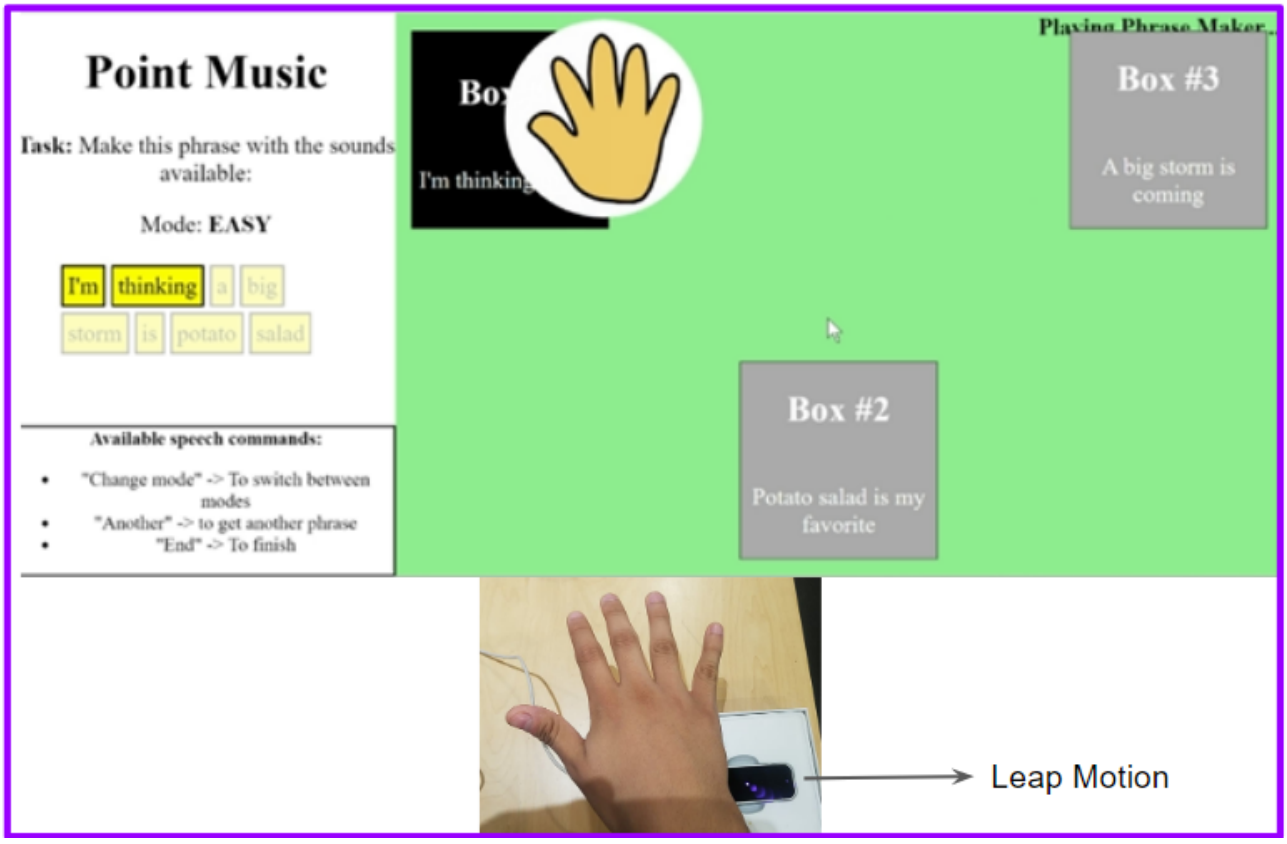
Point Music

Raul Alcantara



I built a system that served both as a game for creating spoken music (i.e., *Phrase Maker*), as well as a music mixing tool (i.e., *Music Mix*). In both modes, a sound is played every time the user's hand cause the virtual hand to hover over one of the "music boxes".

The system used both the Leap Motion to track the 2D position of the user's hand, as well as the computer's microphone to listen for speech commands. The Javascript library Howler.js[1] was used for controlling when each given sound was played.

Overall, Phrase Maker was found to be very entertaining on its own and a good introduction to the capabilities of the system. On the other hand, Music Mix was not found as fun/creative as originally intended. presumably because of the low level of user customization. One of the main critiques about the system's usability was the not super accurate speech detection, which prompted the user to repeat the speech commands several times until it worked.

[1] https://github.com/goldfire/howler.js/

Introduction

Motivation

The process of combining different music tracks into one piece (i.e., audio mixing) is an extensive, and sometimes painful, process. A common pop song has about 10 tracks, according to different music forums (i.e., reddit, gearspace). However, within each track there is usually a lot of repetition of instruments, and even vocals, because it helps the song to be memorable and predictable. This suggests that there should be a more efficient way to perform this task.

Overview

Point Music's main focus is to provide an easy and fun way to mix different sounds. As a result, the system allows its users to perform the mixing process by simply moving their hand to different places in real-life. The reasoning behind this model is to mimic an artist playing several instruments (e.g., drums) by putting their hand close to them. The hope is that the users will find this interaction intuitive and fun.

Apart from the music mixing capabilities, Point Music also offers a game called Phrase Maker that introduces the user to the system's main capabilities (i.e., music playing when hand hovering), but with a gamified view that asks users to mix given sounds to make up a predetermined phrase.

System description

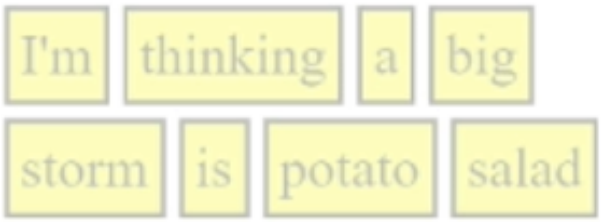
Even though Phrase Maker and Musix Mix are very related to one another, they are fundamentally independent, so I'll describe them both separately. It's worth noting that from the original menu the user can go to either by just saying "play phrase maker" or "music mix"

Phrase Maker

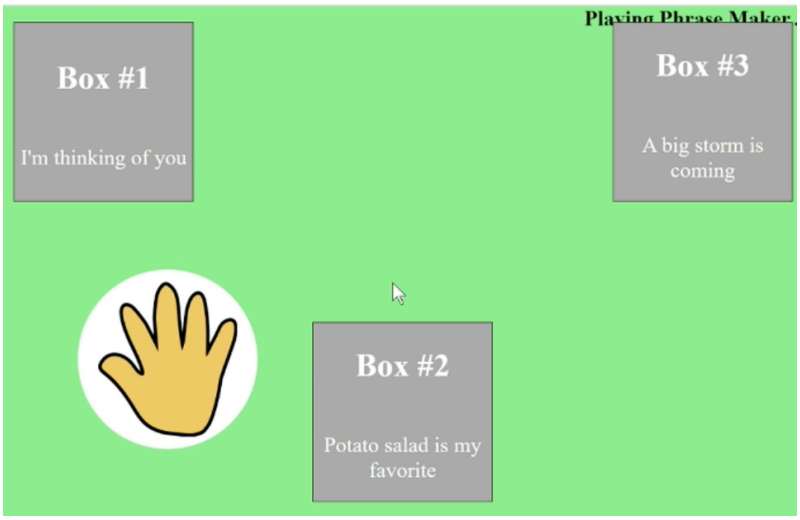
At any given time, the user has an "Available speech commands" box on the bottom left part of the screen that explains what the user can say.

Upon entering the game, the user is asked to choose a "template" which is just a pre-determined arrangement of the "music boxes". The user can see other templates by saying "next". Once they are okay with their choice, they can say "continue" to move to the game.

Once the game starts the user can see the phrase they are trying to replicate. For example, if the phrase is "*I'm thinking a big storm is potato salad*", then the user would see the following:

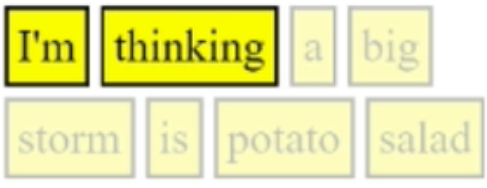


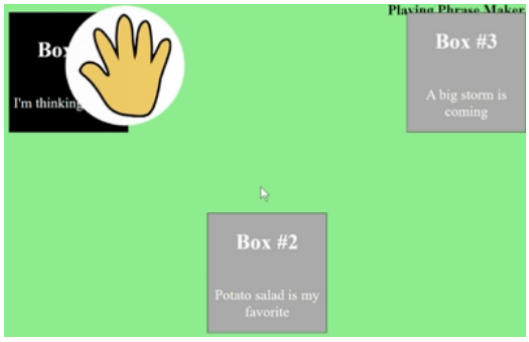
Apart from this, they can see the instructions and, most importantly, the main music boxes. On one of the examples, there are 3 boxes that contain audios that spell “I’m thinking of you”, “A big storm is coming” and “Potato salad is my favorite”. As such, the user would see the following:



Ideally, the hand icon would be a good enough hint to the user that they should start moving their hand above the Leap. When they start doing that, they will see that when the (virtual) hand hovers over the boxes, they get highlighted and immediately after that they start hearing the audio. For example, when their hand hovers over Box #1 from the example above, the user will be hearing “I’m thinking of you”. It’s also worth noting that the sound will stop if the hand exits the box.

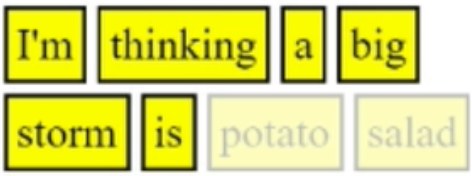
On the other hand (pun intended), while the user is hearing something, the boxes from the target phrase will lit up as appropriate. This means that if, for example, the hand hovers over Box #1 and then it exits after the system just said “I’m thinking” then the target phrase will look as follows:

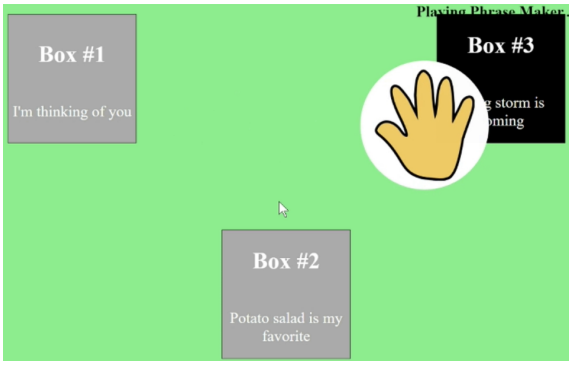




If the hand stays on Box #1 for too long (i.e., if the user hears ‘I’m thinking **of’**) then the target phrase resets, as the phrase made doesn’t match with the target phrase.

Now if the hand moves away from Box #1 and goes to Box #3 and they hear “a big storm is” then the target phrase would look as follows:





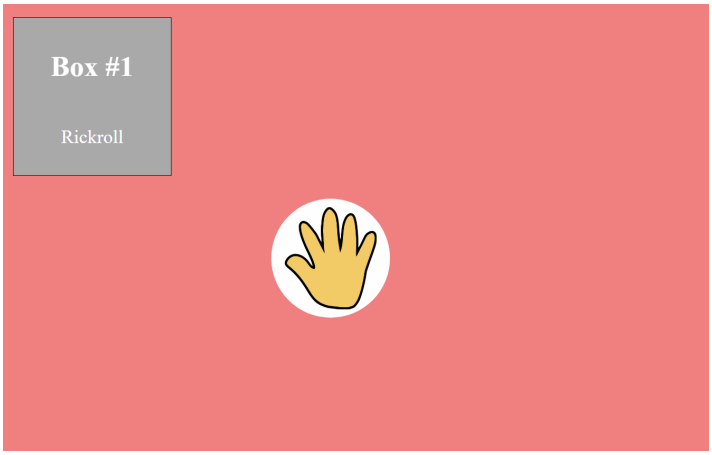
Finally, if the user completes the full phrase then they hear a “yayy” winning sound, and they get a new phrase.

A few extra features are:

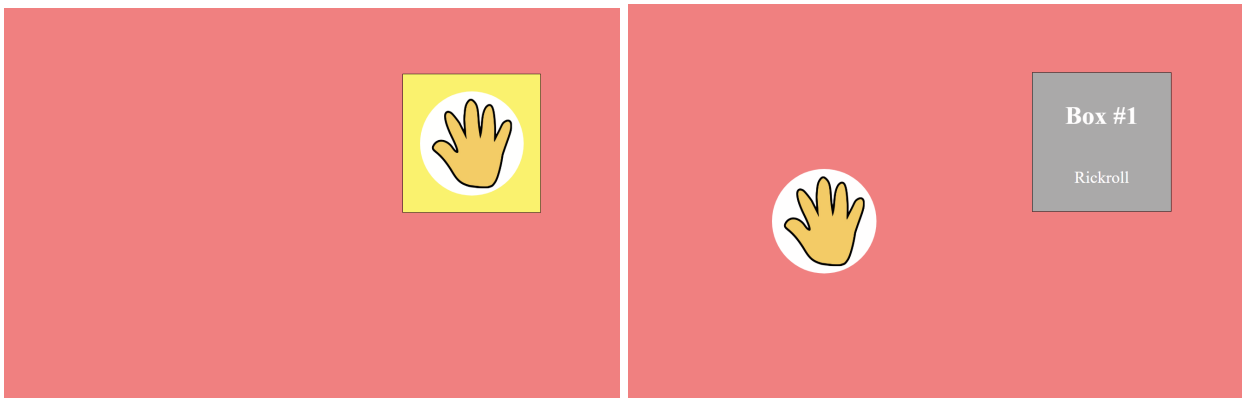
1. “easy level”(default) vs “hard level”. Hard level means the same box sounds but they are a bit faster so it’s harder to stop hovering the box on time.
2. The user can say “another” to get another phrase if they want to keep playing but the random phrase is too difficult.

Music Mix

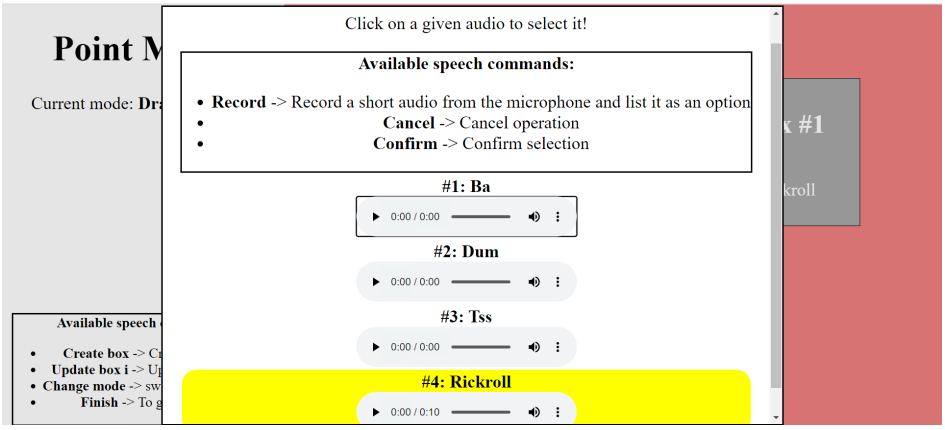
Upon entering this mode, the user will see an empty red screen. Ideally, after playing with Phrase Maker, the user will intuitively know that boxes need to be placed there. As the speech commands box suggest, the user can say “Create box”. Once they do that a new box will appear on the screen, like follows:



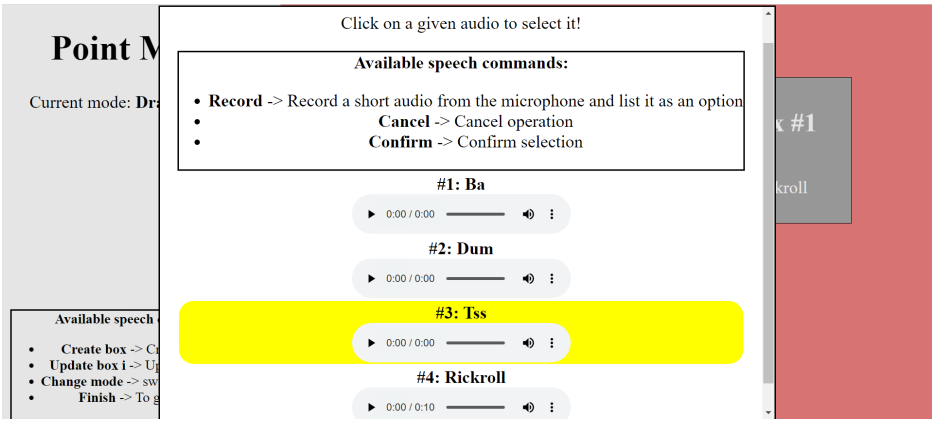
Now, once the user hovers over the given box then they’ll see that they can drag it:



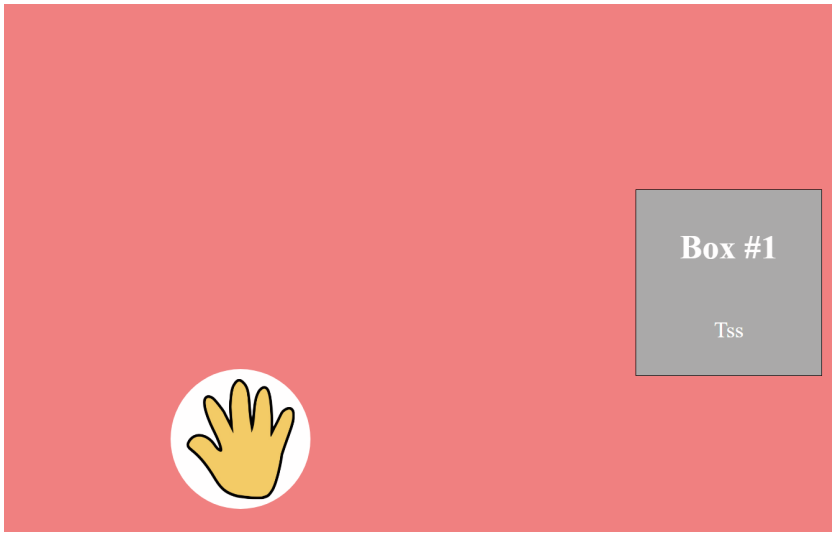
Once a box is created, they are assigned a random audio (in the example it was an audio called “Rickroll”), chosen from a list of predefined audios. However, chances are that the user will want to change this audio, which is why if they say “Update Box i” (where i is a number) then the following popup will show:



Which is the update menu for a given box. Withjin here, the user can click on the bounding boxes of the audio, to select it. For example, if the user clicks on “Tss” then the menu would look like:

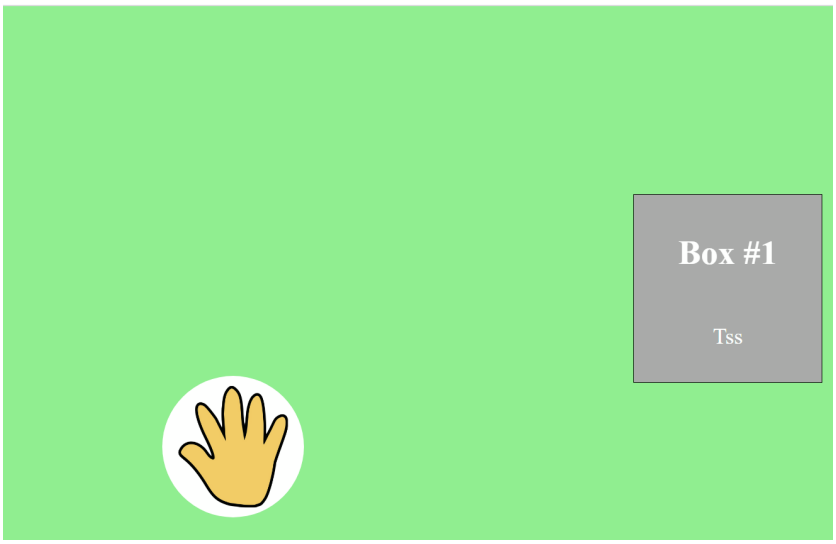


Finally, if the user says “Confirm” then the box would look like:



Not only that but if, within the update menu, the user says “Record” then a 5s countdown will start and after that the system will record any sound coming to the computer microphone. After the sound is recorded, it will be saved locally (is suggested to save it on the *assets/recordings/* folder), and the user will be able to see this audio as an option in the update menu. This can allow the user to add any custom sounds.

Finally, you may be wondering “but when will the user be able to mix music!”. The answer to that is that, as long as there is one music box setup, the user can say “change mode” to go from “Dragging Boxes” mode to “Mixing music” mode, which looks like:

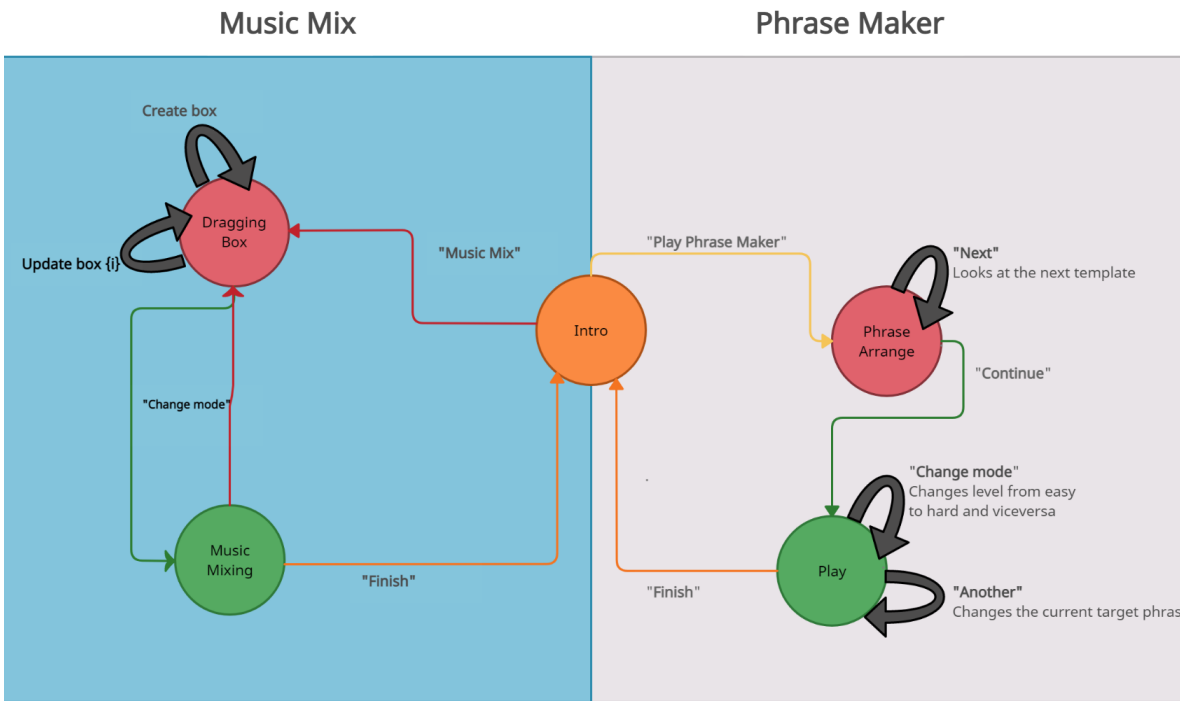


And now with going back to the green background hopefully users realize that this is similar to Phrase Maker and so now when they hover over the boxes now they can make sounds! At any time the user can change mode, and anytime on dragging mode the user can create boxes and updating the songs of each box. Once in “music mixing” mode, the user can make custom music by just moving their hand!

How the system works

State Machine

Everything starts on the “intro” state, after that it can go to either Phrase Maker or Music Mix, so here we show the state machines for both:



Most of this has been explained on the System Overview, but just to briefly explain the states:

1. **Intro**: Beginning default state, before choosing to go through Phrase Maker or Music Mix.
2. **PhraseArrange**: Here is when the user chooses the template they want to go over
3. **Play**: Main state of “Phrase Maker”, where it can change easy/hard mode and update the current target phrase
4. **Dragging Box**: State where users can drag the music boxes. They can also create a box in the screen and update the internal sound behind every song.
5. **Music Mixing**: State where users can make music when hovering over the music boxes.

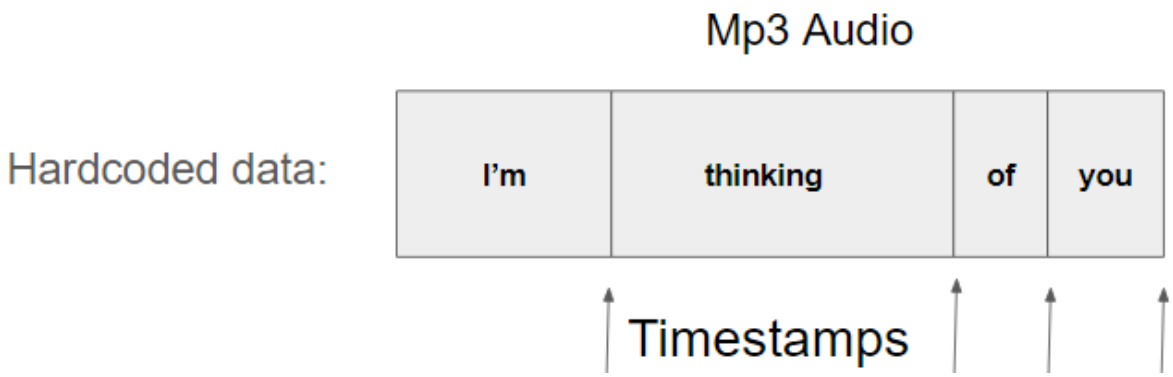
Finally, when relevant, the virtual hand can be moved by having the user move their hand on top of the Leap.

How well it works

This system works well for most of the time, but sometimes it does require to 1) be on a quite space and 2) speak loud because otherwise the speech recognition doesn’t work as one would expect. This is quite unfortunate because a good amount of the interaction is speech-dependent. On the other hand, I’d say that the hand detection is very smooth and stable, something that wasn’t the case at the beginning of the implementation. I think what

helped is that I averaged the positions of the last 10 positions, and so even when the Leap data is shaky, the screen position used for drawing the virtual data is quite stable.

Another important feature of the system is to be able to detect what word has been played in Phrase Maker. For this to work, each audio had to have a (hardcoded) timestamp, that represented the end time of each given word, as follows:



Once these timestamps are decided, the audio gets chopped into different segments, one per word. During the game, after each chunk is finished playing, a word is officially detected (for the purposes of making up the target phrase), and this triggers the next audio to play. This means that when we say "Playing the audi, that just means playing the first chunk of the audio.

Experiments

To evaluate how good the speech detection was, I went over all relevant speech commands for the app and checked how many of them were correctly identified by the system. There were some phrases that were very hard to detect (e.g., Update Box i, 30% success rate), and some of them that were easy to detect (e.g., Continue, 100% rate). Overall, among 12 phrases I tested, the system averaged success rate was 65%. Full data can be seen below:

Phrase	Total tries	Successful tries	Success %
Play phrase maker	10	8	80
Next	10	5	50
Continue	10	10	100
Change Mode	10	8	80
Another	10	7	70
Finish	10	8	80
Music Mix	10	7	70
Create box	10	4	40
Update box i	10	3	30
Confirm	10	3	30
Cancel	10	9	90
Record	10	6	60
		Average	65

The testing was performed by me at a bit higher volume than normal speaking voice. Given that I have an accent (as a non-native speaker), there is a chance that the system performance could improve if a native speaker were to try it.

Interesting Failure

Given of the processing that goes behind the scenes, there is a certain amount of lag from when the user moves their hand in real life, to when they hear (or stop hearing) a given sound. As such, for Phrase Maker there are some target phrases that are almost-impossible to achieve. This happens when the breaking point of an audio is on a small word (e.g., the, of, etc). To solve this, I tried to make sure the target phrases break on slightly longer words. For example, if there were two boxes “I’m thinking of you” and “A big storm is coming” then the target phrase “I’m thinking **of** a big storm” would be incredibly difficult, whereas “I’m **thinking** a big storm is coming” is more manageable (break words are bolded).

Unexpected Difficulty

Playing audio for the Phrase Maker game was unexpectedly difficult. The main difficulties came from the fact that it wasn’t just playing audio but I also needed to keep track when a given word was said. Even after having the hardcoded timestamps, it wasn’t clear how to check when those timestamps had passed. I wished that Howler.js (the library I used to play audios) would offer a callback every 1 ms an audio is playing, but that wasn’t the case. One option was to remember the timestamp (i.e., Date.now()) when the hand hovered a box, and then in every loop of the Leap check how much time had passed since then and match that with a given word. The problem with this is that it assumed that as soon as the hand hovered the box, the audio would start playing. Instead, there was some lag so this solution introduced somme unwanted latency. I knew that I wanted a solution that involved only the audio itself, since that’s the thing the user would be hearing. That’s how I arrived with the idea of dividing into chunks and having a callback when a chunk is finished. In this way, I can be sure that all detected words have been heard by the user.

What worked easily

Using the Leap data of the position of the hand and transforming it into “screen coordinates” was easier than I expected. It just needed a bit of tuning at the beginning and then there I was able to achieve an smooth feel between moving the hand in real life and seeing it moving on the screen.

Modifications to the system

Towards the beginning of the project (around Prototype Studio), my idea was to have a “tap” gesture that would repeat a given sound when the user keeps tapping with the finger. However, I found the Leap data to be very unreliable when it came to detect keytapping. It sometimes

didn't detect it, and sometimes when it did it detected it multiple times when it should've been just one. I sometimes even needed to do exaggerated tapping motion. As a result, I chose to move away from the tapping gesture and focused on position information. I still believe that it should be possible to implement this gesture feature, it just may take some extra care to make sure that the hand position doesn't get lost when doing that gesture.

Probably one of the main pivots I had is to go away from the 1D view of "pointing in a giving direction" to the 2d view of "moving to a given position". Early prototypes of the system used Leap data to calculate the angle offset of one of the fingers and use that info to decide which audio to play. However, I found that this was not very escalable if we wanted to have multiple boxes (for example). For these reasons, I went with the idea of having 2d boxes that the user needed to "reach" to play the sound. Now the system allows for many more boxes (i.e., sounds) to coexist.

Finally, one addition to the system (rather than a modification) was the creation of Phrase Maker. This was due to seeing many people's feedback on the Prototype and Implementation studios that "building up phrases" seems like a fun thing to do. I realized that this idea was not only entertaining, but also a good introduction to the system's capabilities (i.e., playing a sound by hovering over a box). This means that, in a way, you could consider Phrase Maker as a fun, gamified "tutorial" of Point Music.

User Study

For the user study, I simply asked three people to test both Phrase Maker and Music Mix and then asked them for feedback about the usabilitiy of the system, as well as how entertaining each of them were. Here is the general feedback I got:

1. Phrase Maker:
    a. Everyone thought that Phrase Maker wason a fun game.
    b. The easy mode was easy enough that all of them were able to finish i
    c. One person said the hard mode was a bit too fast, that there could be a middle point between easy and hard.
2. Music Mix:
    a. They said that this mode wasn't thaat fun, but it was easier to understand what needed to be done.
    b. One person said that the virtual hand was a bit too big, which made dragging music boxes confusing given that you only saw the hand moving.

What I did not anticipate

This didn't happen on the last round of feedback, but earlier I was surprised by the fact that test users usually prefer as much feedback about what they're doing as possible! I got recommendations for adding sound effects, change of colors, and things like that as a way of visual/auditory feedback.

Performance

I think Phrase Maker performed quite well, and I believe one of the main reasons why is because users have a clear goal, to make up a phrase. This makes them more engaged with the system. On the other hand, the open-endedness of Music Mix plus the lack of customization of the music makes it perform very well, in terms of user engagement.

On the other hand, I think both Phrase Maker and Music Mix suffered from the same problem: synchronization hand-ear. The system suggests that it's possible to go back and forth between two different sounds very quickly. However, if one does this then sometimes it's stuck on one sound. I believe that this behavior comes from the fact that the audio-playing javascript library Howler.js was not made for such fast behaviors, and so it may crash (or do unexpected things) once on a while.

Out–of-reach example

An example out of the scope of the current system would be to have Phrase Maker allow target phrases to be inputted by the user, rather than hardcoded. There could be a list of template phrases (i.e., just text) that are hardcoded, but the system would ask the user to repeat them and record that audio. The system would need to automatically detect the timestamps for every word or, in turn, allow the user to define those parameters themselves.

Improvements

The first thing would be to use a custom, more reliable Speech Recognition system instead of relying on the one from the browser. Alternatively, we could reduce the amount of speech commands the users needs to make (e.g., by allowing some gestures on the Leap) so that the interactivity is not fully hindered if the Speech Recognition is not great.

Tools/packages/libraries used

Package/Tool/Library	Where is it available? (eg url)	What does it do?	How well did it work?	Did it work out of the box? If not, what did you have to do to use it?
LeapJS	https://github.com/leapmotion/leapjs	It interfaces the Leap motion with Javascript that can be used on the browser	I'd say that hand position (in 3D) was smooth and accurate enough. Gesture detection (especially keytapping was a bit troublesome)	It worked by using the leap.min.js and leap-plugins.min.js provided by the staff on Miniproject 3. I think I had problems with using the "official" link
Howler.js	https://github.com/goldfire/howler.js	It allows the system to programmatically play and stop sounds, either full sound or just part of it (i.e., sprite)	It worked pretty well imo. The .stop() method for audios sometimes (very few)	Yes. With following the examples on the github it should be fine.

			doesn't stop the audio.	
Underscore.js	https://underscorejs.org/	Offers some useful helpers functions. For my purposes I only used it for the debounce method	Pretty well.	Yes.
Backbone.js	https://backbonejs.org/	Allows creation of "models" to help with modularity on a JS app.	Pretty well.	Yes. Followed the examples used in Miniproject 3.
Famous	http://famous.org/learn/components.html	Rendering engine. Makes it easier to do things like "Create a box of a given size" and add it to the DOM.	Pretty well.	Yes. Followed the examples used in Miniproject 3.
TTSMp3	https://ttsmp3.com/	To generate the voice-like audio of different phrases and export them to mp3. It even allows to add silent breaks, which was very useful for the easy version of PhraseMaker!	Pretty well.	Yes.