Scott Jacobson
John Bradley
Joo Keng

Loop Loop Revolution

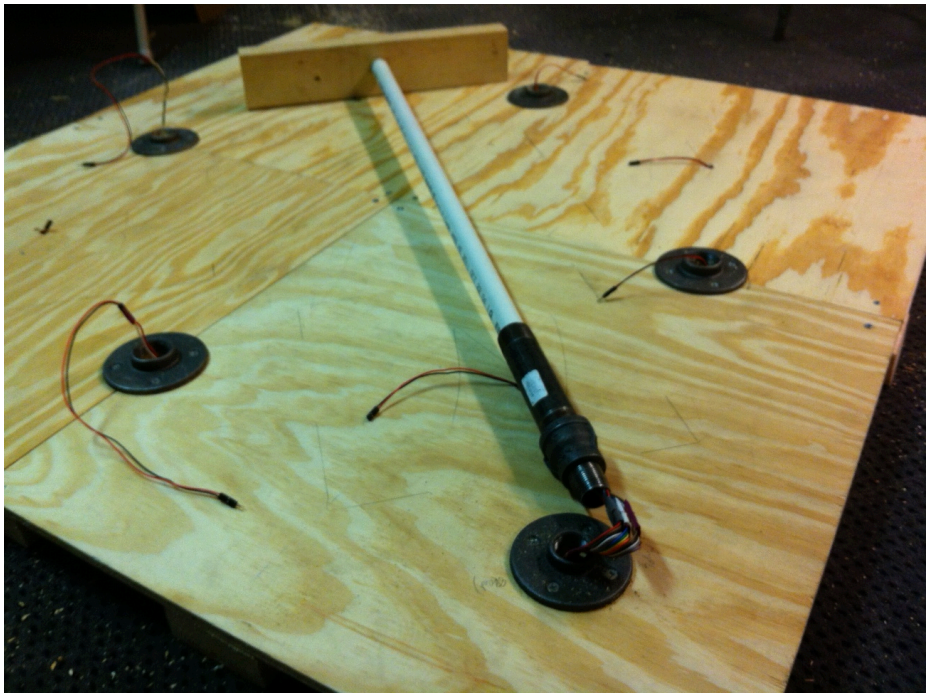
Introduction and Goals:

We wanted to make an instrument that incorporated large, visible movements as part of playing it as well as resulted in a dancing-like performance of the instrument. We used the videogame Dance Dance Revolution as inspiration, and decided 4 floor panels wouldn't be enough so we added 4 hand panels to give a total of 8 sensors to control the playing. Hip hop was an area that intrigued all of us, so we decided we wanted to make a looping, sample-based hip hop beat making dance station that was completely controllable by the performer.

Parts and assembly:

Platform:

The platform was made out of a 2'x2' frame of 2x4s with a plywood floor on top of them. We connected PVC pipes for the hand panels and control panel with 1" flanges and nipples that fit into the pipes so they could be screwed in and unscrewed as needed. The floor panels were rubber cemented on. The wiring was all done through holes in the frame that went up through the necessary flange as well as the control panel flange.



Hand Panels:

The hand panels were about 3 feet of PVC going into a small 4x4 square of 2x4 that had a hole drilled on the bottom and an FSR attached to the top. The wire was sent through a hole in the side of the PVC into the base, where it hooked up with its connector.

Foot Panels:

The foot panels were an interesting challenge, as we had to find a way to make the base reading on the FSRs zero, and then transfer force to FSRs on the floor when the large 1'x1' aluminum panels were stepped on without transferring too much force and. We ended up mounting the panels onto a special high-recovery rubber foam, cutting a square piece out in the middle where the FSR was going to be, and then placing a extremely thin piece inside to make contact with the FSR when it was pressed.

Control Panel:

The control panel was a special type of hand panel that was attached to a 2' x 1' piece of particleboard on which 12 buttons, 2 toggle switches, and 2 slide potentiometers were mounted, as well as the arduino. A ribbon cable carried the signal from the platform's sensors to the control panel, and a breadboard was used to provide the ground to all of the switches and FSRs.

Programming, Arduino-Level

The FSRs and slide potentiometers were hooked into the first 10 analog inputs of an Arduino Mega, while the buttons and toggles were assigned digital inputs 22-36, all connected through a breadboard to a common ground and running off of the 5v signal provided by the Arduino. The pull-up resistors of the Arduino's inputs were set to High for all inputs to try to prevent any noise, however, having all digital channels simultaneously transmitting data clogged the serial connection and made the entire operation incredibly noisy. Instead, the 3 unused analog ports 13-16 were hijacked to simultaneously transmit a 10-bit binary number that represented the states of each of the digital ports (and by extension the buttons) as being active or not active by assigning each button in a 10-button string its own bit (button 0 was 2^0, button 1 was 2^1, ect) and transmitting the sum. Interestingly, due to the internal exponent function within the Arduino (and presumably as a result of truncating beyond 2 bit numbers), I had to modify the code even further to add an extra 0-bit number to every number greater than 2^1. Finally, the serial code was transmitting instantaneously and ready to be sent into Max.

Programming, Max-Level

Using the Arduino Max object, expanded for the Mega, we used an unpack function for the digital read-ins written by a former student that included the '&' – bitwise intersection – object to give a 1 or 0 on each button. The analog inputs used the standard

Arduino analog input feature of the patch, with the sends of all objects renamed as needed. Besides the Arduino Patch (renamed LoopLoopDuino), there was one more patch within the project:

The Looper Patch

The looper patch contains what was probably the most difficult usage of sensors: how to interpret the FSR data. We decided on duplicating the success we had in the Muziks Cube with creating two thresholds for the FSR – since its input is fairly noisy, there would be a minimum read in of 250 in order to trigger a note on, with a slight delay to allow the FSR to achieve its peak reading before sending a MIDI Note with the scaled velocity from the FSR – then, once the FSR dipped back below 200, it would flush and send a Note Off. These two thresholds had to be different because the FSR would modulate almost instantaneously between thresholds when they were equal, resulting in a plethora of note on and offs that left the MIDI stream too unstable to play anything. Each Mode represented a Master Transposition of all the notes on the patch that corresponded with the regions on the NN-XT, and also gated the signals through a separate "seq" object, one for each region, that accepted a record and stop command. The master tempo was controlled by one of the slide pots and resulted in a metronome that sent a metronome sound to reason as well as assigning the length of the 4-bar loop for recording (after 4 bars, it sent a "play" command).

Programming, Reason Level

The Reason rack consisted of a NN-XT sampler mapped to 6 different regions, each one representing a different patch or mode, and each mapped to a separate stereo output in the module. Each region consisted of 8 chromatically triggered samples, representing the 8 different controllers on the LLR. The samples were taken from some songs we found to be interesting sources, among which was We Are The Champions by Queen, Fire by Jimi Hendrix, and Larger Than Life by The Backstreet Boys.

Problems / Conclusion

It was unfortunate that the seq object is only designed to take in MIDI notes after they have been generated – because of this, we struggled to find a way to send the notes through seq and then through output – makenote requires a duration beforehand, and MIDIformat stopped working intermittently. Because of this, we had to make an adjustment and use a MIDIPipe to provide the pipe between the notes being created and the MIDIIn and then seq object. It resulted in some interesting things with the seq object, one of which being that the object loops back when the sequence is over, but not necessarily when the 4 bars are over, resulting in some awkward-length loops that sounded "off". In hindsight, maybe we should have used the sequencer built into reason to be sure that it snapped to a grid we could control and to have the benefit of visual sequencing.