Table of contents

Introduction

The goal of the Squidbox project is to develop a low-cost Bluetooth MIDI controller that triggers velocity-sensitive diatonic chords. This project draws on ideas developed in the Chord Board and Bluetooth Potentiometer projects to develop a new instrument. In short, the intended design features at least 10 buttons: 8 velocity-sensitive buttons to trigger the 8 diatonic chords in a chosen scale, and two buttons to transpose up and down by half-steps. Velocity sensitivity refers to the ability of a MIDI instrument to record the velocity or force with which a key is pressed, typically to control the volume of a note.

Project overview

This project involved consideration of various different design aspects, giving rise to several closely related but independent sub-projects: physical design, electrical design, software and MIDI control, and velocity-sensitive button design. Originally, the intended goal was to develop and present a fully functioning Squidbox with 8 velocity-sensitive buttons. Developing a robust velocity-sensitive button proved challenging, so the scope of the project was refined into presenting two separate deliverables: a Bluetooth MIDI controller with traditional (non-velocity-sensitive) buttons and a prototype of a single velocity sensitive key, with software capable of unifying the functionalities of both.

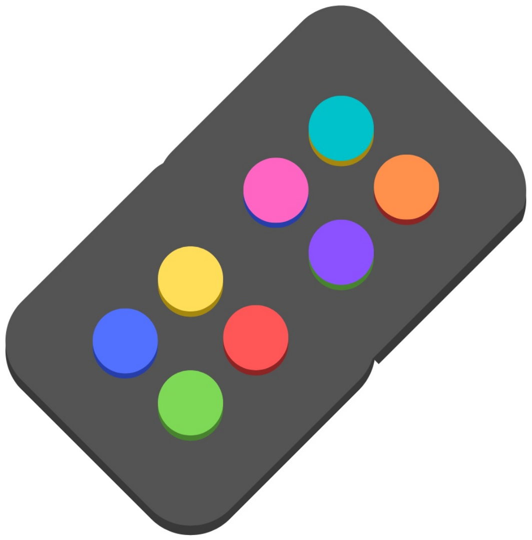


Figure 1. A basic visualization of the Squidbox controller

Our first prototype consisted of a piezo buzzer and three small Arduino buttons: two transposition buttons and a button to trigger a tone. Transposition was handled by multiplying a root pitch by the equal-temperament half-step ratio of $2^{1/12}$. The software was modified so the button could play a major chord arpeggio on the buzzer, using true temperament pitch ratios above the root pitch (Figure 2). The second prototype expanded the hardware to eight buttons that played the diatonic chords on a piezo buzzer, with transposition moving all eight chords up or down a half step. We then developed designs to transition to MIDI control over USB, first an automatic loop from the Arduino, then a single button control, then all eight buttons sending melodic MIDI values. Finally, we modified the codebase to send MIDI information over Bluetooth Low Energy (BLE), which can be received by smartphone synth applications as well as by a DAW given an appropriate MIDI over Bluetooth software bridge. We also made efforts towards designing and testing a velocity sensitive button using capacitive touch sensors.

I	ii	iii	IV	V	vi	vii dim
0	2	4	5	7	9	11
4	5	7	9	11	12	14
7	9	11	12	14	16	17

Figure 2. Chart for calculating diatonic chords from a root note in a major scale

Velocity Sensitivity

Velocity sensitivity is the ability of a MIDI instrument to register the speed at which a user presses any musical key. Typically, this functionality allows for the user to control the dynamics of the instrument similarly to a physical piano, but velocity data can also be used to modify other aspects of the note, such as intensity of an effect. Velocity sensitivity differs subtly from pressure or force sensitivity, which continuously measure the force of a key press, even after the note is sent, allowing for “aftertouch” effects such as volume swells.

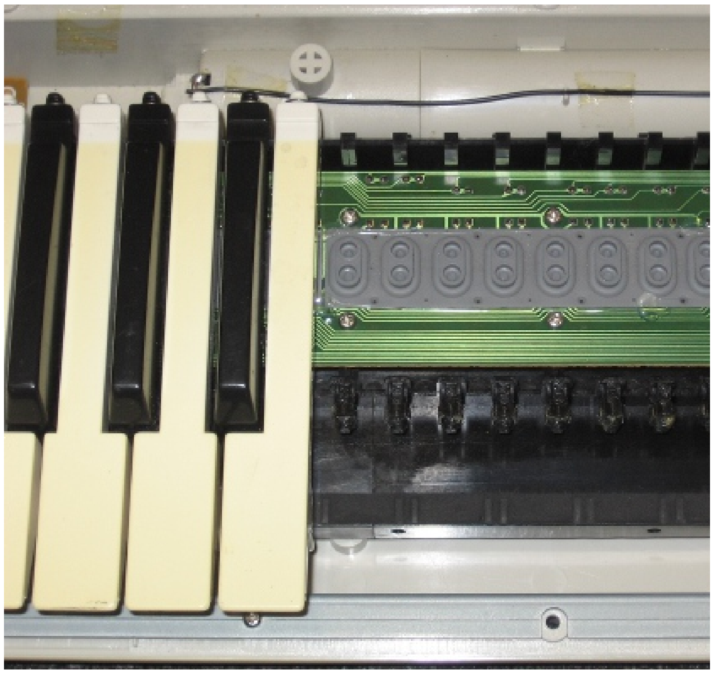


Figure 3. A linear array of pairs of silicone domes in a velocity sensitive keyboard

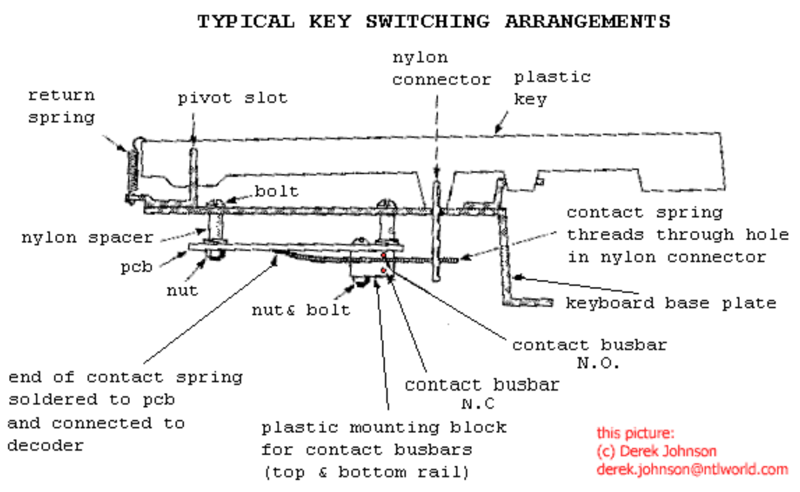

TYPICAL KEY SWITCHING ARRANGEMENTS
nylon connector
plastic key
return spring
pivot slot
bolt
contact spring threads through hole in nylon connector
nylon spacer
pcb
nut
keyboard base plate
nut & bolt
contact busbar N.O.
end of contact spring soldered to pcb and connected to decoder
contact busbar N.C
plastic mounting block for contact busbars (top & bottom rail)
this picture: (c) Derek Johnson derek.johnson@ntlworld.com

Figure 4. Another velocity sensitive mechanism, a sheet of spring steel moves between two contacts (red dots) as the key is pressed

We can differentiate between "analog" and "digital" methods of implementing velocity sensitivity. Digital pianos, keyboards, and other similar instruments commonly follow the "digital" approach, in which velocity sensitivity is achieved by measuring the time difference between two button presses at different depths of a single instrument key press (Figure 3, 4). A lower time difference corresponds to higher velocity, as the key moves more quickly through its range of motion. The "analog" approach involves an explicitly pressure-sensitive device outputting an analog signal that is processed by the microcontroller. A naive example is a force-sensitive resistor (FSR), a component with conductance proportional to pressure. Though FSRs are sensitive to a very wide range of forces, they often suffer from slow response times, making them inadequate for use in musical instruments. For this reason, we followed the "digital" method, but future projects may investigate the effectiveness of Velostat, a rubber sheet material that is pressure sensitive, functioning similarly to pressure sensitive synth pads, which may offer better performance than FSRs.

Our approach towards establishing velocity sensitivity involved using capacitive touch sensors placed at different depths to achieve the digital method described above. More detail regarding our implementation of the touch sensors will be described in the following Electronics section.

Electronics

The development platform for this project was an Arduino MKR WiFi 1010, a Cortex M0+-based Arduino development board with several appealing onboard features. In addition to Micro-USB, the board natively supports WiFi, Bluetooth, and Bluetooth Low Energy (BLE) connectivity, and has a LiPo battery socket and onboard charging circuit. Programming is done over USB, but in this project typical operation is entirely wireless over BLE and powered by a LiPo battery.

The eight non-velocity sensitive musical buttons and two transpose buttons are connected between digital input pins and ground, and are configured to use the internal pull-up resistors of the microcontroller, so they are active-low buttons. In an initial prototype, a piezoelectric buzzer and series resistor was used as output on a PWM pin of the Arduino, before we configured MIDI output (Figure 5). In a later prototype, the buttons were migrated from a breadboard to a soldered perfboard for robustness and compactibility (Figure 6).

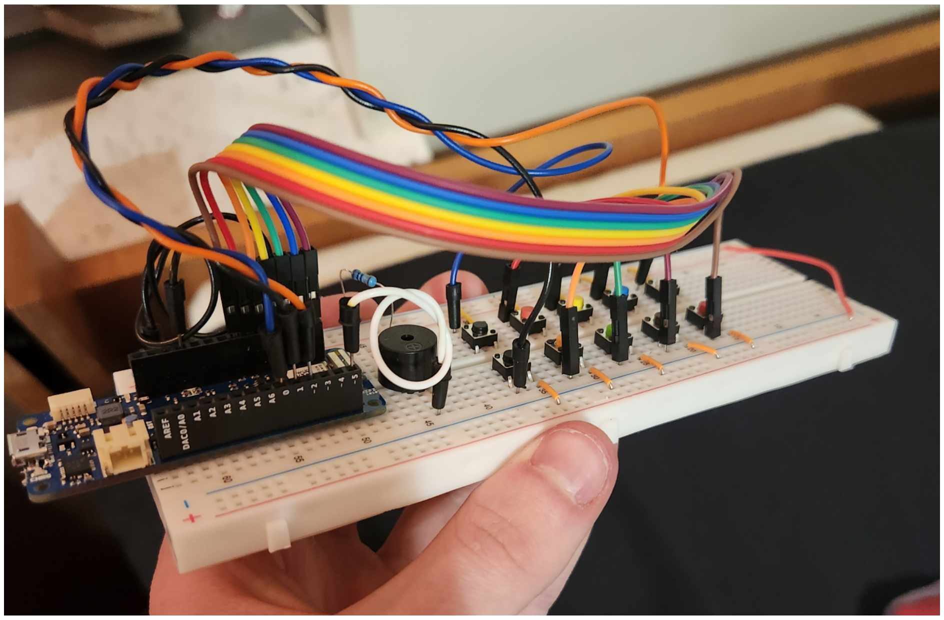


Figure 5. Initial prototype. L to R: MKR WiFi 1010, piezo buzzer, 10 buttons

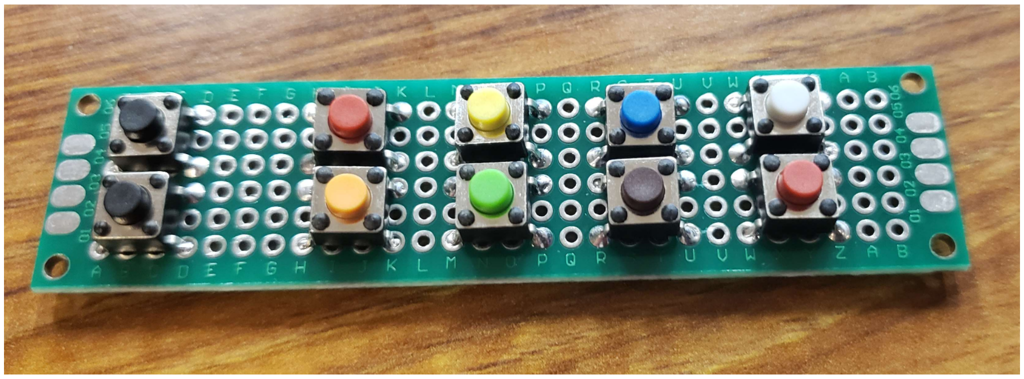


Figure 6. Soldered buttons. Two transpose buttons (black, left) and eight musical buttons

To design the velocity-sensitive buttons, we settled on a very interesting capacitive touch sensor, which detects the change in capacitance when an object comes close to a metal plate on the PCB, and outputs a high voltage signal when the sensor is triggered. The main advantages of the capacitive touch sensor is its sensitivity, requiring no force to actuate, absence of moving parts, and the lack of button bounce or noise that would need debouncing or filtering. Furthermore, this touch sensor registered most everyday objects, not limited to the human finger. Rather than a mechanical contact that bounces, an IC on the sensor outputs a clean 1 or 0 signal that can be read without any debouncing circuit. We would encourage future teams to consider designs using micro limit switches, which are mechanical switches with similar sensitivity and easy actuation but may be cheaper.

The touch sensor has three pins: +3.3 V power, GND, and signal. The signal pins go directly to a digital pin on the Arduino, and do not need pulldown resistors. Each key requires two sensors and thus two digital pins, the final design will require a total of 2x8+2 = 18 digital pins for all 10 buttons. The MKR WiFi 1010 has 21 digital pins, but that would involve using all the analog pins, which may not be desirable. Future groups may improve on this design with a number of approaches: multiplexing the buttons or designing a small discrete digital circuit that handles time-difference sensing and reads out each over serial communication.

Physical Design

The ultimate goals and main focuses of physical design were buttons and size. Ideally, the MIDI board would use buttons instead of keys for simplicity and size, and we wanted the device to have handheld capabilities. Although we began with initial designs for the "case", they had to be regularly updated as the formfactor of our buttons was continually changing. Ultimately, we put the shell off to the side and decided to focus our efforts on the MIDI functionality, and refining a design for velocity sensitive buttons. Our button designs went through many different iterations as our ideas for implementing velocity sensitivity changed (including limit switches, a simple lever, and others, shown in Figures 7 and 8). For our final velocity sensitive button design, we decided to "sandwich" two touch sensors back to back, using the top touch sensor to register the initial time of touch, and the second sensor to measure the time taken for it to be triggered. We accomplished this by having the top touch sensor uncovered and triggered by a finger, and when pushed down far enough, can trigger the bottom touch sensor. This is done with a bar attached to the bottom of the top button covered in aluminum foil to create a metal-on-metal contact that would best trigger the second button. To bounce the first button back into position, we placed springs in between the two buttons. This final design can be seen in Figure 9.

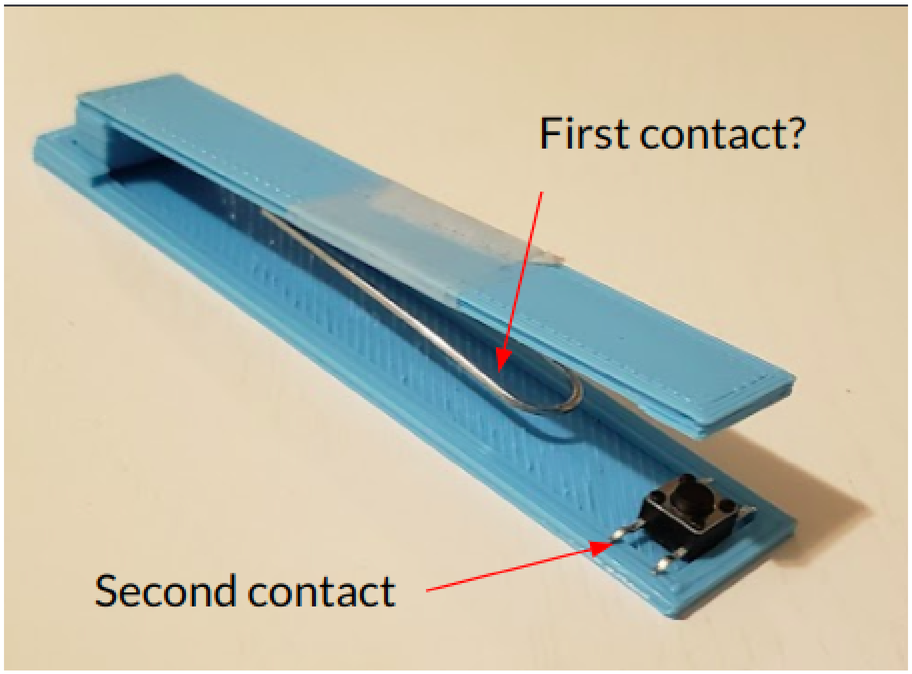


Figure 7: Simple lever design with paper clip and button as two contact points

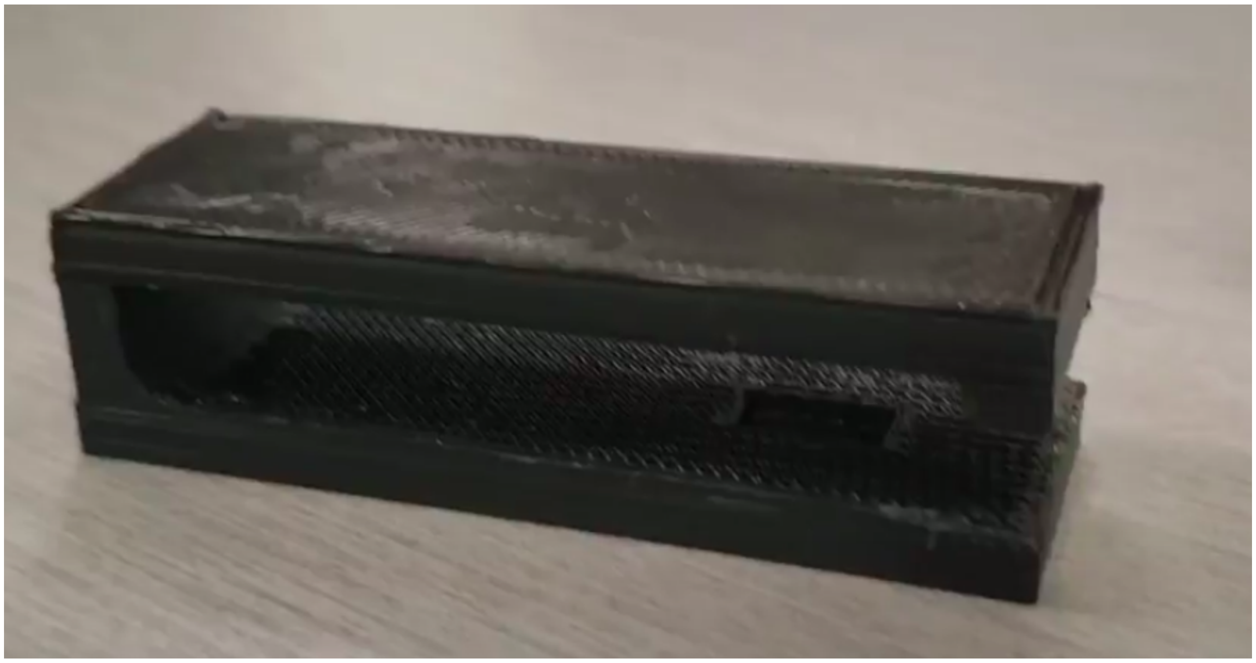


Figure 8: More robust lever button design for limit switches

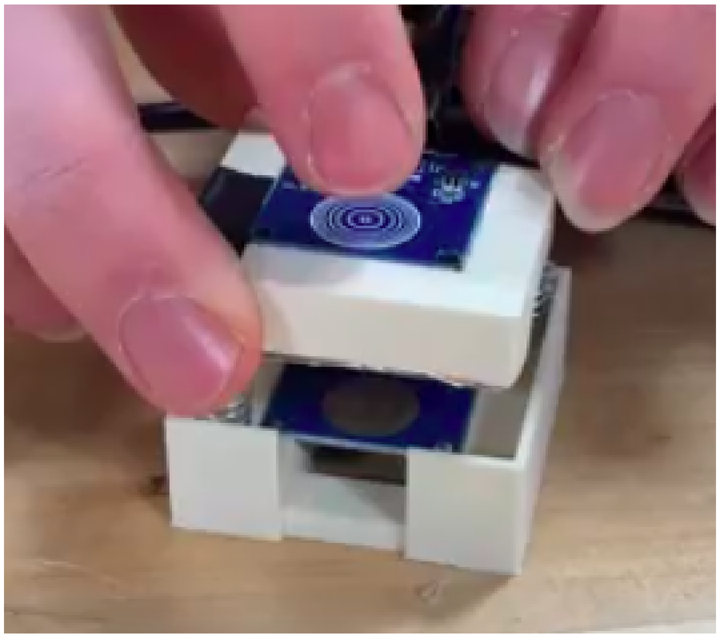
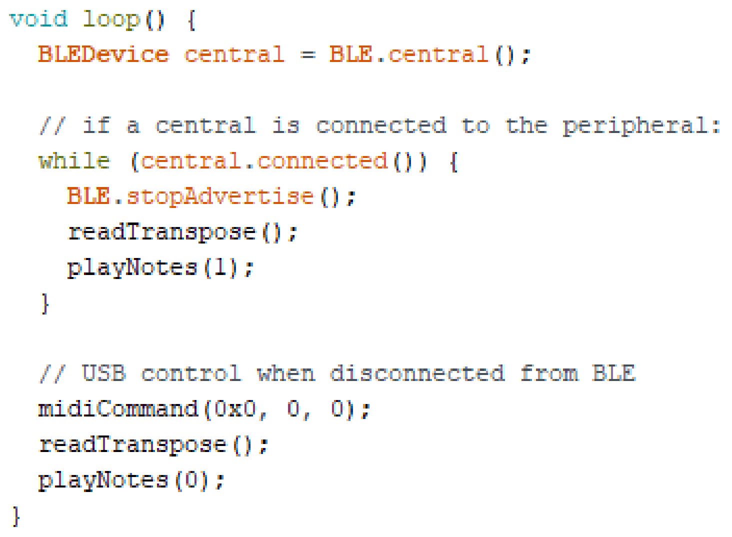


Figure 9: Final button design incorporating touch sensors

Code and BLE

For programming the MIDI device itself, we were able to take advantage of a few native Arduino libraries that offered excellent encapsulation of the MIDI data as well as the transmission methods used to send to a host machine: the MIDIUSB and ArduinoBLE libraries. The most essential and basic portion of the code lies in the definition of each button-pin combination on the Arduino along with a starting note value given to each (aside from two buttons, which acted as a way to “transpose” those starter note values up and down). For each iteration of the “loop()” function, our code would first attempt to establish a connection over Bluetooth Low Energy. This is done by having the Arduino, acting as the BLE Peripheral device, broadcast a local name and MIDI service for which any Central device (i.e. another computer) can establish a connection with and read MIDI data from. If this connection is unsuccessful, our code will then default to sending MIDI messages over the USB serial port to whatever device is connected to it.



```
void loop() {
  BLEDevice central = BLE.central();

  // if a central is connected to the peripheral:
  while (central.connected()) {
    BLE.stopAdvertise();
    readTranspose();
    playNotes(1);
  }

  // USB control when disconnected from BLE
  midiCommand(0x0, 0, 0);
  readTranspose();
  playNotes(0);
}
```

Figure 7. Main loop() function

Once pairing is successful and a transportation method has been established (both USB and BLE work the same), the board will listen to any button presses received by the pins and map to a specified note value, with the base values being the C major scale. A “note on” MIDI message is then packaged with the given note value, set to channel 1 and given a default velocity, and then received by the connected host device to be played. To prevent button presses from being read multiple times a second, an array of previously pressed notes is updated immediately when a button is pressed, which is then checked each loop to make sure that that note is still being pressed and to not resend the MIDI value over and over again. Upon lifting up from the button, the array is updated immediately to show that the button is no longer pressed and a “note off” MIDI message is sent. If a user would like to access notes above or below the C major scale, there are the previously mentioned transposition buttons which update all of the set values up or down a half step (up or down a value of 1 for actual MIDI translation), so that any notes between a MIDI value of 0 and 127 can be played.

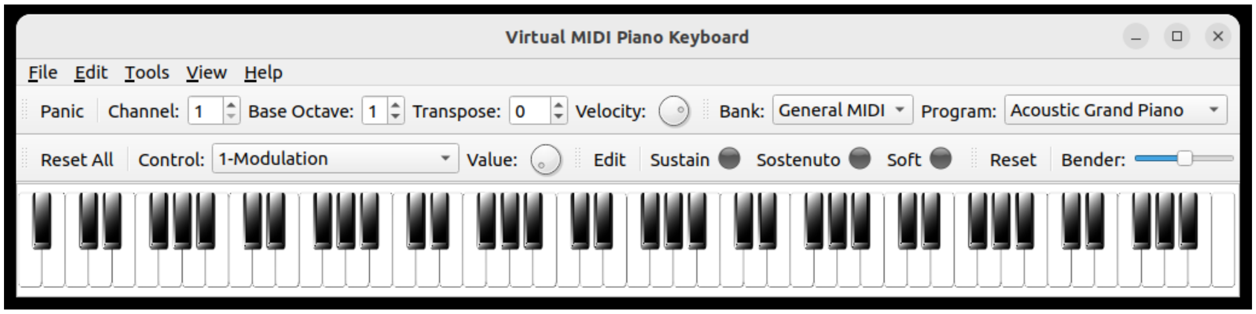


Figure 8. Virtual MIDI Piano Keyboard on Ubuntu

On the host computer side, the desired software to play the received MIDI data is freely up to the user. For testing purposes, we used the free Virtual MIDI Piano Keyboard software on Ubuntu, the FluidSynth MIDI Android app, as well as Ableton Live on MacOS, with successful connections and interactions on each. There are a few limitations, however, with the biggest one being that the MIDI controller does not seem compatible with the Windows OS default audio API. The device is able to be seen and connected to wired and wirelessly, but no MIDI messages are able to be successfully received. On the other hand, MacOS/iOS as well as most Linux distributions that support the Advanced Linux Sound Architecture (ALSA) API are able to automatically recognize and set up the controller once connected, allowing the board to act as a plug-and-play device with minimal setup. Another limitation also lies in the host system’s support for BLE, as some systems only support regular Bluetooth and are incompatible with BLE. On MacOS, there is a special setting specifically for Bluetooth MIDI controllers, which is where Squidbox will be able to be seen and connected to. The physical USB connection can also be used to bypass this complication at the cost of wireless connectivity.

Conclusion and Future Development

For the final prototype, we were successfully able to accomplish our smaller goal of creating both a wireless MIDI controller as well as a single velocity-sensitive key, both operating on the same software and circuit. For future development, the code currently only supports checking the state of one velocity sensitive button, so the next step would involve incorporating all eight buttons with velocity sensitivity in a more robust and efficient button design, and independently checking their state for both being pressed and at what velocity. Once these are accomplished, a following goal could be a visually appealing casing design to encase all the buttons and electronics. A really cool final working project could be to compose music with the Squidbox.