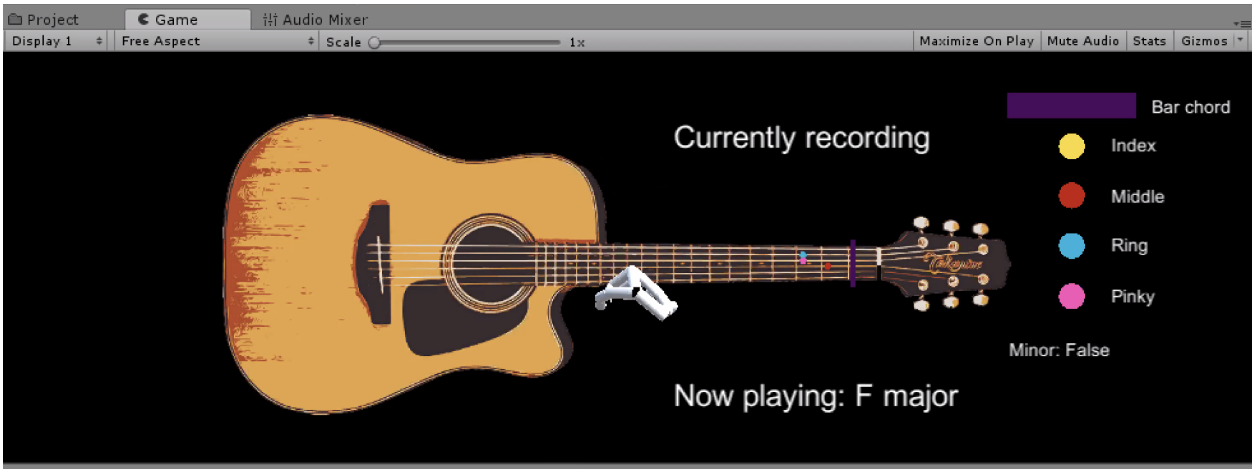
6.835 Final Project: Air Guitar

Nada Hussein, Nisha Devasia

Spring 2020



For our 6.835 final project, we built a multimodal virtual guitar player called AirGuitar. This program was built in Unity and interfaced with a LeapMotion, a laptop keyboard, and a laptop microphone to allow a user to input chords and strumming patterns to the program. AirGuitar is able to detect these chords and strumming patterns, visualize them on screen for the user, and play the corresponding music back to them real-time. The program also allows a user to record their music and play it back as well, with the voice commands "Start", "Stop", and "Playback".

Air Guitar worked very well in detecting chords and strumming motions, but was weaker when it came to detecting voice commands. Air Guitar can currently register 14 total major and minor chords and correctly detects them 100% of the time, and correctly detects a strum on the LeapMotion 90% of the time. While it correctly detected voice commands over 90% of the time, the recognition latency was high. This was due to the Unity speech package that we used, which had high latency, and perhaps low sensitivity to microphone inputs that made it hard to recognize certain words.

Collaboration: Both of us collaborated on most sections of this project.

Code: https://github.com/ndevasia/AirGuitar

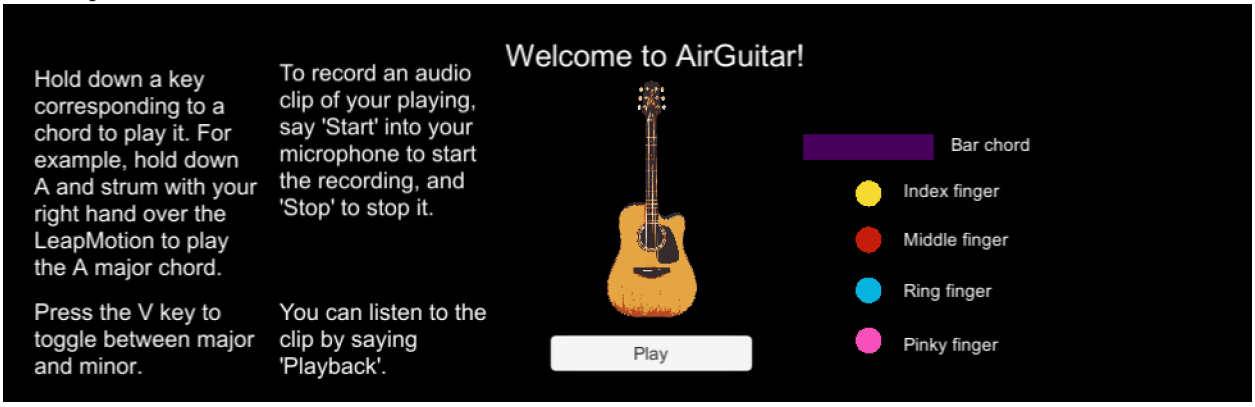
1 Introduction and Overview

AirGuitar was inspired by the problems that musicians may face in having to purchase, transport, and travel with large instruments. We wanted to come up with a way to allow musicians to be able to create music from anywhere, without worrying too much about having large equipment with them. Thus, we created AirGuitar: a multimodal virtual guitar player that requires only a laptop and a LeapMotion to play guitar. AirGuitar allows a user to set up anywhere and very easily input chords and strumming patterns to digitally generate a song played on guitar, with the option of recording and saving the song when they are done. This project allows musicians to play guitar without needing to purchase one, and allows them to play while traveling, or when they do not have their guitar with them.

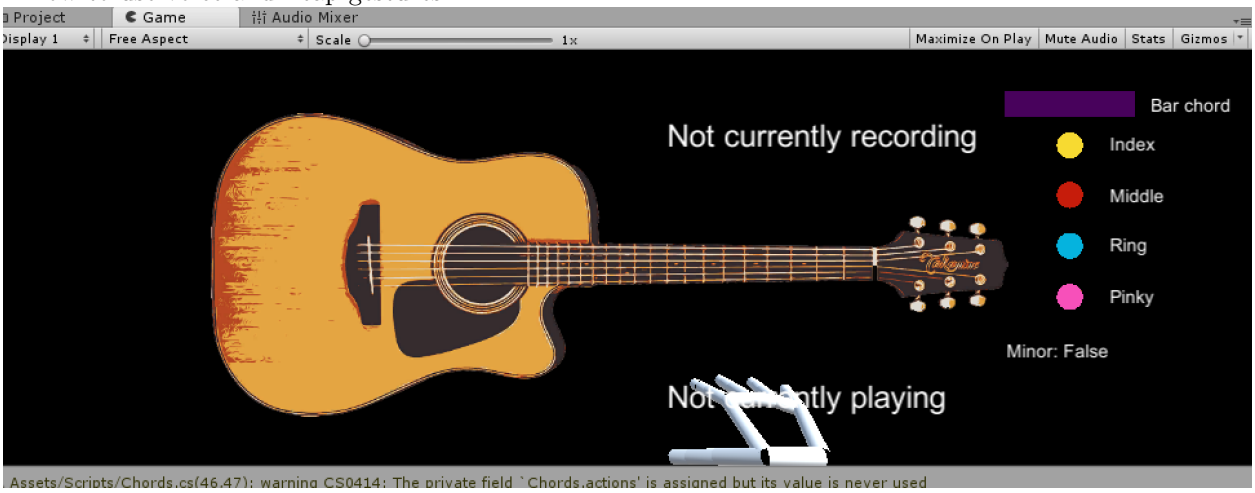
2 System Design

2.1 What Does It Do?

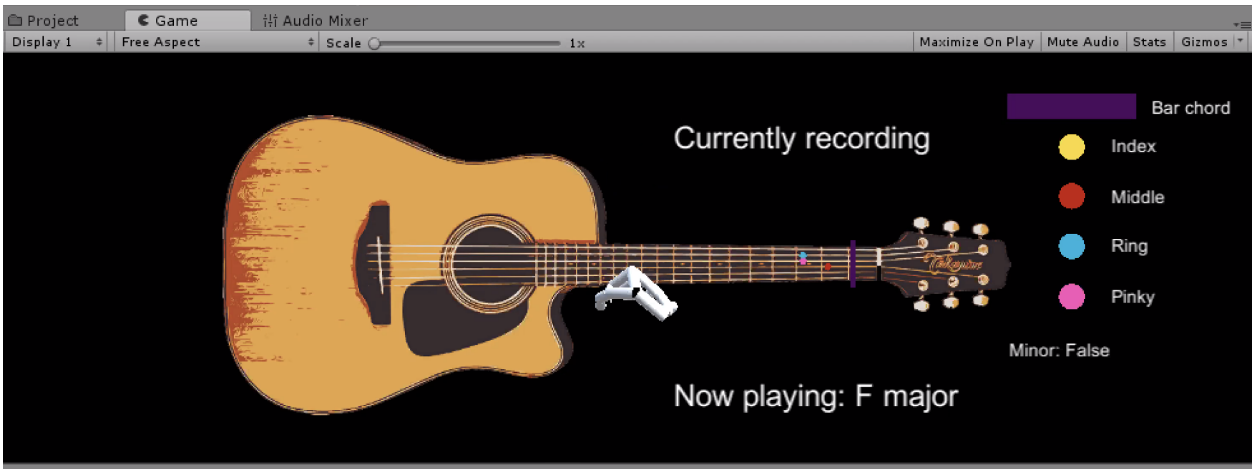
Here we will walk through a step-by-step of how the program worked with real input, showing screenshots at each step.



First, a user sees an intro seen as shown above. It instructs them that they can play chords by typing the letter, and type 'v' to turn on the minor chord functionality instead of major chords. It also explains to them how to use voice and Leap gestures.



On the game screen, the user sees a guitar, as well as the color code for each finger.



When a user types a letter, the chord is visualized on the guitar. When the user strums on the Leap, the program then plays the corresponding chord. If no chord has been pressed, the program plays open strings. There is also a recording option - 'start' to start recording, 'stop' to stop it, and 'playback' to listen to it.

2.2 System Architecture

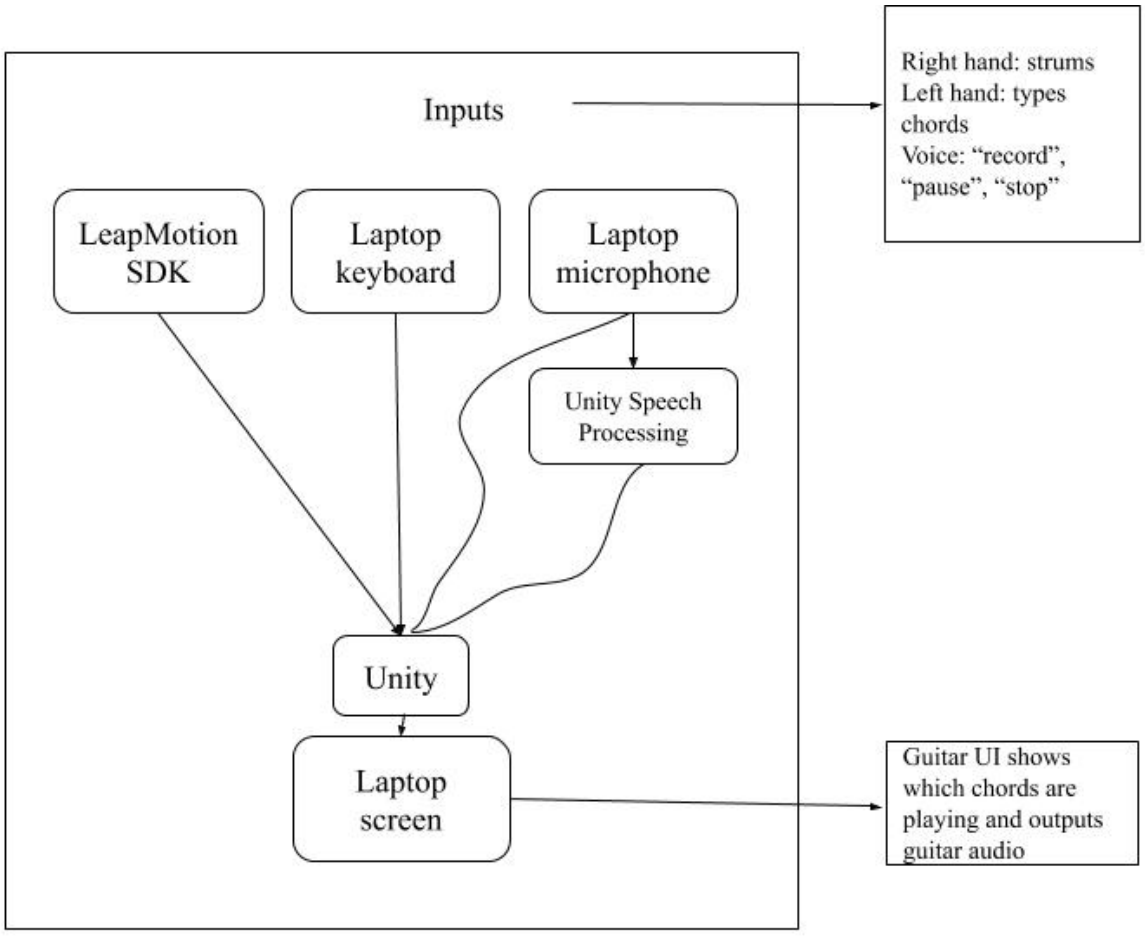


Figure 1: System Architecture Diagram

Here you can see our system architecture diagram. AirGuitar is based in Unity. When a user wants to play, they are presented with an intro screen that explains to them the mechanics of the system. From here, they hit 'Start,' and are directed to a screen showing a guitar. From here, a user can choose a chord they want to play by typing the letter of the chord name (for example, typing 'A' to play an A chord.) The system responds by visualizing the finger position of playing that chord, color coding each finger position based on which finger is placed where. Now a user can mime a strumming motion over the LeapMotion, and the system will see the upstrums and downstrums and play the chords accordingly. When a user wants to switch chords, they simply type a new chord and repeat the process. If a user chooses to record their playing, they can simply say 'Start' before playing their desired chord patterns. Once they are done, they simply say 'Stop', and can say 'Playback' to hear the recording played back to them.

2.3 Performance

AirGuitar performed fairly well. We were able to successfully get all the parts implemented with ¿ 90% accuracy. Our chord visualizations were consistently accurate, as we tested by simply inputting all of the chords that our system supported and making sure that they displayed the correct chord. Strumming also worked relatively well; we were able to very consistently detect strums and play the chord accordingly. We tested this by trying to strum at various speeds and gauging the responsiveness of the system, while ensuring we didn't detect more strums than we should have. However, we did not have as much success with trying

to distinguish between a strum and a hand position reset, so currently our system just strums each time it detects the hand motion. We also received almost 100% accuracy on recording ability, though the audio quality was a little poor.

An interesting failure to AirGuitar was the strumming. We had to experiment a lot with our strumming metrics, and tried to use position displacement, velocity, and inflection points to detect strums. In most cases, the strumming functionality detected too many strums, so we had to continuously tune the parameters and use a combination of metrics before it became less sensitive. We also struggled with the hand reset vs strum problem as mentioned, which we handled by trying to use pinsh strength as a metric (i.e. if a user is pinching, they are strumming, if not, they are resetting). However, this was difficult as it turned out a user is not able to switch directions for a strum at the exact same time that they start or stop pinching, so this metric still resulted in too many strums. We then tried to use a metric of having some percentage of the strum be a pinch, which worked somewhat well but still needs some work.

2.4 Difficulties

We had a lot of difficulty with detecting strumming, as well as recording the audio. We initially tried detecting strumming by doing a displacement threshold, but this seemed to detect too many strums. We then tried a velocity threshold, but since it was an instantaneous velocity measurement, the program often saw a user slowing down mid-strum as a separation between strums, so this also detected too many strums. We finally settled on a combination of a displacement threshold and keeping track of direction changes, such that a strum was only detected if a user moved far enough and they also switched directions at the end of the strum. This resulted in a very small delay as the strum did not happen on the program until it was completed by the user, but it was very responsive and accurate in detecting strums from the user.

Voice commands were also difficult to implement because we were using Unity's built-in library for this. This library had really high latency, so it took the system a really long time to detect a user's voice commands. The library could only record external audio as well, which resulted in poor audio quality since we had to record the speakers playing the user's audio rather than being able to internally record.

2.5 Easy To Implement

The chord visualization was relatively straightforward to implement. We were able to easily process chord letter inputs from the keyboard, and visualize the chord accordingly. It was also relatively easy to set up the playing of the chords, by adding all of the chord sound files to Unity and being able to call any specific file to play based on which chord had been chosen by the user.

3 Modifications

We had to make some significant modifications to our system along the way. Our initial system design involved a user using the LeapMotion attached to a plexiglass fretboard in order to actually 'play' the chords on the fretboard. However, in practice this turned out to be more complicated to implement and use than we expected. When we did our first user test, we had just gotten our system working such that chord inputs just happened by typing the chord name. This actually ended up being much more straightforward and intuitive to use, and allowed us to then switch gears into making strumming the Leap input rather than chord fingering. This modification happened because the Leap was not accurate enough for us to read finger positions that were so close together so accurately, and because we were unable to machine a plexiglass fretboard prior to leaving campus.

4 User Study

Due to the COVID-19 situation, we were unable to conduct a user study for AirGuitar.

5 Performance

In the process of creating AirGuitar, we learned a lot. We were able to very successfully detect chords and visualize/play them back. However, strumming and recording ended up being very difficult to implement. We had to experiment a lot with the optimal metrics to determine when a user was strumming over the LeapMotion, ranging from velocity thresholds to displacement thresholds to keeping track of inflection points to a combination of these options. We learned a lot about how to work with the sensitivity of hardware and work with it to come up with an implementation that would allow us to meet our goals.

An interesting next step for AirGuitar would involve much more streamlined playing and recording. Currently, the strumming sounds not very smooth, and is not as sensitive as we would like. Our recording quality and process is also kind of difficult to work with due to latency and poor audio quality.

To improve this system, we would first improve the strumming functionality. We would figure out how to get the Leap to better distinguish between a strum and a hand reset, by experimenting with different hand positions, distances, etc that might allow a user to clarify what they are trying to do. We would also rerecord our strumming sounds such that we also have a different sound for upstrums, such that we can play music that more accurately represents an actual guitar. We would also work on how to process the soundbites such that quick strumming would not cause overlapping sounds that cut each other off and sounded digital.

As far as recording, we would work on improving the latency of the voice recognition, perhaps by using a different library than the built-in Unity one (ex. Google Speech API). We would also change our recording implementation such that we internally record the audio, rather than having to record from the mic (which requires a user play their music out loud, and results in poor quality audio). We would also add functionality for actually naming and saving the recordings.

6 Tools Used

6.1 Unity

We primarily developed this project in Unity 2018 1.1 on a Windows machine. This particular version was used because it is the latest version that is compatible with the LeapSDK. We imported the Microphone feature from the Universal Windows Platform in order to have voice functionality, and use key presses to indicate chords. For strumming, we utilized the Hand Module from the LeapMotion Unity assets, and attached a script to the right hand model to detect strumming.

6.2 Adobe Suite

All the assets for this game were created in the Adobe Suite, primarily Photoshop and Illustrator. In particular, the semi-realistic picture of the guitar took an hour in Photoshop and was a labor of love which the creator is very proud of.

6.3 LeapSDK

For the SDK, we used LeapMotion Orion 4.0.0. We followed the setup at the 'Unity Assets for Leap Motion Orion Beta' page on the LeapMotion website in order to integrate the LeapMotion with Unity.

6.4 TeamViewer

Unfortunately, the LeapSDK integration for Unity does not work for Mac computers. In order to split the work on this project relatively evenly, we used TeamViewer so that the team member with the Mac could access the other team member's Windows machine. Testing merely required that the Windows team member wave their hand in front of the LeapMotion. However, this was still inconvenient to both team members, and would have likely been much easier if we were both on campus with direct access to each other's machines.