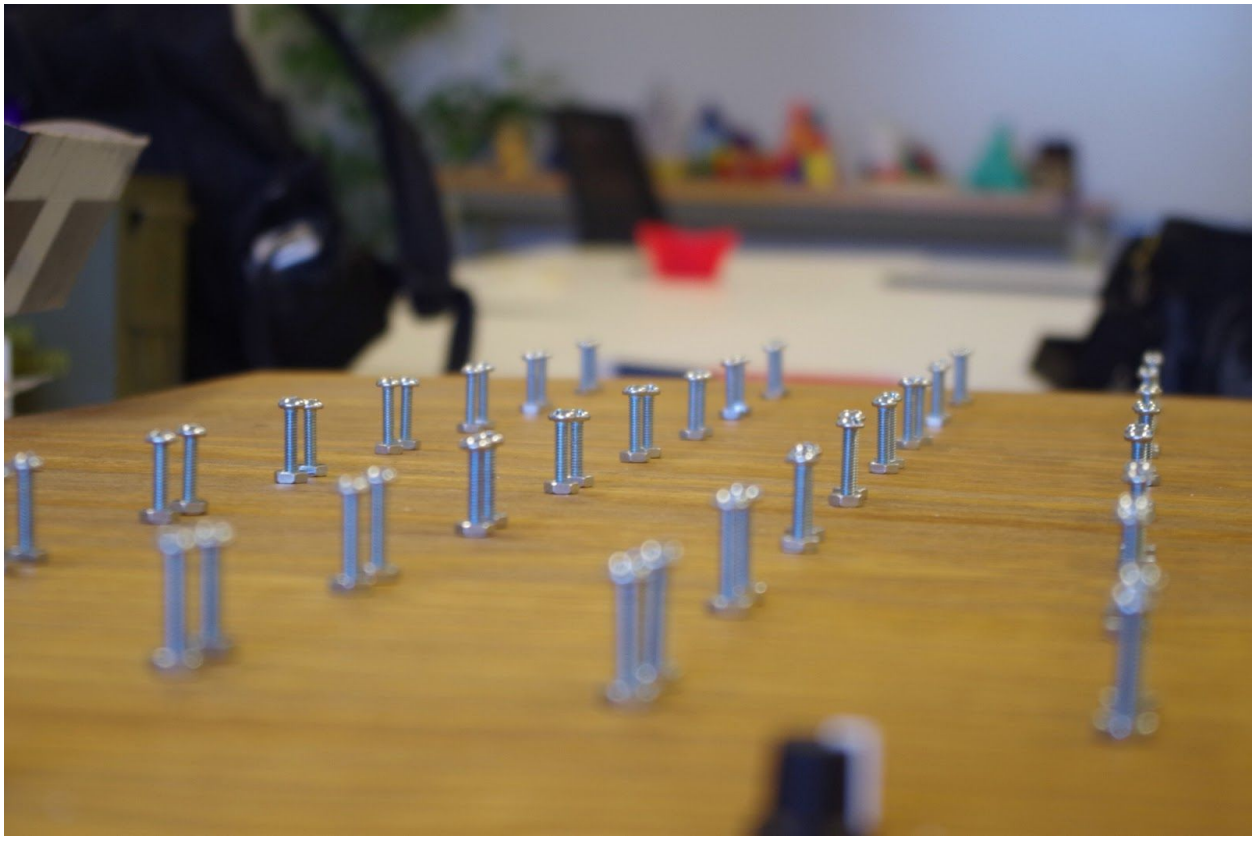


Physical Step Sequencer

David Schofield

INTRODUCTION

This project was completed as an assignment for my Interaction Design class (ARTM3220) taught by Professor Andrew Ames. Our objective was to create a "Meaningful Controller," which met the following criteria:

Objectives:

1. Create a physical interface for a single piece of existing or original software.
2. Improve prototyping skills and enhance object making techniques.
3. Continue to practoce user research and testing.
4. Implement basic electronic skills like soldering and wiring switches.
5. Utilize a micro-controller or micro-processor and explore various approaches to programming them.

Requirements:

1. *Uniqueness*: It must not already exist;
2. *Interaction*: It must be good for a particular type of game or software;
3. *Reality*: It should be constructed of off-the-shelf parts, including other controllers;
4. *Platform*: It must connect to a Mac or PC;
5. *Durability*: It must last during the opening reception of the gold show or senior showcase;

I did deviate from a couple of these requirements, opting to go with a self contained interface/program that didn't rely on an external computer.

THE IDEA

My idea was to create a physical step sequencer. A step sequencer is essentially a device that allows the creation of patterns in music, usually used with drum sounds, by allowing users to decide which sounds should be played at which part of a loop.

By having an physical step sequencer, users would be able to interact with the beat directly, and by having it playing continuously while the user arranges the pattern, they get to hear the beat evolve.

The actual interface I set out to build revolved around placing metal balls on pegs mounted on a wooden board. I chose this because I think materials play a big role in a user's experience, and both nice wood and polished metal evoke a sense of quality.

PROOF OF CONCEPT

My first step was to create a proof of concept for my idea, in order to see if it was feasible and work out as many bugs as possible before construction.

For this initial prototype, I elected to create a model in Autodesk's 123D Circuits simulator, since it also allows you to simulate an Arduino and write code for it.

Click here to view the latest iteration of my simulated proof of concept.

The simulator revealed my first major bug. When multiple connections were happening, the rows were reading erratically, often with all of them returning true. After examining the problem and tracing where the electricity would be going in my sketchbook, I discovered that I was experiencing electrical back-flow, where electricity would "piggy-back" on other rows and columns to create erroneous readings. The solution to this was to put a diode on each set of pegs, allowing electricity to only flow onto the rows needed.

Once I had a working proof of concept and most of the software written in the simulator, it was time to start collecting materials.

THE INGREDIENTS

1. Arduino Mega - The brains of the device.
2. R-606 Vintage Drum Synthesizer chip - The voice of the device. This chip takes trigger and volume inputs on different pins, and outputs an audio signal. It emulates the Roland TR-606 This awesome chip and more are made in Sweden by DSP Synths.
3. 1" Steel Ball Bearings
4. White LEDs - These illuminate the current step.
5. 1N4148 Signal Diodes - To solve the backflow problem.
6. Potentiometers - To serve as volume controlls for each instrument.
7. Hookup wire
8. Assorted resistors, capacitors, and jacks - To build the output circuit for the voice chip, condition power to the LEDs, and act as pulldown resistors (described later).
9. Plywood
10. Screws (2/switch), Nuts (6/switch), and Washers (2/switch)

CONSTRUCTION

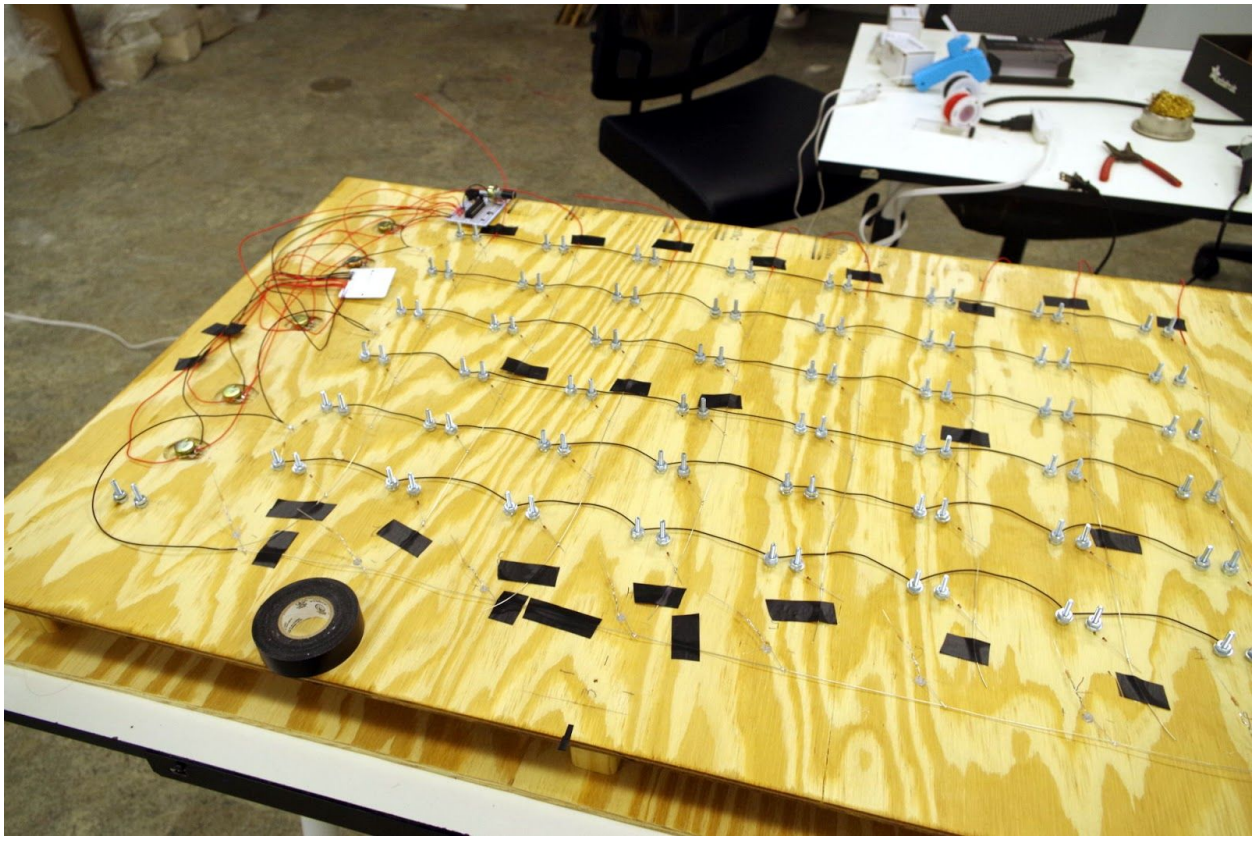
The physical construction proved to be more involved than I first thought. The process began with making the structural parts of the sequencer, cutting down plywood and other wood parts to create it.

After that, I prepared the front face of the board by sanding and staining it. Then came drilling all the holes for the screws and using a router to cut out recesses for the potentiometers to sit in.

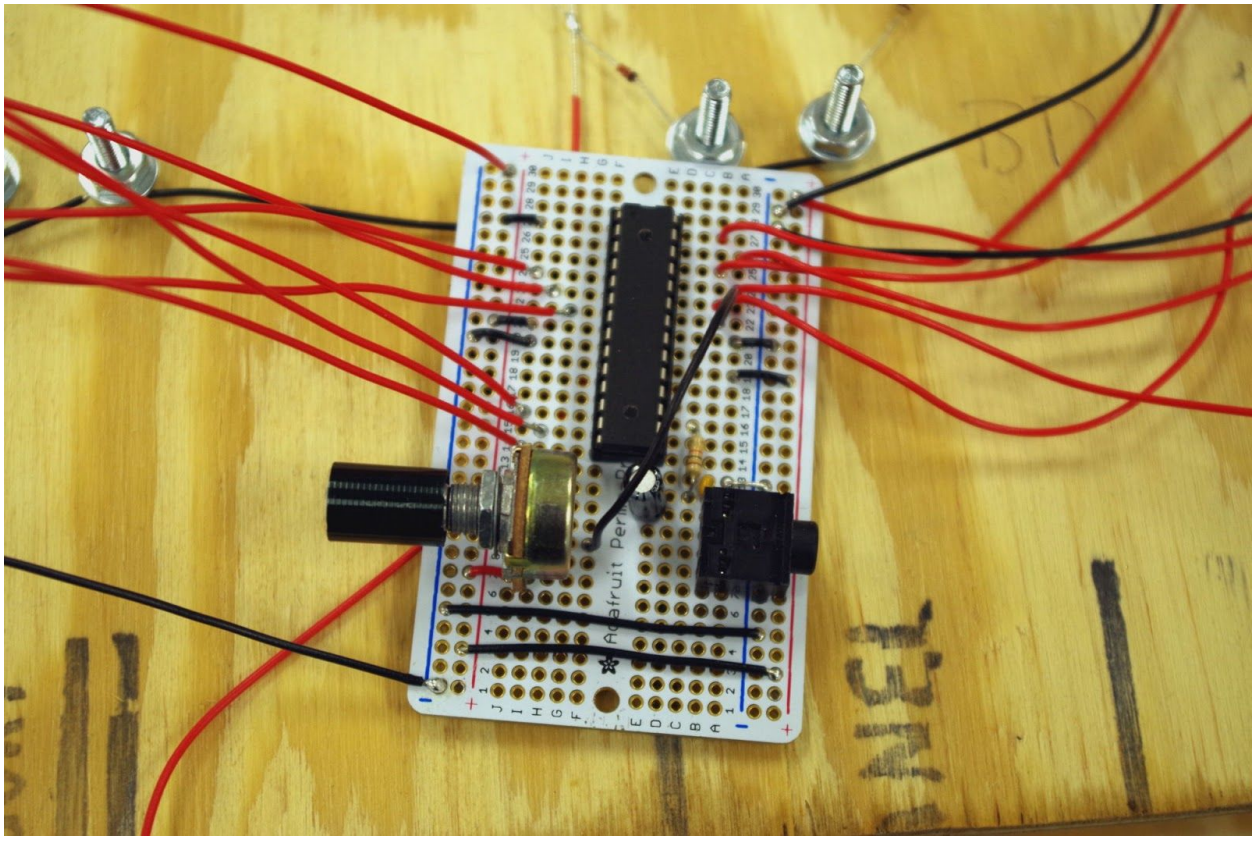
After I got all the screws into the holes I drilled, but before I started with wiring, I made up a quick minimal viable product test, using a simple Arduino sketch and a single pair of screws with a ball.

It was here I encountered my second major bug.

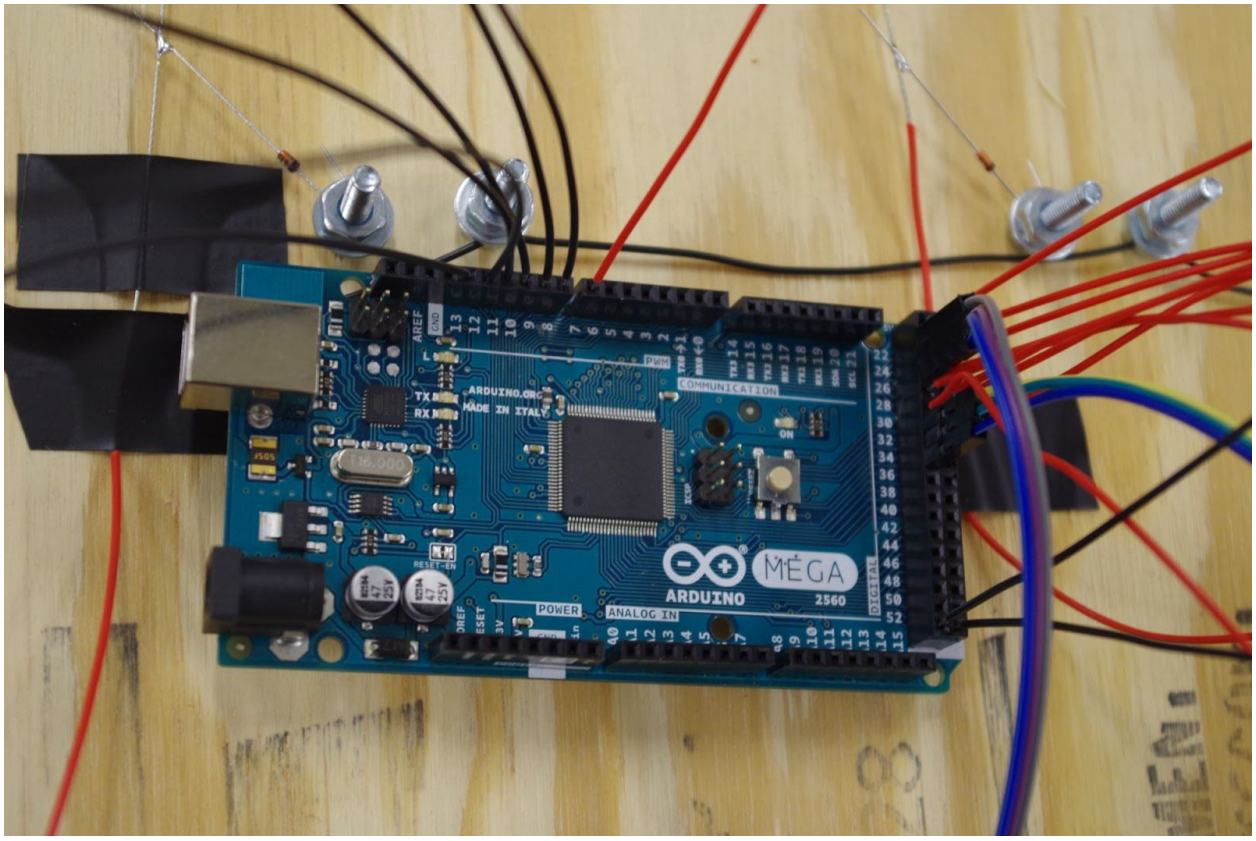
When the ball was in place, the sensor returned "true". When the ball was not in place, the sensor returned... a seemingly random and always changing steam of values. This was my first encounter with the "floating pin" problem, where an Arduino reading a value off a pin that isn't tied to voltage or ground picks up huge amounts of interference, and can return any value, unpredictably. To remedy this problem, a "Pullup" or "Pulldown" resistor is used. This is a high-resistance resistor tied to the pin and to voltage (Pullup) or ground (Pulldown), in order to give the pin a "default" state. Because of the resistance, the default value is overridden by any other inputs.



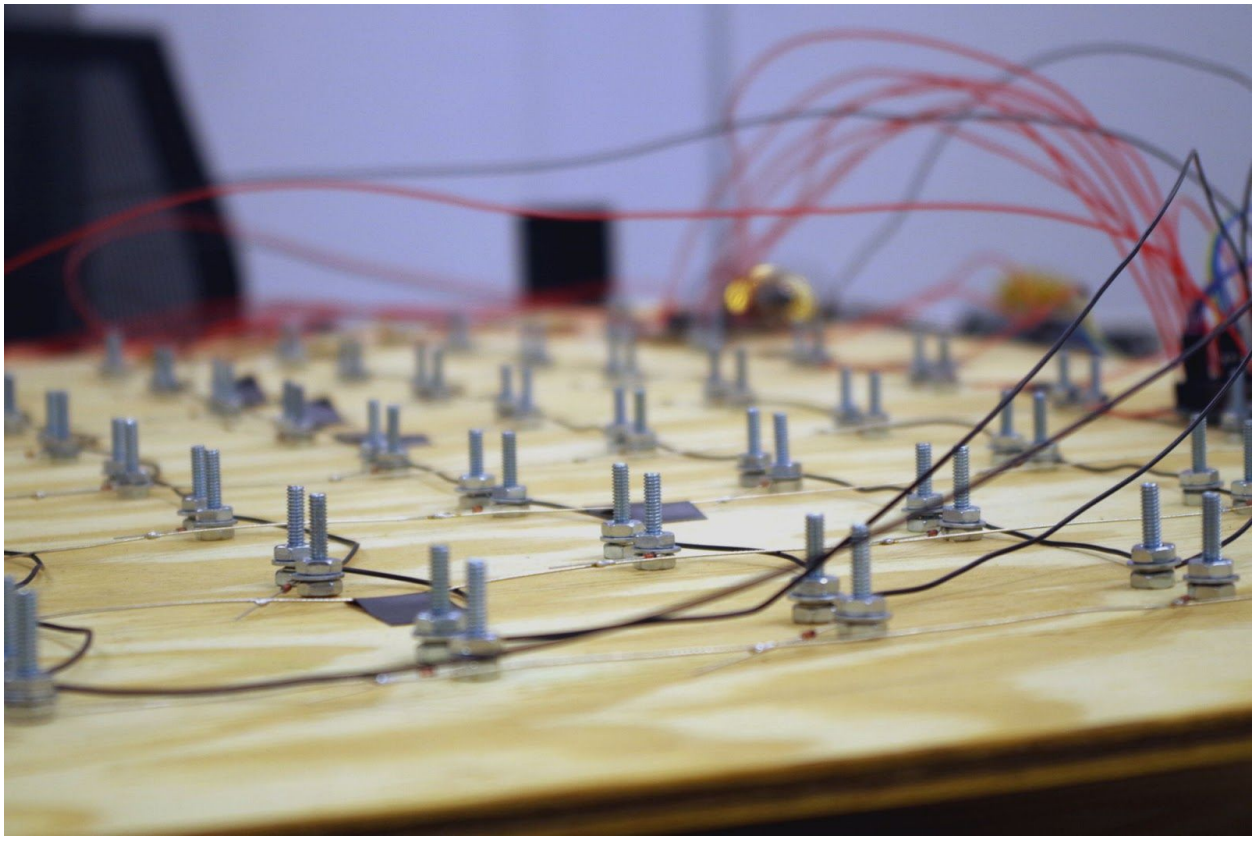
On the subject of wiring, there were two design choices that saved me a significant amount of work. The first was the screw and nut setup, so instead of making solder joints, I just clamped down the bare ends of a wire between two nuts. The second choice was that for the columns, instead of having many short wire sections, I stripped the entire wire and just soldered to that, like a rail.



The R-606 Voice Chip



The Arduino Mega



Wiring Detail

RESULTS

After constructing and wiring the sequencer, I uploaded the software to the Arduino, ironed out some quirks (the sequence ran backwards at first? An easy software fix though), and plugged it in, and it worked perfectly.

The project was unveiled at the class final, and seemed to make an excellent impression. As I talked about the challenges and ideas I'd had throughout the process, people played with it, and seemed to really enjoy it.

GOING FORWARD

My next steps with this project will be:

1. Use a wood burning tool to add labels - I've always liked the look of burnt wood.

2. Increase the base tempo - people seemed to enjoy making beats in double time a lot more than at 1 hit/beat, so I'll increase the base tempo.
3. Record a video of someone playing along to it in action.

ARDUINO CODE

```
/*
The original project and code was created by
David Schofield @ davidjschofield.com
You are welcome to adapt, modify, and improve upon this work
as you wish as long as it is for non-commercial purposes.
Attributions and shout-outs are always appreciated but not neccecary.
If you have any questions or are using this to do something cool,
let me know! I'd love to see your project.
*/
//Variables (editable)
const int totalsteps = 8; //How many steps you want
const int BPM = 140; //The BPM you want the beats to run at
const float duty = .30; //The duty cycle for hitting the beat. A
decimal between 0 and 1, inclusive.
const int instrumentnumber = 6; // The number of instruments you want
to trigger

//Variables (automatic, probably shouldn't touch.)
float beatdelay;
int stepquant = totalsteps - 1;
int stepnum = 0;
int instquant = instrumentnumber - 1;

//Pin Setup
//Step pins turn on an LED for the current step
//Note: Step pins were not used in the final version. Instead, wire
LEDs to the top of each column. These pins are still usable but not
neccecary.
int stepPins[] = {45, 46, 47, 48, 49, 50, 51, 52, 53};
//Column pins send power to the current column. The number of pins
should equal the number of steps.
int columnPins[] = {36, 34, 32, 30, 28, 26, 24, 22};
//Row pins read the values for the given columns. The number of pins
```

```
should equal the number of instruments below.
int rowPins[] = {13, 12, 11, 10, 9, 8};
//Hit pins send the trigger signal to the drum generator
int hitPins[] = {23, 25, 27, 29, 31, 33};
//Doubletime pin. This is a pullup pin, so the switch should connect
to ground.
int doubletime = 7;

// the setup routine runs once when you press reset:
void setup() {
  //debugging
  Serial.begin(9600);

  //LED outs
  for (int i=0; i <= stepquant; i++){
      pinMode(stepPins[i], OUTPUT);
  }
  //Column Outputs
  for (int i=0; i <= stepquant; i++){
      pinMode(columnPins[i], OUTPUT);
  }
  //Row Inputs
  for (int i=0; i <= instquant; i++){
      pinMode(rowPins[i], INPUT);
  }
  //hit pins
  for (int i=0; i <= instquant; i++){
      pinMode(hitPins[i], OUTPUT);
  }
  //doubletime pin
  pinMode(doubletime, INPUT_PULLUP);

  //debugging
  Serial.println("Setup Complete.");
  delay(2000);

}

// the loop routine runs over and over again forever:
// This counts steps from 0 - stepquant, or totalsteps -1
void loop() {
  //Set the tempo at the top of each bar
  if (stepnum == 0) {
```

```
  beatdelay = 30000 / BPM;
  if (digitalRead(doubletime) == HIGH) {
    beatdelay = beatdelay * 2;
  }
}

//Debugging
Serial.print("step number");
Serial.println(stepnum);

//Turn on the LED for the step we're on
digitalWrite(stepPins[stepnum], HIGH);

//Activate the column we're on
digitalWrite(columnPins[stepnum], HIGH);

delay(5);//Output settling

//cycle through the rows and hit the instrument beat!
for (int i=0; i <= instquant; i++){
     digitalWrite(hitPins[i], digitalRead(rowPins[i]));
}


// Delay the on part of the duty cycle
delay(beatdelay * duty);

//turn off all the beat pins to prepare
for (int i=0; i <= instquant; i++){
    digitalWrite(hitPins[i], LOW);
}

//delay the off part of the duty cycle
delay(beatdelay * (1 - duty));

//Turn off the current LED
digitalWrite(stepPins[stepnum], LOW);

//Turn off the current column
digitalWrite(columnPins[stepnum], LOW);

//Move to the next step to prepare for the next beat
stepnum++;
```

```
  //If the number of steps exceeds the proper amount, start the bar
over
  if (stepnum > stepquant) {
    stepnum = 0;
  }

}
```