COSMOS UC San Diego
California State Summer School for Mathematics and Science

# Light Jams

•••

Utilizing an Arduino Microcontroller to Read Potentiometers, Buttons, and Multiplexer Analog Inputs from Photoresistors, and Generating MIDI Output Based on the Input to a Pure Data Synthesizer

Ramon Magana, Jeremy Mui, Daniel Wehara

COSMOS UC San Diego
California State Summer School for Mathematics and Science

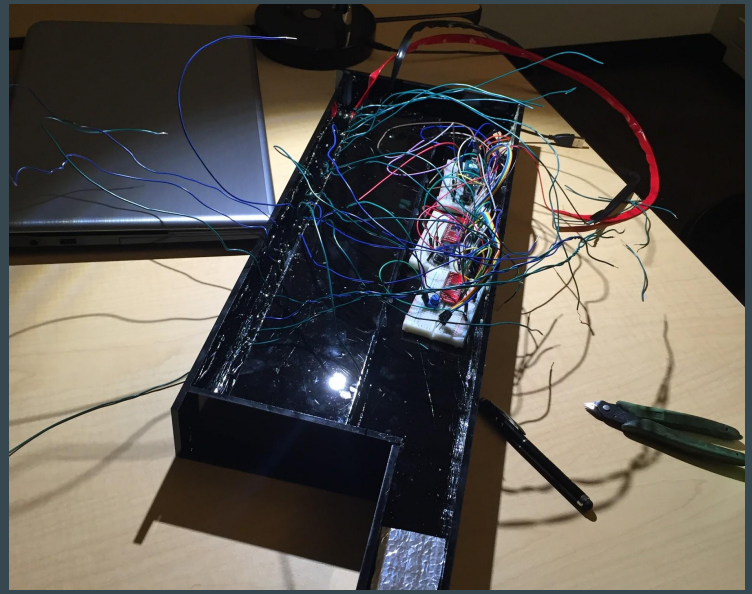
# Product Features

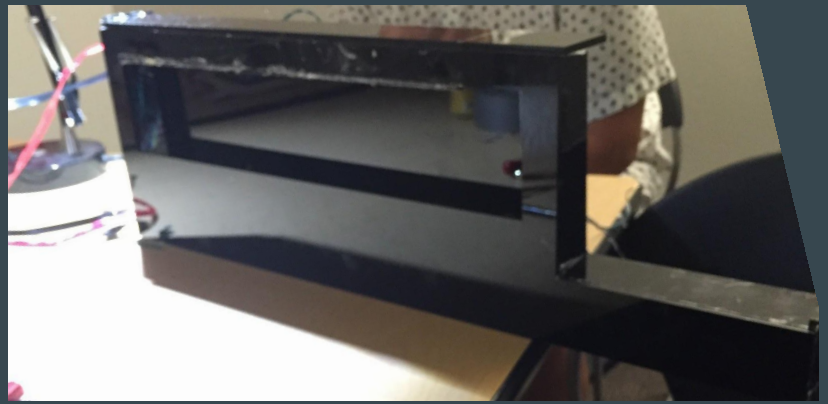
- Sleek glossy black acrylic body
- Octave changes, recording, and playback possible through buttons
- 25 “keys” allow for the playing of every possible chord inversion
- Octave changes allow the range to extend from MIDI note 12 (C0) to MIDI note 120 (C9), which is greater than that of a piano
- Dynamics can be achieved through a potentiometer that as a dial that changes the MIDI notes’ velocities

Keytar and Light Keytar

COSMOS UC San Diego
California State Summer School for Mathematics and Science

# Finished Product





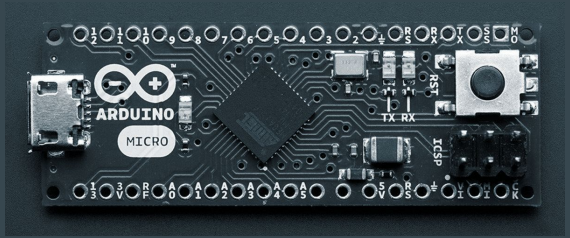
# Hardware

- Arduino Micro
- Photoresistors
- Buttons
- Potentiometer
- Multiplexers
- Acrylic Body

COSMOS UC San Diego
California State Summer School for Mathematics and Science

# Arduino Micro

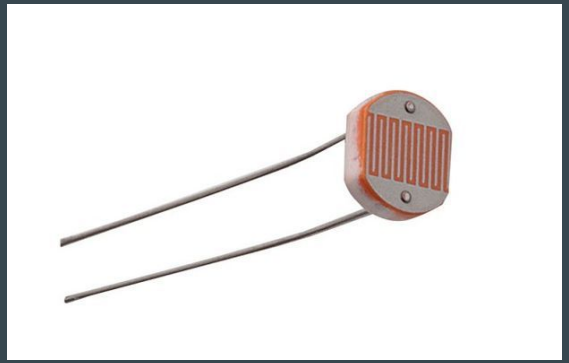
- Microcontroller
  - A processor that takes  a preprogrammed command and executes it continuously
- 8-bits
- 6 Analog Pins
- 11 Digital Pins
- Can read many different inputs such as buttons, potentiometers, and photoresistors
- Used to read input and send MIDI output

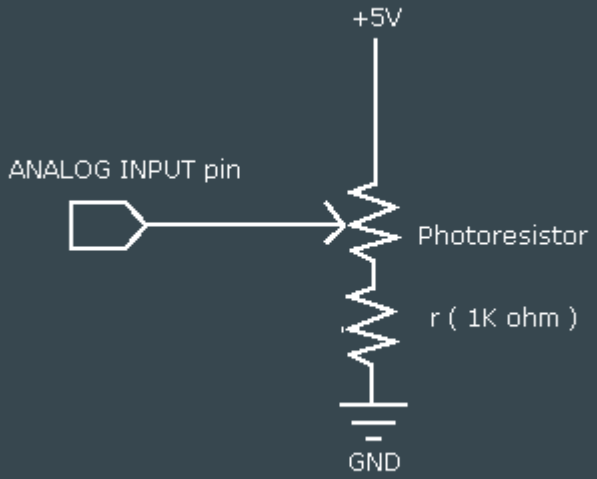


COSMOS UC San Diego
California State Summer School for Mathematics and Science

# Photoresistors

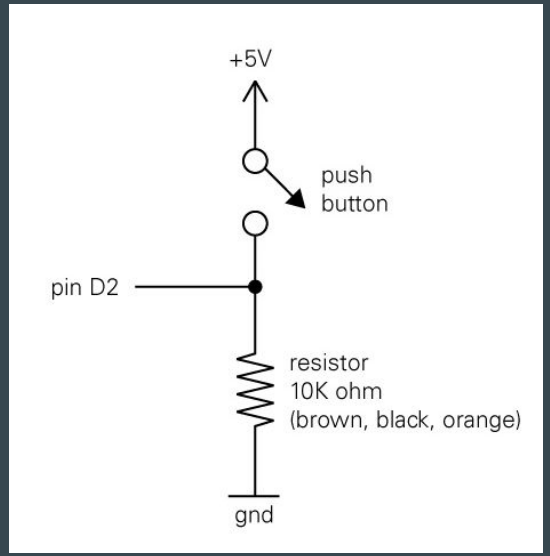
- Changes amount of voltage passed through based on the intensity of light shining on it
- Voltage decreases when dark, and increases when in the light
- Acts as the “keys” for the instrument





# Buttons

When the button is pressed, a high voltage is sent to the input which can be used to trigger an action, such as octave changes, recording, and playback.




COSMOS UC San Diego
California State Summer School for Mathematics and Science

# Potentiometer

- By turning the knob, the amount of voltage flowing through changes
- Through connecting it to an analog pin, it can be used as a volume dial for dynamics during performances

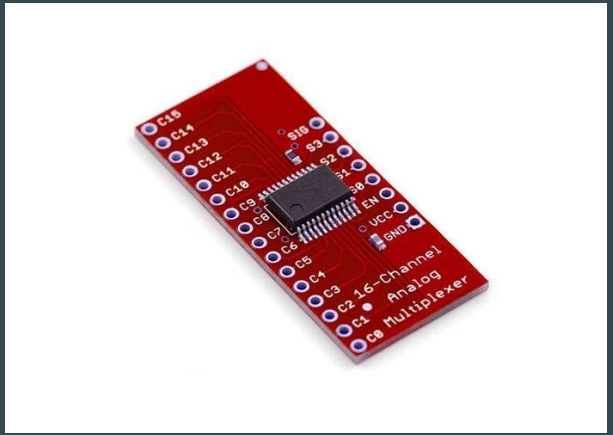
# Body

- For the instrument's body, we initially used a foam prototype
  - This design proved to be unstable and not very durable
- The inferior foam model was abandoned in favor of acrylic, a stronger, superior material
- We laser cut ⅛ inch thick black acrylic into the shape of a keytar, using individually cut faces and holes to route cables

COSMOS UC San Diego
California State Summer School for Mathematics and Science

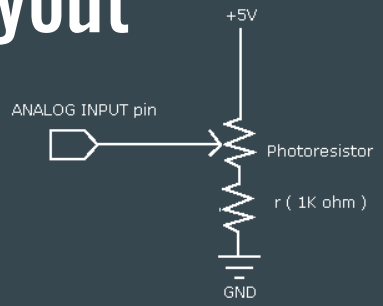
# Multiplexers

- Each has 16 analog inputs
- Need 2 in order to read all the inputs from our 25 key photoresistors
- By writing commands to the multiplexer, the Arduino can read specific analog pins on the multiplexer, effectively expanding our available inputs from ~12 to over 50

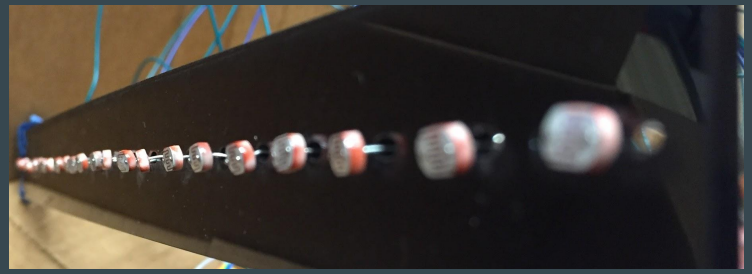


COSMOS UC San Diego
California State Summer School for Mathematics and Science

# Layout

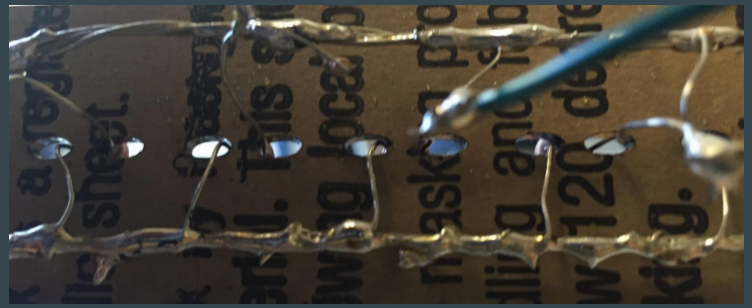


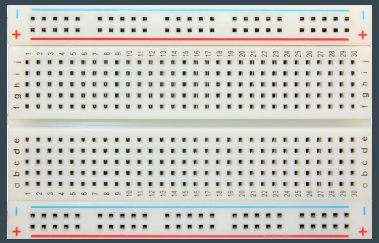

Sensor bridge - 25 photoresistors with legs through holes



Under sensor bridge -

- Leg of photoresistor connected to power (5v)
- Leg of photoresistor and 330 ohm resistor connected to analog input
- Leg of resistor connected to ground

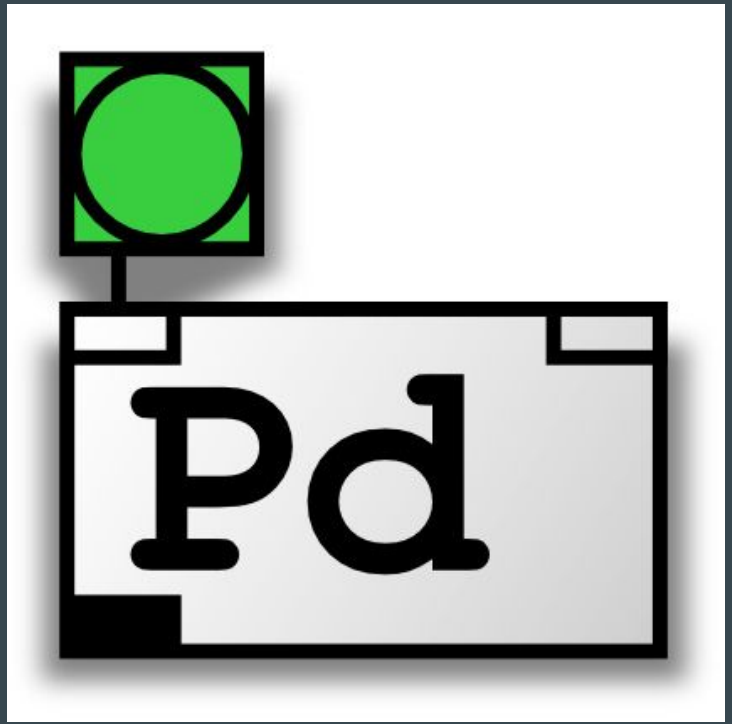




Power and Ground cables connect to the Arduino through the breadboard

# Software

- Arduino IDE
- Pure Data
- Hairless MIDI
- loopMIDI

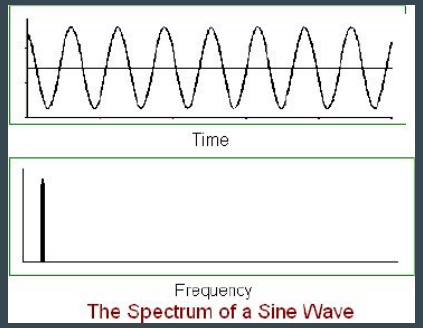



# Pure Data

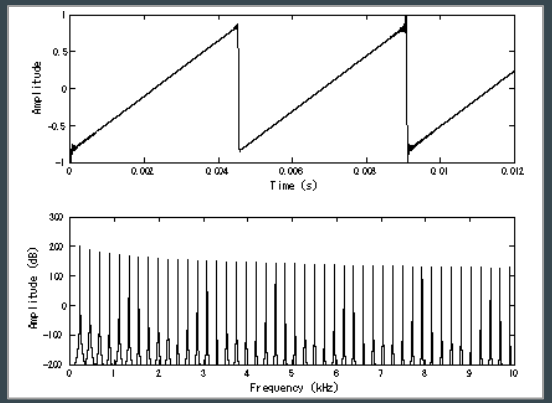
- Pd is a visual programming language that is used for computer music and multimedia
  - Pd is used to make many things ranging from digital audio workstations to synthesizers and other software instruments
- Made a synthesizer that takes midi signals and converts them to notes with frequency and velocity
- Subtractive synthesis - uses a sawtooth wave with many harmonics and cuts out the unwanted frequencies to form a sound with the desired timbre

COSMOS UC San Diego
California State Summer School for Mathematics and Science

# What is subtractive synthesis?




Boring sine wave - no harmonics, one frequency

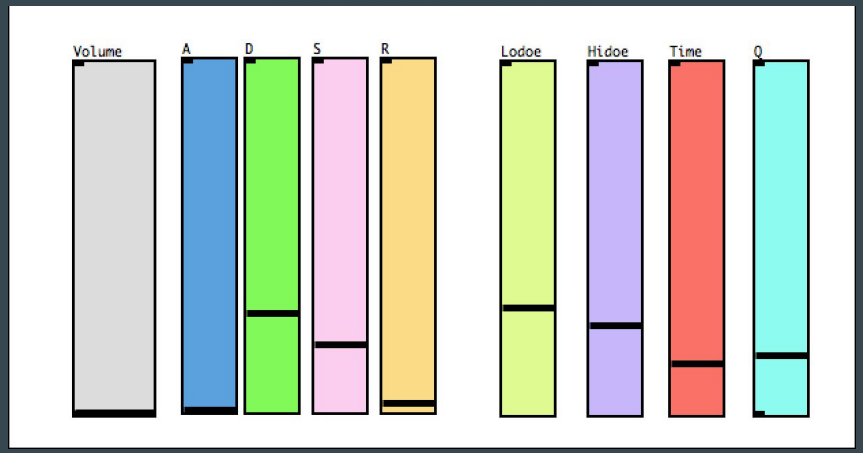



Sawtooth wave - many harmonics, excess frequencies (sounds too dirty and harsh)

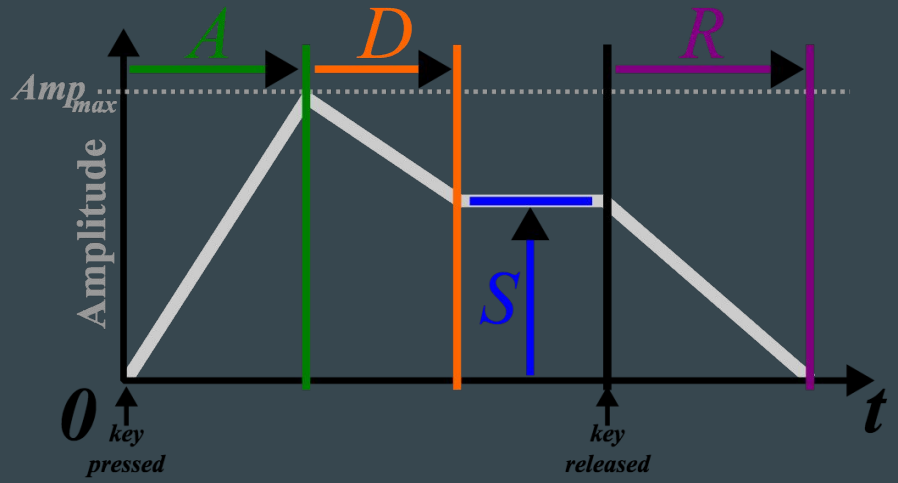

Filters select and modify which frequencies can pass through so only desired harmonics are heard.

Low pass filter lets frequencies below a determined frequency pass through
High pass filter lets frequencies above a determined frequency pass through

# Pure Data Synthesizer




User interface with volume, attack, decay, sustain, release (ASDR), low pass, high pass, time, and q factor



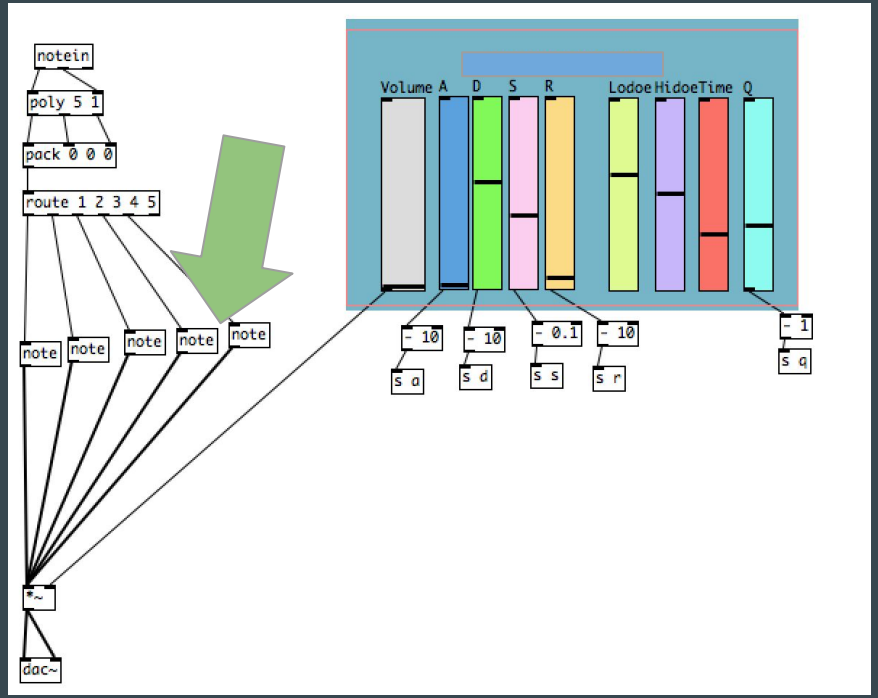

Attack - time until peak volume
Decay - time from peak until constant volume
Sustain - time of constant volume
Release - time until note stops after key is released

# Polyphony

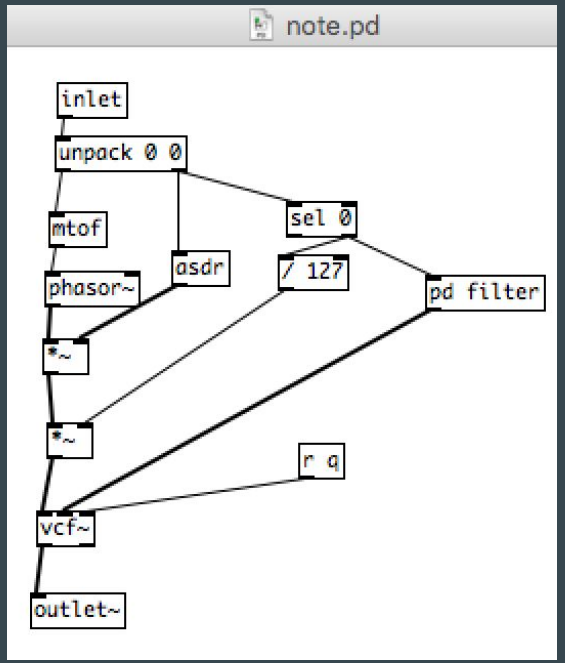
Poly 5 1 and route 1 2 3 4 5

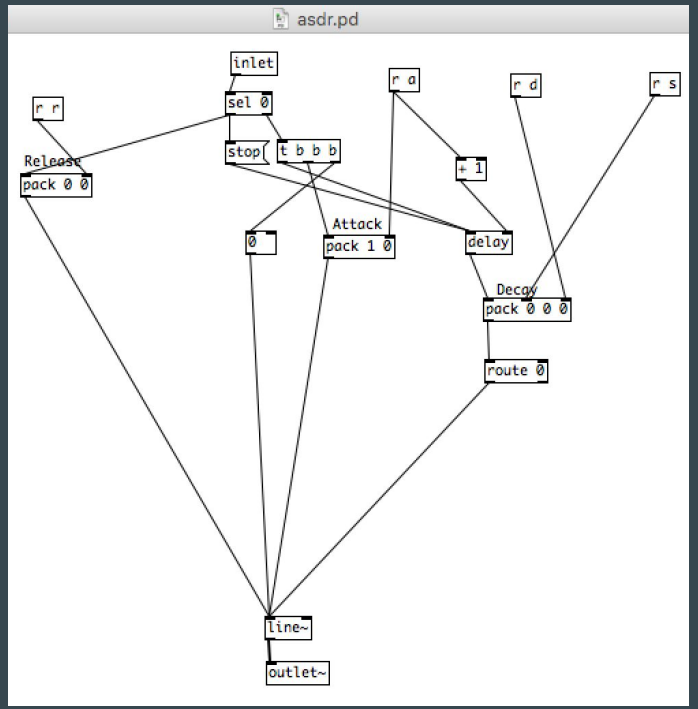
-allow for five “note” objects to connect

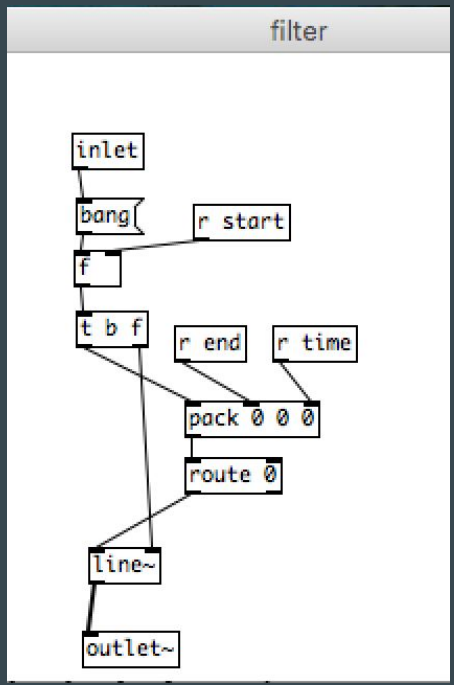
-play up to 5 notes at a time (Polyphony)



# Inside the "note" object


Internal subpatches in the synthesizer that modify the ASDR and frequencies that pass through. These components are in subpatch format to make things cleaner

# Arduino Programmed Functions

```
/*************************************************************************************************************
Arduino Light Keytar - Click Here for Source Code
By Jeremy Mui
readKeyLevels() for reading multiplexer analog inputs provided by Colin Zyskowski and modified by Jeremy Mui
August 2016
*************************************************************************************************************/
void readKeyLevels(int* arr) {
void blinkThrice() {
void generateKeyDefault() {
void setup() {
void noteOn(int pitch) {
void noteOff(int pitch) {
void play() {
boolean isButtonPressed(int pin) {
void detectLowerOctave() {
void detectRaiseOctave() {
void detectRecord() {
void detectVelocity() {
boolean isFinishedRecording() {
void record() {
void playback() {
void detectPlayback() {
void eraseRecording() {
void detectEraseRecording() {
void loop() {
```

# void readKeyLevels(int* arr)

COSMOS UC San Diego
California State Summer School for Mathematics and Science

```
/*
* Reads both multiplexers' analog inputs and stores them into parameter arr.
* Provided by Colin Zyskowski and modified by Jeremy Mui
* Parameter int* arr is the array where the photoresistor inputs are to be stored
*/
void readKeyLevels(int* arr) {
	int index1 = 0;
	int index2 = 16;
	for (int i = 0; i < 16; i++)
	{
		//The following 4 commands set the correct logic for the control pins to select the desired
		input
		//See the Arduino Bitwise AND Reference: http://www.arduino.cc/en/Reference/BitwiseAnd
		//See the Aruino Bitshift Reference: http://www.arduino.cc/en/Reference/Bitshift
		digitalWrite(CONTROL0, (i & 15) >> 3);
		digitalWrite(CONTROL1, (i & 7) >> 2);
		digitalWrite(CONTROL2, (i & 3) >> 1);
		digitalWrite(CONTROL3, (i & 1));
		//Read and store the input value at a location in the array
		arr[index1] = analogRead(multiplexer1);
		if (index2 <= 24)
			arr[index2] = analogRead(multiplexer2);
		index1++;
		Index2++;
	}
}
```

# void generateKeyDefault()

```
/*
* Calibrates photoresistors by setting baseline values for each
* Reads all photoresistor inputs using readKeyLevels(), then waits for 1 second
* Does this for a total of 3 times
* The average of each sensor's 3 readings are stored in keyDefault
*/
void generateKeyDefault() {
      int raw[3][25];
      for (int row = 0; row < 3; row++)
      {
            readKeyLevels(raw[row]);
            delay(1000);
      }
      for (int col = 0; col < 25; col++)
      {
            for (int row = 0; row < 3; row++)
            {
                  keyDefault[col] += raw[row][col];
            }
            keyDefault[col] /= 3;
      }
}
```

# void noteOn(int pitch)

```
/*
* Writes MIDI signal to Serial for turning on a note
* Parameter int pitch is the MIDI number for the note to be
turned on
*/
void noteOn(int pitch) {
    Serial.write(on); //on = 144 for channel 1
    Serial.write(pitch);
    Serial.write(velocity);
}
```

# void noteOff(int pitch)

```
/*
* Writes MIDI signal to Serial for turning off a note
* Parameter int pitch is the MIDI number for the note to be
turned on
*/
void noteOff(int pitch) {
    Serial.write(off); //off = 128 for channel 1
    Serial.write(pitch);
    Serial.write(velocity);
}
```

# void play()

COSMOS UC San Diego
California State Summer School for Mathematics and Science

```
/*
* Method that detects changes in the current photoresistor values and the keyDefault
* If the change is greater than delta, the MIDI signal for turning on the note
corresponding to the photoresistor will be sent using noteOn()
* If the change is no longer greater than delta, the MIDI signal for turning off the
note corresponding to the photoresistor will be sent using noteOff()
*/
void play() {
	readKeyLevels(keySensorLevels);
	for (int i = 0; i < 25; i++) //size here
	{
		int diff = keyDefault[i] - keySensorLevels[i];
		if (diff > delta)
		{
			if (!keyPressed[i])
			{
				keyPressed[i] = true;
				noteOn(i + transpose);
		}
			}
			else
			{
			if (keyPressed[i])
			{
			keyPressed[i] = false;
			noteOff(i + transpose);
			}
		}
	}
}
```

# boolean isButtonPressed(int pin)

```
}
/*
* Returns true if the button at parameter pin is pressed, and
false if not
*/
boolean isButtonPressed(int pin) {
    return digitalRead(pin) == HIGH;
}
```

# void detect[...] ()

COSMOS UC San Diego
California State Summer School for Mathematics and Science

void detectLowerOctave()
void detectRaiseOctave()
void detectRecord()
void detectVelocity()
void detectEraseRecording()

```
/*
* If the lower octave button is pressed and the lowest key is still in range, lowers transpose by 12
* All notes are then played one octave lower
* Program execution is delayed for buttonDelayTime ms for button debouncing
*/
void detectLowerOctave() {
	if (isButtonPressed(lowerOctaveButton) && transpose - octave >= 12)
	{
		transpose -= octave;
		delay(buttonDelayTime) ;
	}
}
```

```
*
* If a recording is saved and the playback button is held for 3 seconds, executes eraseRecording()
*/
void detectEraseRecording() {
	if (recordingAvailable && isButtonPressed(playbackButton))
	{
		unsigned long timer = millis() ;
		while (isButtonPressed(playbackButton))
		{
			;
		}
		if (millis() - timer > 3000)
		{
			eraseRecording() ;
			blinkThrice() ;
		}
	}
}
```

# void record()



```
/*
* Records each key press as data structure note, which has timeStart, timeEnd, pitch, and
velocity
* Sends a call to noteOn() for each new key press
* Octaves and velocity can be changed during recording
*/
void record() {
	recordStartTime = millis();
	while (isRecording && !isFinishedRecording()) {
		detectLowerOctave(); //detect octave changes during recording
		detectRaiseOctave();
		detectVelocity(); //detect velocity changes during recording
		readKeyLevels(keySensorLevels);
		for (int i = 0; i < 25; i++)
		{
			int diff = keyDefault[i] - keySensorLevels[i];
			if (diff > delta && nextIndex < 119) //recording size here
			{
				if (!keyPressed[i])
				{
					keyPressed[i] = true;
					int pitch = i + transpose;
					notes[nextIndex].timeStart = millis();
					notes[nextIndex].pitch = pitch;
					notes[nextIndex].velocity = velocity;
					tempRecordingIndices[pitch] = nextIndex;
					noteOn(pitch);
					nextIndex++;
				}
			}
```

# void record() -continued

COSMOS UC San Diego
California State Summer School for Mathematics and Science

```
/*
* Sends a call to noteOff() at the end of each key press
* Calls detectRecord() each loop to determine if user has stopped recording
* Plays C Major arpeggio when recording limit has been reached
* Recording limit = 120 notes because of Arduino Micro memory limitations
* Sets recordingAvailble to true and isRecording to false at the end of the function call
*/
void record() {
                        else
                        {
                                if (keyPressed[i])
                                {
                                        keyPressed[i] = false;
                                        int pitch = i + transpose;
                                        notes[tempRecordingIndices[pitch]].timeEnd = millis();
                                        tempRecordingIndices[pitch] = -1;
                                        noteOff(pitch);
                                }
                        }
                }
                detectRecord(); //detects if the user has stopped recording
        }
        noteOn(48);
        delay(200);
        noteOn(52);
        delay(200);
        noteOn(55);
        delay(200);
        noteOn(60);
        delay(200);
        noteOff(48);
        noteOff(52);
        noteOff(55);
        noteOff(60);
        recordingAvailable = true;
        isRecording = false;
}
```

COSMOS UC San Diego
California State Summer School for Mathematics and Science

# void record() - A Visual Explanation

Each key press is stored as a “note”

```
/*
* Data structure for holding
notes' information
*/
typedef struct {
     unsigned long timeStart;
     unsigned long timeEnd;
     int pitch;
     int velocity;
} note;
```

Each note is stored in an array of notes called “notes”

```
note notes[120]; //120 = recording limit
```

A4: pitch = 57
Velocity = 127

timeStart = 5000 ms  timeEnd = 7000 ms

C#5: pitch = 61
Velocity = 127

timeStart = 6000 ms  timeEnd = 8000 ms

# void record() - A Visual Explanation (Continued)

An array is like a file cabinet

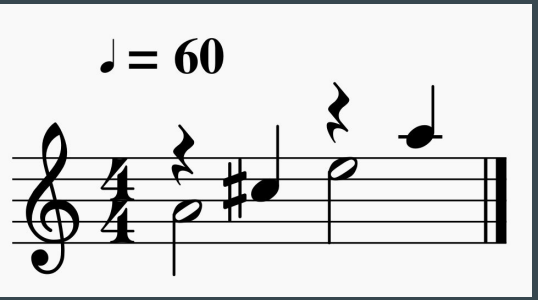
| **notes[0]** | **notes[1]** | **notes[2]** | **notes[3]** | **notes[4]** | **notes[5]** | **notes[6]** | **notes[7]** | **notes[...]** |
|---|---|---|---|---|---|---|---|---|
| timeStart = 5000<br>timeEnd = 5500<br>pitch = 57 (A4)<br>velocity = 127 | timeStart = 5500<br>timeEnd = 6000<br>pitch = 59 (B4)<br>velocity = 127 | timeStart = 6000<br>timeEnd = 6500<br>pitch = 61 (C# 5)<br>velocity = 127 | timeStart = 6500<br>timeEnd = 7000<br>pitch = 62 (D5)<br>velocity = 127 | timeStart = 7000<br>timeEnd = 7500<br>pitch = 64 (E5)<br>velocity = 127 | timeStart = 7500<br>timeEnd = 8000<br>pitch = 66 (F# 5)<br>velocity = 127 | timeStart = 8000<br>timeEnd = 8500<br>pitch = 68 (G# 5)<br>velocity = 127 | timeStart = 8500<br>timeEnd = 9000<br>pitch = 69 (A5)<br>velocity = 127 | timeStart<br>timeEnd<br>pitch<br>velocity |

# void playback() - Opening the File Cabinet

```
/*
* Plays back recorded notes from the note array notes
*/
void playback() {
	playbackStartTime = millis();
	int backIndex = 0;
	int frontIndex = 0;
	while (frontIndex >= backIndex)
	{
		timeElapsed = millis() - playbackStartTime;
		for (int i = backIndex; i <= frontIndex; i++)
		{
			if (!recordingIndexPlayed[i] && timeElapsed > notes[i].timeStart - recordStartTime &&
			frontIndex < 118) //recording size - 2
			{
				noteOn(notes[i].pitch);
				if (notes[frontIndex + 1].velocity != -1)
					frontIndex++;
				recordingIndexPlayed[i] = true;
			}
			if (timeElapsed > notes[i].timeEnd - recordStartTime)
			{
				noteOff(notes[i].pitch);
				if (i == backIndex)
					backIndex++;
			}
		}
	}
}
```

COSMOS UC San Diego
California State Summer School for Mathematics and Science

# void playback() - Opening the File Cabinet (Continued)




notes[3]
**A5**: pitch = 69
Velocity = 100

timeStart = 8000 ms  timeEnd = 9000 ms

notes[2]
**E5**: pitch = 64
Velocity = 100

timeStart = 7000 ms  timeEnd = 9000 ms

notes[1]
**C#5**: pitch = 61
Velocity = 100

timeStart = 6000 ms  timeEnd = 7000 ms

record StartTime = 4000

notes[0]
**A4**: pitch = 57
Velocity = 100

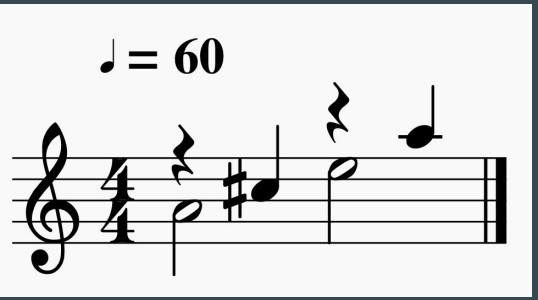
timeStart = 5000 ms  timeEnd = 7000 ms

```
unsigned long recordStartTime = 4000;
```

| notes[0] | notes[1] | notes[2] | notes[3] |
|---|---|---|---|
| timeStart = 5000 | timeStart = 6000 | timeStart = 7000 | timeStart = 8000 |
| timeEnd = 7000 | timeEnd = 7000 | timeEnd = 9000 | timeEnd = 9000 |
| pitch = 57 (A4) | pitch = 61 (C# 5) | pitch = 64 (E5) | pitch = 69 (A5) |
| velocity = 100 | velocity = 100 | velocity = 100 | velocity = 100 |

COSMOS UC San Diego
California State Summer School for Mathematics and Science

# void playback() - Opening the File Cabinet (Continued)




notes[3]
**A5**: pitch = 69
Velocity = 100

timeStart = 18000 ms  timeEnd = 19000 ms

notes[2]
**E5**: pitch = 64
Velocity = 100

timeStart = 17000 ms  timeEnd = 19000 ms

notes[1]
**C#5**: pitch = 61
Velocity = 100

timeStart = 16000 ms  timeEnd = 17000 ms

playbackStart Time = 14000

notes[0]
**A4**: pitch = 57
Velocity = 100

timeStart = 15000 ms  timeEnd = 17000 ms

```
unsigned long recordStartTime = 4000;
unsigned long playbackStartTime = 14000;
```

Arduino subtracts timeStart from recordStartTime, timeEnd from recordStartTime, and current time from playbackStartTime to determine when notes should be played and when they should be released

Uses an algorithm to efficiently loop through all the notes

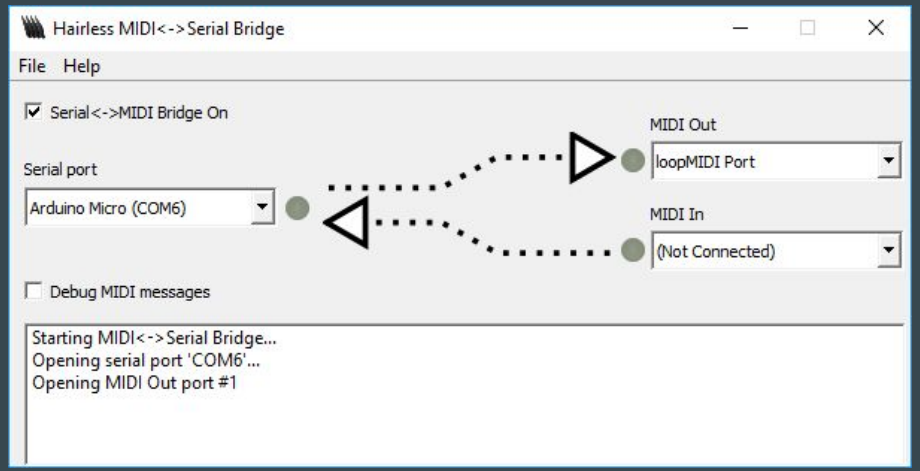
# void eraseRecording()

```
/*
* Erases recording by setting recordingAvailable to false and all notes' velocities to -1
*/
void eraseRecording() {
	recordingAvailable = false;
	for (int i = 0; i < 120; i++) //recording limit
	{
		notes[i].velocity = - 1;
	}
	nextIndex = 0;
}
```

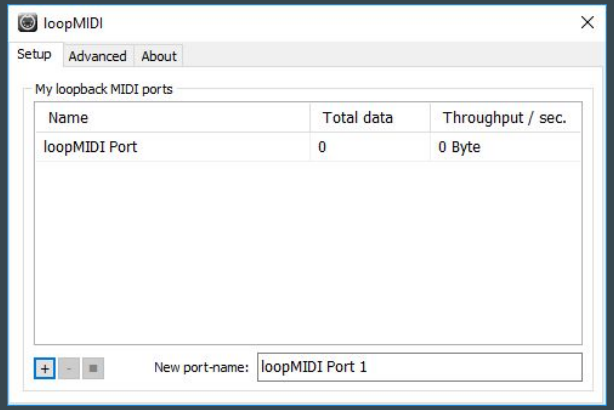
# void loop() - Putting it all together

```
/*
* Main loop that Arduino continuously executes
*/
void loop() {
// put your main code here, to run repeatedly:
      detectLowerOctave() ;
      detectRaiseOctave() ;
      detectVelocity() ;
      detectRecord() ;
      detectPlayback() ;
      isRecording ? record() : play() ;
      detectEraseRecording() ;
}
```

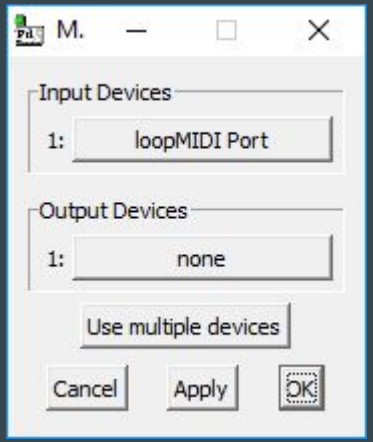
# From Arduino to Pure Data




Hairless MIDI is used to route MIDI from Arduino to loopMIDI




loopMIDI is used to create a virtual loopback cable




PD can read MIDI input from the loopMIDI port and synthesize sounds

COSMOS UC San Diego
California State Summer School for Mathematics and Science

# Difficulties

- Wiring
  - Lots of short circuits
  - Input wires came loose
  - Not enough analog input pins on the Arduino Micro
- Solution
  - Covering exposed wires with electrical tape
  - Soldering to secure photoresistors, resistors, and wires
  - Using multiplexers to expand the amount of inputs

# Possible Prototype Extensions

- Using a Raspberry Pi running Pure Data as a synthesizer, allowing for a mobile, untethered instrument
- Implementing analog output from the Arduino, removing the need for a synthesizer and allowing the Arduino to connect directly to speakers
- Reprogramming photoresistor inputs to play chords and possibly algorithmic harmonize with the user

COSMOS UC San Diego
California State Summer School for Mathematics and Science

# Media


Sensor bridge

Playing around & testing

Synth recorded on Logic & keytar MIDI inputs as software instruments

https://drive.google.com/open?id=0Bxme2p2ltu4gRkxRcEZibHJwajg MP3 of this →

COSMOS UC San Diego
California State Summer School for Mathematics and Science

# Acknowledgments


- Maurício de Oliveira, Kim Morris, and Kevin Haywood for their guidance in wiring
- Shlomo Dubnov for inspiring us to use technology in new ways to create sound
- Colin Zyskowski for providing us with code for the multiplexers and laser cutting the body for our instrument.