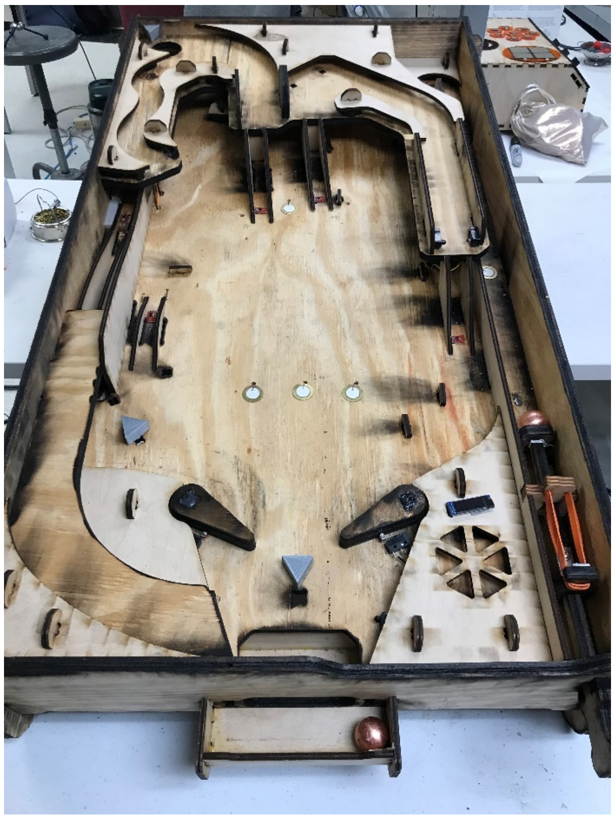
Pinball Wizard

Naomi Durand, 15 December 2020
Teammates: Travis Clawson, Taiki Tashiro

Instrument Overview

The Pinball Wizard is an instrument that creates music based on an active pinball game. When the player launches the ball, the song begins. As the ball hits various sensors throughout the game, various musical parameters (such as pitchbend, semitone increases, and tempo) increase.

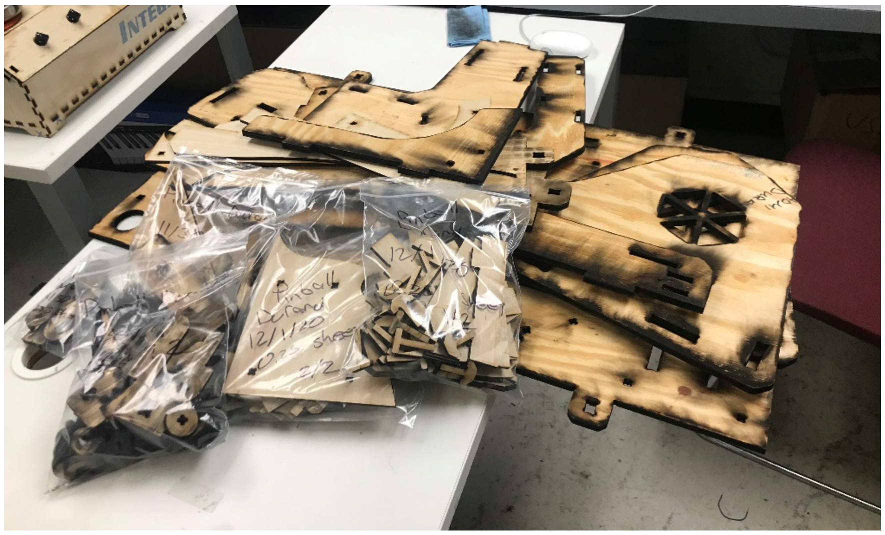
This instrument is fundamentally different from a classic pinball machine because it doesn't just have pre-recorded sound effects that play when various sensors are hit. Instead, there is an underlying melody that plays throughout the entire game, and this melody itself is changed based on the gameplay.



Pinball Base

Our base pinball machine was built from laser-cut plywood. The plans were designed by Aribabox and purchased as DXFs online; we sent the files to Bray for laser cutting.

The parts were intended to be cut with a CNC and the joints were designed to be press-fit. However, because the X-carve bed was not large enough, we had to laser cut our pieces. Therefore, there was a slight margin of error and they were not tight enough to press-fit. This looseness came in handy when we had to implement the sensors, which required the machine to be taken apart and reassembled a few times.



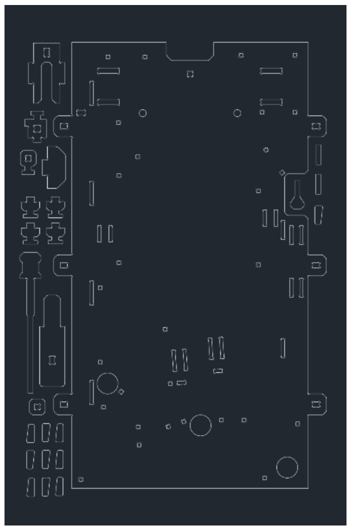


Figure 1 (left): Plywood pieces after laser cutting
Figure 2 (right): Screenshot of one DXF for laser-cutting (four total)

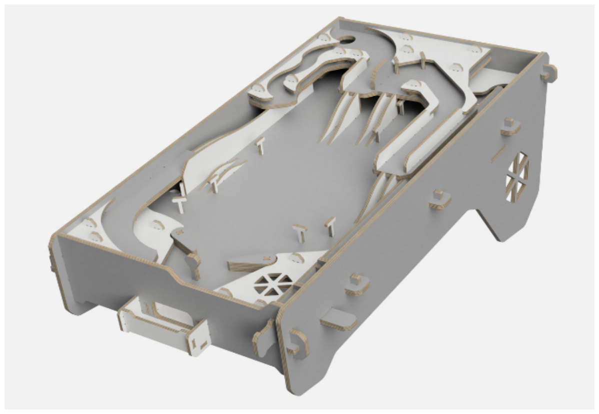
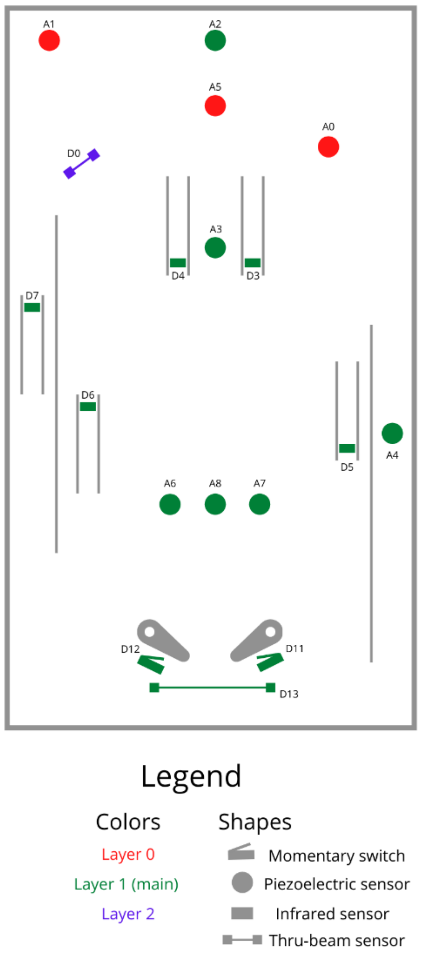


Figure 3: Image of constructed pinball machine from [Aribabox's website]()

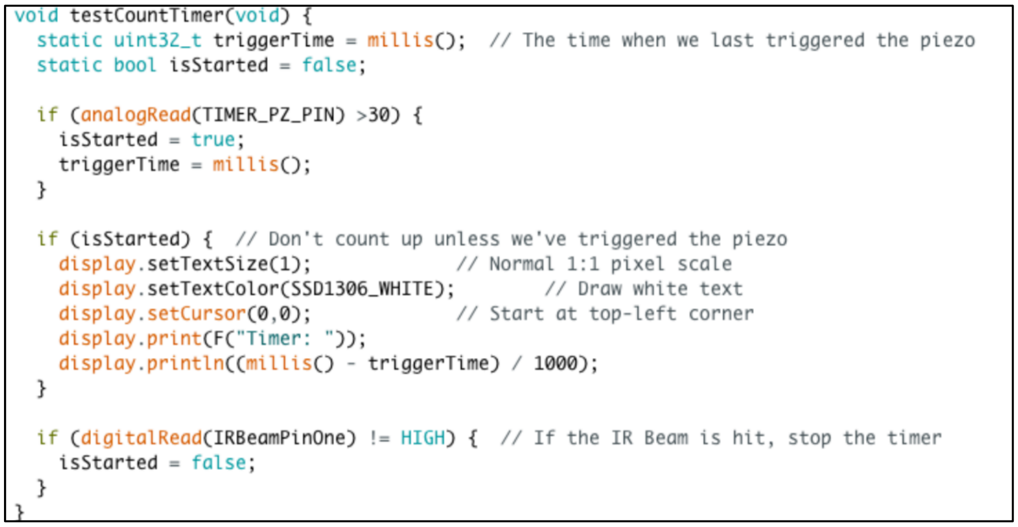
Electronics

The electronics are comprised of one Arduino Mega and 18 sensors. The following diagram highlights all of our sensors with their corresponding locations, the layer of the machine that they sit on, and the Arduino pin that they connect to:

Arduino Sensor/Pin Diagram



We placed an infrared sensor at all ramps entering and leaving each layer in order to detect the location of the ball. When experimenting with our IR sensors, we learned that the IR sensor will not detect a glass marble, even if it's slightly pigmented. Therefore, we used black spray-paint on the marbles… only to learn that the IR sensor doesn't easily detect black bodies either! In the end, we covered the marbles with copper tape, which is very reflective and was consistently detected by our sensors.



```
void testCountTimer(void) {
  static uint32_t triggerTime = millis();  // The time when we last triggered the piezo
  static bool isStarted = false;

  if (analogRead(TIMER_PZ_PIN) >30) {
    isStarted = true;
    triggerTime = millis();
  }

  if (isStarted) {  // Don't count up unless we've triggered the piezo
    display.setTextSize(1);             // Normal 1:1 pixel scale
    display.setTextColor(SSD1306_WHITE);        // Draw white text
    display.setCursor(0,0);             // Start at top-left corner
    display.print(F("Timer: "));
    display.println((millis() - triggerTime) / 1000);
  }

  if (digitalRead(IRBeamPinOne) != HIGH) {  // If the IR Beam is hit, stop the timer
    isStarted = false;
  }
}
```

An Arduino sketch was written for calculating and displaying the timer from the game. The timer starts when the launch-piezo is hit, and ends when the bottom trip-sensor is flagged. Although this display is not fully functioning in the final product, the display was designed to show a timer and keep score for the game.

Unfortunately, we did not understand that the thru-beam sensors required more current than the Arduino could supply. We discovered this quite late, when we were trying to troubleshoot an Arduino that had already been fried… thankfully Joe saved the day and taught us about buck converters! We used a buck converter and an additional 5V supply for the thru-beam sensors and started fresh with another Arduino.

The piezo elements gave us the most trouble. They were quite finicky and often took a long time to return to their baseline state. Therefore, we had to write a subpatch to identify peaks in the piezo values.

Merging Base with Electronics

In order to implement the electronics, we drilled holes in the pinball base to feed all wiring to the underside. The sensors were secured with hot glue and double-sided tape.

Originally, we had hoped to x-carve slots for the individual sensors to be embedded in the machine. If we had extra time, this is definitely something we would have done. However, we wanted to prioritize the musical features of the instrument rather than the aesthetics. Additionally, embedding the sensors would not allow for us to make further adjustments to the location and orientation of the sensors, which was essential to our final process.

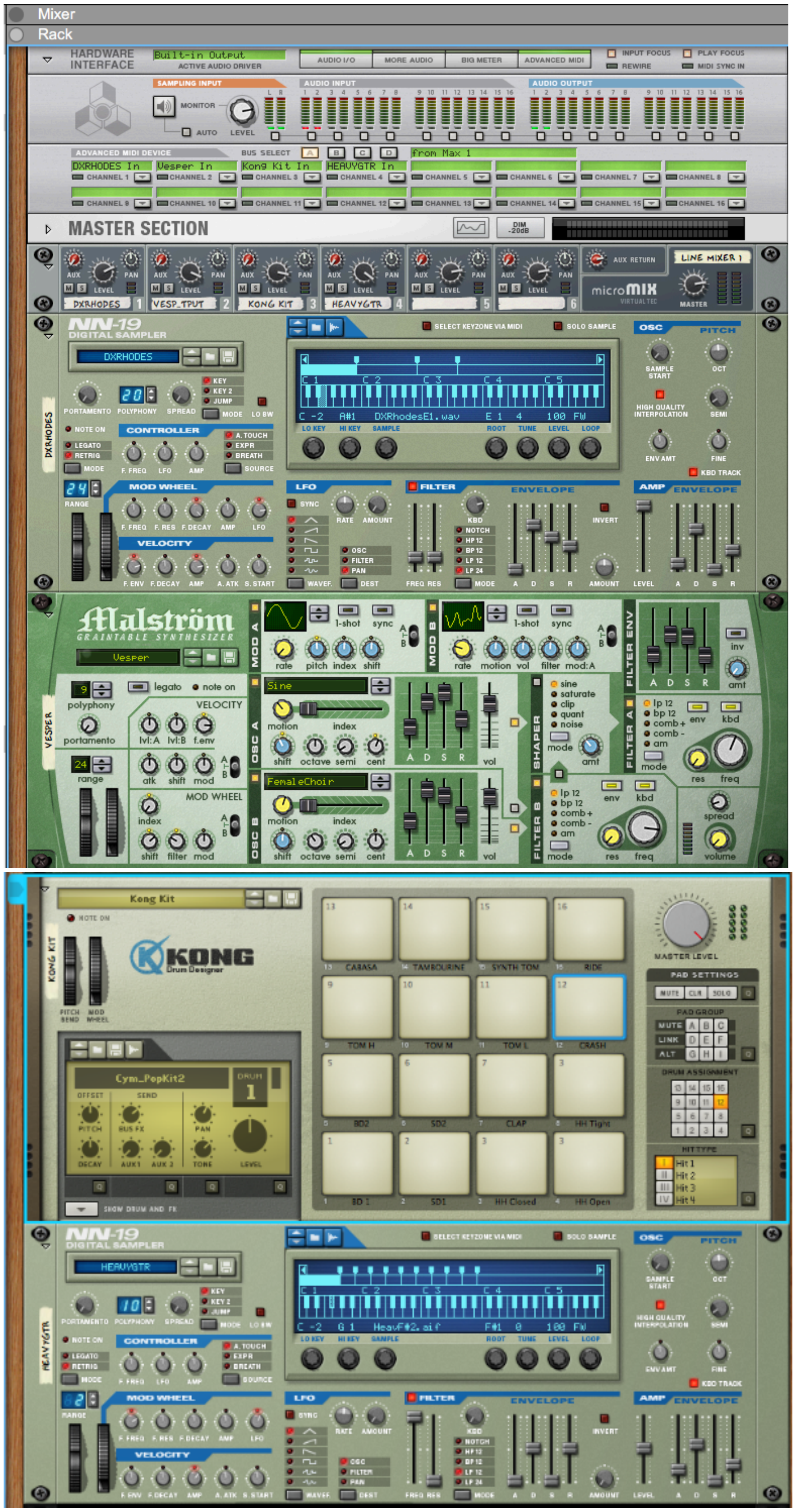
Reason

Our reason patch contains 4 channels:

1. **NN-19** for melody notes using synthesizer effect
2. **Malstrom** for base chord
3. **KONG** for percussion
4. **NN-19** for electric guitar strum

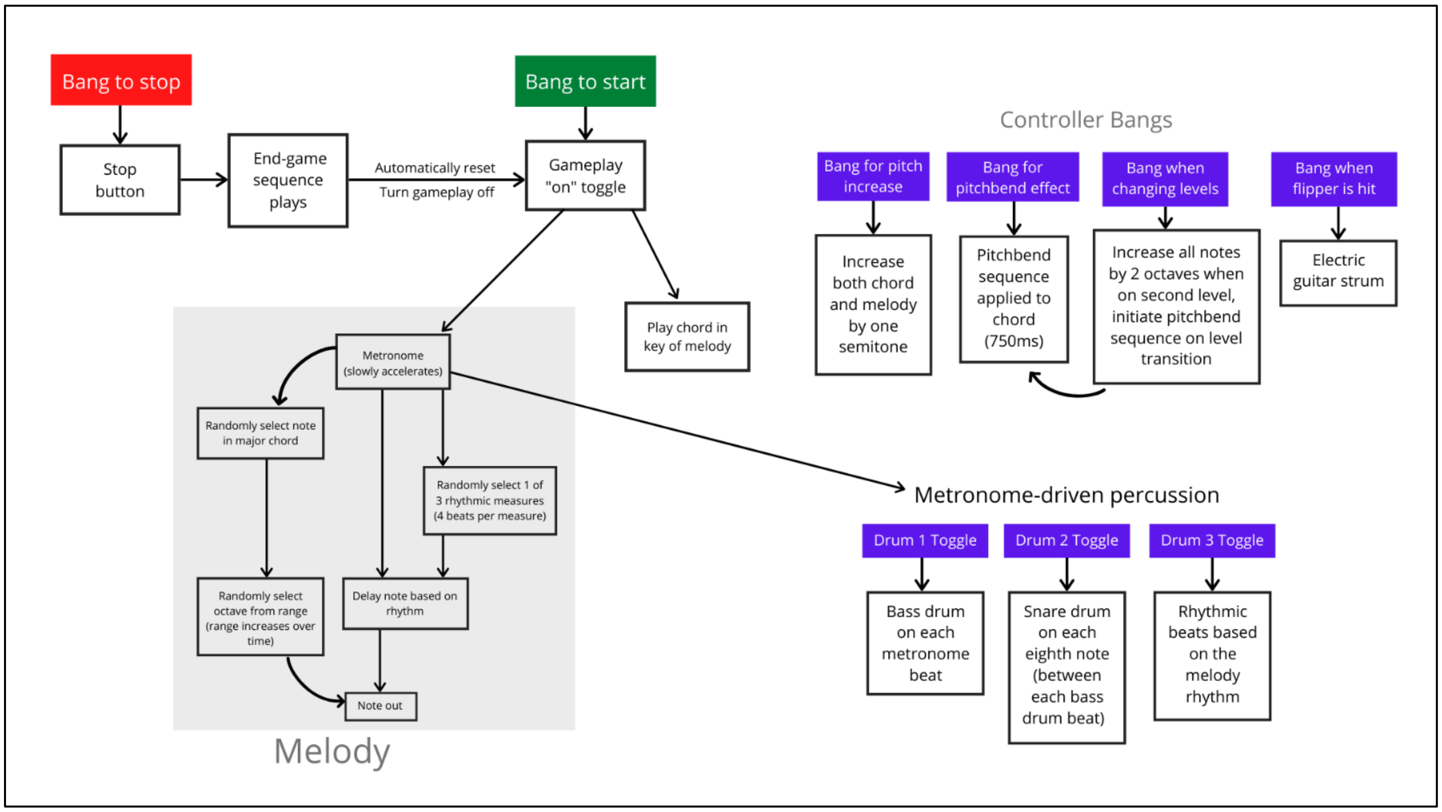
Primarily there are three different sound effects are were sent to Reason:

1. **Pitchbend effect**: This effect is created by adjusting the pitch knob of Modulator A on the Malstrom
2. **Semitone increase**: In order to increase the entire instrument by a semitone, the semitone knobs for the Malstrom and the NN-19s are simultaneously increased.
3. **Level state effect**: When the ball is up on the "second level", the pitch of the entire instrument shifts upwards by 2 octaves. When the state of the level is changed, the pitchbend effect is called to highlight the transition.


Mixer
Rack
HARDWARE INTERFACE
Built-in Output
ACTIVE AUDIO DRIVER
AUDIO I/O
MORE AUDIO
BIG METER
ADVANCED MIDI
INPUT FOCUS
PLAY FOCUS
REWIRE
MIDI SYNC IN
SAMPLING INPUT
AUDIO INPUT
AUDIO OUTPUT
MONITOR
AUTO
LEVEL
ADVANCED MIDI DEVICE
BUS SELECT
From Max 1
DXRHODES In
Vesper In
Kong Kit In
HEAVYGTR In
MASTER SECTION
DXRHODES
VESP_TPUT
KONG KIT
HEAVYGTR
AUX RETURN
LINE MIXER 1
micromix
MASTER
NN-19
DIGITAL SAMPLER
DXRHODES
SELECT KEYZONE VIA MIDI
SOLO SAMPLE
DXRhodesE1.wav
OSC
PITCH
CONTROLLER
MOD WHEEL
VELOCITY
LFO
FILTER
ENVELOPE
AMP
Malström
GRAINTABLE SYNTHESIZER
Vesper
MOD A
MOD B
FILTER ENV
OSC A
OSC B
SHAPER
FILTER A
FILTER B
Sine
FemaleChoir
Kong Kit
KONG
Drum Designer
Cym_PopKit2
CABASA
TAMBOURINE
SYNTH TOM
RIDE
TOM H
TOM M
TOM L
CRASH
BD2
SD2
CLAP
HH Tight
BD 1
SD1
HH Closed
HH Open
MASTER LEVEL
PAD SETTINGS
PAD GROUP
DRUM ASSIGNMENT
HIT TYPE
Hit 1
Hit 2
Hit 3
Hit 4
SHOW DRUM AND FX
HEAVYGTR
HeavF#2.aif

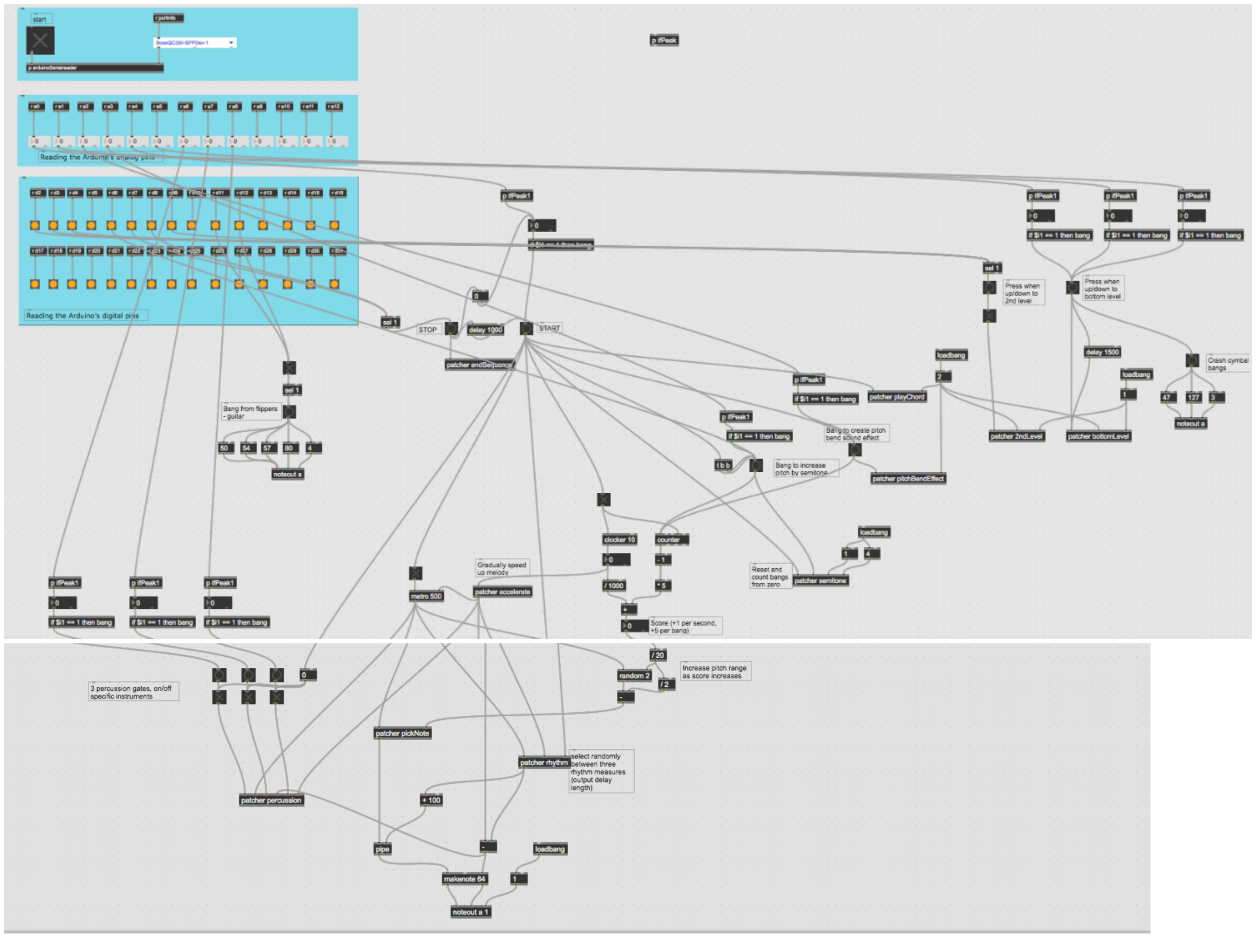
Max

The figure below shows a block diagram of the Max patch and highlights all of its features:

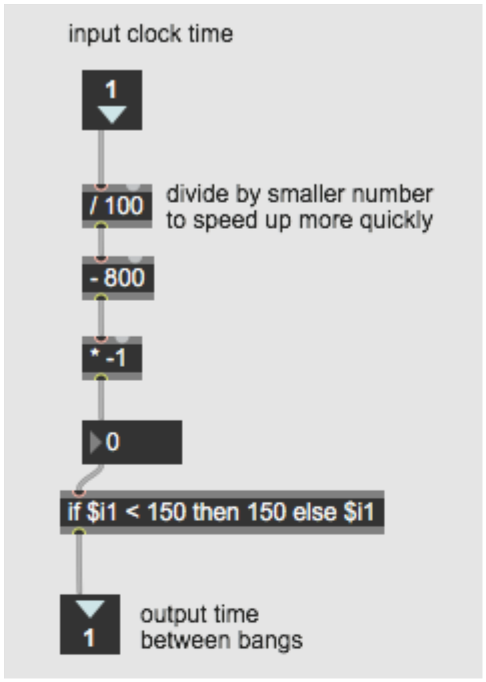

Bang to stop
Stop button
End-game sequence plays
Automatically reset
Turn gameplay off
Bang to start
Gameplay "on" toggle
Play chord in key of melody
Metronome (slowly accelerates)
Randomly select note in major chord
Randomly select 1 of 3 rhythmic measures (4 beats per measure)
Randomly select octave from range (range increases over time)
Delay note based on rhythm
Note out
Melody
Controller Bangs
Bang for pitch increase
Increase both chord and melody by one semitone
Bang for pitchbend effect
Pitchbend sequence applied to chord (750ms)
Bang when changing levels
Increase all notes by 2 octaves when on second level, initiate pitchbend sequence on level transition
Bang when flipper is hit
Electric guitar strum
Metronome-driven percussion
Drum 1 Toggle
Bass drum on each metronome beat
Drum 2 Toggle
Snare drum on each eighth note (between each bass drum beat)
Drum 3 Toggle
Rhythmic beats based on the melody rhythm

The primary Max patch was made of many subpatches. The most notable was *accelerate*, which increased the speed of the melody. The second was the *rhythm* patch, which selected one random rhythm from three measures.

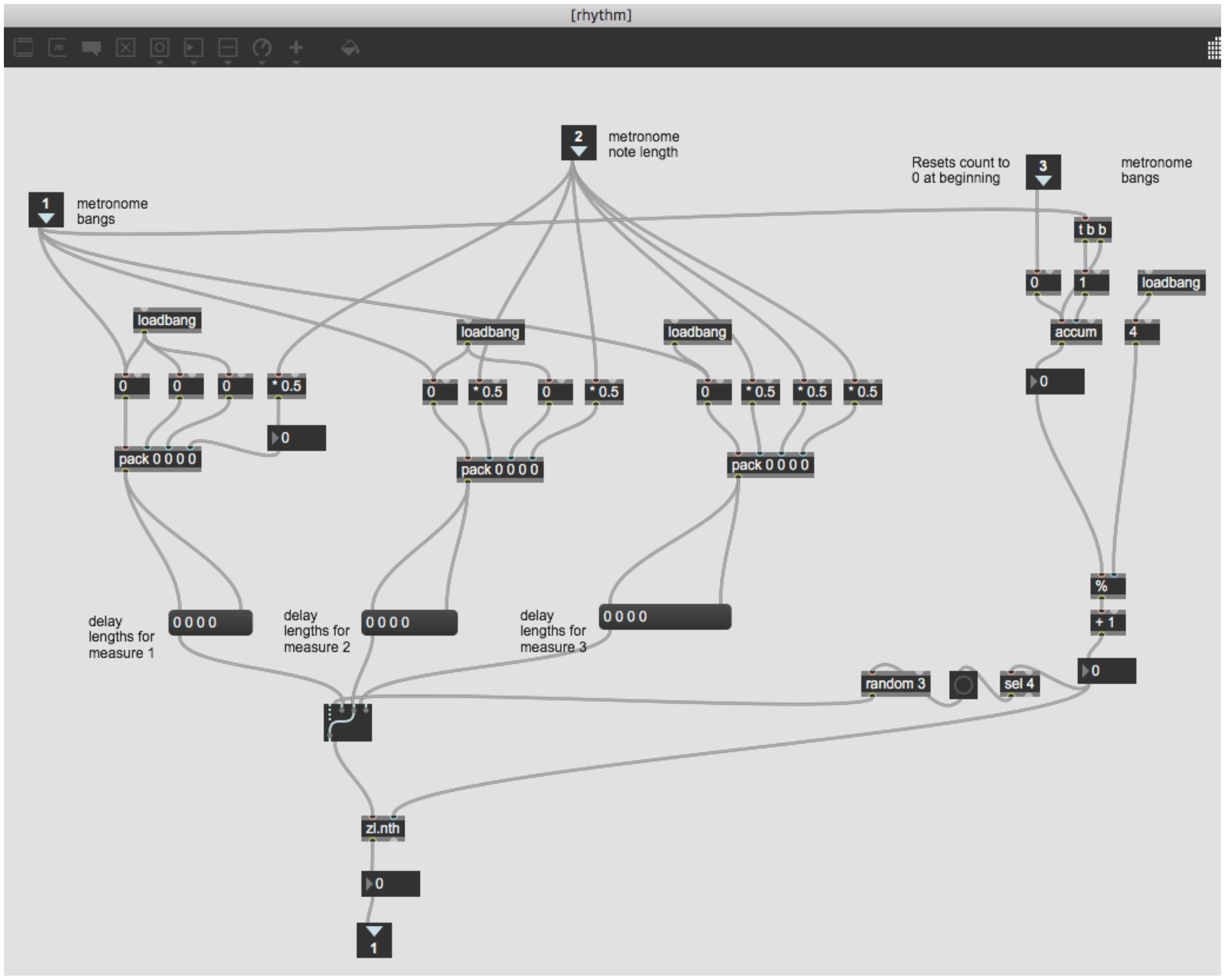
Below is a screenshot of the primary Max patch, as well as the *accelerate* and *rhythm* patches.


start
p arduinoSerialreader
Reading the Arduino's analog pins
Reading the Arduino's digital pins
p ifPeak
p ifPeak1
if $i1 == 1 then bang
sel 1
STOP
delay 1000
START
patcher endSequence
Bang from flippers - guitar
noteout a
Press when up/down to 2nd level
Press when up/down to bottom level
loadbang
delay 1500
Crash cymbal bangs
patcher playChord
patcher 2ndLevel
patcher bottomLevel
Bang to create pitch bend sound effect
Bang to increase pitch by semitone
patcher pitchBendEffect
clocker 10
counter
Gradually speed up melody
patcher accelerate
metro 500
Reset and count bangs from zero
patcher semitone
Score (+1 per second, +5 per bang)
3 percussion gates, on/off specific instruments
random 2
Increase pitch range as score increases
patcher pickNote
patcher rhythm
select randomly between three rhythm measures (output delay length)
patcher percussion
pipe
makenote 64
noteout a 1

Accelerate:


input clock time
1
/ 100
divide by smaller number to speed up more quickly
- 800
* -1
0
if $i1 < 150 then 150 else $i1
1
output time between bangs

Rhythm:


[rhythm]
2
metronome note length
Resets count to 0 at beginning
3
metronome bangs
1
metronome bangs
t b b
0
1
loadbang
accum
4
loadbang
loadbang
loadbang
0
0
0
* 0.5
0
* 0.5
0
* 0.5
0
* 0.5
* 0.5
* 0.5
0
0
pack 0 0 0 0
pack 0 0 0 0
pack 0 0 0 0
%
+ 1
delay lengths for measure 1
0 0 0 0
delay lengths for measure 2
0 0 0 0
delay lengths for measure 3
0 0 0 0
random 3
sel 4
0
zl.nth
0
1

Team Member Contributions

I felt that we worked very well as a team in building the Pinball Wizard. Our role breakdown was very similar to the prior project (Kitchen Magician):

- Travis: Building of the base pinball machine
- Taiki: Arduino code and sensor wiring
- Naomi: Max and musical features

Our experience was very similar to the Kitchen Magician; without having an electrical engineer in the group, we spent a significant amount of time working through the wiring for the sensors. It was a great help that Taiki was willing to step up and take charge of this section! Travis also did a great job with the base of the pinball machine and was always working to make tweaks that made the gameplay smoother.

My comfort with Max increased exponentially for this project, which I really enjoyed. I feel much more proud of this Max patch than I did for the Kitchen Magician; this time, I was able to more thoroughly take advantage of Max.