– Independent Work Report Spring, 2024 –

# MiNee Me: Exploring Music Composition Through Robotics

Nicolas Nee

Advisers: Radhika Nagpal and Allison Chen

## 1. Introduction

I had two goals when I first started the project. The first was to create a robot that could play the piano. The second was to train a model that could then create its own music. One of the main motivating factors behind creating a paino-playing-robot was because I had recently lost the ability to play. Growing up I played for at least an hour every day for 8 years but after I switch to the viola, I completely forgot the piano clefs. After years of my grandma telling me she wished I could still play the piano, I tried to relearn last semester, but to no avail. I was too busy with school to commit enough hours to it. Then, when I found this class, I thought it would be the perfect way to "relearn" how to play. Instead of trying to learn again, I could simply create a robot that could play for me! This is where the inspiration for the name "MiNee Me" came from. As for the note predictor model, I wanted to provide more exposure to Asian culture. At first, I wanted to train the model on music from my Vietnamese culture, but quickly realized there wasn't enough data, so I had to pivot to Japanese music by Nintendo. This way, I could still to the theme of Asian cultural representation. I also wanted to make the presentation more interactive, and since the model needs a couple of notes to start generating off of, I thought this would be perfect. It would make the viewer feel like they are a part of the project, as they have helped create a brand new piece of music!

One of the main sources for inspiration for MiNee Me was the MahaDeviBot from the KarmetiK's Machine Orchestra.[1] The orchestra is comprised of many different robot that play

[1] Kapur, Ajay. MahaDeviBot, 2006. https://www.karmetik.com/karmetik-robotics/mahadevibot/.

different instruments, with the MahaDeviBot playing a drum set rigged with 12 different India percussion instruments. Surprisingly, the orchestra still uses a human to play the piano, so I saw an opportunity to create something that might actually get used. Another piece of inspiration was Xiaole, a humanoid robot that could apparently read emotions, which I thought to be interesting but unnecessarily complicated.[2] The fact that the robot was humanoid made it much harder and more expensive to create, and also made it much slower because the robot had to wait to rotate, reach, and press in order to play a single note. I aimed to create a robot that could press any key, whenever it needed to, and at any speed. To do so, I decided to audio files called MIDI files to interpret the music. These files were first invented in 1987, before mp3 files, and were perfect for my project because they listed out all the information about a piece of music in a very simple format. Once I successfully interpreted them, I could send the notes to my arduino, which mapped each note to a specific solenoid that could then press the note. This implementation worked very well, in my opinion, because I was able to reach every goal I had set out for: I was able to play multiple notes, whenever I wanted, however quickly I wanted. The only unfortunate part was that the solenoids were pretty expensive, so I was only able to buy 8, making the range of playable notes very limited.

For the model, I gathered every note played in hundreds of Pokemon and Mario songs and added them to a text file so that I had a string of around 800,000 notes that I could train a next word predictor on. For each note, I kept track of the previous $n$ notes so that if the $n$ note sequence came up when generating, I could pick the note that appeared most often after the sequence. While the model can create new music that has some musical flow, it is inaccurate and does not sound great, for many reasons which I will dive into later, in section 10.

Although the projects are related, the processes for creating them were completely different, so I will first discuss MiNee Me, and then the model. Rather than jumping back and forth between the two I believe that it would be easier to understand them if I focused on each one specifically, and then how they interacted during my final presentation.

[2]Hall, Sophia. "Watch the Piano-Playing Robot Developed by Leading AI Lab That Can Also Read Human Emotions." Classic FM, January 23, 2023. https://www.classicfm.com/music-news/videos/piano-playing-robot-human-emotions/.

## 2. Problem Background and Related Work (MiNee Me)

With a specific goal in mind, I began brainstorming and researching different ways to complete it. The first challenge that I quickly encountered, however, was that there weren't many resources to reference for guidance. Piano Players have been around for a very long time, but were extremely outdated as they used physical paper rolls to play notes. The piano works by shooting a constant stream of air that is only able to pass through if there is a small slit in the paper, in which the air travels through a vacuum tube and hits the corresponding note. This was a good starting point, but I wanted my robot to be able to play based off of digital audio, rather than needing a physical roll of paper. After a couple more weeks, I found Amelia, a robot made by Ramon Yvarra, that is most similar to what I have created.[3] Unfortunately, I found this after I figured out how I was going to implement the project, so I was not able to use it for much guidance. The robot uses a player piano as its base that is modified to accept MIDI files as input rather than using the paper roll. It parses the MIDI file and uses solenoids to control the vacuum tubes that already exist within the player piano, rather than directly pressing the keys with the solenoids like I have done. When activated, the solenoids open, allowing airflow in the tube to rush through and hit the key. Amelia uses Apple's MIDI library to parse the MIDI files and output a note value, but Ramon stated that even Apple did not have a complete library and that he had to add his own adaptations to it so that he could complete his project. I found this project fascinating because it successfully updated old piano players while maintaining their cultural influence.

I, however, did not want to rely on existing vacuum tubes or Apple's incompete MIDI library (and couldn't afford a $10,000 player piano). I also wanted to prove that cool projects such as xiaole or Amelila did not have to be done by million dollar research labs or entire groups, and could be completed in just one semester by a single student with less than $500. Other than these projects, there weren't many resources that could provide further guidance, making the task incredible daunting and risky, because I had to come up with my own design and figure out how to assemble

[3] Ramon, Yvarra. "Adventures in Piano Building." Medium, September 11, 2016. https://medium.com/@hackmancoltaire/adventures-in-piano-building-c3fe5fa0b56b#.yu4ph2vmi.

it with a limited knowledge of electronics and circuitry. I hope that this project will inspire other students, whether in college or younger, to pursue robotic projects that they find interesting without being intimidated by a lack of previous works that they can look upon for help, just like I have with this one.

## 3. Approach (MiNee Me)

When I first came up with the idea to create a piano playing robot, I thought the design would be pretty smooth as I already had a rough idea of what I wanted to accomplish and how I would do so. This was the first time I had created a physical project on my own, so I was extremely naive in thinking this. My original idea was to use public domain programs to read through sheet music and convert it into letters that I could then input into the Arduino to output a key-press by rotating a stepper motor onto the key. This was before I had found the Amelia project by Ramon, which details MIDI files and solenoids. Without this information, however, to convert the sheet music, I wanted to use Optical Music Recognition, otherwise known as ORM, that could take in an image or pdf and output a list of notes and other information about the piece I may need. I quickly ran into many problems as I started doing more research and found that there weren't many ORM software available, and the ones that did exist either did not work consistently and accurately, or cost hundreds of dollars but still weren't perfect. At this point, I realized that the creating the robot would be much harder than anticipated and that I would have to restart many times, leading me to the following design process:

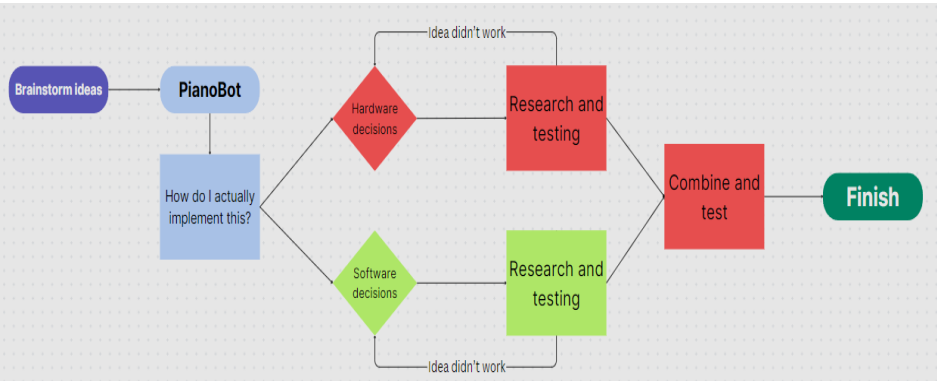


**Figure 1: The project workflow**

The majority of my time was spent in the middle section, where I would make a software or hardware decision, find out it didn't work or wasn't doing exactly what I wanted, and have to try something new.

When I realized that I would have to find a way around ORM's I reached out to the Princeton Music Library and they advised me to look into MIDI files, which clearly laid out all the information about the song. The listed the note value being pressed, whether it is on or off, how long it was being held, and more, captured as 8 bit hexadecimal values. Additionally, I was advised by Brendan Byrne of the Studio Lab to use solenoids instead of stepper motors because they could replicate a finger press more accurately and could also be activated and deactivated quicker than a stepper motor. Thus this brought me to my new approach, MIDI files plus solenoids, which happened to be the same as Amelia, but as I stated before, I wanted to press the notes directly.

### 3.1. Understanding MIDI Files

Once I had this idea, my first task was to understand what exactly a MIDI file was. I needed to fully understand how the music was represented and how the notes were organized so that I could write code to interpret it. Running a MIDI file directly through my file explorer simply played the audio, so I started looking for a way to print it out. While there is a MIDI library on Arduino, the library is only used to accept a live feed of MIDI inputs and interpret the MIDI information, such as playing on a keyboard (or other MIDI device) connected to the Arduino. This, however, was the exact opposite of what I wanted to do. Finding a library that could print the MIDI file was difficult because I couldn't find one that people universally agreed was the best. Every library was made by one or two people and posted to GitHub for public use, making each coding representation for the same file different, as they used different syntax and data structures. After testing different ones, I chose one made by user Raphaël Doursenaud. [4] The library gave me two important functionalities that I needed for both MiNee Me and the model. With it, I could read MIDI files as text and could create my own MIDI files, but I only used to first functionality for this portion of the project. The library represented the files as a list of list, organized by tracks and messages. Each track stored

[4]Doursenaud, Raphaël. Mido. December 15, 2023. https://github.com/mido/mido/tree/main

overarching information about an instrument, such as the tempo, the key, and the messages, which stored all the information about the notes.

Now that I could access all the information of the piece through a list, I had to take time to understand the information so that I could start coding and find a way to easily pass it to the Arduino.

```
MidiFile(type=0, ticks_per_beat=384, tracks=[
  MidiTrack([
    MetaMessage('time_signature', numerator=4, denominator=4, clocks_per_click=24, notated_32nd_notes_per_beat=8, time=0),
    MetaMessage('set_tempo', tempo=1090909, time=0),
    MetaMessage('track_name', name='Electric Piano', time=0),
    Message('program_change', channel=0, program=0, time=0),
    Message('note_on', channel=0, note=48, velocity=50, time=0),
    Message('note_off', channel=0, note=48, velocity=0, time=96),
    Message('note_on', channel=0, note=50, velocity=50, time=0),
    Message('note_off', channel=0, note=50, velocity=0, time=96),
    Message('note_on', channel=0, note=52, velocity=50, time=0),
    Message('note_off', channel=0, note=52, velocity=0, time=96),
```

**Figure 2: The organization of the lists using Doursenaud's library. This is part of the output of the C scale on the 3rd octave.**

Referencing figure 2, the MIDI file is stored in a pretty confusing way. The notes are printed twice, sometimes with a velocity zero? And why do the time variables only equal zero or 96, when you can't play a note for 0 units of time? After creating my own songs to compare with, I came to a couple key conclusions:

1. There is a message for each interaction for a note.
2. The time variable represents the time between the previous message and the current one.

In reference to the first point, I will refer to playing (activating) and releasing (deactivating) a note as an interaction. The library creates a message for each interaction with a note rather that indicating which note is played and for how long it is being played for. For the second point, instead of holding the universal time that the note is played at, it holds the change in time. Back to figure 2, the note 48 (C3) is activated right away, at time 0, then deactivated 96 units of time later. Note 50 (D3) is activated at the exact same time that note 48 is deactivated because its time is 0. The format was pretty complicated and I wanted to keep the code for the Arduino as simple a possible since it limited in storage space and computation speed, so all I wanted to pass was the note value.

## 4. Implementation (MiNee Me)

Now that I knew how to read MIDI files and had a concrete plan of what I wanted to do and what I could use to do it, I could finally start coding and building.

### 4.1. Interpreting MIDI Files

My first attempt at sending the notes to the Arduino was to loop over each message sequentially and pass the note value. While this gave my Arduino the correct inputs, it would send one note every time step (depending on how quickly my computer was performing) and also didn't indicate which notes were part of chords. With this method, my solenoids would just play one note every $x$ seconds and could not play multiple notes at once since I was not taking the time variable into consideration yet. My next attempts utilized a neat property of the MIDI library that allowed me to ignore all of the data in a message except the note value and time. Since a message is created every time a note is interacted with, I noticed that the first time a note value appears, the note is activated, and the second time, the note is deactivated. This meant that any odd appearance of the note was the note being played and every even appearance was the note being released. This way, I could simply keep a boolean in the Arduino code corresponding to whether the note was activated or deactivated.

My second attempt was to loop over each track prior to sending anything to the Arduino and save each time that a specific note was interacted with. With this approach, I would have all the important information about each note and could send it to the Arduino. To do this, I kept track of the total time that the notes were being played by accumulating each time variable, and anytime that a note was interacted with, I would append the total time to a list that belonged to the note. Essentially, I would have a list where the first value was the note value and the next ones are the times in which the note is interacted with. Looking back at figure 2, one of the lists would be [50, 96, 192], indicating that note 50 is interacted with at times 96 and 192. Once I made every list, I could loop over the total time and check whether each note contained the current iteration value (corresponding to the current time). If it did, I appended it to a list that I could then send to the Arduino. By pausing the program for a specified amount of time $t$ after every iteration, where $t$ was

a function of the tempo, I could simulate an accurate tempo for the song. This approach allowed me to play multiple notes at once at the correct time, but as the songs that I tested this method with got longer, the program became extremely slow since I had to check whether hundreds of lists contained a single time stamp at every step.

To decrease the time complexity, I developed one final approach, in which I saved the notes to a time instead of the times to a note. The approach is basically the same, only in reverse. Instead of appending the total time to the corresponding note list, I created a new list for each time in which a note was interacted with. If the time for a note was 0, then I appended it to the current list, because this indicated that the note was being interacted with at the same time as the previous notes. If the time was not zero, then I created a new list. In figure 2, one of the lists would be [96, 48, 50], indicating that at time 96, the notes 48 and 50 are interacted with. This approach worked much better than the previous one because the code was much less complicated and I only had to check whether a specific time existed in the first index of each list. This is the approach that I use in MiNee Me. Now that I could accurately and efficiently pass note values, the software portion of the robot was pretty much done, and we can now discuss the hardware side of things.

### 4.2. Controlling The Solenoids

As I stated before, I originally planned pressing the keys using the rotation of the servo motors, but quickly switched to solenoids as they could press and release quicker. While they were the perfect tool to produce the output, there weren't many references that I could use to help me understand how to build the circuitry. They are not included in the starter kit that we were given at the beginning of the year, so there were no direct instructions that I could follow. There were also many constraints that I had to consider when buying them. The first was that it had to have a long enough activation distance to push the key. In order to fully press the key down from its resting position, it had to move around 8mm, which doesn't sound like a lot but many of the solenoids I found were not long enough for this. I also needed them to be powerful enough to force the key down. Before I discuss further challenges that the solenoids presented, I should clarify how exactly

a solenoid, which is typically just a tight coil of wires, could press a key.

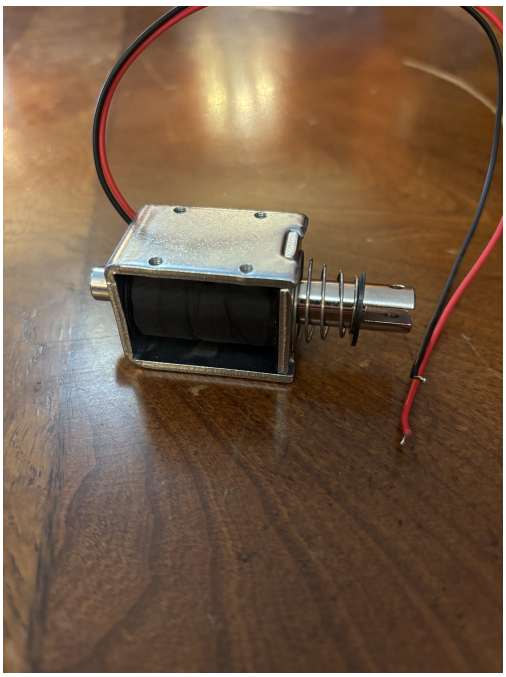

**Figure 3: One of the solenoids used in MiNee Me**

Inside the black casing in the middle of the metal box on figure 3, is the actual solenoid part, with the tight coil that produce an electric field. When a large enough current runs through the coil, the metal rod that rests in the middle of the coil is driven in the direction of the magnetic field (left, in figure 3), pushing the key. By attaching a spring to the other side, the rod will return back to its starting position when the current is cut off, ready for another activation.

The solenoids use 12V and 250mA so I had to add many parts to the circuitry to ensure that the solenoids were safely being used in my project. This was a big challenge because up until now the only hardware experience I had was from two labs in Physics 104. The circuit diagram that the company provided me with was very complicated and used notation that I had never seen before, so I had to look online to try and get help assembling my specific solenoid. There were videos by the Arduino company that controlled smaller ones, but these used power directly from the Arduino and the circuit was also overly complicated. After much trial and error, I was finally able to control a single solenoid, and expanding to control multiple was mundane because I just had to copy the same setup.

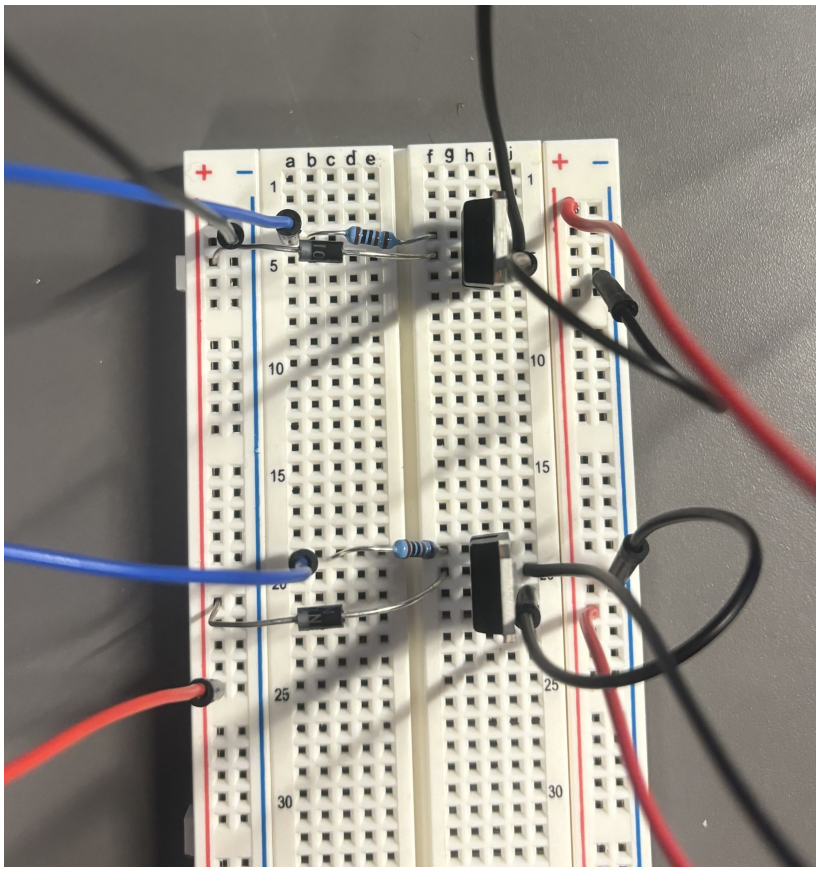


**Figure 4: The final circuit used to control two solenoids.**
**Not pictured: two wires connecting the left and right power rails.**

Each wire in figure 4 is color coded - naturally, the red and black wires are used for power and ground respectively and the blue wires are connected to the Arduino digital pins. Since the solenoids needed 12V of power and a large enough current, I had to use a power chord instead of the Arduino power, which is capped at 9V, so the only other wire that is connected to the Arduino is a ground wire. After passing through a resistor, the digital input travels into the heat transistor, which connects the Arduino command to the solenoid and power. The transistor has three pins, the collector, which collects charge from the power wires, the emitter, which is connected to all ground wires, and the base, which accepts the input to open or close the circuit. Additionally, there is a diode connected directly to the postitive power rail and the collector to protect everything from any voltage that is sent in the wrong direction. When the current to the solenoid is cut off, it creates a voltage spike that travels down the wire in the opposite direction, so the diode captures that voltage and protects the entire circuit from being damaged. This could easily destroy the Arduino, so it is

imperative that the diode is here! Another thing that I learned while attempting to construct the circuit was that solenoids are very specific, meaning they have to be compatible with the transistors and diodes being used. I was running into many issues because I was the wrong transistors, even though my circuit was correct. All the parts used in this project will also be listed at the end of the paper, so that if someone were to use solenoids, you would not run into this same issue. One last thing that I added to the solenoids were rubber stoppers to sound proof them. Each time the solenoids activated and deactivated, there would be a loud clank sound from metal hitting metal, taking away from the experience of listening to the piano. The only issue with this was that the rubber would sometimes get pushed out of place by the repeated activation of the solenoid, so the parts would get stuck together on the sticky pads that were only supposed to touch the rubber. Depending on how much of the metal was connected to each other, the solenoids could completely because they would not have enough strength to disconnect from the sticky part. With more time, I would have just glued the parts of the rubber directly to the metal, ensuring that the rubber could not move.

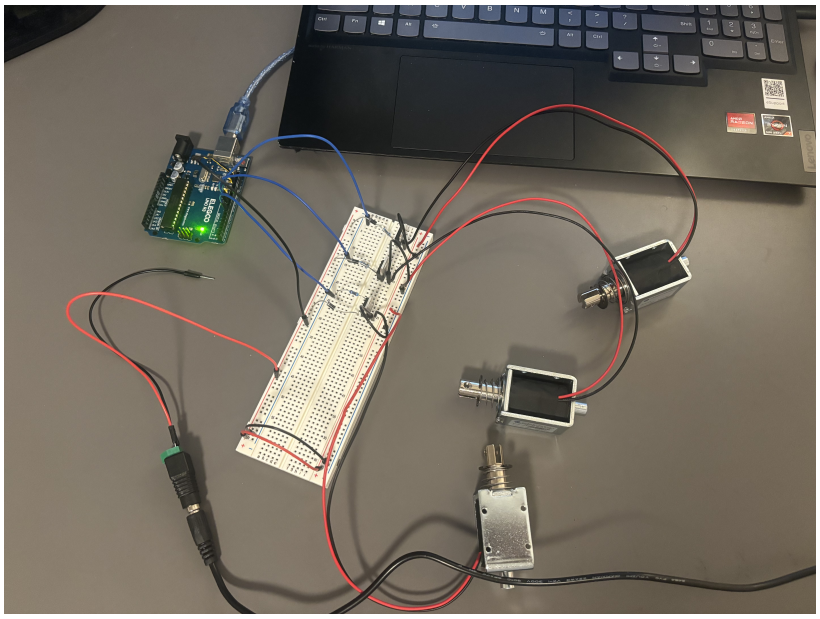

**Figure 5: A wider view of the circuit.**

### 4.3. Creating a 3D Print

Now that I had a way to press the keys consistently and parse the MIDI files, I could easily combine the two. My next task was to create a rig that could support the weight of 7 solenoids and contain the circuitry needed to activate them. There were a couple of key things I had to keep in mind when creating the model:

- Strong enough to support the weight
- Long enough to rest over the entire keyboard
- Wide enough to hold two solenoids
- Locations to put the breadboards and Arduino

My first idea was to use wood to create the model, as I had never worked with 3D modeling software before, but there were many problems with this, such as not having any access to wood or materials. With the help of friends, I was able to learn Autodesk Fusion and began creating my model. The first setback I experienced was that the model had to be 16 inches wide, to rest over the entirety of the keyboard, but the 3D printers at Princeton could only fit a maximum of 9 inches. I decided to split the print in half, but this meant that I would have trouble ensuring that it could support enough weight, especially since I was only putting the solenoids on one half of rig (because the keyboard had two octaves and I was only playing on one). To try and disperse the load, I added pegs to one half of the model and holes to the other half so that they could connect and had the back side of the rig rest on part of the keyboard so that the load could be dispersed into the ground.

Another problem that came up was that the solenoids were slightly larger than the keys. This meant that I could not put them side by side on the same wall and had to create another wall to stagger them so that they could fit above the keys. This would have been a big problem if I was playing the black keys, but luckily I did not have to deal with that since I did not have enough solenoids. If I were to expand the project, I could fix this project by simply buying a keyboard where the keys were larger than the width of the solenoids. That way, I could put all the solenoids controlling the white keys on one wall and the solenoids controlling the black keys on the other wall. In this project, however, I had to ensure that the width was wide enough so that I could fit two

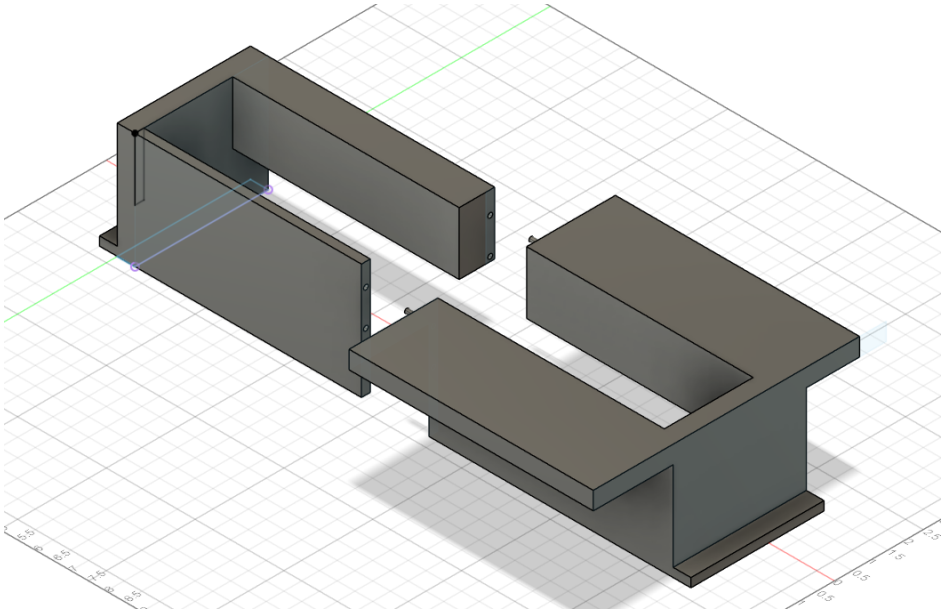

**Figure 6: The 3D model made using Autodesk Fusion.**

solenoids but not too wide to the point where the solenoids could no longer press the key. I also had to allocate room for something to connect the solenoids to the wall. Originally, I planned to use Velcro, but I only measured enough space for it on one wall, forgetting that I needed it on both sides, so I had to switch to strong double adhesive tape.

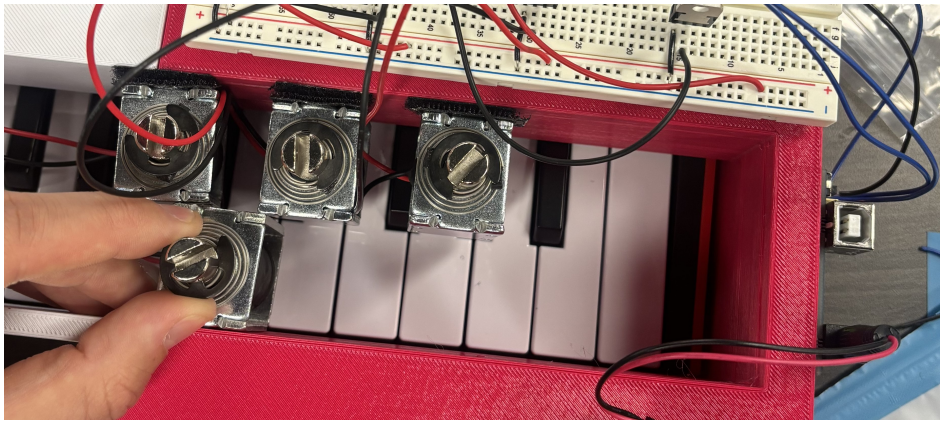

**Figure 7: My attempt to use Velcro. There is just enough space for the solenoid, but not for another set of Velcro.**

Finally, I added two shelves to hold the bread boards. While I was constructing the model, I was solely focused on functionality, so it was created out of opaque material, but with more time I would have liked to create a transparent rig so that a viewer could see the solenoids activating and hitting the keys.

## 5. Results (MiNee Me)

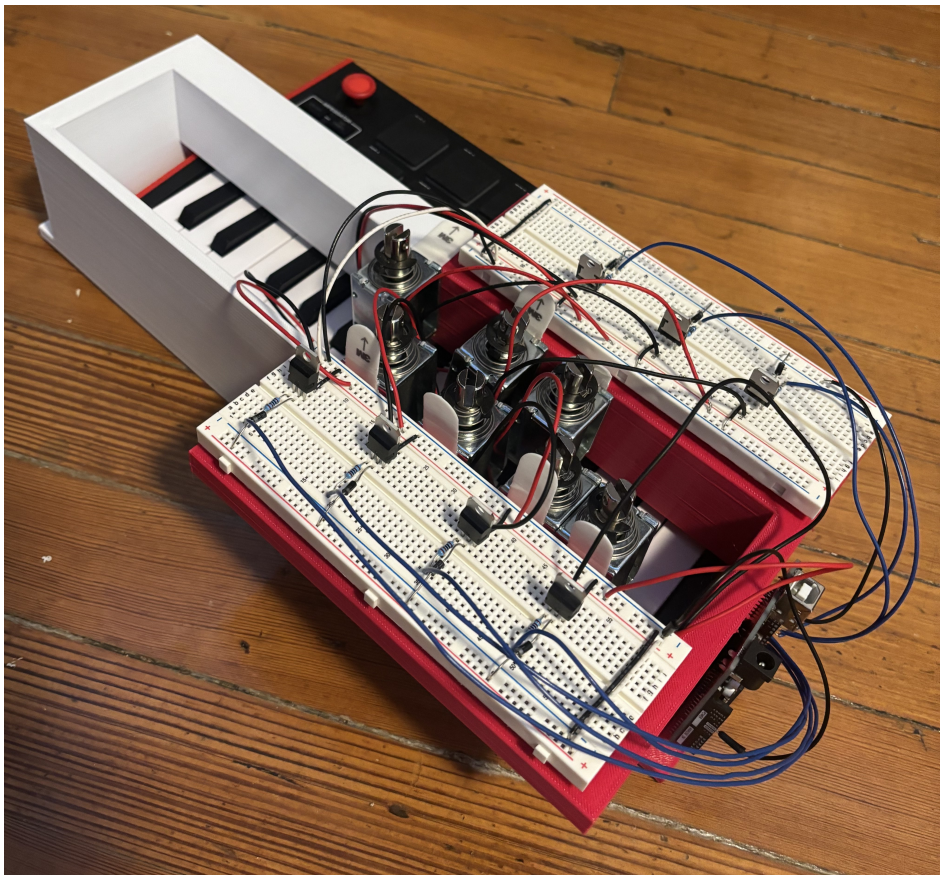

**Figure 8: MiNee Me in its final form.**

Once the solenoids were connected to the walls and circuit, I could successfully press each white key in middle C whenever I wanted. I made the code so that it would not fail if a note that was outside this range was sent, so the robot is able to accept any MIDI file and play the part that corresponds to middle C perfectly. Obviously, playing only the middle C part of a song isn't a good output, so the songs that I chose to download for the exhibition were simple ones that only used the 7 notes. Compared to the other Piano playing robots that I found, I believe that this prototype has the potential to be much better when fully developed. It is faster and more accurate than Xiaole, the humanoid robot, and it does not rely on having access to preexisting parts like Amelia, making it much more feasible to create on a budget so that it could be easily replicated by a future student using the explanations in this paper and the code provided on the git.

## 6. Summary (MiNee Me)

Throughout the paper I have mentioned different moments in which the robot could be slightly improved, including but not limited to ensuring the keys are wider than the solenoid and gluing the rubber stoppers. Beyond adding more solenoids to play more keys, one of the main ways that this project could be improved would be by adding a way to control the force at which the keys are pressed. Currently, it is binary - the key is either activated or it is not. In the songs, however, there are fluctuations in volume and ways in which the keys are pressed. Adding a way to control the speed of the solenoid would cause the keys to be hit with different amounts of force, changing the volume of each note. I believe that this could be done using pulse width modulation, or PWM. PWM is a different type of digital output that allows the user to control the time in which the voltage is released, as opposed to what I have used, which acts like a switch. By allowing the voltage to pass over a longer period of time, the strength and distance of the solenoid could decrease. I am not sure this would work because you still need to solenoid to go far enough to activate the key, but this would be a good starting point to try and add a way to control volume.

I do not think there are many other limitations besides only having 7 keys and not being able to control the volume, one of which is easily fixable with more funding. That concludes the portion of the paper about MiNee Me, we can now discuss the next note predictor that I created to accompany the robot.

## 7. Problem Background and Related Work (Next Note Predictor)

For this portion of the project, there were many related works that I could use as guidance, such as the music generation LSTM by Karnika Kapoor, which has the same goal, but uses a different training model and trains off of Chopin music rather than Japanese Music.[5] Another piece of work

[5]Kapoor, Karnika. "Music Generation: LSTM ." Kaggle, October 11, 2021. https://www.kaggle.com/code/karnikakapoor/music-generation-lstm.

that I used to write most of the training/generating code was a next word predictor by NeuralNine. [6] My goal was to adapt the code from their project and use it to predict notes instead of words.

## 8. Approach (Next Note Predictor)

I began this part of the project after MiNee Me, so I already knew how to read MIDI files and decided to use them to gather data. My approach was mimic the word files that NerualNine was using to train as closely as possible by creating a text document that held every note in a song separated by a space. I reasoned that this would be a good idea because notes were similar to words in that music operates in patterns - it is likely that when you see a specific sequence of notes, the same notes will come after it. I mistakenly thought that it would not be important to differentiate between chords and singular notes, because it would be the appear in the same order in the text document, but once I needed to generate, I had no way of outputting a chord. This portion will get more clear once I have discussed the implementation of the model.

## 9. Implementation (Next Note Predictor)

There were four major steps to completing the next note predictor. The first was data gathering, in which I needed to download enough songs so that I could get over 800,000 notes to make a good model. The next was training the model, which I will discuss first because it provides context for the data gathering. The final two were generating and creating a new MIDI file.

### 9.1. Training

To train the model, I counted of the number of appearances for each unique note and kept track of the 10 previous notes for a specific input. So for each note $p$, I knew the $n$ previous notes, so that when a similar sequence appeared in the generation phase, the note $p$ would have a high probability of being chosen. I then used the TensorFlow library to train the model, using softmax activation and a batch size of 128. I created multiple different models, most of which used $n = 10$, so that I

[6] NeuralNine. "Text Generation AI - next Word Prediction in Python." YouTube, March 9, 2023. https://www.youtube.com/watch?v=tEV_Jtmx2cc.

could find one that had the highest validation accuracy and sounded the best, and then a model that only trained off of the 12 notes in middle C. This model bridged the gap between this portion of the project and MiNee Me, as I could play generated songs on the robot.

The first model that I trained was completely unfiltered, using every note in every song. The generated notes using this model, however, sounded weird because the high and low notes seemed out of place. This led me to create a filtered model, in which I ensured that the notes were above 40 (E2) and below 80 (G#5). This sounded only slightly better, but it wasn't until I created a graph of the accuracy and loss for the validation set that I realized that the model was inherently faulty. The third model I made was the one that my model could play, trained off only notes in middle C. Instead of completely disregarding notes outside that range, I mapped the notes to the middle octave, so if a note was C1, it became C4. Each model took around 20 minutes to train since they had to process around 650,000 notes and then validate using a 0.8/0.2 split between training and validating.

### 9.2. Data Gathering

Gathering enough data for the model was extremely time consuming because I had to find MIDI files for each song that I wanted to include. As I stated in the introduction, I originally wanted to train off of Vietnamese music, but there wasn't enough data for this. Even finding enough Nintendo music was hard because they don't create public MIDI files for their songs, so each file that I found was created by a random party. When I started looking for songs, I wanted each song to have a similar theme so that the patterns in the notes would be consistent, but after spending hours filtering through songs and realizing that I had around 5% of what I needed, I just started downloading whatever music I could find. This process was incredibly frustrating at times, because I would find a song that fit the theme perfectly but would not be able to find a MIDI file for it.

I finished with 223 different Nintendo songs from Mario and Pokemon, totaling to 860,233 notes. For each song, I read through the MIDI file using the git library and added each note to a new text file. Even though there would be parts in the text file where notes from a previous song would

influence the training for notes in a new song, I concluded that this would not affect the model much because it only happened in 222 spots.

### 9.3. Generating

To generate new notes, the model examines the $n$ previous notes, predicts the three most likely notes to occur, and randomly picks one of the three. The first $n$ notes that I used for the generations were simply the $n$ most common notes, but this part can easily be changed. This is where I hope to make the exhibit interactive, where a viewer can input their own $n$ notes and generate their own piece. The model added each note it predicted to an list and would then use it as an input for the next prediction, and would only finish once it had generated 100 notes.

### 9.4. Creating a New Midi File

To create a new MIDI file, I took the each note in the list and created a new message, specifying whether the note was being activated or deactivated, the time, and the velocity. To check whether a note was being activated or deactivated, I created a dictionary object that tracked the number of occurrences for a specific note $p$. If $p$ was even then it was being deactivated and had a velocity of 0 but if it was odd then the note was being activated with a velocity of 100. I set the time for each message to be 240, so a note was interacted with every 240 units of time (which depended on the tempo that I set).

## 10. Results and Summary (Next Note Predictor)

Unfortunately, the generated MIDI files do not sound good for a variety of reasons. The first is they are heavily constrained by the output format. The model simply returns a sequence of notes with no indication of whether the notes are part of a chord, so I was only able to activate or deactivate one note every time-step. This makes the song seem pretty random, but there are certainly instances where there is some musical flow. To train off chords, I could have appended the notes to each other and then added them to the text file, so if a chord of 40, 46, and 48 was played, I would add 404648 rather than each note individually. This way, the model would recognize each chord

as a unique token and when it came time to create a new MIDI file, I could simply break apart the notes again.

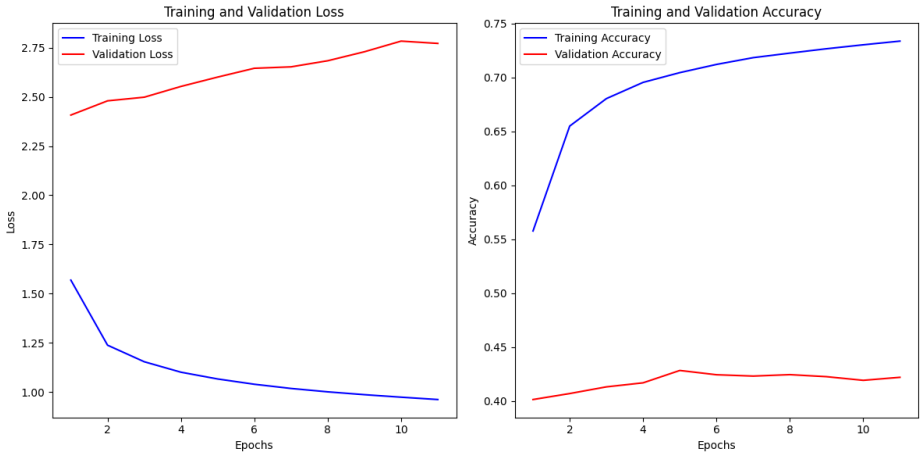


**Figure 9: The training and validation loss and accuracy from the unfiltered model.**

Additionally, since it was too time consuming to properly collect data, the model was trained off songs that had a different theme and were in different keys. I think this caused the validation accuracy to be so low, because the model would have no way to tell what key the notes should be generated in. The main way that I could have improved the model would be by ensuring that each song was in the same key, because this would solidify the patterns in the songs that the model could recognize. Unfortunately, this meant that one of my original goals to generate songs that replicated Asian music was not met. The notes did not follow the same logical sequence and I did not think that it sounded anything like the Nintendo music I had trained it off of.

Despite the validation being low, I do not think that this portion of the project was a complete failure because it performs relatively the same compared to other models. The model by Karnika Kapoor, trained on Chopin music, generates music that sounds like it has the same randomness. She also prints a graph that appears to show her loss is training low, but it follows the exact same trend as mine and she avoids showing the validation loss and accuracy, which are more important in analyzing a model. I think that it is important to show the low validation accuracy because it demonstrates that there is room for improvement. The concept of AI music is very interesting and

has not been perfected by anybody, whether that is an individual or an entire organization, so I hope that this could provide guidance for someone else.

## 11. Conclusion

Overall, I believe that the project was a success, because I was able to create a robot that could read through notes and play them accordingly, and was able to generate new music based on 223 Nintendo songs. While MiNee Me can only press 7 notes, it would be easily scalable with more solenoids and extra Arduino's to control them (since each Arduino only has 14 digital output pins), allowing it to play all 88 keys simultaneously. This would allow it to play songs that even professional musicians could not play, which I find incredibly impressive. With more time, I could have also improved the model by adding new ways to identify chords and ensuring the all the music I trained off of was in the same key.

I really enjoyed this project because it gave me the opportunity to learn so many different skills that I never would've touched at Princeton. While this made development exponentially more difficult, learning how to create my own circuits, how to 3D model and print, and experiencing the grueling process of designing a project will prove invaluable to me in the future. Beyond this project, I hope to take the skills that I learned to attempt my own cool projects such as this one.

This paper represents my own work in accordance with University regulations.
/s/ *Nicolas Nee* /s/

## Parts For The Circuit

Solenoids: https://www.adafruit.com/product/413#technical-details

Diodes: https://www.amazon.com/dp/B07Q3HBM63?psc=1&ref=ppx_yo2ov_dt_b_product_details

Heat Transistors: https://www.amazon.com/dp/B07TWMT52G?psc=1&ref=ppx_yo2ov_dt_b_product_details

Power Supply: https://www.amazon.com/dp/B01GEA8PQA?psc=1&ref=ppx_yo2ov_dt_b_product_details

## References


1. Kapur, Ajay. MahaDeviBot, 2006. https://www.karmetik.com/karmetik-robotics/mahadevibot/.
2. Hall, Sophia. "Watch the Piano-Playing Robot Developed by Leading AI Lab That Can Also Read Human Emotions." Classic FM, January 23, 2023. https://www.classicfm.com/music-news/videos/piano-playing-robot-human-emotions/.
3. Ramon, Yvarra. "Adventures in Piano Building." Medium, September 11, 2016. https://medium.com/@hackman in-piano-building-c3fe5fa0b56b#.yu4ph2vmi.
4. Doursenaud, Raphaël. Mido. December 15, 2023. https://github.com/mido/mido/tree/main
5. Kapoor, Karnika. "Music Generation: LSTM ." Kaggle, October 11, 2021. https://www.kaggle.com/code/karnika generation-lstm.
6. NeuralNine. "Text Generation AI - next Word Prediction in Python." YouTube, March 9, 2023. https://www.youtube.com/watch?v=tEV_Jtmx2cc.