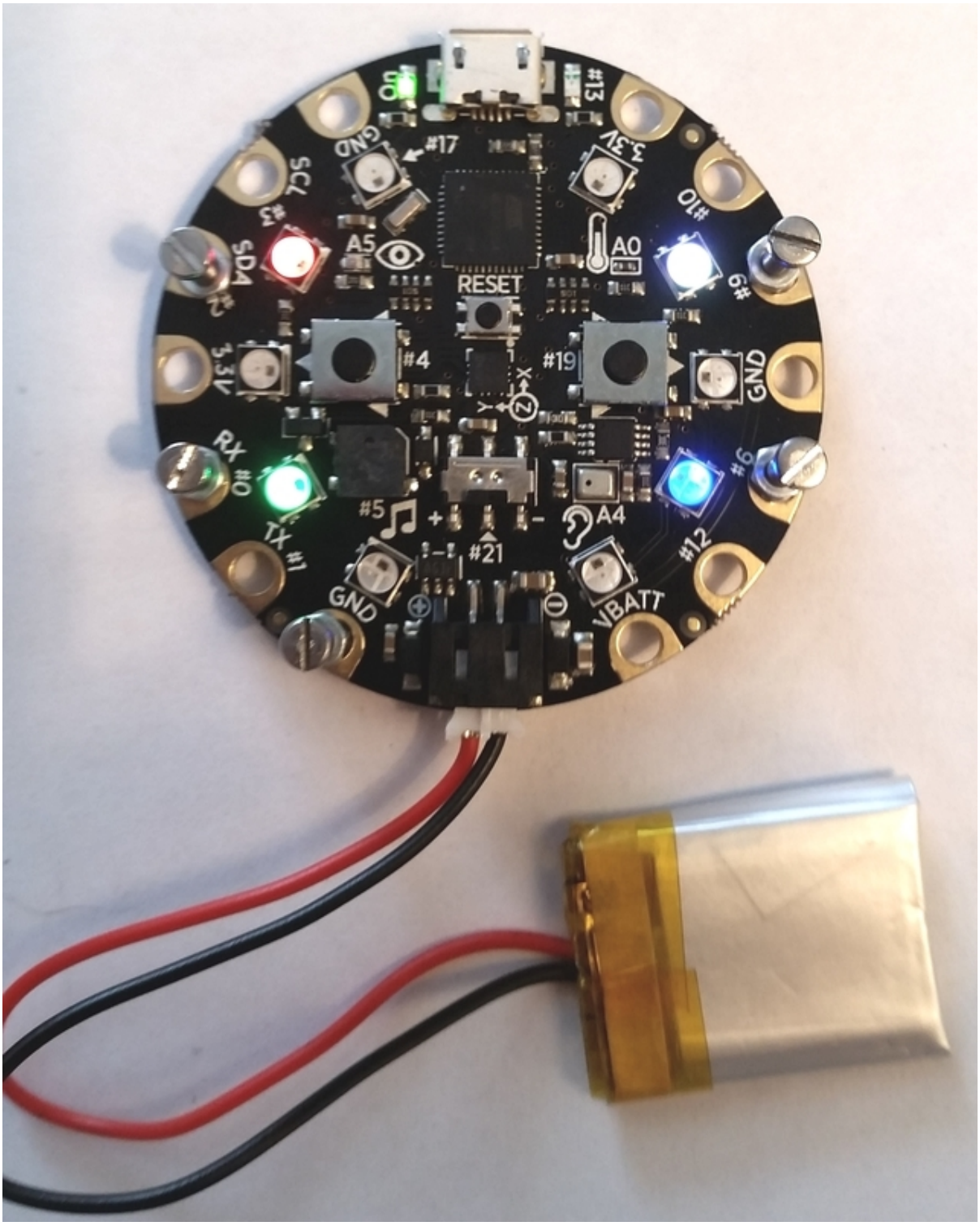
```
    // flush everything until no switch is pressed
    delay(50);
  }
  while (get1stPressedSwitch()<0) {
    // wait for next press
    delay(50);
  }
  return get1stPressedSwitch();
}

int get1stPressedSwitch() {   // Poll the switches, return the switch being pressed or -1 for none
  for (int i=0; i<NLEDS; i++) {
    if(CircuitPlayground.readCap(SWITCHPINS[i], CAP_SAMPLES) >= CAP_THRESHOLD) { // read CP capacitive
//      Serial.print("Switch ");                                    // switchpad
//      Serial.print(SWITCHPINS[i]); // print switch presses if you like
//      Serial.print(", ");
//      Serial.println(i);
      return i;                     // return number of the switch pressed
    }
  }
  return -1;                         // no switch pressed this poll cycle
}

void lightPixel(uint8_t ledToLight, uint8_t intensity) {  // light the NeoPixel at passed intensity
   switch(ledToLight) {
     case(0):
       CircuitPlayground.setPixelColor(LEDPINS[ledToLight],intensity,0,0);  // red
       break;
     case(1):
       CircuitPlayground.setPixelColor(LEDPINS[ledToLight],0,intensity,0);  // green
       break;
     case(2):
       CircuitPlayground.setPixelColor(LEDPINS[ledToLight],0,0,intensity);  // blue
       break;
     case(3):
       CircuitPlayground.setPixelColor(LEDPINS[ledToLight],intensity,intensity,intensity);  // white
       break;
   }
}
```

Game Play:

Press the left or right button pushbutton to start the round. Simon will play a sequence. You then mimic the sequence by touching the pads next to the Neopixels Simon lit. The four buttons are SDA #2, RX #0, #6 and #9. If you repeat the sequence right, the lights flash and Circuit Playground plays a brief tone. The game will then increase the difficulty level by one more light. This variation keeps the lower sequences the same, adding a random light to each round. If the tone is annoying, move the slideswitch to "-" to silence the tones. If you get to the maximum level and repeat it right, you've won and you get a long winning song! If you miss a sequence, the game plays a losing tone. To play a whole new game, press the RESET button.



The code uses the capacitive touch pads on Pins marked #2, #0, #6, and #9. When the LED next to these pads blinks (and plays a unique tone), you need to memorize the sequence. Then you play back the sequence by touching the above numbered gold pads next to the LEDs you saw. Get through the maximum level and you're the big winner. Great at keeping people entertained.

You can see I places small crews and nuts on the button pads - it makes it easier to press the right pads as you play. One more is put on ground. The capacitive switch effect works best with one

hand on ground and one hand's finger touching the button pads. Your body acts as a high resistance to electricity making a good capacitive sensing capability, similar to touchscreens on many phones and tablets.

You can make the game harder (like the original author @TheRealDod did for an Arduino version). Change initGameSequence to make each level unique - that modification requires alot more memorization.

Having the tones play makes the game easier to remember as you go through the sequences - memory association is stronger with multiple senses, in this case position, light, and sound. You can turn the sound off with the slide switch (like in prior examples) but you may find playing requires more concentration with "just" your eyes on the lights. The power of association and music. Viva Circuit Playground's speaker.