The Wonderstick

A Strange Thing that Makes Strange Sounds

designed and built by Ben Machlin, Edward Simendinger, and Will Mairs

overview

The Wonderstick, an iteration of the original Wondertube, is a three player generative music performance system consisting of a large PVC tube and three handheld controller units. The three players each hold a controller unit and stand equally spaced around the tube. They each use their position and distance from the center tube, in addition to data from the controller, to manipulate meta events that generate sounds for three components: percussion, melody, and harmony. Each component responds differently to its player's position and the data sent by that player's controller. The three components come together to make up an immersive, atmospheric, and unique sound, shaped in real time by the players movements and actions.

building off the previous version

The design of the Wonderstick is informed in great part by the successes and failures of the Wondertube, its spiritual and material predecessor. Whereas the Wondertube could only measure a player's movement with respect to the center tube in one dimension, the Wonderstick captures two different dimensions of movement. This is achieved through additional sensors and data manipulation, described in greater detail below. A great drawback in the playability of the original Wondertube was the clunkiness and obscurity of the controller units, rigid 4x3x3 boxes lacking visual feedback. The Wonderstick has more ergonomically designed controllers that provide feedback to the user, allowing for easier and more intentional manipulation of

meta events and parameters in each component. A major success of the original Wondertube was how much the hand controller's sensors added to the playability of the instrument. When designed the Wonderstick's hand controllers, we chose to add more sensors to increase the depth of interaction.

sensors & materials

The Wonderstick system is composed of the Center Tube and three hand controller unit, each connected to the Center Tube by ribbon cables. The tube itself is a 4" section of 6" diameter PVC piping. Six IR distance sensors are mounted 4" from the top of the tube, divided into three groupings of two sensors, equidistant from each other. The groupings of sensors divide the tube into three sectors (figure 1). Each sector represents the playing zone for one player; the two IR sensors measure the player's distance from the Center Tube, and the difference between readings from the two sensors can be used to calculate the player's angular position relative to their sector. The Center Tube serves as a housing for the breadboards and

Connected by three 8' lengths of sour-straw colored rainbow ribbon cables are the three controllers. Each controller contains two tactile momentary buttons, two toggle buttons, one linear soft potentiometer, and one 25mm translucent RGB LED (figure 2). These are mounted flat against a 5x3" piece of 1/4" plywood. We modeled the placement of these sensors after the design used on the controller from the original Wondertube. To improve on the clunky and unclear controllers from the first iteration, we chose a flat design that would fit comfortably in your hands (figure 3). Toggle buttons allowed us to replace the tall switches, and flat-mount tactile buttons were an

excellent replacement for the 3” tall arcade buttons. The soft pots were an ideal flat and ergonomic alternative for the linear slide potentiometers, and solved resolved another issue we had experienced with the slide potentiometers. Whereas the sliders sent a continuous stream of data, which had to be gated carefully within Max to avoid errant values sending accidentally, the soft pots only send a value when you touch them, and otherwise stay at 0. This allowed up to simplify our data reading patches in Max.

We chose to add an LED to the front of each controller to solve an issue encountered with the first iteration. Although the switches themselves did provide some feedback as to the values they were sending, it was very difficult from a glance to know what values the switches held. Since all three components of the Wonderstick implement different functions for the buttons and soft pot depending on the state of the toggles, being able to discern the state of the toggles was very important, and the quicker this could be done the better since that allows for fluid and intentional manipulation of the sound. Adding the LED allowed us to give the user additional visual feedback as to the state of the controllers toggle buttons; the LED shines a different color depending on what combination of toggle buttons are on or off. The four possible toggle combinations are represented with a red, yellow, blue, or purple light. The colors provided an easy way to tell just from a glance what functions are available from the hand controller.

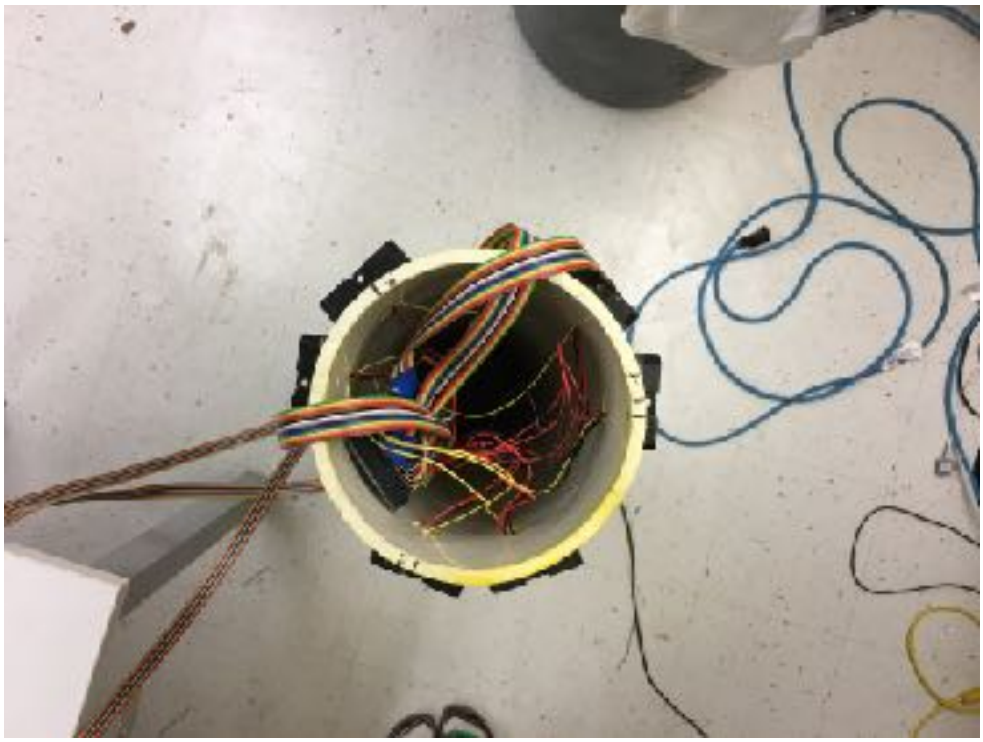

figure 1. top view of the Center Tube. IR sensors are the black, outward facing rectangles

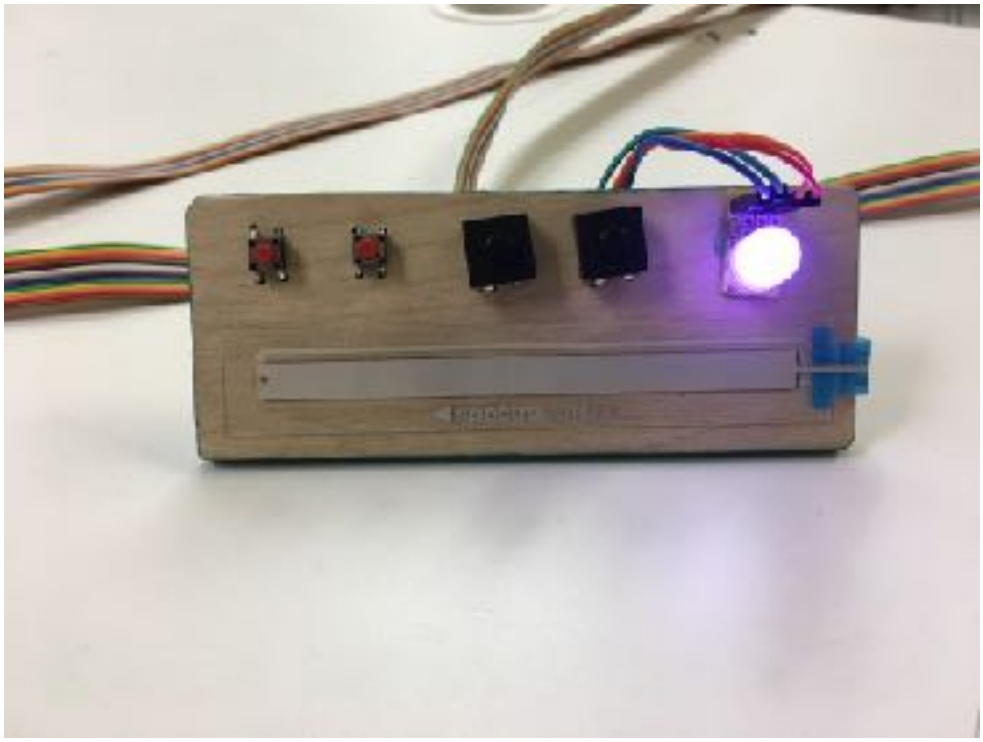

figure 2. front view of a controller. momentary buttons shown in red, toggle buttons in black

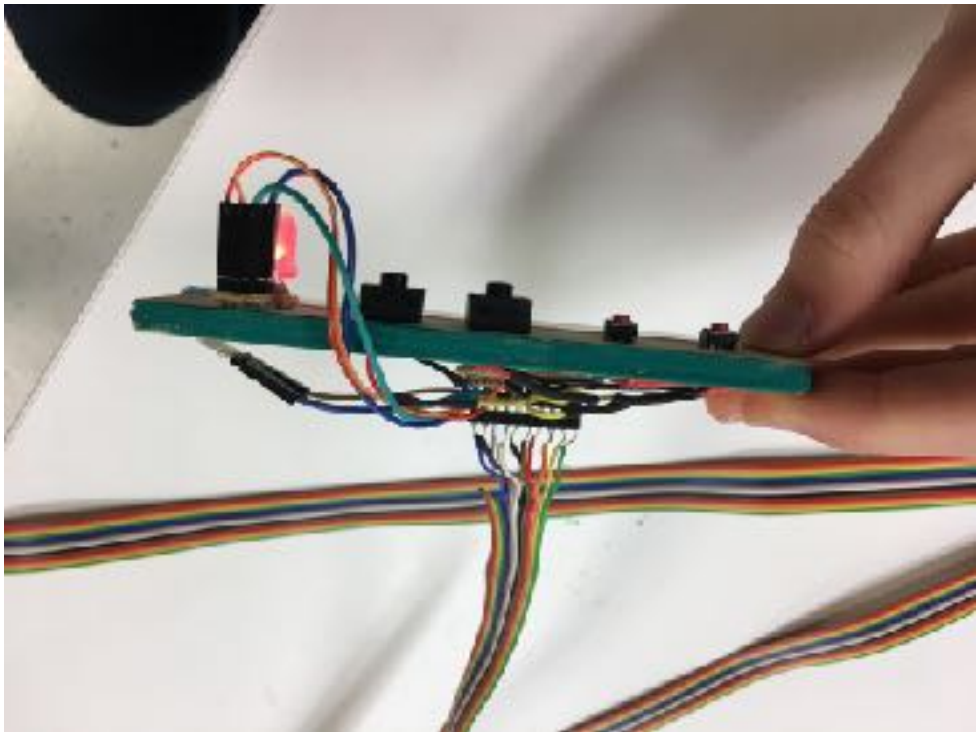

figure 3. our surface-mount sensors and LED allowed us to achieve a very slim and ergonomic design

figure 4. hot glue, though ugly, gave us the best adhesion since the sensors were somewhat too delicate to clamp securely

construction

Construction on the tube itself was very straightforward, as we were mostly following a protocol established during the first phase of this project. Each of the three IR sensors being added to the tube (to comprise six sensors in total mounted to the tube) were disassembled, had their original male connectors desoldered, and had new wire leads soldered directly to the boards. Three holes were drilled 2” from the already installed IR sensors, so that the new sensors could be installed adjacent, creating the three banks of two sensors. We had trouble with the fragility of the IR sensors when one day we had discovered the casing of an already mounted sensor badly cracked and coming apart. Not wanting to take chances applying great pressure to clamp the sensors while glue dried, we opted for hot-glue, which did the trick, though leaves something to be desired in terms of finish (figure 4). A coat of yellow spray paint and some decorative flowers finished off the Center Tube.

To make the bases for the three controllers, we cut out three 3x5” blanks from a piece of 1/4” plywood. These blanks were clamped together to sand their sides flush and to drill matching holes to mount the buttons. Short red and black leads were soldered to the terminals on the buttons, and these were in turn threaded through the holes in the bases. A spot of glue beneath each of the buttons affixed them to the board. The adhesive backing on the soft pot was enough to hold it beneath the buttons.

In order to mount the LEDs, and maintain an orderly, modular design, we created three small LED modules. These modules consisted of a small piece of perf board, four header pins, three 1kOhm resistors, and one common anode RGB LED.

The resisters were wired in series to the RGB leads of the LED, and then these, along with the common anode, were in turn connected to the header pins. Each of these modules could then be connected quickly friction fit other header wires or directly to the Arduino and easily receive signals.

We applied a similar modular philosophy in connecting our sensors and ribbon cables. To one end of the ribbon cable, we soldered header pins to make easy connections to our Arduino, and to the other end we soldered small ten-pin connectors. Similarly to the leads of the soft pots we soldered three header pins so that we could easily connect to the soft pots. Beneath each controller is a small bread board with a ten-pin connector that the ribbon cable can plug into. This bread board serves to make connections between ground and the buttons and soft pot, send 5V from the Arduino to the LED and a 1kOhm resistor in series with the soft pot, and rout digital input signals from the buttons to the Arduino, analog signals from the soft pot to the Arduino, and digital output from the Arduino to the LED (figure 5). This modular approach helped us maintain order in what had become a tangle of messy connections in the previous version of the controllers. Some sanding around the edges and corners and a quick stripe of green around the edges finished off the controllers.

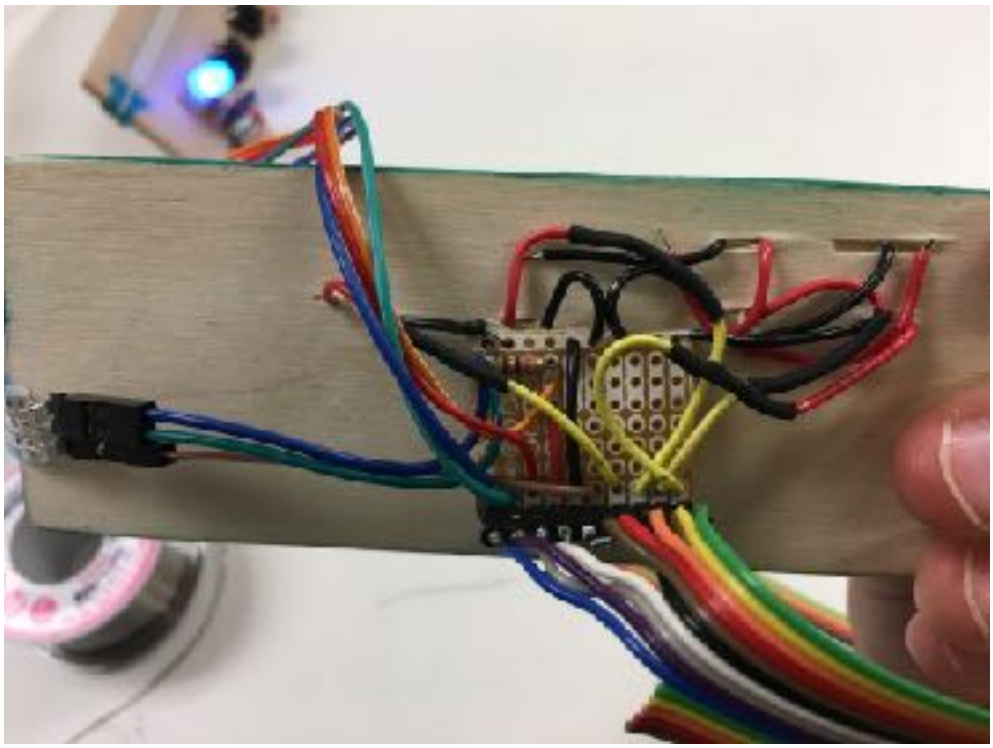

figure 5. the small wire-wrangling breadboard beneath each controller

```
if(digitalRead(p---Toggle1Pin)==HIGH) {
  if(digitalRead(per-Toggle2Pin)==HIGH) {
    digitalWrite(perRedPin, LOW);
    digitalWrite(perGreenPin, HIGH);
    digitalWrite(perBluePin, HIGH);
  } else {
    digitalWrite(perRedPin, LOW);
    digitalWrite(perGreenPin, LOW);
    digitalWrite(perBluePin, HIGH);
  }
} else {
  if(digitalRead(per-Toggle2Pin)== HIGH) {
    digitalWrite(perRedPin, LOW);
    digitalWrite(perGreenPin, HIGH);
    digitalWrite(per-BluePin, LOW);
  } else {
    digitalWrite(perRedPin, HIGH);
    digitalWrite(per-GreenPin, LOW);
    digitalWrite(perBluePin, LOW);
  }
}
```

figure 6. Arduino code reading the state of the toggles and coloring the LED accordingly

Arduino

Our Arduino code is almost identical to the default Arduino2Max program, with a small addition to handle the state of the toggle switches. Rather than send all of our sensor data straight to Max, then parse the state of the toggle switches, and send LED signals from Max to Arduino, we included a brief snippet of code within the main loop of the Arduino program that reads the values of the toggles, and colors the LED one of four colors based on those values (figure 6). The digital values of these toggles, along with the digital values of the buttons and the analog values from the IR sensors and the soft pots, are sent to Max, where they are parsed.

Max

Max contained the heart of the Wonderstick. The generic Arduino2Max patch was our basis, and we used it to route data to each of the three components (figure 7). In addition, this central master patch contained the metro object that served as a central timekeeper for all of the sub patches that implemented the three components of the Wonderstick. The three of us each continued with the components we had chosen when building the Wondertube (I chose percussion, Ben melody, and Edward harmony) and refined and rebuilt our patches.

The heart of my percussion patch (figure 8) is two sequences (shown in purple and yellow) containing note values that are looped through when the patch receives bangs from the central time keeper. One of these sequences stores only bass-drum values, while the other can contain any of six values corresponding to six sounds I designed in Reason, which are discussed below. While researching generative music I

found a paper by a computer scientist at McGill University, in which the author describes an algorithm that generates world beats (http://cgm.cs.mcgill.ca/~godfried/publications/banff.pdf). I was struck by this article, and an implementation of this algorithm in a Max js obeject (figure 9), named after its designer, a physicist named Bjorklund, allowed me to generate more cohesive and interesting sounding beats.

I used one mode of the controller, the Yellow mode, to implement new sequence generation. Using the slider, I can select the number of pulses to generate in the sequence, and pressing the right button populates both sequences with those pulses. Pressing the left button uses these pulses as a seed to generate new notes in the topmost sequence, based on a probability table. The probability table defines it as more likely for bass driven notes to become click-y, high frequency toms and hi hat hits, than for these high-frequency sound to go to bass sounds. This way, by keeping one sequence dedicated to the kick drum, continually generating new sequences based on this probability table creates a diverse spectrum of sounds.

The Red mode toggles the two sequences, with the slider adjusting dry/wet balance on a gated filter in Reason, while the Blue mode controls the global metronome. The Purple mode controls the duration of notes being sent to Reason, in addition to implementing a retrig effect, a continuation of a feature from the previous iteration that I found added enormously to the playability and performability of the instrument.

In order to use my movement to instill as much drama and action as possible on the sounds being generated, I mapped the data from one of my IR sensors to midi velocity. This way the farther I stood from the Center Tube, the quieter and calmer the

sounds, and as I approached the Tube, the sounds crescendoed dramatically. I used the second IR sensor to control the number of steps in the sequences being generated, so that I could toggle between even sounding 16 step sequence, and more unusual 6 and 5 step sequences.

Reason

My reason patch was built around six sounds designed using the Kong drum designer. I preferred the noises generated by the synth engines, rather than the physical models, and used compressions and filters to achieve click-y, spacey tones (figure 10).

The real beauty of my Reason patch came from the Alligator triple filtered gate. Though the intricacies of what exactly is going on are somewhat beyond me, I modified a preset patch to create a filter that randomly boosted resonant frequencies of what came into it. This created lovely reverberating and echoing space tones, which together with phase distorting and delay, created a terrific atmospheric quality.

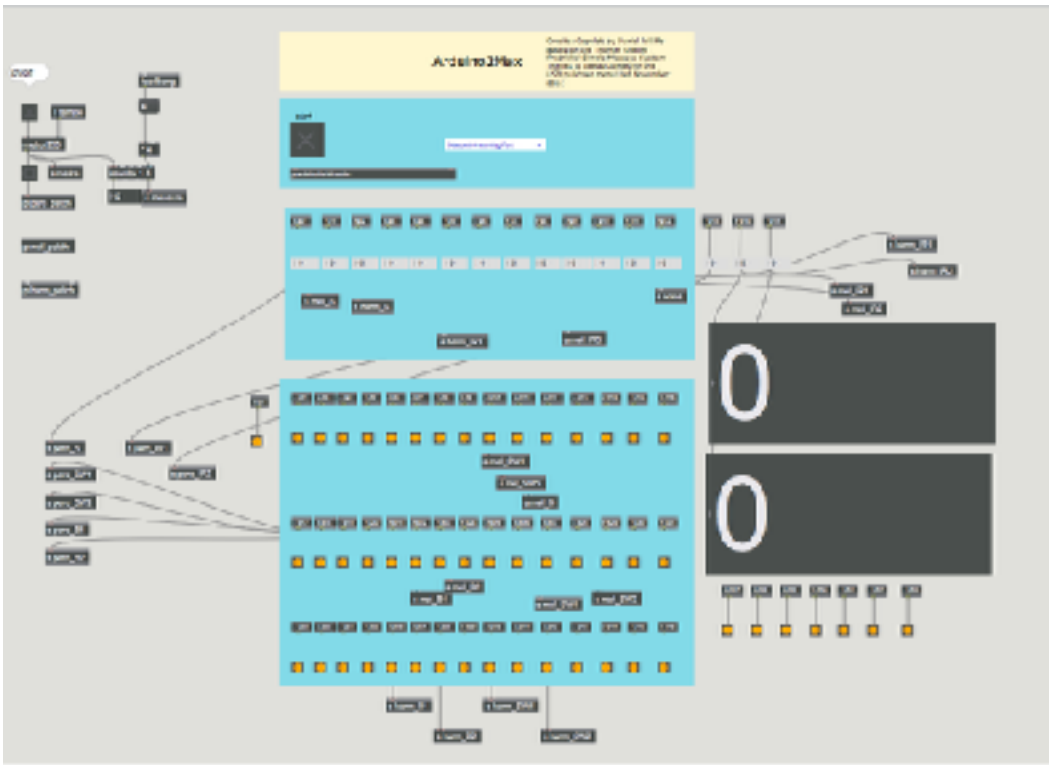

figure 7. the main patch sending data from arduino to sub patches as well as bangs from a global time keeping metro

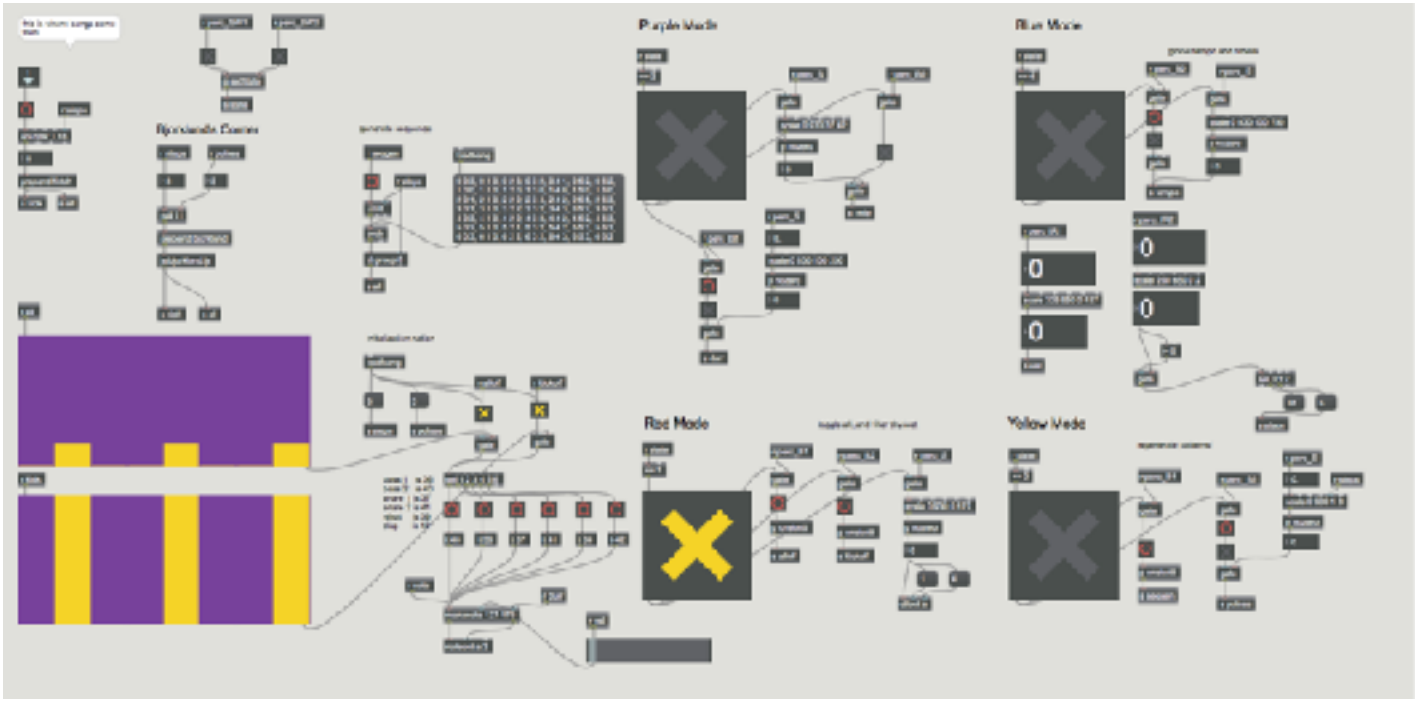

figure 8. the percussion patch, showing button and slider functions for the four controller states, as well as the two sequences

```
inlets = 1;
outlets = 1;

function bjorklund(steps, pulses) {

    steps = Math.round(steps);
    pulses = Math.round(pulses);

    if(pulses > steps || pulses == 0 || steps == 0) {
        return new Array();
    }

    pattern = [];
       counts = [];
       remainders = [];
       divisor = steps - pulses;
    remainders.push(pulses);
    level = 0;

    while(true) {
        counts.push(Math.floor(divisor / remainders[level]));
        remainders.push(divisor % remainders[level]);
        divisor = remainders[level];
           level += 1;
        if (remainders[level] <= 1) {
            break;
        }
    }

    counts.push(divisor);

    var r = 0;
    var build = function(level) {
        r++;
        if (level > -1) {
            for (var i=0; i < counts[level]; i++) {
                build(level-1);
            }
            if (remainders[level] != 0) {
                build(level-2);
            }
        } else if (level == -1) {
               pattern.push(0);
        } else if (level == -2) {
            pattern.push(1);
        }
    };

    build(level);
    outlet(0, pattern);
}
```

figure 9. a javascript implementation of Bjorklund's algorithm

figure 10. the kong drum designer and the alligator filter

difficulties and successes

Many of our greatest difficulties with this project came from fragility of connectors, be they in the IR sensors, between headers, or between the ribbon cable and the small bread boards beneath each controller. We struggled with this on demo

day, unfortunately, and we regrettably could not demo all of the features we had worked so hard to develop. Although I still find Max to be a fairly confusing paradigm, I did feel much more at ease during this project. In particular I felt very pleased at my implementation of Bjorklund's algorithm to build kick drum sequences. I am also happy to look back on how my soldering and small parts assembly skills have improved. After desoldering and attaching new leads to six IR sensors, and putting header pins on all the soft pots and LEDs, I feel like my soldering chops have developed tremendously. Diving into Reason taught me more than I knew existed about synthesis, and opened a new world of music and sound creation. All in all, I am very pleased with the final version of the Wonderstick, the sounds that we taught it how to generate, and the things I learned along the way.