ECE 5725 Final Project
Guanqun Wu gw284, Zhaopeng Xu zx273
Week 2 Progress Update / Proposal
Submission Date: 22 Nov 2019
Thursday Lab

Piano Game

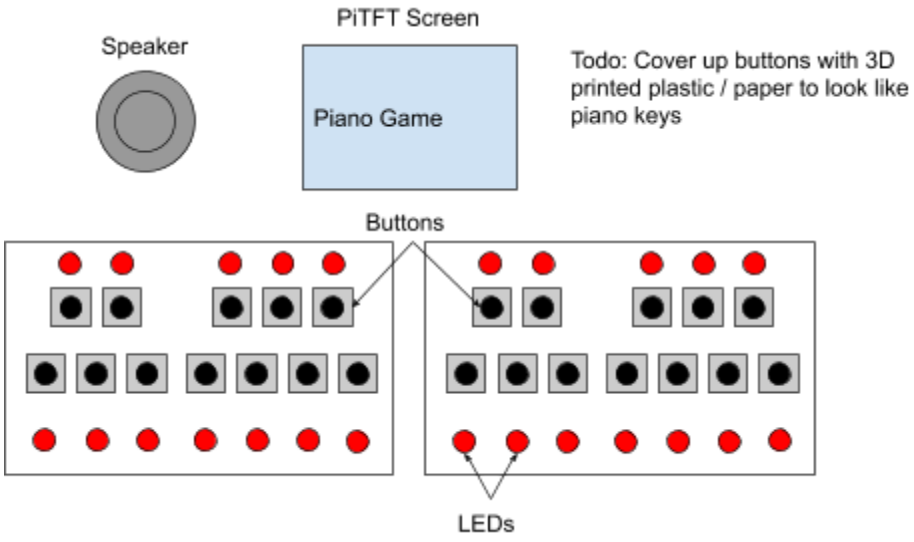
Bill of Materials

2 x Breadboard	Take from Lab	$0
30 x 12mm Buttons	https://www.adafruit.com/product/1119	$7.50
25 x 5mm LEDs	https://www.adafruit.com/product/4203	$2.95
1 x 8Ω 1W Speaker	https://www.adafruit.com/product/1313	$1.95
1 x 16 Port Expander	https://www.adafruit.com/product/732	$2.95
1 x USB Microphone	https://www.adafruit.com/product/3367	$5.95
30 x 1kΩ Resistors	https://www.amazon.com/gp/product/B0185FGTSS/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1	$1.91
25 x 10kΩ Resistors	https://www.adafruit.com/product/2784	$0.75
1 x USB Speaker (?)	https://www.adafruit.com/product/3369	$12.50
Total		$34.46

Software Used

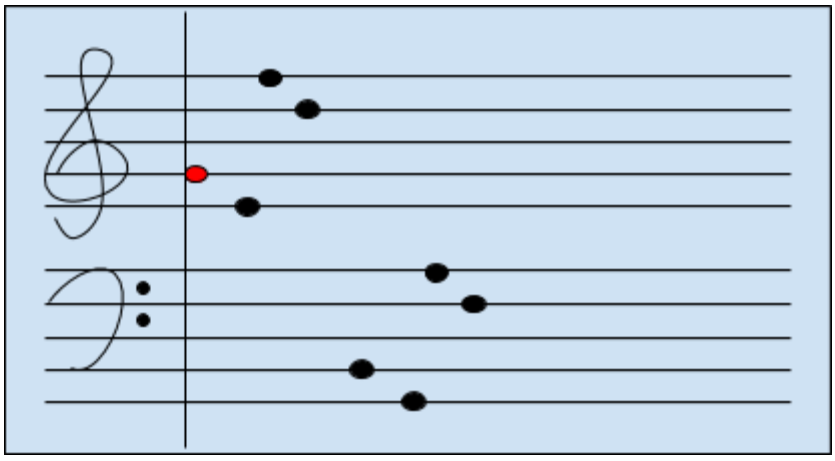
RPi.GPIO, (Possibly PiGPIO / wiringPi) => GPIO
Pydub => To mix sounds around and play them on a USB speaker
Pygame => To draw a GUI and display
Possibly a PostgreSQL database to store player information

Hardware



Using our LEDs, buttons, breadboard and port expander (Not enough ports on the Raspberry Pi to control 24 piano keys), we create 24 piano keys that each have their own LEDs and are each controlled by a port (physical or virtual). We attach our speaker to our Raspberry Pi and we use the speaker to play the appropriate note on the piano.

Training Mode

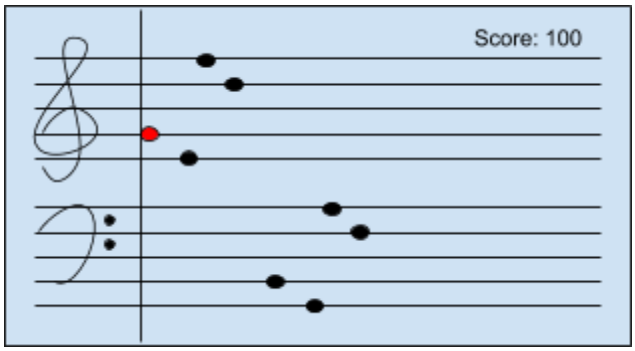


During training mode, we want to train the player to play a certain song. We have the vertical bar moving from the left to right of the screen that displays our current position in the song which would be based on the rhythm of the music. This bar would move along as the key is being played. If the wrong key is played, we would highlight the note in red. If the right key is played, we would highlight the note in green. We also use the LED lights to light up the correct key(s) on the piano that currently need to be played.

Listening Mode

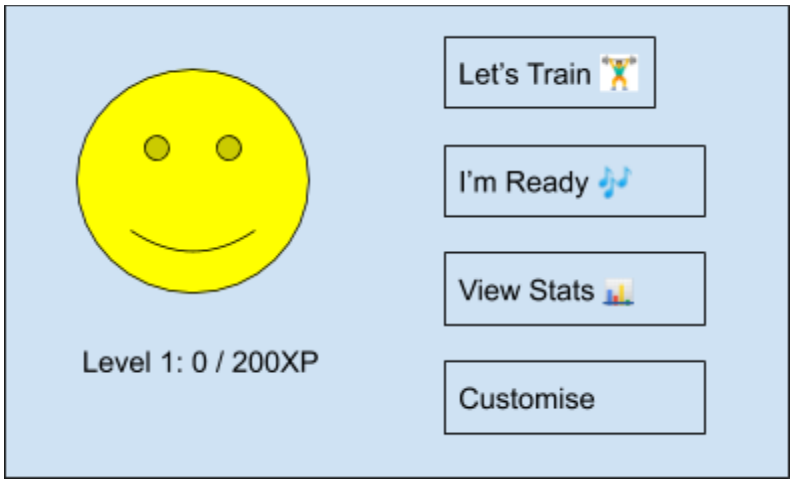
We will provide a listening mode which is a variation of the training mode except that the piano will light up all the required keys and play the music at the required pace and thus not require any input from the player. We will also allow the player to adjust the speed at which the music is played as well as the position of the music that we're playing (by enabling pausing and fast forwarding of the music). We could possibly combine the listening mode with the training mode so that a user is able to listen to part of the music first, then train on it.

Game Mode



In the game mode, the player would play the music at a set rate (ie at 1x speed, 0.5x speed etc). In this mode, the music bar would keep moving at the fixed rate and the player would need to keep up with the rhythm and are scored based on the number of notes they get correct and based on how well they time their playing with the actual timing of the song. Based on their final score, players would be assigned a rating for that particular piece.

UI and Gamification



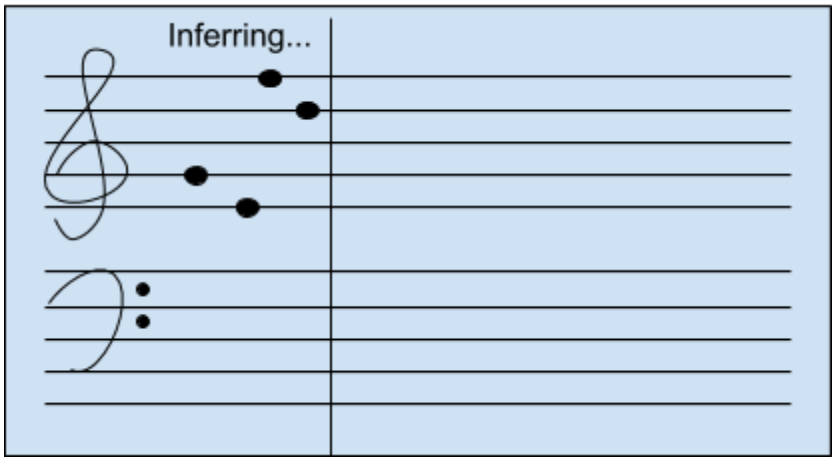
The screen above shows a possible UI screen that provides for the different modes. We are thinking of including some form of gamification that provides a level and some experience points. The user could perhaps earn experience for training on the piano during consecutive

days or getting good scores on certain songs. By levelling up, the user could unlock different customisations that allow them to, for instance have a cool background or a nice avatar.

When the user selects the training/game modes, they'll be presented with a screen that shows a list of song names as well as their grade and score on each of the songs. We'll probably come up with a way to sort these songs and we could probably jump to the next song that we recommend them to play and display that as the first song in the default display list.

View stats would allow the user to view their statistics on all the songs as well as for a particular song. Customise would allow the user to change their background image, their theme and their avatar.

Recording Music



We would like to add a feature using a USB speaker where the user can play a piece of music and our system would be able to read in the music, perform FFT, then create a piece from the music on which the player can practice. This might be difficult in practice because of background noise and the fact that pieces can overlay sounds from multiple instruments, but we will do our best to come up with a piano approximation for these songs.

Milestones and Schedule

14 Nov - 20 Nov => Confirm project idea, write up proposal, order parts, get button and speaker to work with simple square wave
21 Nov - 27 Nov => Produce more complex sounds, get some portions of GUI working. Get at least a single keyboard working
28 Nov - 4 Dec => Finish up, do whatever extensions that we have sufficient time to do. Get both keyboards working
5 Dec => Project Demo

Why this project is an embedded device / system

The Raspberry Pi is a central piece of our system because it provides a GUI for the user to interact with the piano game and it also handles all of the GPIO inputs and outputs for the user to interact with the piano game and produce sound on the speakers. If we were to power the Raspberry Pi with a portable charger / batteries, we would be able to bring this game around with us and practice anywhere.

Demo

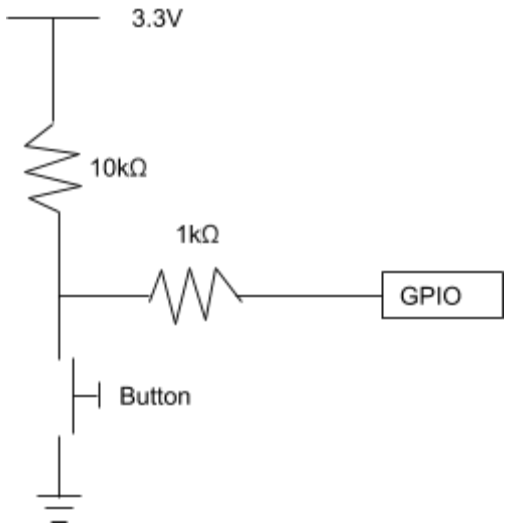
We expect to show off the ability of the piano to play music during the training mode and we expect to show off how we can interact with the piano during the training mode. We also expect to show ourselves playing a round of the piano game on game mode and showing that the game works correctly. We would also demo additional stretch goals if they're available.

Week 1 (14 Nov - 20 Nov) Progress

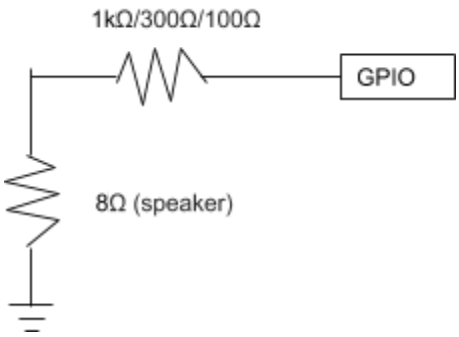
For this week, we've been working on a scaled down version of the piano circuit and have been tuning the 8Ω speaker to work correctly.

Our initial task was to wire a single button and to wire the external speaker and check that it can produce A4 (440 Hz) continuously while the button is pressed down and stop the sound while the button is lifted.

We used the following wiring diagram (taken from Lab 2) for the button.



We used the following wiring diagram for the speaker

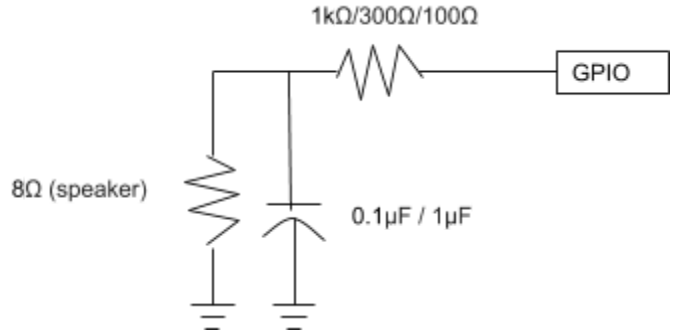


We tried different resistors for the speaker and we found that a 100Ω resistor produced the best results (a moderately loud sound). A 300Ω resistor produces acceptable results while a 1kΩ resistor is too soft.

We used RPi.GPIO to get the buttons to sync with the speaker. The button is active low. We use a callback on both the rising and falling edge of the button to test whether we should be playing music. We create a list that keeps track of what notes we're currently playing (could contain multiple elements if we're playing a chord which we can't currently do). We then create a lock on that list to preserve concurrency. When a button is pressed, we acquire the lock, then add the associated frequency to the list. We then look at the list's contents and play the frequency associated with the first element of the list (can't currently play chords).

We play the music by creating a square wave using RPi.GPIO's PWM functionality. We use a wave of frequency 440Hz with duty cycle of 50%. We noticed that we're unable to produce a stable frequency using software PWM with Python (The sound is somewhat distorted) so we would have to try C in order to produce a stable PWM. We switched to C instead and were able to get a stable wave.

Now we wanted to examine how we could produce an arbitrary sound wave. Unfortunately the Raspberry Pi does not have analog output so we're unable to produce such a wave through the usual method. We instead try to approximate a continuous wave using PWM and a low pass filter. We update our speaker circuit by adding a capacitor to act as a low pass filter and to smoothen out our waveform.



We ran our program at the fastest interval we can simulate and produce a stable waveform while computing sin(2πft) and using clock_nanosleep (around 15kHz). We then vary the PWM

duty cycle by the value of sin(2πft), normalising its range to be within 0 and +1. We tried frequencies above 15kHz, but we were unable to get an accurate duty cycle using those frequencies. We then tried a 0.1μF capacitor using our 100Ω resistor but were unable to get much smoothing out of it. We then used a 1μF capacitor and a 300Ω resistor and were able to get a waveform that resembles a sine wave but with jagged edges around it. We realised that we need a higher frequency (~60kHz) to get a smooth sine wave and so would need to resort to using just a for loop and using the Preempt RT libraries to gain priority. We haven't tried that yet and will try that next session. We would also need some way for our python program to communicate with our C program and would probably have to use a FIFO to do that.
Another issue we observed is that the impedance on the speaker itself is not purely resistive and is somewhat capacitive and this produces audio distortion. We would need to use an amplifier to remove this capacitive effect and we'll try that next session with a MOSFET / BJT. I've ordered in a USB Speaker and we're considering switching to that and grabbing some wav files from a bunch of piano recordings then using pydub to compose them together if we're unable to get this speaker circuit working correctly (We don't have much time to get it to work because we only have 3 weeks for this project from 14 Nov - 5 Dec).

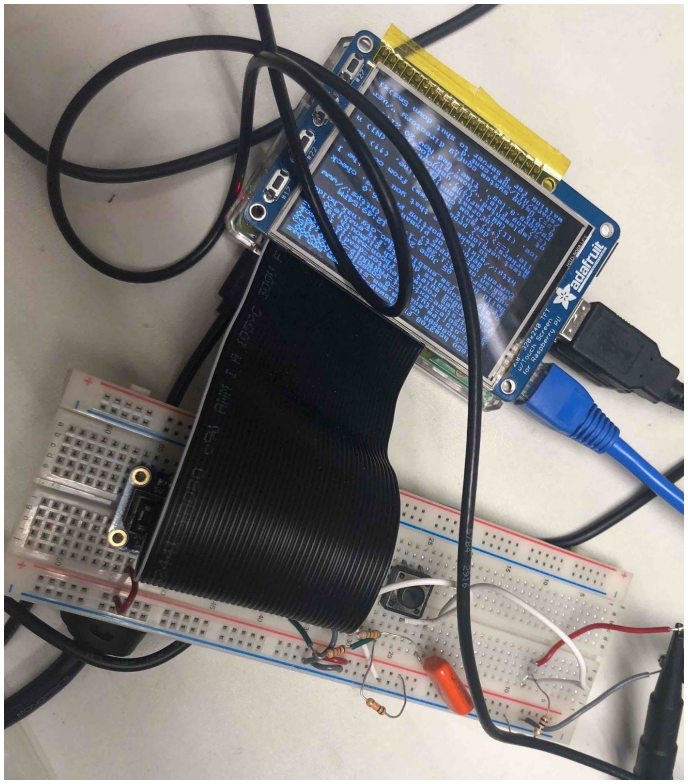


Image of our testing circuit

For the coming week, we expect to try fixes for the problems above and then work on the GUI and wire up our piano keys to get some semblance of a piano game working.

The project turned out to be harder than we expected (especially with respect to the speaker and its real time nature). We're currently behind schedule but we plan to make up for it by going to lab on weekends and during the Thanksgiving break. I had a prelim this week and had to write up Lab Report 4 so I'm quite busy and I think I'll have more time during the next week to work on our project.