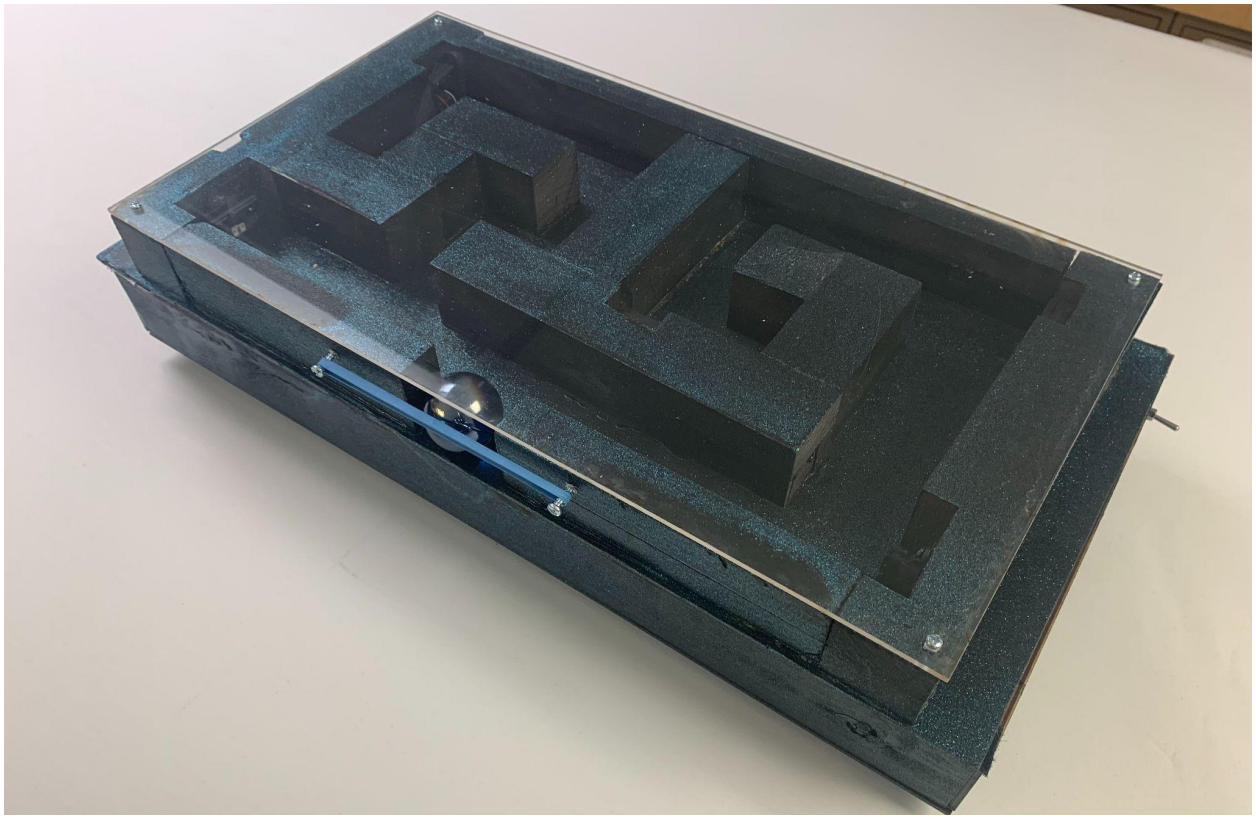
The MMMM
(Marble Maze Music Maker)

Jonah D'Alessandro
Group Members: Julia Huckaby and Ismael Diaz Mateo



Summary

The Marble Maze Music Maker (MMMM) was designed with the goal of being an easily playable instrument with highly variable, random elements. The instrument consists of a wood maze that controls the timbre based on the position of a marble, and has buttons on the bottom to directly play specific notes. We were inspired by marble puzzles where you aim to roll a marble through a maze by tilting the entire structure — the difficulty to control the marble made us think that it would be an interesting way to get random musical effects. The instrument plays a randomized backing track that includes drums, bass, and chords, while the user can play additional lead notes in a major scale using buttons underneath the maze. There are five IR sensors throughout the maze which track the marble's position and mixes between different instrument sounds for each of these instrument categories.

Structure

Materials used:

- 1.4" diameter marble (x1)
- Arcade push button (x8)
- 3 pin on/off toggle switch (x1)
- 10 kΩ resistors (x9)
- Sharp GP2Y0A51SK0F Infrared Proximity Sensor 2-15 cm (x3)
- Sharp GP2Y0A41SK0F Infrared Proximity Sensor 4-30 cm (x2)
- Arduino Uno (x1)
- Solderable breadboards (x2)
- 3 mm plywood sheets for laser cutting (x3)
- 3 mm clear acrylic sheet (x1)
- 4'x8" wood planks (x2)
- Black spray paint
- Glitter spray paint
- Miscellaneous fasteners, wires, wood glue, and tape

The structure of the MMMM consists of two main parts: the maze itself and the electronics housing/base. The maze was designed using a random maze generator in Easel, the software for the X-Carve. Due to difficulties finding appropriately thick wood to fit our marble, we decided that it would be easier to cut the walls manually and put the maze together rather than using the X-Carve. To aid this process, I drew the maze design on a grid to determine what wall size we wanted without making the maze too big and to find the dimensions of each wall that we needed to cut, shown in Fig. 1. We decided to make the walls and pathways 1.6" wide to fit the 1.4" marble while keeping the maze at a reasonable size of 17.6"x9.6".

We glued two 4'x8" planks that we wood glued together to get a sufficient wall height of 1.5". Wood strips were then cut for the walls using a miter saw and a table saw, as specified by Fig. 1(b). A band saw was used to cut rectangular slots out from some walls to make room for the IR sensors in the locations in Fig. 1(a). The walls were wood glued to one of the 3mm plywood sheets which would serve as the top of the electronics housing. Holes were drilled in the locations of each IR sensor on the top of the box after the maze walls. The IR sensors were glued into their positions with their wires feeding through the holes.

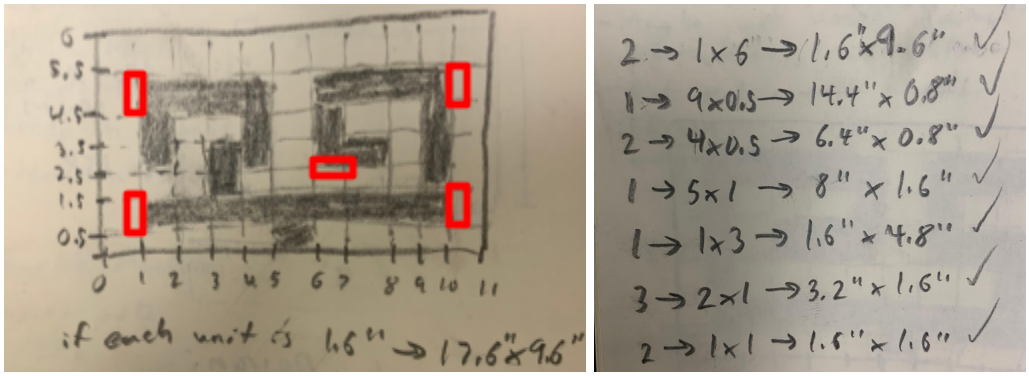


Fig. 1: (a) The maze design sketched on a unitless grid. Specifying each grid square to be 1.6”x1.6”, the full design is 17.6”x9.6”. The red rectangles indicate the locations of IR sensors, where slots would be cut out. (b) List of the wall dimensions required to build the maze with the wall thickness as 1.6”.

The remaining plywood sheets were laser cut to create the bottom and sides of the base under the maze. The bottom of the box had eight hexagonal holes cut out to fit the buttons through. The side walls are thin and just meant as covers without adding structural support. The front and back walls were made from ¾” thick scrap wood, cut to the correct length. The bottom piece was screwed into these walls in four corners before the side walls were glued to the front/back walls. Velcro strips were glued to four corners of the top of the walls and the bottom of the maze base to close the box but keep it available to open for prototyping. All of the electronics, including an Arduino, are housed inside this box, with a hole drilled in the back wall for the Arduino cable to be plugged in. A hole was also drilled through one of the side walls to fit the on/off switch through. Finally, the 3mm clear acrylic was laser cut to the size of the maze and screwed into the top of the walls to prevent the marble from escaping out of the maze during use, while a rubber band was stretched between two screws outside of the maze entrance to prevent the marble from falling out the front.

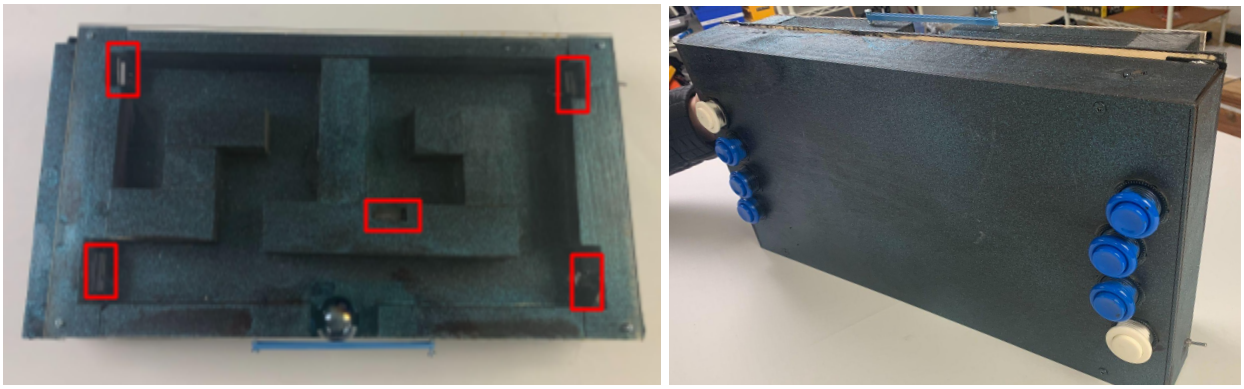


Fig. 2 and 3: Top and Bottom view of the MMMM. The top shows the structure of the maze with the IR sensor locations/functions while the bottom shows the eight buttons.

Circuitry

The circuitry for the MMMM was done on two solderable breadboards, to connect the buttons and IR sensors on opposite sides to a central Arduino Uno. The boards are both connected to the 5V and ground pins of the Arduino. The board on the right has four buttons, three IR sensors, and the switch, while the board on the left has four buttons and two IR sensors. The wiring assumes that any code will set the buttons as 1-4 moving up the left side and 5-8 moving up the right side.

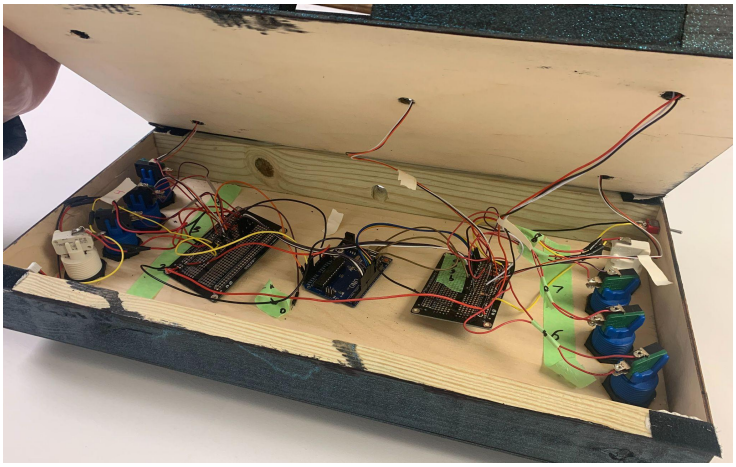


Fig. 4: Photo of circuitry. Shows the intended order of the buttons, IR sensors being wired through holes in the top of the box, and the interconnected breadboards.

The wiring for each IR sensor is very simple — they have a red wire which connects to power, a black wire which goes to ground, and a white wire which outputs to the Arduino directly. The white wires were soldered to rows of their respective breadboards, before being wired to specific Arduino pins. The buttons are also relatively simple. One of their wires connects to power while the other splits between the Arduino and a 10k pull-down resistor to ground. The pin numbers for each connection is as follows:

- Buttons 1-8 in digital pins 4-11
- IR Sensors on left in A4-A5
- IR Sensors on right in A0-A3

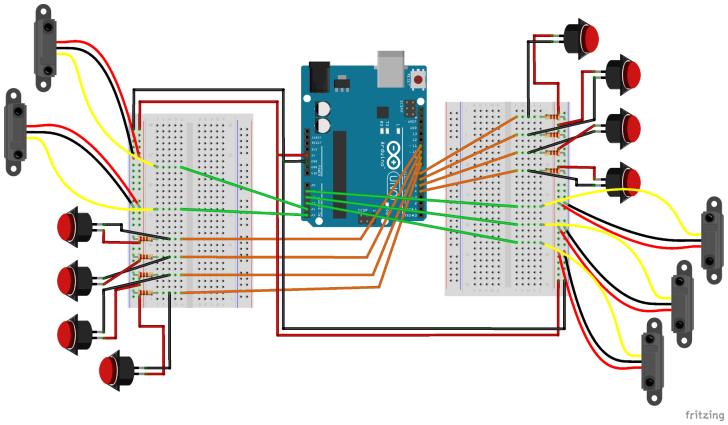


Fig. 5: Wiring diagram of the sensors and Arduino. Does not include the switch, but it is similar to the buttons.

Max and Reason

Our Max patch receives data from the Arduino serial port and processes it differently for buttons vs IR sensors. The full patch is shown in Fig. 6, but is shown in more detailed parts below.

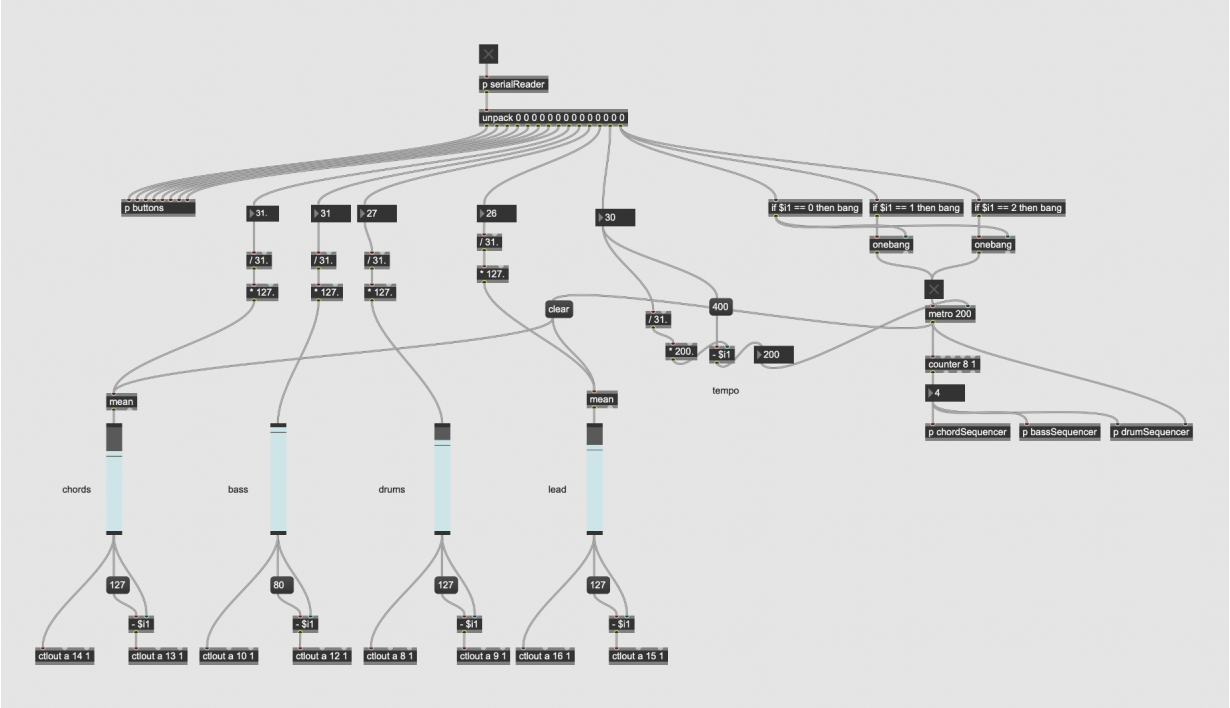


Fig. 6: Entire MMMM Max patch. Includes reading in data from Arduino and the processing for the buttons, IR sensors, and switch. The switch turns on a sequencer for drums, bass, and chords which are each encapsulated in sub-patches. The buttons play a lead instrument and are also encapsulated. The IR sensors control the sliders seen here, which mix between different instrument sounds.

The first part of the patch is a simple serial reader to read data from Arduino. There is a toggle which acts as an on/off switch for the whole instrument in that it starts a metronome for the serialReader sub-patch in Fig. 7. This reads from the serial port of the specified port at the specified baud rate in the serial object, set to match the Arduino port and code. Some additional processing follows to ignore newline characters and convert from ASCII back into the actual numbers that the Arduino code intends to print. Finally, the data is unpacked outside of the subpatch to specific locations; the message that Arduino prints to the serial port is in a specific order so that each unpacked value easily corresponds to a specific button or IR sensor.

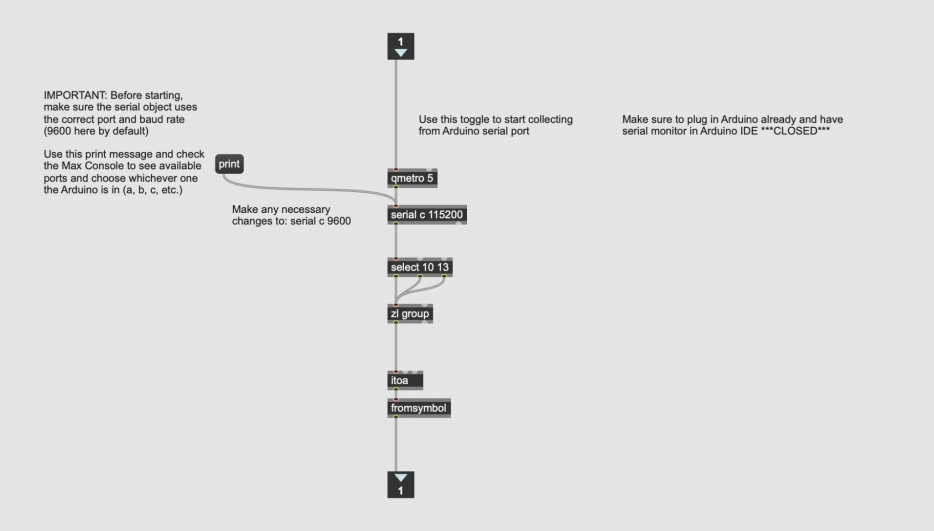


Fig. 7: The serialReader subpatch. The single input comes from a toggle that activates a metronome to continuously read from the serial port when activated. The serial messages from the Arduino are processed and sent out of the subpatch to be unpacked in a specified order for each button, IR sensor, and switch.

Each unpacked message of the data goes to code to handle buttons or IR sensors. They are printed and unpacked in the order of buttons 1-8 then IR sensors 1-5 and finally the on/off switch. Each button uses Arduino code which follows the same logic: the Arduino prints 1 when the button goes from a released to pressed state, 2 when the button goes from a pressed to released state, and 0 otherwise. The Max patch then handles this by starting a note when it receives a 1 and flushing that note to turn it off when it receives a 2. It starts the note by sending a bang to a message with a specific midi note value (44 or G#3 for button 1), which then gets sent to a makenote with a set velocity of 100 and an excessively long duration. This makenote object is then wired into flush before wiring to noteout. This way, when the condition for 2 sends a bang directly to flush, the note stops. Two makenote objects are used for each button, so each is playing two midi channels which correspond to different instrument sounds simultaneously. The buttons all play to channels 8 and 9 — the instruments for each channel are outlined in Table 1. Each button corresponds to a different note in a major scale with the root note at button 1 and an octave at button 8.

The IR sensors have a proximity value with an approximate range of 4-31 printed from the Arduino. The general processing for each sensor divides the data by 31 and multiplies it by 127, normalizing it to the 1-127 range for easy midi control. The wider range of data is shown on a slider before being sent to a ctlout object, which controls a specified midi parameter. IR sensors 1-4 each control a mixer in reason, which mixes the volume between two different instruments corresponding to each of four instrument categories: chords, bass, drums, and lead. The leads are the instruments which are played by the buttons, while the other instruments are played by sequencers built into the Max patch. The last IR sensor controls the tempo of the sequencers, making it slower as the marble approaches it. The 1st and 4th IR sensors also use moving averages to smooth the data — this moving average is cleared out on the same time interval that the sequencers metronome uses. These two IR sensors are long range sensors on the long front path of the maze, and tend to have noisier data at long ranges, which is the default when the marble wasn't in their path. Thus these were the most important sensors to smooth out the data of.

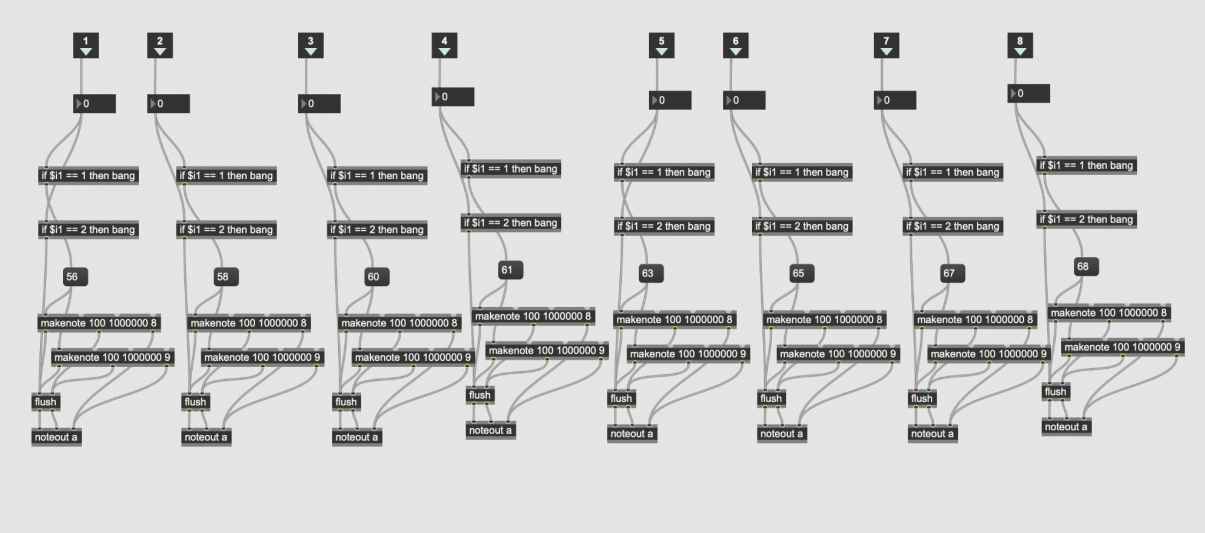


Fig. 8: The buttonStuff subpatch for eight buttons controlling midi channels 8 and 9. Buttons receive 0, 1, or 2, sending a bang to two very long duration makenote objects for each channel on 1 and sending a bang to flush those notes on 2.

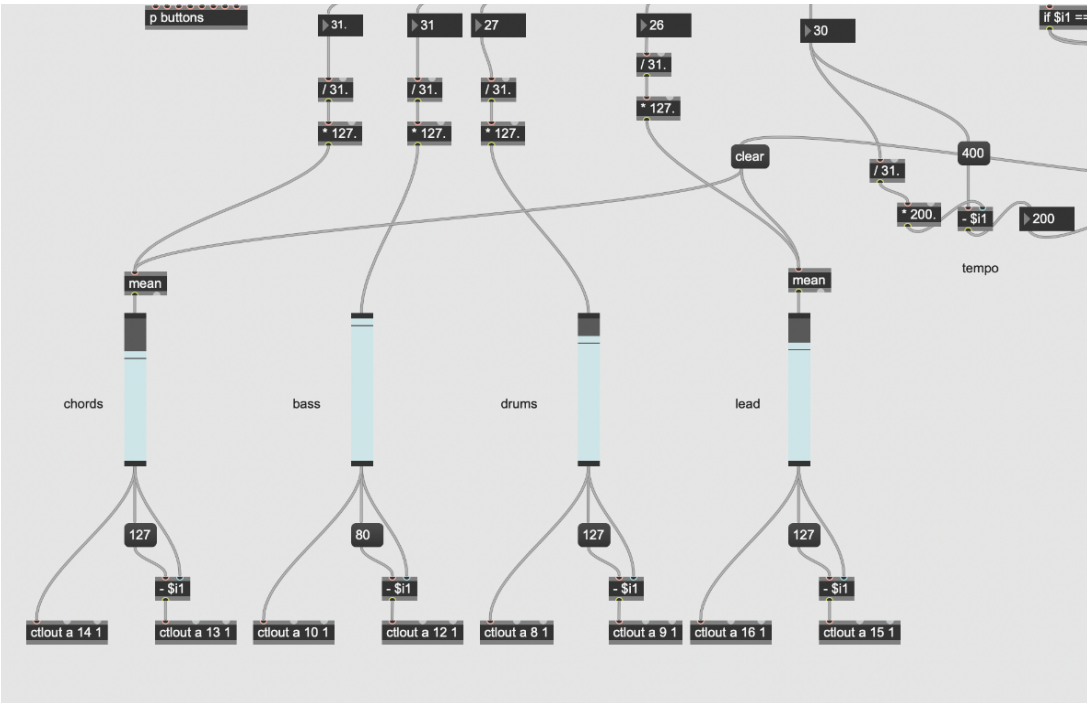


Fig. 9: Zoom in off the full Max patch for the five IR sensors controls. Four IR sensors are processed to have specific ranges of values that control the mixer volumes between two separate instruments assigned to play for each instrument type: chords, bass, drums, and lead. The last IR sensor controls tempo instead, slowing down as the distance decreases.

The on/off switch serves to toggle the sequencers playing a randomized backing track of drums, bass, and chords. The switch simply toggles the metronome which controls the tempo of all three sequencers simultaneously, which are encapsulated separately in chordSequencer, bassSequencer, and

drumSequencer. The bass sequencer is the simplest: it is a probabilistic 8-note sequencer that plays random notes in a scale for each note that plays. More specifically, a counter in the main patch counts up to 8 and each number 1-8 is selected with a sel object and sent out to separate random objects inside of the sub-patch. Each random object generates a random number up to 100 before comparing it to a set probability value. If the number is below this probability value, a random note is played to two separate channels with different bass instruments. These probabilities can be altered with knobs in the sub-patch, but are preset to values based on testing what sounded good to us. The chord sequencer is nearly identical, but uses different probabilities to have chord changes less often and plays chords in a major key instead of individual notes in that key.

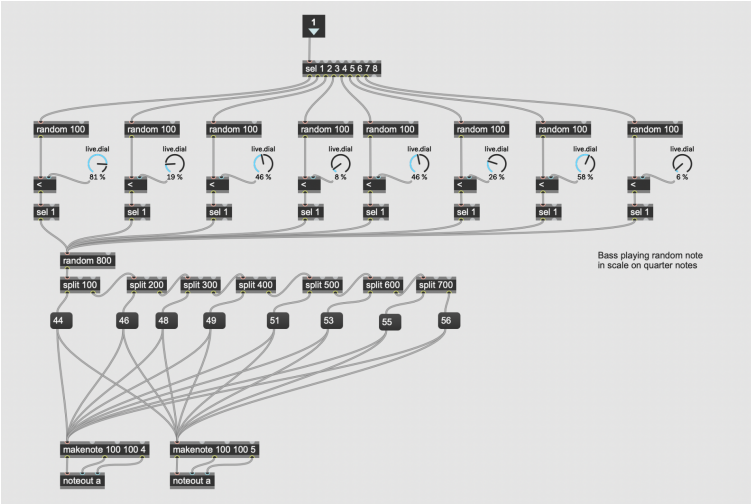


Fig. 10: The bassSequencer sub-patch. Two different bass channels play random notes in a major scale on random notes in an 8-note sequence based on preset probabilities.

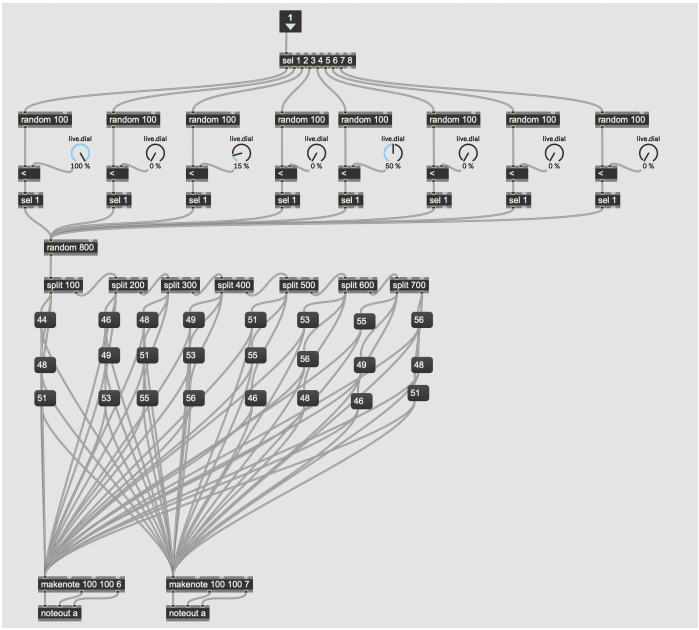


Fig. 11: The chordSequencer sub-patch. Two different channels play random chords in a major key on random notes in an 8-note sequence based on preset probabilities.

The drumSequencer sub-patch works slightly differently, but with some similar probabilistic controls. The main differences are that it mostly isn't connected to the 8-note counter and the notes aren't randomized. The first input does come from the counter and consistently plays on notes corresponding to a kick drum on 1 and 5 (1 and 3 if you are thinking about eighth notes). The second input, however, comes directly from the metronome and connects to random objects to play a kick, snare, hi-hat, and shaker. On any count, there is a random chance to play any or all of these different drum sounds. On each count, the probability to play each of these drums changes as well, based on a random function with additional math functions to scale the probability differently (the highest to lowest probabilities are shaker, hi-hat, snare, and then kick). Thus, this sequencer is semi-generative, playing a simple but consistent kick drum beat with additional random notes changing in probability with time.

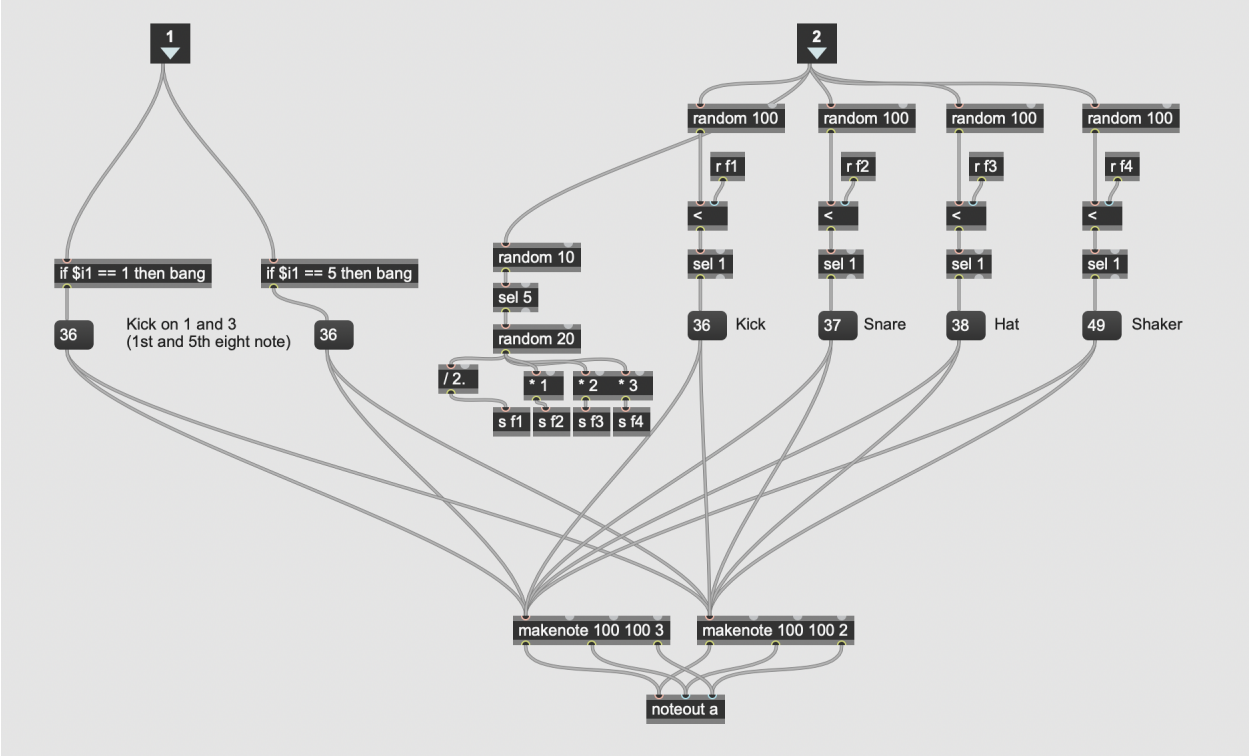


Fig. 12: The drumSequencer sub-patch. Two different channels play kick, snare, hi-hat, and shaker sounds on random metronome ticks, with the probability to play randomly changing with each tick. Additionally, two consistent kick drum notes are played on 1 and 5 in an 8-note sequence.

Finally, all of these controls get sent to several channels in a Reason instrument. The IR sensor controls go to channel 1, which is a 14:2 mixer. The channels for the buttons, and each sequencer are outlined in Table 1. The instruments themselves are shown in Appendices 1-4, for drum, bass, chord, and lead instruments respectively. This is the same order that they show up in pairs of two instruments per category in both the interface and mixer in Fig. 13.

Table 1: Reason Channels and Corresponding Instruments/Controls

Channel	Instrument	IR Sensor Mixer	Note Controller
2	Pop Kit 1	3	drumSequencer
3	Funt Kit	3	drumSequencer
4	808 Bass	2	bassSequencer
5	101 Bass	2	bassSequencer
6	Bottles	1	chordSequencer
7	Soft Resonance	1	chordSequencer
8	Evolved Organ	4	buttonStuff
9	Vibraphone	4	buttonStuff

Figure 13: MMMM Reason rack interface and mixer. Channel 1 on the interface is the mixer, while channels 2-9 are different instruments. Mixer channels 1-8 correspond to each instrument in the same order as the interface. Each instrument type comes in pairs of two, in the order: drums, bass, chords, lead.

Future Work

The MMMM functions well in that all of the intended controls were successfully implemented. The buttons play assigned notes, allowing for a simple musical control, while the IR sensors make for an interesting control scheme for altering the sounds of the instrument. The sequencers also provide a backing track that can be controlled by these sensors and provide additional interesting effects on top of having a full band of instruments being played on its own. Improvements for the future would mainly be adding additional sensors for more ways to control the sounds.

One idea for this would be developing a control scheme for the sequencers themselves. Having a separate box with 16 physical knobs to control the knobs on the bass and chord sequencers could be one way to do this. Although it would be separate from the main maze, it could be preset to different rhythms for different songs or a second person could effectively “play” the backing track by tweaking it separate from the maze. In addition, they could be advanced to 16-note sequencers with a faster metronome and more counter splits. This could allow for more rhythmic possibilities, especially for the drums and bass.

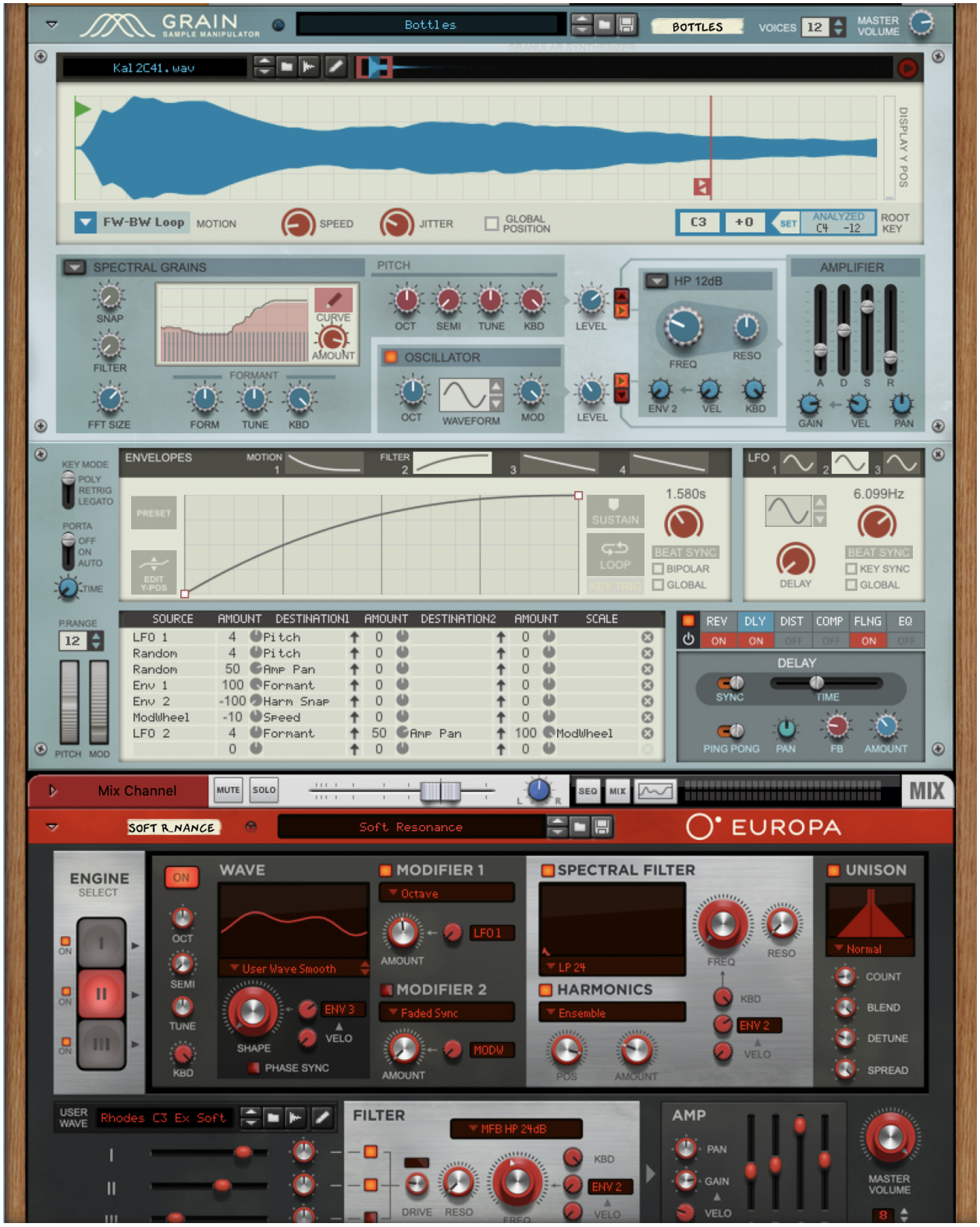
Another possible improvement could be adding another switch to change instrument modes. In a different mode, the buttons could play chords instead of single notes. In this mode, you probably wouldn’t want a backing track, so this would turn off and the IR sensors would instead be used to control other parameters of the chord instrument sound.

A final thought would be rebuilding the maze to better optimize the number of IR sensors that could be used. With more IR sensors, more aspects of the sound could be controlled. This could make for a more interesting sounding instrument, with more variability throughout sections of the maze. In a larger maze, the user could take time to explore what each section changes in the sound. A more frictional surface between the floor of the maze and the marble could make this process slower and easier to control, adding to this exploration. Even building a large gyroscopic stand could make small rotations easier to control and make the instrument easier to hold, as it holds itself up and the user can focus on exploration and playing the buttons.

Appendix 1 - Drum Instruments

Appendix 2 - Bass Instruments

Appendix 3 - Chord Instruments



Appendix 4 - Lead Instruments