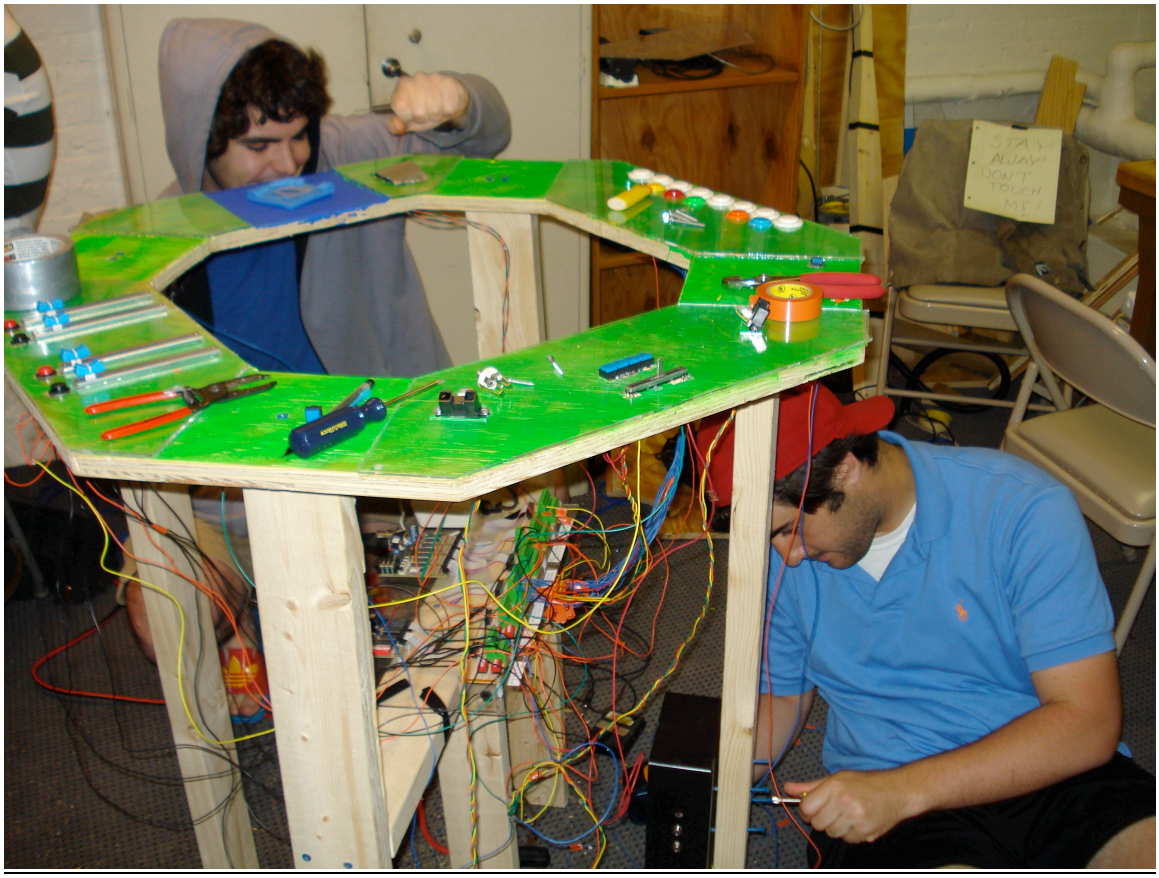
Electronic Musical Instrument Design
Spring 2008
Name: Jason Clark
Group: Jimmy Hughes
Jacob Fromer
Peter Fallon

The Octable



Introduction:

You know what they say: two is company, three is a crowd, and four is a party. It seems that so often, musical instruments are about the individual. Musicians grow special intimate relationships with their guitars, violins, drumsets, etc, and this is quite understandable: playing a typical musical instrument is very personal. Our objective with this project was to abandon this convention for a much more cooperative, collaborative, and symbiotic playing experience. We wanted personal musical responsibilities to overlap, expression to be multi-dimensional, and interaction and improvisation to not only be encouraged but to also be required. What we came out with was the Octable.

The octable is a four section, octagonal musical instrument that requires four individuals to play it. The instrument contains a bass zone, a drum zone, a lead zone, and

a mixing zone, and as a whole it includes 26 binary switches in addition to 8 continuous controllers. These sensors were housed on a wooden structure coated with plexiglass. Switches and controllers are wired to a Doepfer box, which converts the information into MIDI data and sends it to the computer. MAX was programmed to collect and organize the midi data and send it to Reason for musical synthesis.

Distribution of Tasks:

A lot of work went into this project as a whole as each section of the instrument had a hardware component, a Max programming component, a Reason programming component, and a few had a separate musical composition component. Unfortunately for our group, our collective expertise did not include electrical engineering, and so such a fundamental element of the project had to be done in a very tedious guess-and-check fashion, and that put on hold being able to progress in the software side of the project. That being said, however, our group did however distributed responsibilities effectively, and this allowed us to create such an in depth, over the top device.

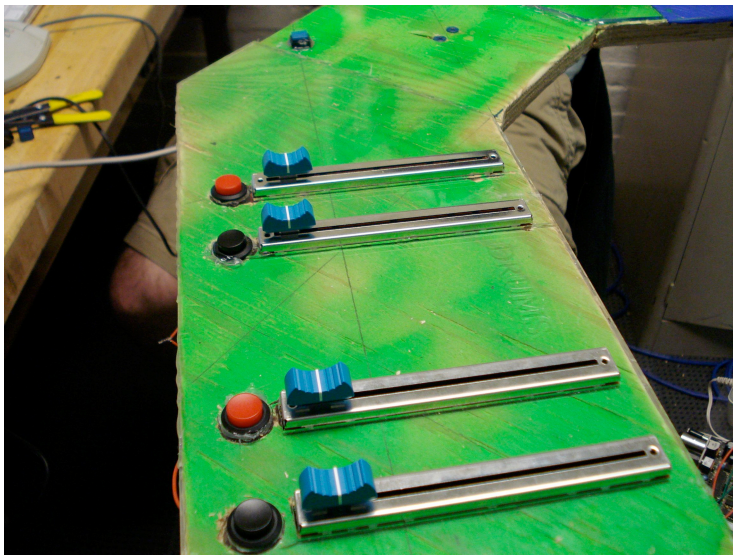
Jimmy and Pete were responsible for the wiring side of hardware, while Jake and I focused more on the acquisition and assembly of the housing. My main task, though, was max programming, which although in retrospect was quite the undertaking, I managed to keep a great schedule and pace and got most of it done before the first sensor was connected. Jake and I were responsible for the composition side of the project, which called for us to sit down and come up with drum beats for each time signature and different levels of magnitude for each beat. We also were responsible for choosing the pieces of each drum kit, as well as the sounds for each instrument in reason. Finally, Jimmy programmed the mixer in both max and reason.

Sections of the Instrument:

I) Drums

a) Hardware/ Operation

The drumming zone of our instrument was made up of four sliders with a push button switch at the base of each slider. There was also one other switch mounted to the left of all the sliders.



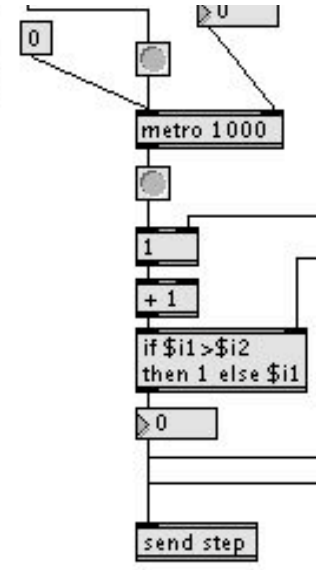
The concept behind the drums is that instead of controlling when each element of the drum kit would sound, the player would instead control *how much* of that element was in his drumbeat. The four sliders represent the bass drum, the snare and toms, the cymbals, and percussion.

At the lowest setting for a slider, the beat would not contain any of that element. As the slider is increased, that

element becomes more and more prominent in the beat, until the maximum, where that section is most complexly integrated. The buttons below each slider controls the pattern of that section within its time signature. For instance, one pattern might emphasis the on beats, while another might emphasize the off beats. Mixing and matching patterns resulted in some really cool experimental stuff. Finally, the button at the far left changes the drum kit.

b) Software

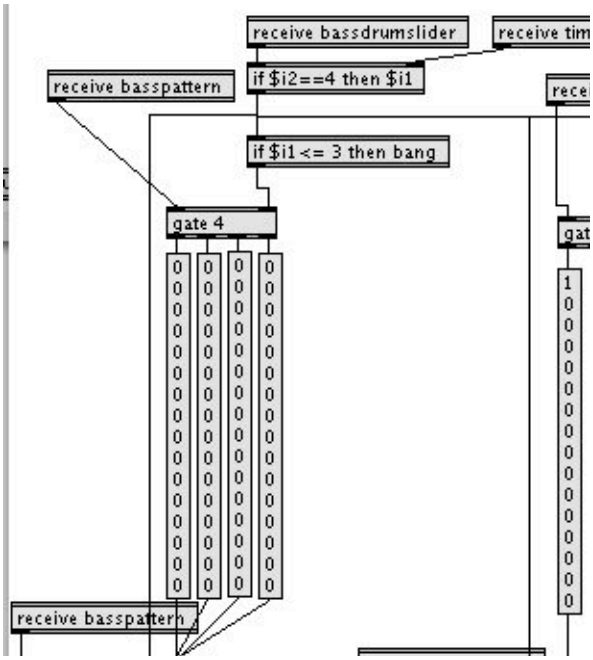
The drums, as anticipated, was the largest programming hurdle in this project, but I had a clear path in mind when the concept of the drums were first discussed and so the task was quite manageable. Each time signature had a bank of patterns, and each pattern had a level of intensity, and each level of intensity was a list of numbers (16 long for 4/4, 20 long for 5/4, etc.) full of either zeros or ones (later some twos and threes). The idea was that these numbers, in order, represented the steps of the drum sequence. A zero represented a rest, and a 1 represented a hit.



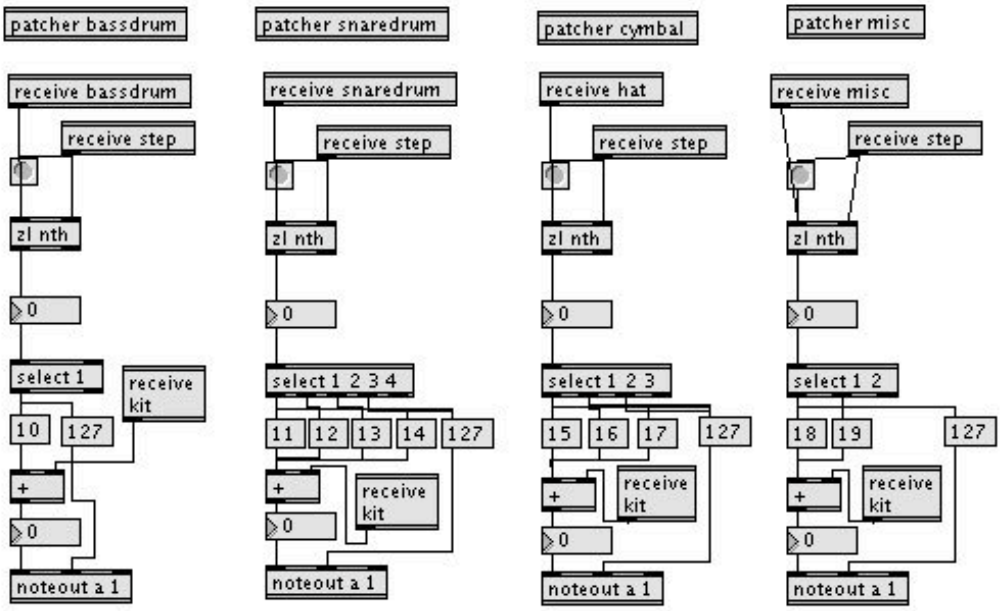
The sequencer was made by a metro object (with a tempo set by the mixer). Every bang of the metro increased the step of the sequencer by one, until it reached the end (determined by the time signature) and started over. This gave me a running step counter that I could use an argument to call the corresponding command of each section of the drums.

In order to select which bank of numbers to choose from, ranges of the slider were determined to bang in certain levels of intensity of the beat.

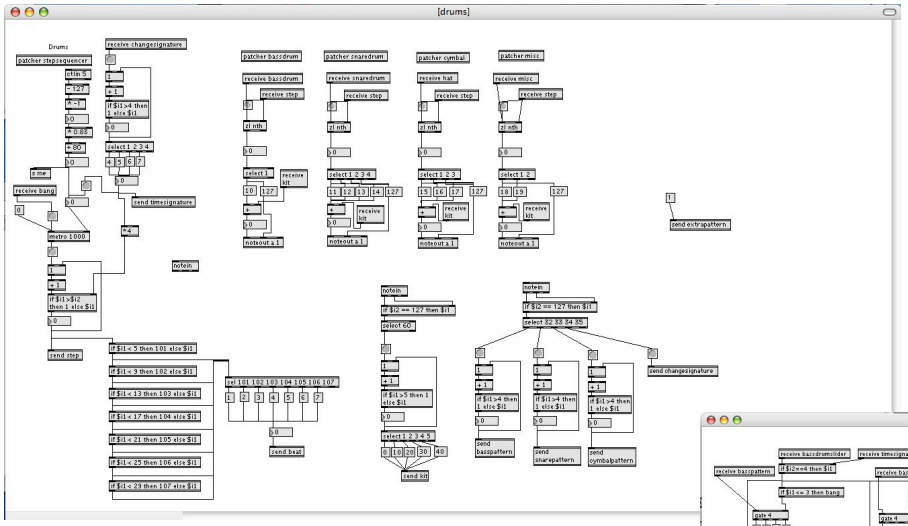
An if statement was used to make sure that the beat is in this time signature before it sends out a bank. If it is, the patcher takes the receives a slider value, and then evlautes it in a series of 9 if then statements to figure out what range that slider is in. A gate was set to determine which pattern should be sent (chosen by cycling through using the button below the sliders), and so when a slider is in the range, a bank is banged. (This screen shot shows the lowest range of the slider, where all patterns are off, and therefore there are all zeros)

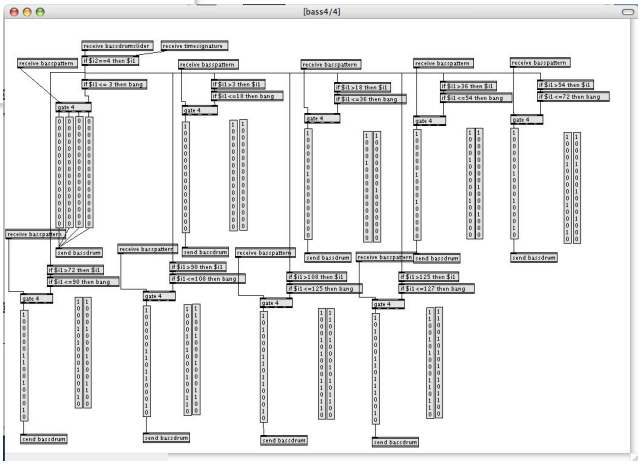


This bank gets banged into a ZL nth object which is used to “take the nth argument from this list.” Therefore, if I have the list and a running steps sequencer, I now can look for 1’s using a select command and send out notes to reason.



A change in drum kit command scales the notes that are sent into the NN-19 module, which was armed with 5 different kits of 10 hits each. There was a rock kit, a conga kit, a hip hop kit, a natural disaster kit, and a glitch kit.



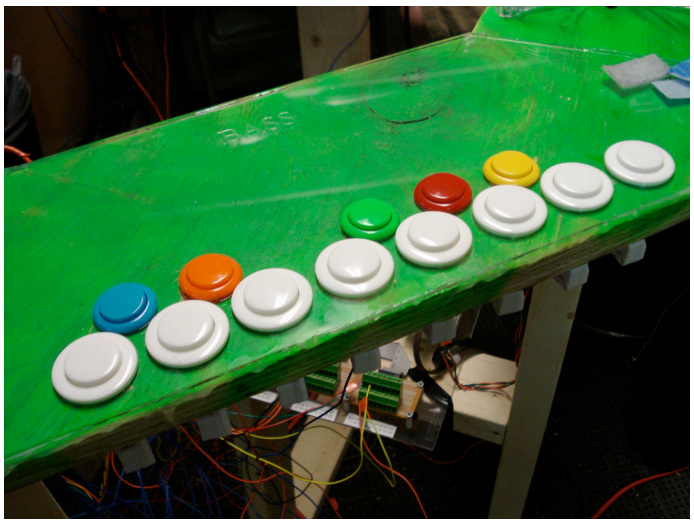


Drum patch with (early version of) one bank of note banks for the bass drum in 4/4. There is an identical patcher for each section of the kit in each signature

II) Bass

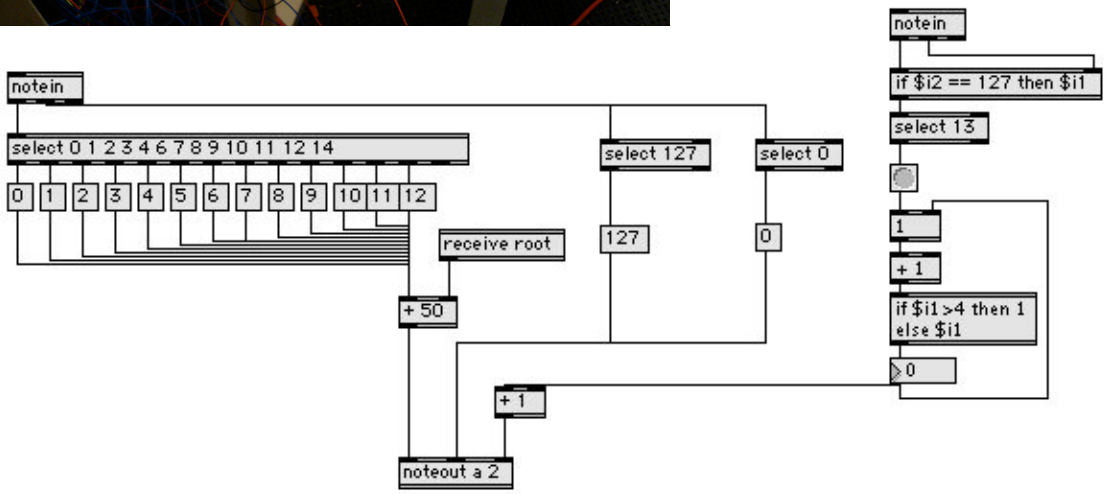
a) Hardware/ Operation

The bass section consisted of 13 arcade style push button switches, and one additional small push button switch. Operation is identical to a keyboard, and the additional switch selects voice.



b)Software

Programming the bass was very basic. From the note in, max looked for specific values corresponding to the buttons and assigned them their new pitch value. Channel was determined by the voice select button. In reason, subtractor patches were created on the corresponding channels to change the sound of the bass.

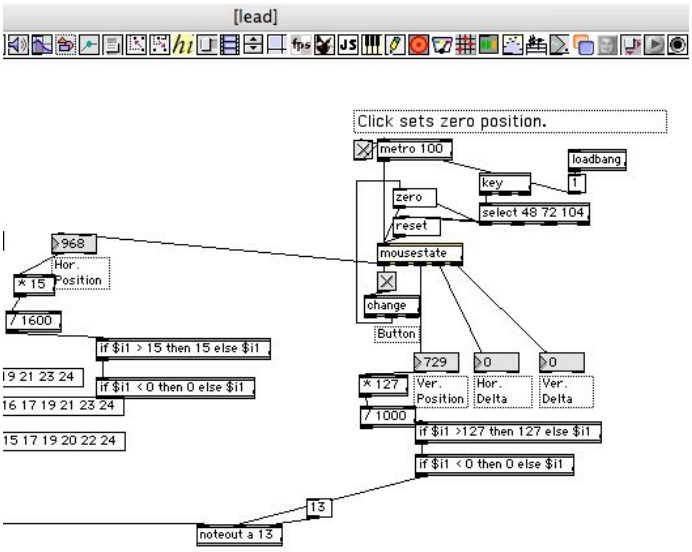


III) Lead

a) Hardware/ Operation

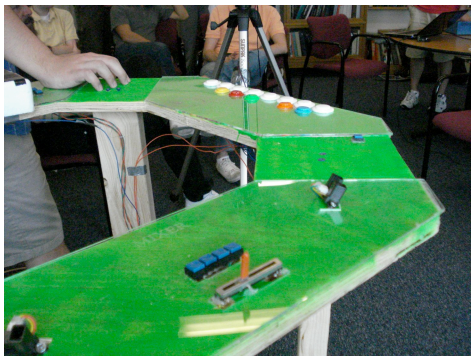
The lead originally was going to be an XY pad, but after we could not get that to work, we opted for staying in the two axis family and we used an optical mouse mounted upside down. X- coordinates determined relative pitch, determined from either a major, a minor, or a pentatonic scale, and Y-coordinates determined note velocity. The player moves a pad over the exposed optical reader in the mouse in order to change his "coordinates."

b) Software



I created a modified version of the example patch from reason which makes use of the "mousestate" object. This object retrieves coordinates of a connected USB mouse. For the X-axis, I scaled the maximum and minimum coordinates over 15 values. Using those 15 values and a ZL nth object, I could select relative values from a bank of notes, corresponding to the major, minor, and pentatonic scales (chosen by one of the buttons) For the Y-axis, I scaled the maximum and minimum coordinates over 127 values to correspond to note velocity. Notes are generated either every step, every other step, or every 4th step of the sequence as determined by one of the buttons

IV) Mixer

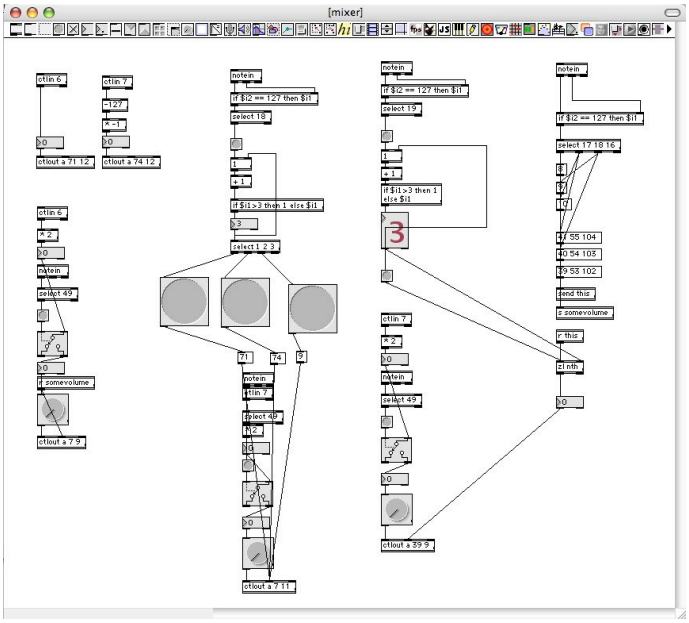


a) Hardware/Operation

The mixer consists of two IR distance sensors, one slider, 4 buttons, and a footswitch. The slider determines the master tempo of both of the tempo dependent units. The four buttons are used to select which instrument to mix, and which effect to change. On each instrument, the left IR sensor always determines volume. The right cycled through delay, filter frequency, and filter resonance. The foot pedal, when engaged, tells the IR sensors to start reading values, and when disengaged, tells them to hold that value.

b) Software

Jimmy did the programming for the mixer, but by the looks of his patch, it appears that he is using a graphic gate, which is banged by the foot pedal in order to start and stop taking values. Jimmy used max to control mixer settings in reason, and he put all of the instruments and effects into that mixer. He used a ZL command to determine

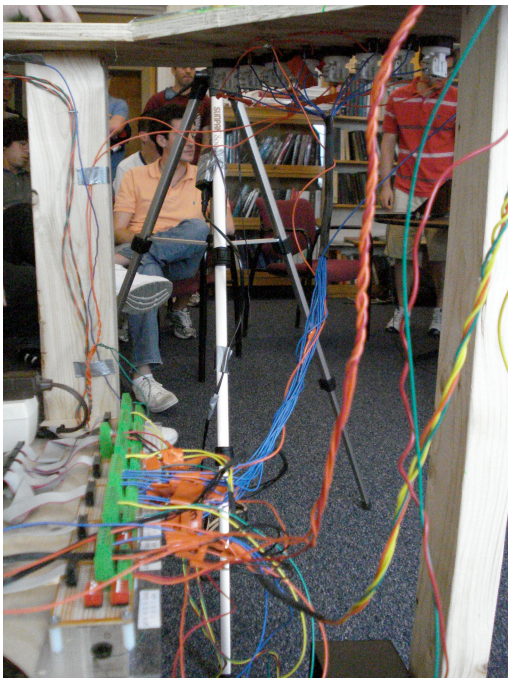


which effect from a list of effects for each instrument to modify.

Reason patch

Obstacles/Problems:

The biggest problem we encountered in this project was trying to get the Tactex MTC xy pad to work. We needed to use a serial to USB adapter that we had to make ourselves using an online wiring diagram, and we had to also find a max object that knew how to interpret its data. Unfortunately, this didn't work out, and we had to swing into a less glamorous, less fun to play plan b: The optical mouse.



Other than that, the only issues involved other academic responsibility and time. It would have been nice to have gotten it wired much earlier on, but it really was not a possibility with 0 electrical engineers in the group. If we had more time, I would have liked to have seen some more expression control in the bass zone, as well as a cooler lead instrument.