# Code

## General Set Up:

* The Capacitive Keyboard requires the Adafruit\_MPR121 library found here:

<https://github.com/adafruit/Adafruit_MPR121>

* + Make sure the .h file is named "Adafruit\_MPR121.h"
* To edit and change code make sure to have the Teensyduino add-on for Arduino <https://www.pjrc.com/teensy/td_download.html>
* Download main.ino from the Capacitive Keyboard Folder, (Community Engagement 🡪 On the Road 🡪 Demos-Shows-Supplies 🡪 Adafruit Musical Fruit 🡪 Capacitive Keyboard)
* Copy Audio files (in Capacitive Keyboard folder) to microSD card

## Main Variables

* Interval is how long the audio recording will be in ms
* touchedPins holds the output from the MPR121 board
* currentKeyboard is whichever keyboard is currently being used
* numNotes is how many notes are in the current chord
* cGain is the Gain for each mixer depending on how long the chord is
* chord[] holds notes in current array
* Audio Connections
  + 4 different AudioPlaySdWav because the max amount of notes in a chord is 4
  + 2 different AudioMixer4, need one for each side of output
  + audiooutputI2S for audio output
  + audioinputI2S ,audioRecordQueue, AudioPlaySdRAw, AudioAnalyzePeak, for audio recoding
  + patchChord1-patchCord10 set up audio out from playing SD .wavFiles
  + patchCord11-patchCord14 set up audio recording

## Setup Function

* sets up buttons are Inputs
* enables the audio board and SD card
* sets up MPR121 board as I2C device
* If the audio board or MPR121 board don’t set up currently, check connections and use the Wire library Scanner example code, include following lines to enable audio board:
* #include <Audio.h>
* AudioOutputI2S audioOutput;

## createChord Function

* Input: notes (output of MPR121 board), chord (array to store notes)
* Clears previous chord by replacing old chord with zeros
* Resets numNotes to 0
* Bitwise reads notes (the output of the MPR121 board is a 12 bit long binary number, if the position i has a 1 then input i of the board was pressed),
  + If it reads 1 then add currentNote to chord and increment numNotes by 1
* Returns numNotes

## playChord function

* Inputs: chord array, notes (number of notes returned from createChord function), cKeyboard (current keyboard)
* First changes gain for each mixer depending on how many notes in chord (notes variable)
* To find the corresponding .wav file to play in the 2D keyboards array, the first argument is cKeyboard and the second is chord[i], chord[i] stores the ith note in the chord
  + Ex, keyboards[cKeyboard][chord[2]] plays the second note in the chord

## Recording Functions

The following functions came from the Teensy Audio Example Recorder (<https://github.com/PaulStoffregen/Audio/tree/master/examples/Recorder>)

* startRecording()
* continueRecording()
* stopRecording()

The recordRaw() function calls the previous functions and records a sample that is as long as the inputted seconds variable

playRawFile() function plays a .RAW file in its entirety

## loop()

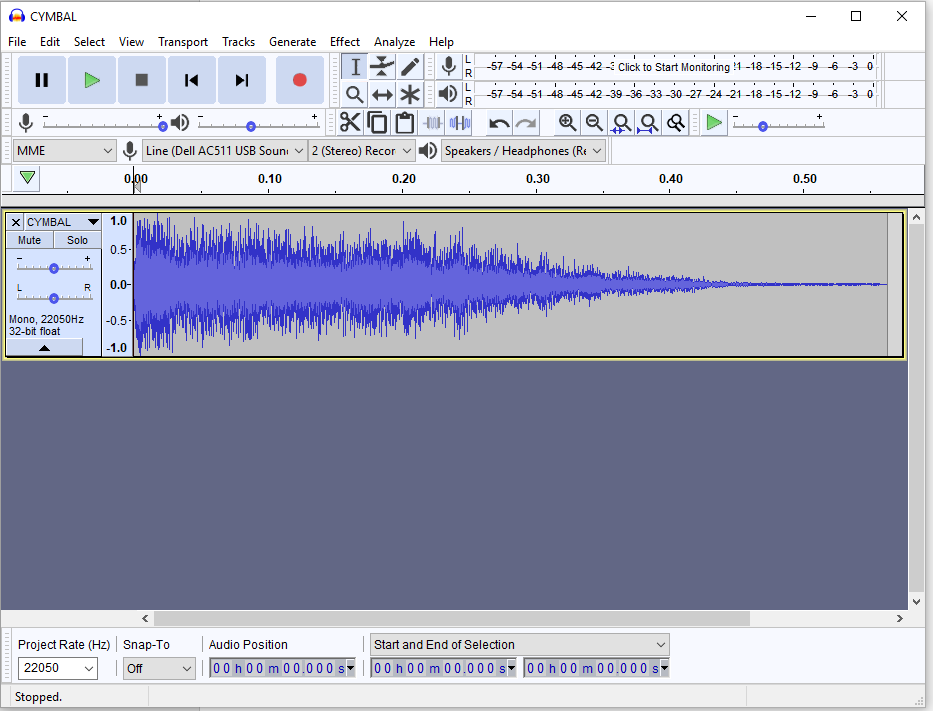
* checks if keyboard has been switched (the black keyboard\_switch button has been pressed)
  + changes keyboard by incrementing currentKeyboard by 1
* checks if the record button has been pressed and records a sample
  + changes currentKeyboard to keyboardLength (this means that the currentKeyboard will not be found in the keyboards 2-d array
* If the currentKeyboard equals keyboardLength then the current keyboard is the recorded sample one
  + As long as any pin has been pressed, play the recorded sample
* Otherwise call the createChord and playChord functions to play the current chord

# Audio File Sources:

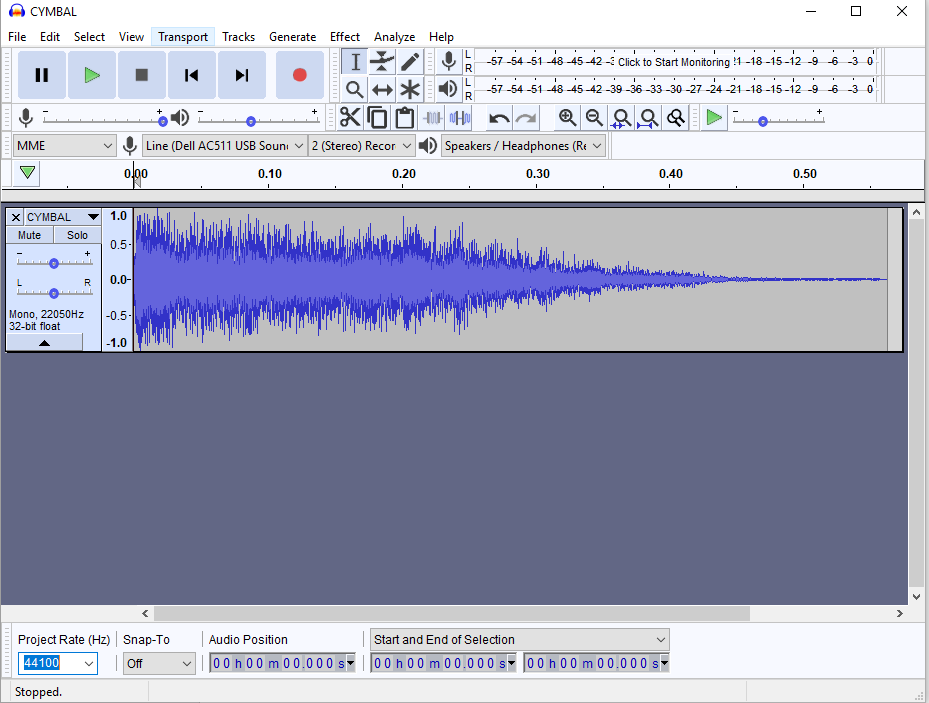
* Cats:: <https://freesound.org/people/tuberatanka/sounds/110011/>
  + The sample is then pitch shifted in Audacity to create new notes
* Piano Keyboard: auto generated tones in Audacity
* Drums: samples from Adafruit Playground Drum Machine (<https://learn.adafruit.com/adafruit-circuit-playground-express/playground-drum-machine>) and Tama Drum Kit by Snapper4298 (<https://freesound.org/people/Snapper4298/packs/11125/>)
  + Samples were adjusted and clipped to match microSD card requirements for Teensy Audio Board

# Adding New Keyboards

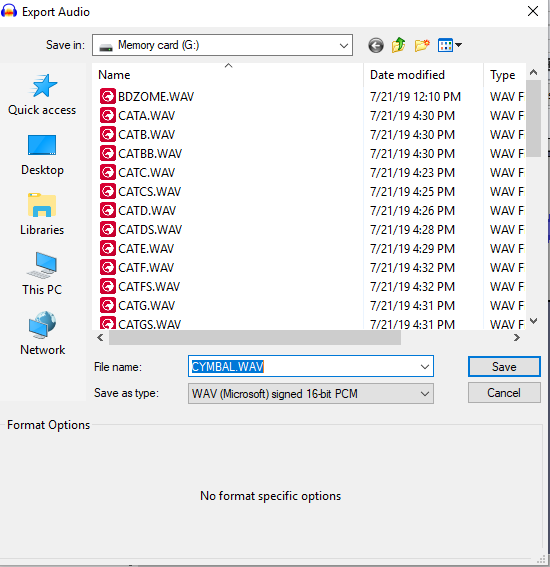
* To add a new keyboard you will need a mircoUSB cable, microSD card reader and 12 .wav files
* Make sure the .wav files are formatted as 16-bit 44.1kHz and the name is in 8.3 format
  + What does that mean and how do I make the file that if it’s not the case?
  + 8.3 format name means the name is in all caps and is no longer than 8 characters long and the file extension is up to 3 characters long EX. ELECBLIP.WAV is an acceptable/ideal name
  + It’s easy to fix the .wav format in Audacity:
    - Open file in Audacity, you can choose to import as a copy if you want to keep the original file as is
    - At the bottom left of the window there will be a Project Rate Hz, make sure it is 44100 Hz
    - export file as .wav, change the file name so the .wav is also capitalized (the file Save As Type should read “signed 16-bit PCM)
    - there’s no need to save the project or change the metadata of the file when saving



Original File with Incorrect Frequency



File with fixed frequency



* To access the microSD card on the Teensy Audio Board, unplug the microUSB cable connecting the Teensy 3.2 to the power supply, lift the battery pack up and remove the microSD card
* Using a microSD card reader, save the .wav files onto the microSD card and insert the microSD card back into the Audio Board
* Connect the Teensy 3.2 to a computer via microUSB
* Open up the Arduino software with Teensyduino and Adafruit\_MPR121 library installed (if not installed, follow instructions under General Set UP for Code
* Open the program main.ino from the Capacitive Keyboard Folder, (Community Engagement 🡪 On the Road 🡪 Demos-Shows-Supplies 🡪 Adafruit Musical Fruit 🡪 Capacitive Keyboard)
* Scroll to IF ADDING KEYBOARDS ONLY CHANGE THIS SECTION STARTS HERE
* Increase the keyboard Length variable by 1
* Add a comma after the } of the last keyboard
* Add the list of new .wav files, keep the keyboards spaced as pervious ones so it’s easy to read and modify

