The BirdBox

Ian Jones, Allie Lam, Nihal Pai

I. Introduction

The BirdBox is an exploratory instrument that augments a randomised sequencer with a number of musical parameters and their associated controllers. With just enough guidance to get beginners started but no set 'instructions' to limit them, *The BirdBox* encourages players of any skill level to discover new ways to combine its unique functionalities, all in the name of more structured and pleasant-sounding musical improvisation.

II. Evolution

Our group's initial motivation was to create a collaborative, improvisation-based instrument where each player would be able to control a different musical parameter (or set of parameters) of the overall piece. These musical features consisted of pitch (via a set of note banks, each constituting a certain scale), rhythm (via on & off-beats of a standard 4/4 groove), BPM & special effects (such as LFO rate modulation, etc.). The code for each parameter would be implemented in Max in conjunction with Arduino2Max, while the corresponding controllers & sensors would be organised on a circular instrument body according to the musical parameter they contributed to. That is, the two linear soft pots that were to control pitch (thresholded such that each segment corresponded to a different note bank) would have occupied a separate section from the 8 3-way switches / BPM potentiometer and the remaining 4 potentiometer knobs, which controlled rhythm and special effects respectively. The wiring would have been housed in a clear acrylic dome in the center, as per Figure 1 below:

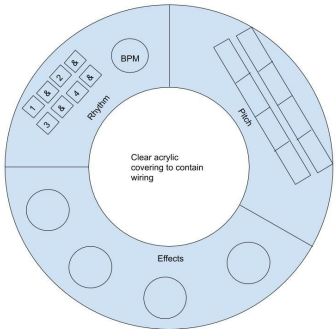


Figure 1: Initial Outline

However, structural & wiring constraints caused us to naturally pivot to a more compact box form that housed the wiring in its interior with protruding 4-way switches (instead of 3-way switches due to an ordering error on our part), rotating potentiometers, and a single linear soft pot (as opposed to 2). The musical parameters mapped to these controllers stayed the same, but the scale of the overall instrument was reduced dramatically such that it was just large enough to house the Arduino, breadboards, and wiring. This effectively made it a single-player instrument.

Our final iteration of the instrument built on this boxed concept with a few changes and interesting additions:

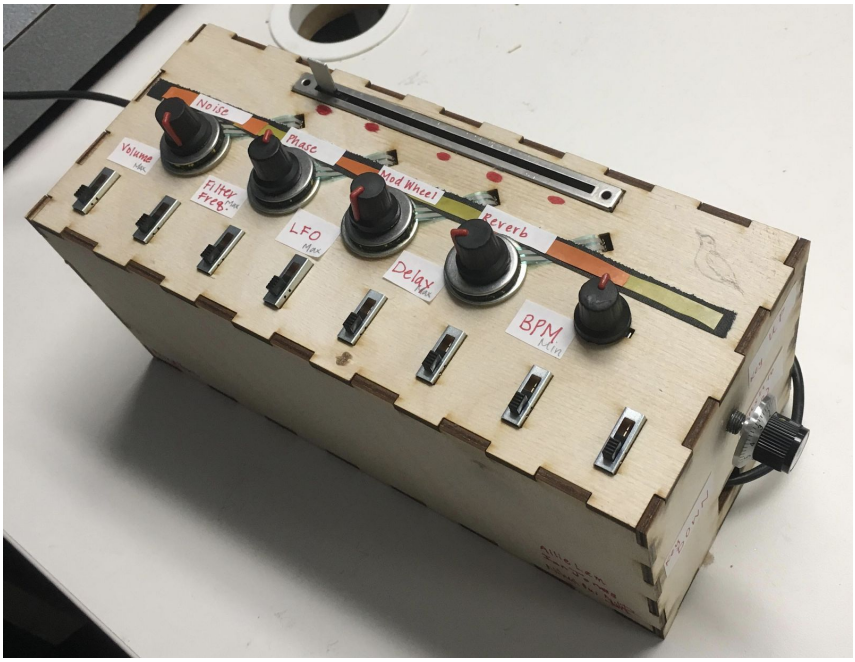
a) The linear soft pot required the player to continuously apply pressure to a certain section to 'activate' the desired scale / note bank, which did not provide a practical method of scale selection. This was hence replaced with a slide potentiometer with the same underlying logic (i.e. segmented such that each segment triggered a different scale). This provided a more consistent and stable method of changing the scale.

b) The structures of 4 knobs were manually reconstructed with springs and washers to house FSR's underneath. Each FSR would map to a new musical parameter, essentially allowing the user to control 2 musical parameters per knob via pressure application.

c) A rotary encoder with a push button was added to the side of the instrument to allow the player to control key and trigger looping respectively.

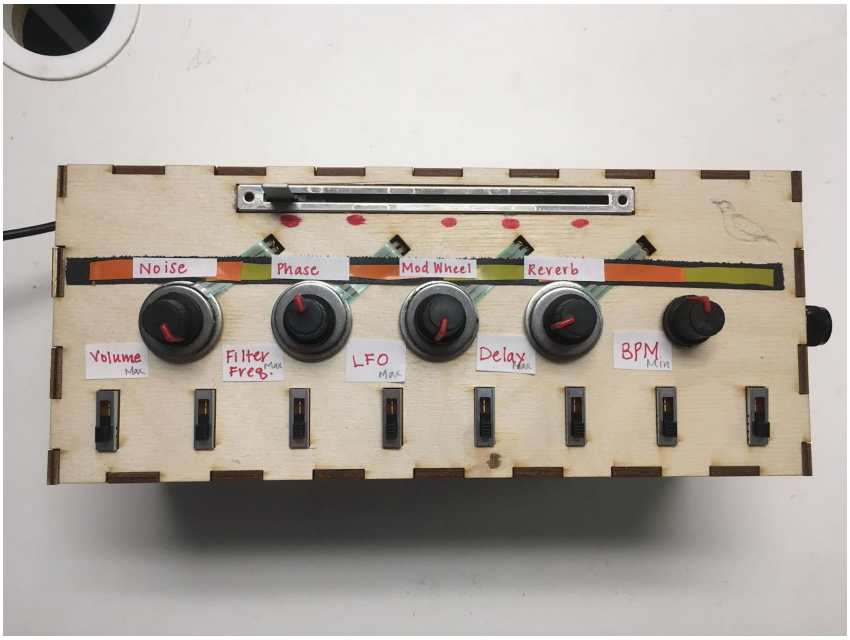
d) The set of available scales was changed to Tritonic, Tetratonic, Pentatonic, Whole Tone, and Western Standard, which we felt were the most common scales used in modern music.

These alterations greatly expanded *The BirdBox*'s range of functionality and made it much more interesting and holistic instrument. The final version can be seen in Figures 2 & 3 below:



`

Figures 2 & 3: The BirdBox



III. List of Parts

Sensor / Component	Use(s)	
Rotating Potentiometers (5x)	Volume	
	Notch filter frequency	
	LFO amount	
	Delay	
	BPM	
FSR's (4x)	Noise	
	Phase oscillator	
	Modulation wheel	
	Reverb	
Rubber Washers (8x) & Springs (4x)	Apply even pressure to FSR's	
4-Way Switches (8x)	Velocity for each on-beat & off-beat in a standard 4/4 groove (i.e. 1-&-2-&-3-&-4-&). Each switch position corresponded to an incremental change in velocity (OFF - 0, LOW - 30, MEDIUM - 70, and HIGH - 120)	
Slide Potentiometer (1x)	Tritonic Scale	MIDI Values: 64, 67, 69
	Tetratonic Scale	MIDI Values: 60, 64, 65, 67, 72
	Pentatonic Scale	MIDI Values: 60, 62, 64, 67, 69, 72
	Whole Tone Scale	MIDI Values: 60, 62, 64, 66, 68, 70, 72
	Western Standard Scale	MIDI Values: 60, 62, 64, 65, 67, 69, 71, 72
Rotary Encoder (1x)	Key selector	
	Looper	
Arduino Mega 2560, Breadboards with Wires	Microcontroller & wiring kit	

1/8th inch Plywood	Instrument housing / body

Table 1: Parts & Functions

In addition, copious amounts of hot glue, solder, and tape were used for sealing different sections of the body together, wiring, and decoration.

IV. Construction

In terms of the body / housing, a basic plywood box was designed in Solidworks and laser cut in Nolop. The top face comprised of a front panel and back panel - the former was 'user-facing' and presented the switches, sensors, knobs, and slide through holes, while the latter provided vertical support for the components that extended below the top panel. We originally had living hinges constituting the edges of the box, but these proved too delicate and cracked during our mid-term project, so this iteration used a simple serrated ‘jigsaw’ pattern that allowed each panel to fit together. The interior of the box contained the wiring. Wires were soldered to each sensor and plugged into a central Arduino Mega 2560 with 2 lateral bread boards for each component’s ground and 5V access. The most important structural change to our instrument was the addition of a series of springs, metallic washers, and manually-cut rubber washers to each knob (Figure 4). This provided some amount of cushioning such that each knob could hold an FSR between its washers, allowing the user to apply pressure downwards and activate the parameters linked to the FSR’s.

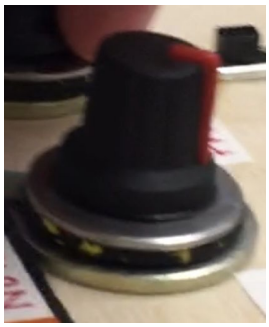


Figure 4: Modified Knobs

V. Circuit Modifications

Our project did not involve physically modifying the circuit. However, it did require appropriately thresholding the ranges of incoming signals from the Arduino-based components to Max. This allowed our interfaces to accurately capture the physical motion range afforded by the sensors as well as mitigate the effects of noise (which were significant for the FSR's). We also had to alter the consecutive digital & analog wire-to-pin scheme to avoid (what we believe to be) Arduino's serialised shifting of digital data to analog pins (for context - this was a frustrating but interesting bug we encountered whenever we left our initial circuit running for more than a couple of minutes). We also had to edit the Arduino2Max code accordingly so that it would accept the correct digital inputs that were being wired to since the default GUI did not display all of the data ports we needed to use for our additional components. Lastly, we had to prevent Max from sending interfering data streams for '*off*' positions when other switches were toggled to '*on*' (e.g. if only 1 switch was '*on*', and a second switch was turned on, Max would initially send data for the first switch turning off and the second switch turning on instead of *just* the second switch turning on).

VI. Max Patch Functionality

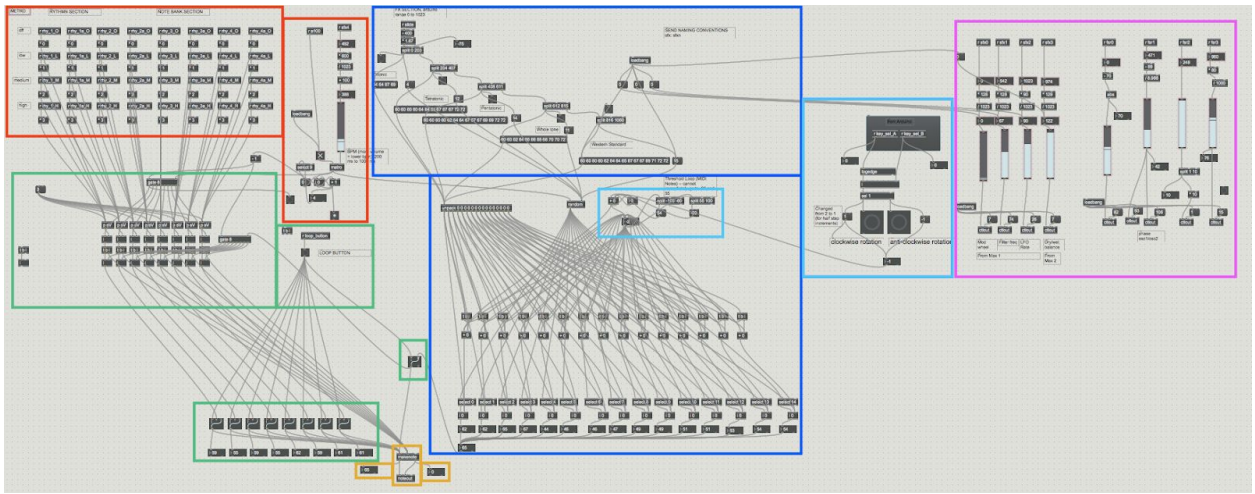


Figure 5: Max Patch

Our Max patch was essentially modeled after a randomised sequencer:

- A *Metro* sends a *bang* every *x* number of milliseconds based on the BPM to track which beat the note is on
- The Rhythm section would read a value for each specified beat and select the appropriate note velocity
- Pitch selection from the specified note bank would occur randomly, with 1 note being sent per beat

The overall patch consisted of a 2 main modules that provided the sequencer's randomised functionality:

1. Rhythm: (red section in Figure 5)
   Each of the 8 beats can assume 1 of 4 possible velocities (OFF - 0, LOW - 30, MEDIUM - 70, and HIGH - 120). These values would be selected from data read in from Arduino2Max and allow the user to toggle between note velocities. The BPM would directly correspond to a rotating potentiometer (i.e. rotating it anticlockwise would reduce the BPM, rotating it clockwise would increase it). A ceiling for BPM had to be programmed so that the linked Reason patch didn't overload.

2. Pitch: (blue section in Figure 5)
   Pitch consisted of 5 different note banks with a varying number of notes (the scale specified by each note bank is explained in Table 1) such that once a certain bank is selected, the notes would sound off in a randomised fashion. The scales offered (Tritonic, Tetratonic, Pentatonic, Whole Tone, and Western Standard) were selected because they were the most commonly used as well as distinct from each other - we felt that this would provide the ideal musical foundation for new players to start creating as seamlessly as possible. We also peppered each note bank with certain repeated note values so that they would have a higher incidence in the randomised sequence. We felt that these note values would provide the essence of the scale, allowing the untrained ear to quickly distinguish between notes.

The Rhythm (velocity) and Pitch data were combined using a *Makenote* object to actually create the musical notes (yellow section in Figure 5).

Apart from these, 3 additional modules provided the meat of our project:

3. Special Effects: (pink section in Figure 5)
   Since the Arduino values from each potentiometer consistently ranged from 0-1023, we were able to scale them to the appropriate MIDI values via simple multiplication and division. These were mapped to different effects in the Reason Subtractor / Reverb / Echo rack. Each rotating potentiometer and its FSR were mapped to different effects in the following scheme:

Group	Rotating Potentiometer	FSR
Group 1	Volume	Noise
Group 2	Notch filter frequency	Phase oscillator
Group 3	LFO amount	Modulation wheel
Group 4	Delay	Reverb
Group 5	BPM	n/a

Table 2: Rotating Potentiometer - FSR Effects

4. Key: (light blue section in Figure 5)
   Using a custom Max patch we found and later incorporated (light blue section to the far right), the rotary encoder's functionality was leveraged to trigger semitone increments or decrements when turned anticlockwise or clockwise. These increments were applied to the note values in the selected note bank using a *Split*, such that the scales would not start from the default key if a new note bank was selected. Since our scales had a minimum & maximum note values of 60 & 72 respectively and the MIDI range is 0-127, the possible range of note values was set from -60 to +55. We also wanted the user to be able to seamlessly cycle across semitone 'boundaries' i.e. once the maximum note value of 55 (relative to the base scale) was reached, any further increments would reset the semitone to -60 and vice versa. This was accomplished using a pair of *Split*'s linked in a looping fashion (shown in the small light blue section on the left).

5. Looping: (green section in Figure 5)
   The basic function of this section was to store a running memory bank of the previous 8-note sequence and repeat the sequence until the termination of the looping trigger. This was accomplished using a series of 8 *TBI*'s (Trigger Bang) with associated *Integer* blocks (shown in the top left green section). The idea was that note values from the sequences were to be stored in the *Integer* blocks and

combined with their corresponding velocity values from the rhythm section before being sent to the overall *Makenote*. However, we found that note values had to be set prior to velocity values in order for the *Makenote* to function as expected. This is where the *TBI*'s came in - these components are able to send an integer value from from left to right followed by a bang. We configured this so that the integer value initially sent was the note's velocity value and the following bang was sent to the integer component storing the actual note value.

We also included a *Bang* component that featured as our looping button (shown in the top right green section). This was used to switch off the notes from the 8-note bank currently looping (via the 8 gate components shown in the bottom left green section) and switch on the new notes coming in from the randomised scale bank (via the single gate component shown in the bottom right green section). This enabled the patch to only record new notes when new randomised sequences were allowed to come through.

Figure 6 below displays the Arduino2Max interface, colour-coded according to the associated components in the Max patch:

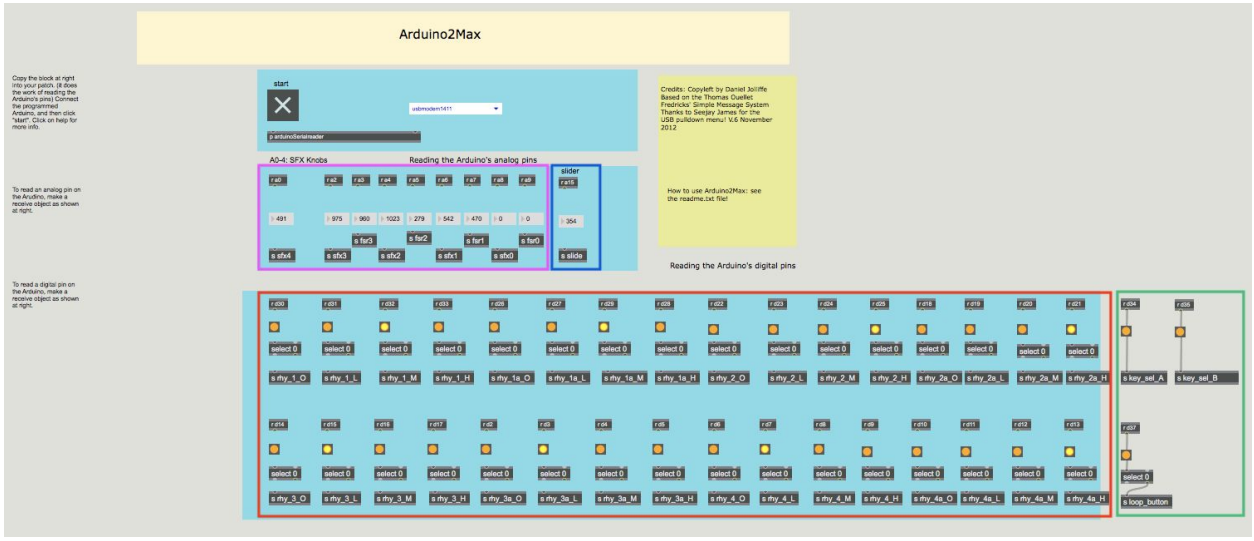


Figure 6: Arduino2Max Interface

VII. Reason Rack

Figure 7: The Echo (Delay)

Figure 8: Subtractor & RV-7 Reverb

Our Reason Rack consisted of 3 separate channels:

1. Bus A: *Subtractor* module with the *Backlash Bass* instrument, which provided a warm, retro, synth-like sound
2. Bus B: *RV-7 Reverb* module with the *Small Room* reverb setting
3. Bus C: *The Echo* module with the *Warm Echo* setting

The potentiometer knobs and FSR's were mapped to effects in these specific modules.

VIII. Looking Back & Next Steps

While we are all extremely pleased with the final results of our instrument, there are a few changes that would likely prove invaluable additions to *The BirdBox*.

Firstly, given additional time & resources, we would have loved to improve the construction of the FSR-Knob complexes. We found that the FSR's were incredibly sensitive and extremely responsive to the weight of the washers as opposed to just the downward force applied by the user. While we were able to reduce the effects of these unwanted signals through thresholding, this also meant that we had to forego a certain degree of accuracy and seamlessness from the instrument's playbility. This also highlighted to us the overall lack of stability in these knobs. We found that the knobs were often easily pulled off and the springs underneath them were too compressed to be fully functional. While this did not affect the functionality of the instrument by much, having access to better quality components would allow us to improve on this design for a more seamless playing experience.

Secondly, we would want to include some sort of digital display for each component that would indicate the current levels of the potentiometer knobs / FSR's, the scale being played, and the key. This addition would truly complete our instrument.

Lastly, we found that our rotary encoder was a little unreliable in that it did not register certain rotations and was erratic when it came to switching directions. To improve this, we would want to perfect the Arduino code we added to debounce the encoder's signals, as shown in Figure 9 below:

```
//encoder = 34, 35
curr_pin = digitalRead(34);
if(!curr_pin && (past_pin)){
  if (millis()-timer > DELAY_TIME) {
      if(digitalRead(34)==LOW) d5 += 1;
      if(digitalRead(35)==LOW) d5 += 2;
      timer = millis();
  }
}
past_pin = curr_pin;
```

Figure 9: Arduino Code to Debounce Rotary Encoder

This code did not improve our debouncing by as much as we expected and requires further debugging.

IX. Team Evaluation

Although we were all present throughout almost every stage of the project, we were able to specialise our tasks to some extent. Ian essentially figured out the looping portion in Max, took the lead on designing the SolidWorks files, spent time laser-cutting the plywood housing in Nolop, and designed & built the knob-FSR complex. Allie worked hard on the the Max logic involving the scale notebanks, assisted Ian with the looping logic, and worked with me to figure out how to incorporate the Rotary Encoder Max Patch and handle the cyclic semitone shifts as well as solder some FSR's. She also spent a lot of time configuring the Arduino2Max interface for our new sensors, working out the math to threshold our signals, and colour-coding my wiring scheme for debugging. Ian and Allie also took the lead on remapping the FSR's and knobs to different Reason module effects and designing the sound scape. I found myself pitching in at various points throughout this process, namely soldering and wiring components (with a great deal of help from Allie and Ian, who both expertly relocated my wiring scheme to a smaller breadboard when it proved too bulky to fit in our housing), working with Allie to figure out the Rotary Encoder / semitone shifting Max logic, assisting with Max debugging when I could, and coding the rotary encoder's debouncing logic in the Arduino code with Ian. While we all contributed a lot of time and effort to this project, both of my teammates were essentially the progenitors of *The BirdBox*, working incredibly hard throughout the process and often outside their skill sets to help each other and myself out to make this instrument the best it could be. They made this an incredible experience and I have been incredibly fortunate to work with & learn from both of them.