CMPO 381 FINAL PROJECT DOCUMENTATION

DECLAN DOHERTY-RAMSAY #300165300

LECTURER: JIM MURPHY

DUE: 01/11/2014

1. Overview

For my final project for CMPO 381, I decided to make an interface that was fun, familiar, and flexible. I wanted to push my self in a direction I hadn't explored before in my degree, with most of my work up until this point being in creating sound sculptures and high-concept projects. My resulting prototype project, *Everyone Loves A Slinky*, brought together those key aspects through a classic toy, as well as some personal touches of my own along with a musical pun or two. Two modified slinkies allow people to modify the sound output, with Slinky 1 controlling a pitch shifter and band pass filter, and Slinky 2 controlling a sample's playback speed and the mix of reverb (creating a "spring reverb"). Slinky 1, which was the larger of the two, was controlled by a pulley system with the slinky contained in a clear tube; Slinky 2 was handheld. Both had their stretching capacities limited by zip ties, based on the limitations of the sensors.

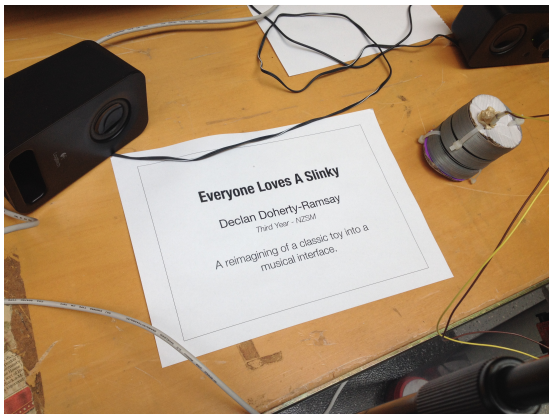

Figure 1: Everyone Loves A Slinky

2. Software

For this project I used two LDRs as my sensors with each LDR located inside the Slinkies, giving information about how open or closed the slinky is. This information was transferred from the Arduino chip to ChucK through USB. The values of Slinky 1 had a rolling average applied before going to ChucK, creating an even smoother response.

```
const int numReadings = 10;

int readings[numReadings];      // the readings from the analog input
int index = 0;                  // the index of the current reading
int total = 0;                  // the running total
int average = 0;                // the average

int inputPin = A5;

void setup()
{
  // initialize serial communication with computer:
  Serial.begin(9600);
  // initialize all the readings to 0:
  for (int thisReading = 0; thisReading < numReadings; thisReading++)
    readings[thisReading] = 0;
}

void loop() {
  // subtract the last reading:
  total= total - readings[index];
  // read from the sensor:
  readings[index] = analogRead(inputPin);
  // add the reading to the total:
  total= total + readings[index];
  // advance to the next position in the array:
  index = index + 1;

  // if we're at the end of the array...
  if (index >= numReadings)
    // ...wrap around to the beginning:
    index = 0;
```

Figure 2: Smoothing

2.1 ChucK

The initial values were very heavily on the bottom end of the scale, between closed and slightly open, and this meant to get a really strong response throughout the range of movement some rather harsh remapping had to take place, including having to multiply the value by $1.0x10^{-25}$. This remapping however meant I had a great response throughout the range of movement.

With the limitation of only 2 sensors I felt I needed to apply multiple parameters to each slinky to increase the complexity and richness of the experience. I decided to use Slinky 1 as a control for pitch-based parameters, using it for pitch modulation and the notch of the bandpass filter. This meant as the modulation increased, the filter followed it.

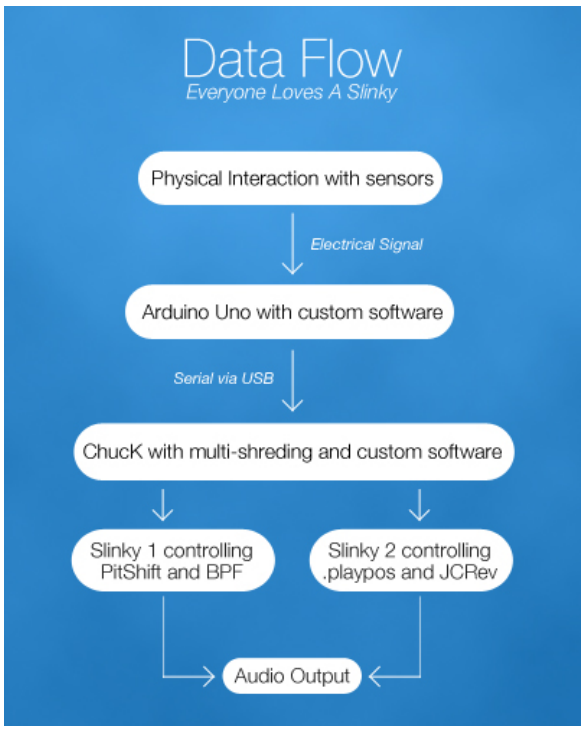

Figure 3: Data Flow Structure

For Slinky 2, I assigned the incoming data to control the playback position of one of my samples between -1 and +1, and to control the mix of a reverb UGen that both samples and effects were feeding in to. While the initial idea of the reverb was for the spring to increase the reverb time, this proved not possible in ChucK and so the best compromise was to simply affect the mix levels of the JCRev.

One fix I was particularly pleased about with the help of Jim was the looping playback of this secondary sample. While there is little issue with looping going forwards, when the sample was moving backwards through the start of the sample it wasn't moving to the end to continue the movement. With this little bit of code below the sample looped fluidly in both directions.

```
124    // This next section makes sure the countries sample loops backwards and forwards
125     if (countries.pos() == countries.samples() ) {
126         1 => countries.pos;
127     }
128
129     else if (countries.pos() == 0) {
130         countries.samples() - 1 => countries.pos;
131     }
```
Figure 4: Forward and Backwards Looping

3. Sonic Elements

For this project I decided to use two samples which I found rich for modifying that also have a personal touch for me. The primary sample, which has its parameters controlled by Slinky 1, was a stretched sample of my own song "Smoke", which had originally been recorded as a final project for CMPO 181. The stretch was made with Paul's Extreme Sound Stretch – a piece of software first introduced to me in CMPO 210. This stretched sample provided a rich range for pitch modulation, and created a sonically dense texture.

Figure 5: Smoke later appeared on my EP, Shakes & Shivers

The second sample is a short snippet of the narrator's monologue from the film *The Virgin Suicides*, which had featured in my final project for CMPO 281. This sample reads, "In the end we had the pieces of the puzzle, but no matter how we put them together, gaps remained, oddly shaped emptinesses mapped by what surrounded them, like countries we couldn't name." I feel this sample reflects the vague and constantly shifting composition.

Choosing to use samples from my previous courses in this thread (CMPO 181 and 281) references the nostalgic nature of using this childhood and historic toy, looking back while creating new mutations.

4. Hardware

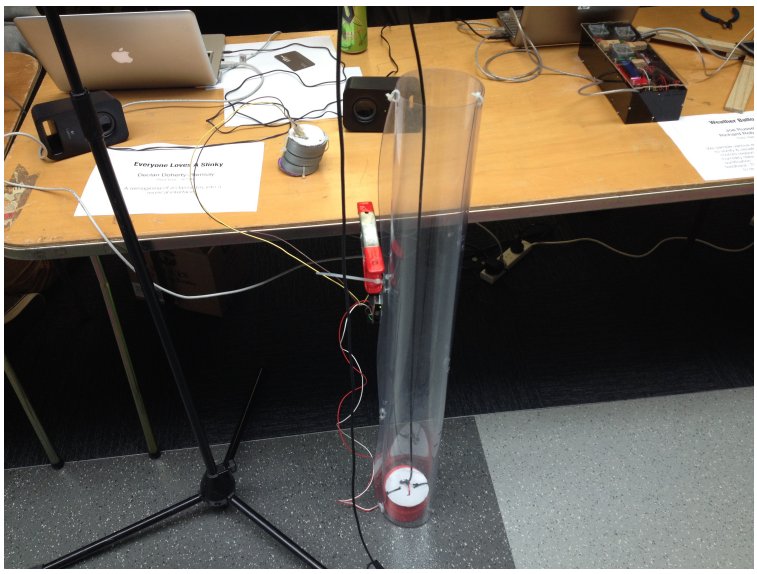

Figure 6: Slinky 1 (Tube) and Slinky 2 (Handheld)

This prototype ran on a Arduino Uno, with two LDRs as my sensors. The sensor for Slinky 1 was simply embedded underneath the clear tube and Slinky 1, while the LDR for Slinky 2 was more permanently affixed in to one end of the Slinky, ensuring stability when in the hands of the public. As it was a prototype, it was mostly held together by zip ties, clamps, masking tape, and hope. Further work on this would certainly start with a more secure and stable housing for the hardware.

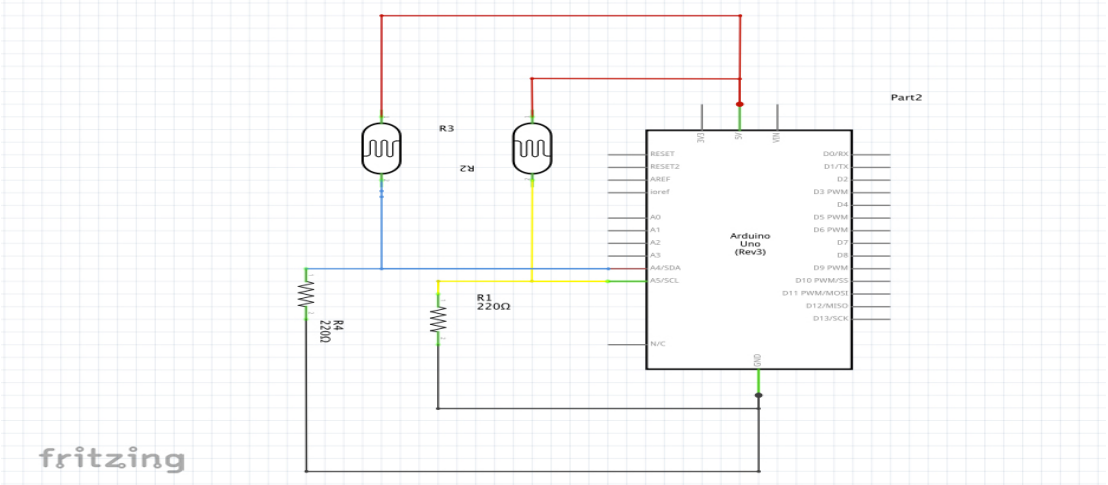

Figure 7: Circuit Diagram

5. Interactivity

Both Slinky 1 and 2 have different mechanisms for people to interactive with, and that's something I'm quite pleased with. Slinky 1 runs on a pulley system where a string is attached to the top of the slinky, allowing people to fluidly pull the slinky nearly a metre high. Slinky 2, held in the hands, is allowed to be pulled apart about 50 centimetres, which is a comfortable length for most people. That being said, in the demonstration a lot of people played with dangling the slinky in one hand above the ground, or grounding it on the table and pulling it up.

I particularly like that this instrument really calls for two people in order to use it to its full capacity, and this resulted in the demonstration with some interesting collaborations and modifications. I feel the multi-user aspect increased the chaotic nature, and also the fun of the piece.

6. Future Work

I feel this piece has great potential to grow and develop from this beginning. I was severely limited in the time I was able to dedicate to this project and had to unfortunately see some of the bigger ideas I had shelved for future expansion, including expressive pitch-bending for Slinky 2 through bending and shaking the slinky. As someone who is incredibly visually-focused, a properly mounting Slinky 1 with some more engaging visual cues to the pitch-shift would be also a priority. In future versions I'd like to move to a wireless system for Slinky 2, and introduce more movements - a musical slinky that moves through a scale or a melody down a set of stairs (steps), different positions of opened/closed creating regions of chords in a sort of mock-accordion – these have just been a few ideas I have had while making this project. I am however very pleased with how *Everyone Loves A Slinky* was received in the demonstration and I feel that really shows how fun this project was and would love to keep developing it.

7. Appendix

7.1 Parts List

1 x Arduino Uno
1 x Arduino ProtoShield
2 x 22kΩ Resistors
2 x Light-Dependent Resistors (LDR)
3 x Female Header Pin sets

7.2 Arduino Code

```
// Define the number of samples to keep track of.  The higher the number,
// the more the readings will be smoothed, but the slower the output will
// respond to the input.  Using a constant rather than a normal variable lets
// use this value to determine the size of the readings array.
const int numReadings = 10;

int readings[numReadings];      // the readings from the analog input
int index = 0;                  // the index of the current reading
int total = 0;                  // the running total
int average = 0;                // the average

int inputPin = A5;

void setup()
{
  // initialize serial communication with computer:
  Serial.begin(9600);
  // initialize all the readings to 0:
  for (int thisReading = 0; thisReading < numReadings; thisReading++)
    readings[thisReading] = 0;
}

void loop() {
  // subtract the last reading:
  total= total - readings[index];
  // read from the sensor:
  readings[index] = analogRead(inputPin);
  // add the reading to the total:
  total= total + readings[index];
  // advance to the next position in the array:
  index = index + 1;

  // if we're at the end of the array...
  if (index >= numReadings)
    // ...wrap around to the beginning:
    index = 0;

 int sensorValue2 = analogRead(A1);

  // calculate the average:
  average = total / numReadings;
  // send it to the computer as ASCII digits
  Serial.print("[");
```

```
  Serial.print(average);
  Serial.print(",");
  Serial.print(sensorValue2);
  Serial.println("]"); //send ] with newline character

  delay(1);        // delay in between reads for stability
}
```

7.3 ChucK Code

```
//UGens
SndBuf back => BPF notch => PitShift swing => JCRev hands => dac;
SndBuf countries => hands;

// Global + Setup Variables

0 => int i;
0 => float pitching;
0 => float mydelay;
1 => notch.Q;
0.5 => swing.mix;
1 => back.loop;

// Samples and Directories
me.dir() => string path;
"/audio/smokestretched.wav" => string myback;
"/audio/vs-countries.aif" => string mycountries;

// changes the filenames to full pathnames
path + myback => myback;
path + mycountries => mycountries;

//read those files
myback => back.read;
mycountries => countries.read;

// Starting Mute
0 => back.gain;
0.1 => countries.gain;

// Serial Stuff
SerialIO serial;
string line;
string stringInts[3];
int data[3];
SerialIO.list() @=> string list[];
// prints out the available serial devices
// uses "C-Style" printing instead of <<< >>>
for(int i; i<list.cap(); i++){
    chout <= i <= ": " <= list[i] <= IO.newline();
}
// open serial port 2 at 9600 baud, set to output ACSII (not binary)
serial.open(2, SerialIO.B9600, SerialIO.ASCII);

// Functions
fun void serialPoller() {
    while(true){
```

```
        serial.onLine() => now; //wait for arrival or serial….
        serial.getLine() => line; // store serial data in string 'line'
        // if it is null, break out and go back to waiting for serial
        if(line$Object == null) continue;
        0 => stringInts.size;
        //if messages are threee comma-separated numbers w/ sq. breackets at
end,
        //then store the three strings in the stringInts array.
        if (RegEx.match("\\[([0-9]+),([0-9]+)\\]", line, stringInts)){

            //now loop through and convert from ascii to int…
            //store these new ints into the "data" array
            for(1 => int i; i < stringInts.cap(); i++){
                Std.atoi(stringInts[i]) => data[i-1];
            }
        }
    }
}

 // Mapping function
fun float map(float x, float in_min, float in_max, float out_min, float
out_max) {
    return (x-in_min)*(out_max-out_min)/(in_max-in_min)+out_min;
}

// Making math!
fun void mathIsHard() {
    map(data[0], 0, 1023, 0, 1000) => float newFloatValue;

    Math.pow(newFloatValue, 10) => float newFloatValue2;
    0.00000000000000000000000001 *=> newFloatValue2;
    200 +=> newFloatValue2;


    // Creates hard top limits to not blow speakers
    if(newFloatValue2 > 19000) {
        19000 => newFloatValue2;
    }

    if(mydelay > .95) {
        .95 => mydelay;
    }

    newFloatValue2 => notch.freq;
}

spork ~ serialPoller();
spork ~ mathIsHard();

while(true) {
    0.8 => back.gain;

    // data[0]
    data[0] => pitching;
    200 /=> pitching;
    pitching => swing.shift;

    // data[1]
```

```
    data[1] => float mydelay;
    data[1] => float playpos;

    // Delay
    1000 /=> mydelay;
    mydelay => hands.mix;

    // Play Position

    1000 /=> playpos; //from 0 to 1
    2 *=> playpos; // 0 to 2
    1 -=> playpos; // -1 to +1
    playpos => countries.rate;

   // This next section makes sure the countries sample loops backwards and
forwards
    if (countries.pos() == countries.samples() ) {
        1 => countries.pos;
    }

    else if (countries.pos() == 0) {
        countries.samples() - 1 => countries.pos;
    }

    <<<data[0], data[1]>>>;
    1::ms => now;
}
```

Gmail by Google

Declan Doherty-Ramsay <declandr@gmail.com>

Final Assignment Extension

Jim Murphy <jim.w.murphy@gmail.com> Wed, Oct 29, 2014 at 12:53 PM
To: Declan Doherty-Ramsay <declandr@gmail.com>

Hi Declan
Sorry to hear that...
How about Wed Nov 5 (11:59PM) as a turn-in date? That gives you a week from today.
Think that'll be okay?
Thanks
-Jim
[Quoted text hidden]
--
- - - http://jimwmurphy.com/