Phillip Anzalone (pra6@columbia.edu)
Sean Ballinger (sbb2151@columbia.edu)

December 14, 2012

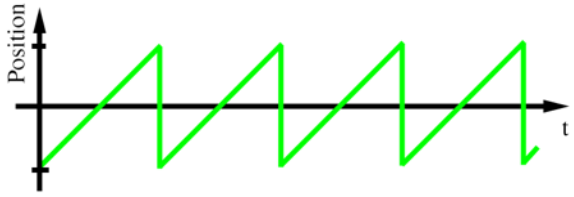
PONG

Abstract

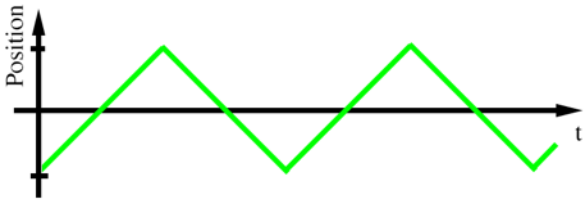
The goal of this project was to create the game of Pong from an array of LEDs, using potentiometers to control the paddles and an Arduino to guide the motion of the ball. The code would also be responsible for detecting the position of the paddles and their contact with the ball. We decided to take two approaches in implementing this project. The first involved making the ball bounce around the array of LEDs, and the second was to make it bounce around the graticule of an oscilloscope by both analog and digital means. For the oscilloscope version, we first used two function generators, and then replaced them with Arduinos and low-pass filters.

Oscilloscope and function generator

Professor Vallancourt proposed that we use function generators to simulate a ball bouncing around. The theory behind this approach is interesting. With an input of constant voltage in channel 1 of the oscilloscope (which dictates movement in the X direction), the trace behaves as follows:

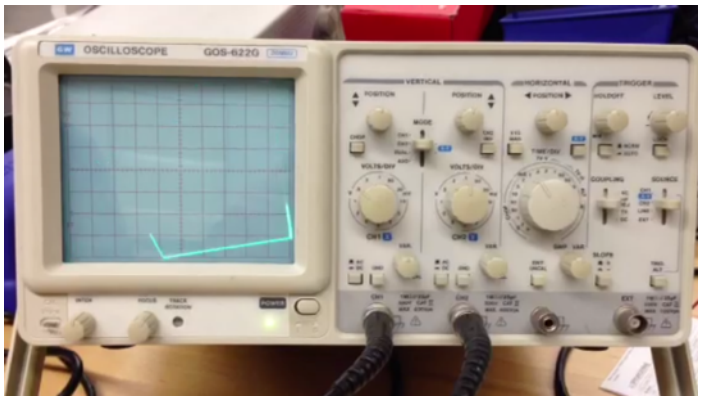


In the above diagram, the slope of the line is analogous to the speed of the ball. If we want to simulate elastic collisions with the walls, we need to make sure that the direction of the ball reverses upon reaching the end of the frame. The velocity must be reversed, which means that the graph of the position should look like this:



The above graph represents a sawtooth wave. If we add another sawtooth wave to channel 2 of the oscilloscope, we're setting up the same rules for the Y direction, and we can quantify the motion of a completely elastic ball colliding with the walls of a box. We can even observe this motion using the X-Y display mode of the oscilloscope.

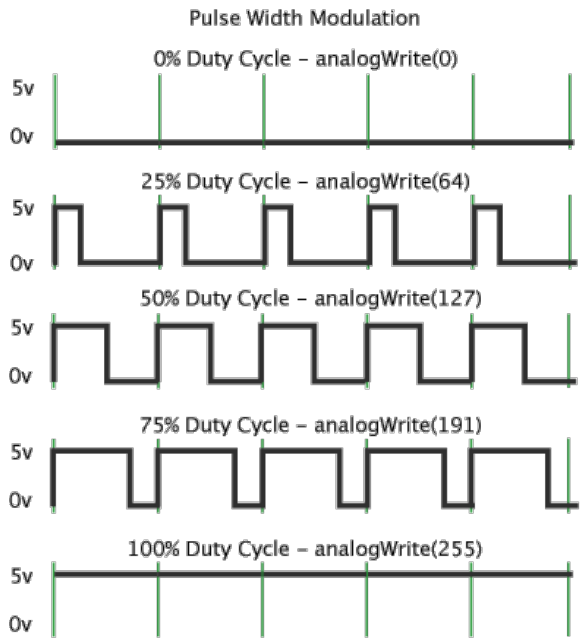
Fortunately, we are able to create a sawtooth wave and control its frequency and amplitude using a function generator (FG). We connected FG1 to channel 1 on the oscilloscope and FG2 to channel 2. We set FG1 to produce a sawtooth waveform of 9 Hz and FG2 to produce a sawtooth waveform of 6 Hz. These settings are particular to the measurement scale used on the oscilloscope, which was 0.5 V/div on Ch1 and 0.1 V/div on Ch2, and with Ch1 on DC mode and Ch2 on AC mode. We activated the X-Y mode and set the X-Y source to Ch2. By varying the amplitude on FG1, we were able to scale the range of movement of the trace in the X direction, and FG2 allowed us to set the scale in the Y direction. We were able to control the motion of the ball by varying the frequency of the sawtooth wave's output from the function generators, and we observed that when one frequency was close to being a multiple of the other, the motion became periodic.



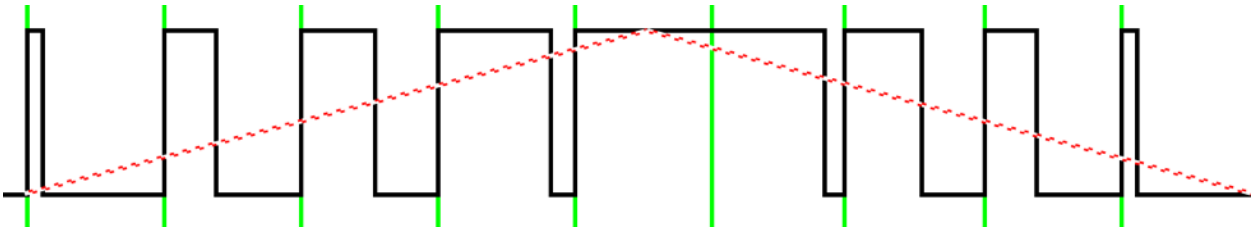
A video of the motion of the ball can be viewed at http://www.columbia.edu/~sbb2151/fg-osc.mov

Oscilloscope and Arduino with low-pass filter

Our next goal was to recreate the sawtooth waves using Arduinos. By default, these do not offer fine-grained control of different waveforms. They are capable of outputting square waveforms alternating between 0 V and 5 V by a process called Pulse Width Modulation (PWM). This allows us only to change the duty cycle: 0% duty cycle outputs a constant 0 V, and 100% outputs a constant 5 V. We discovered that pins 12 and 13 did not offer control over the duty cycle, and we saw better results after switching to pins 9 and 10.



The figure above, taken from the Arduino documentation, shows the control offered by PWM.

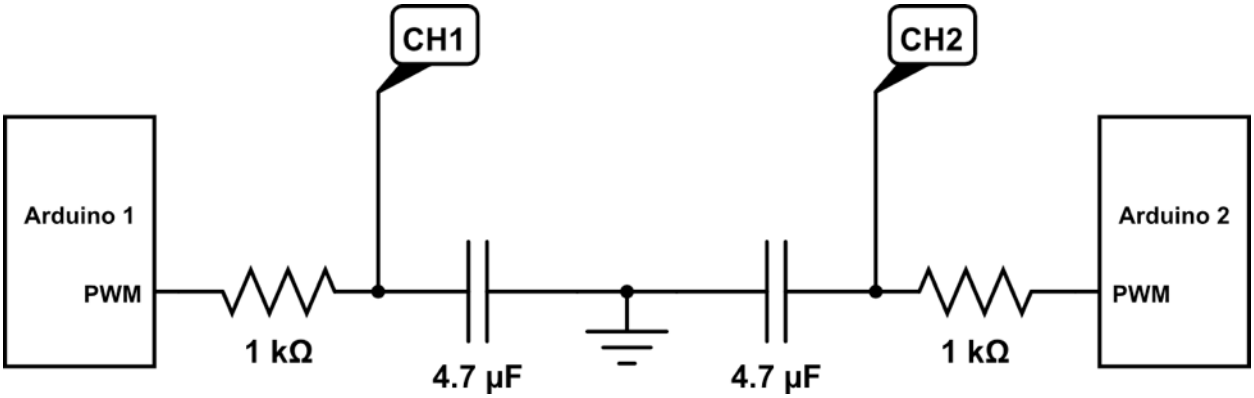


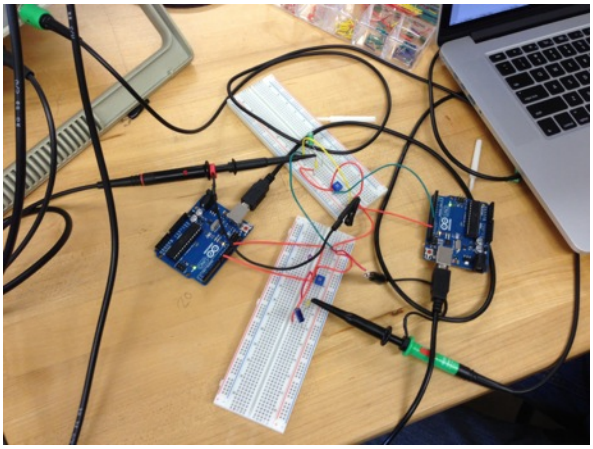
The figure above illustrates the effect we exploited: the average voltage of the PWM pin on the Arduino is a function of the duty cycle, and when it varies linearly, we can produce a sawtooth wave. To achieve maximum resolution, we created a loop that went through all 255 increments, but its PWM origins are plainly visible when viewing the final result in X-Y mode on the oscilloscope.

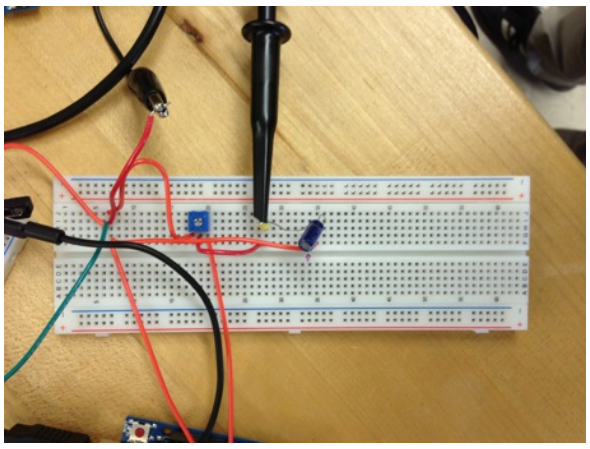
By sending the PWM signal through a low-pass filter with a very small corner frequency, we limited the output to the first Fourier series coefficient, which corresponds to the average value of the signal.

$$\begin{aligned} f_c &= \frac{1}{2\pi RC} \\ &= \frac{1}{2\pi(1\,\mathrm{k\Omega})(470\,\mu\mathrm{F})} \\ &= 0.339\,\mathrm{Hz} \end{aligned}$$

We set up the Arduinos to output to channels 1 and 2 of the oscilloscope as shown below:







We created a program that went through a loop to increase the duty cycle by one increment until it reached the maximum value of 255, and then went back down to 0. This means that we were linearly ramping the duty cycle between 0% and 100%. PWM requires the "delay" command, which has the side effect of incapacitating the entire Arduino on run, and we needed to control the individual periods of two sawtooth waves, so we needed to use two Arduinos. We set one Arduino to a period of 500 µs (2 kHz), and the other 340 µs (3 kHz), in order to achieve the bounc-

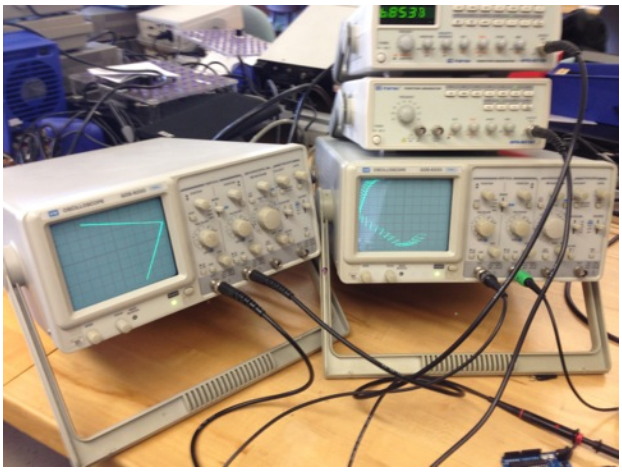
ing effect. We wrote the following program, and ran it with the different values for `delayMicroseconds` on two Arduinos.

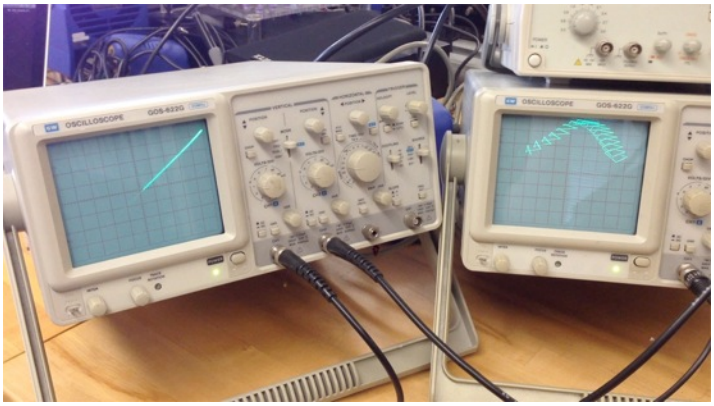
```
int xpin = 9;
int i;

void setup() {
	pinMode(xpin, OUTPUT);
}

void loop() {
	for (i = 0; i < 255; i = i + 1) { // ascent
		analogWrite(xpin, i);
		delayMicroseconds(170); // set half-period value in microseconds
	}
	for (i = 255; i > 0; i = i - 1) { // descent
		analogWrite(xpin, i);
		delayMicroseconds(170); // set half-period value in microseconds
	}
}
```

The images below show the difference between the function generator waveforms (left) and the Arduino generated waveforms (right).

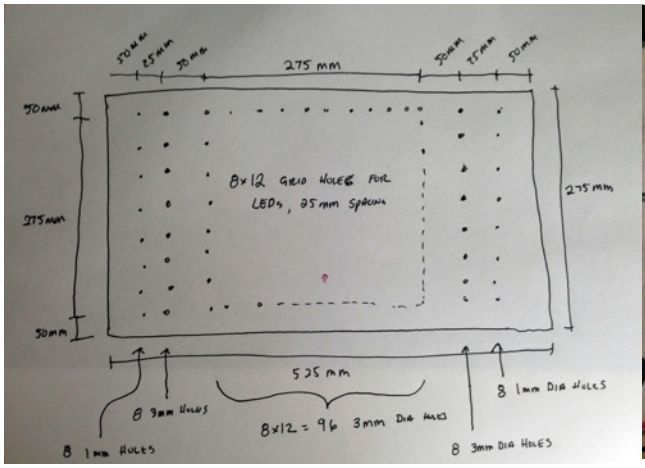


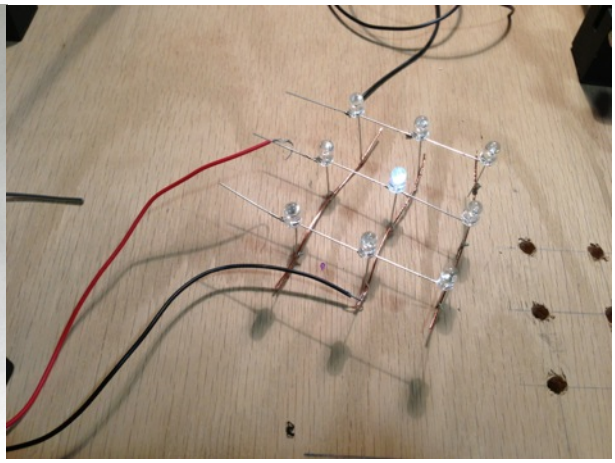


The function generator creates a single smooth trace while the Arduinos create a trace that follows a similar overall trajectory as the analog version, but is reveals the PWM used to crete the average-voltage sawtooth wave.
A video of the two side-by-side: http://www.columbia.edu/~sbb2151/ard-osc.mov

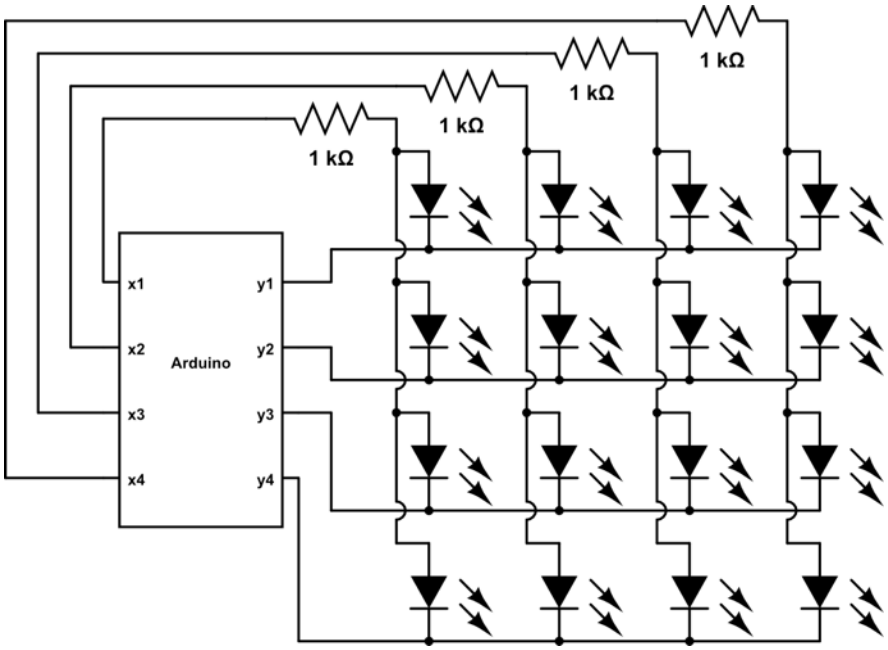
Pong with an array of LEDs

In trying to create a fully digital version of the Pong game, we started the construction of the LED playing field consisting of 96 white 5mm LEDs (8 rows by 12 columns), with an adjacent column of 8 LEDs on either side to represent the paddles for the game. The LEDs were connected by soldering all positive terminals together in a column, and each negative terminal across a row. This allowed us to light a single LED by completing the circuit with a positive lead to one of the columns, and a ground to one of the rows, producing current in only one LED at a time while reducing the number of leads needed to attach to the Arduino to 20 total.



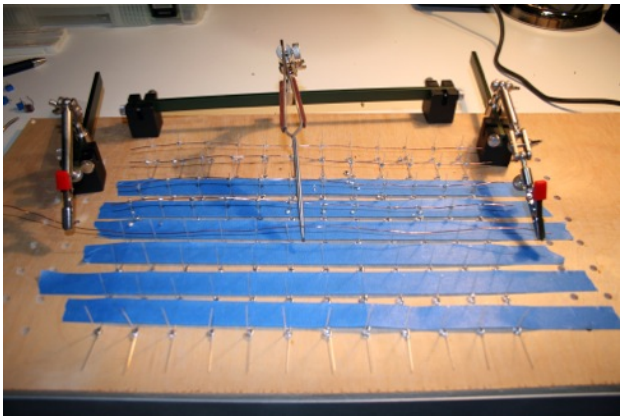


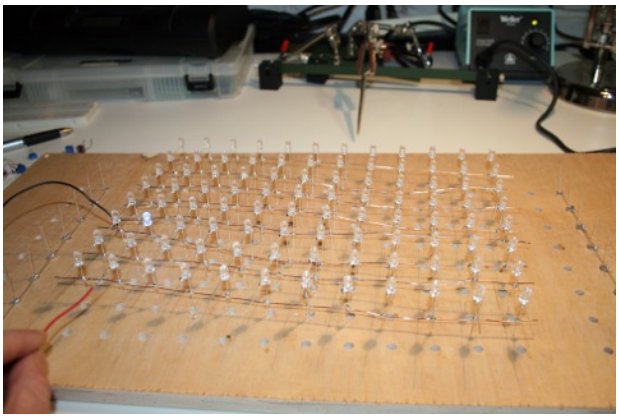
The schematic diagram for a 4x4 portion of the LED field is shown below:



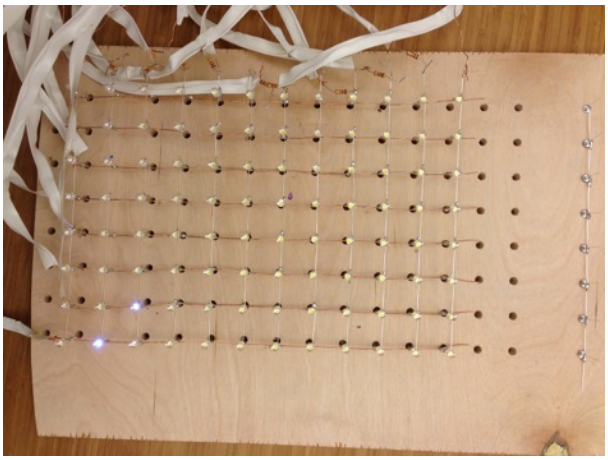
The positive terminals of the LEDs were bent at a 90 degree angle and the LEDs were taped down to a pre-drilled jig with 25 mm hole spacing in the X and Y directions. Each positive terminal was soldered in series to the next positive terminal while careful not to overheat the adjacent LED (some LEDs were discovered as failing to light after being soldered, and overheating might be the cause). Then a bare copper wire was run 90 degrees to the positive terminals and soldered to the negative terminals alone in series, which were still standing upright, allowing spacing such that the negative and positive terminals did not short out.

The construction of the board is diagrammed in the photos below. The resulting mesh of soldered LEDs and wires was sufficiently rigid to be extracted as one mass from the jig and connected to a power source for testing. Each LED was tested extensively: once a column of positive terminal connections was complete, once each row of negative terminal connections was complete and each LED tested once the mesh was complete.





Once constructed, the LED Board was connected to the Arduino by connecting the 12 positive terminal columns each to a digital pin, along with an inline 1 kΩ resistor, and the 8 negative terminal rows each to a digital pin. As a result of Lab closing, we were forced to use bare copper cable insulated with electrical tape to prevent shorting of the LEDs. For the complete project we would solder the resistors and insulated wire leads to the LED mesh and glue this assembly to a acrylic board made with the exact hole spacing as the jig, allowing for cleaner connections to the Arduino.



Programming the Arduino to control ball motion

Because the Arduino only has one ground pin, and does not offer control over it, we had to implement a different sort of ground. Each column (x coordinate) of LEDs that was not meant to fire would be left at "low" output, while the one containing the correct LED would be set to "high". The rows (y coordinates), on the other hand, were all set to "high" for the inactive ones, and "low" for the active one, creating a potential difference only where it was needed.

To create the prototype of the program would use to control the motion of the ball, we used the ncurses library to write a program in C that would make a field of "." characters and show the motion of a ball represented by a "0". Once the core was complete, we modified it to work with the particulars of the Arduino environment. The final version of the code follows:

```
int x1 = 22, x2 = 23, x3 = 24, x4 = 25, x5 = 26, x6 = 27, x7 = 28, x8 = 29, x9 = 30, x10
= 31, x11 = 32, x12 = 33;

y1 = 42, y2 = 43, y3 = 44, y4 = 45, y5 = 46, y6 = 47, y7 = 48, y8 = 49;

int x[12] = {x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12};
int y[8] = {y1, y2, y3, y4, y5, y6, y7, y8};

int i;
int dy = 1, dx = 1; // direction of the ball
int xp = 0, yp = 0; // position of the ball
int bx = 12, by = 7; // dimensions of the board

void setup() {
  for(i = 0; i < bx; i++) {
    pinMode(x[i], OUTPUT); // initialize all x (column) pins
  }
  for(i = 0; i < by; i++) {
    pinMode(y[i], OUTPUT); // initialize all y (row) pins
  }
}

void loop() {
              if (xp + 1 >= bx) { // backward x direction if about to hit right wall
                     dx = -1;
              }
              else if (yp + 1 >= by) { // downward y direction if about to hit ceiling
                     dy = -1;
              }
              else if (xp <= 0){ // forward x direction if about to hit left wall
                     dx = 1;
              }
              else if (yp <= 0) { // up y direction if about to hit floor
                     dy = 1;
              }
              xp = xp + dx; // update x position
              yp = yp + dy; // update y position
              for (i = 0; i < bx; i++) { // update all column pin outputs
                     digitalWrite(x[i], LOW);
```

```
                digitalWrite(x[xp], HIGH);
        }
        for (i = 0; i < by; i++) { // update all row pin outputs
                digitalWrite(y[i], HIGH);
                digitalWrite(y[yp], LOW);
        }
        delay(40);
}
```

Changing the `delay()` value allowed us to control the speed of the ball, and 40ms produced very fast motion. At less than 10 ms it became difficult to discern motion at all.
A video of the LED board is available at http://www.columbia.edu/~sbb2151/led.mov

Once complete, we experimented with the code to produce sequences that lit up the entire board one row at a time, that lit all LEDs at once, and that ran the Pong bouncing ball at different speeds and within different arrays of the LED field.

Future work

The goal of a complete Pong game was unfinished - we intended to connect the potentiometer movement of the paddle, and the circuitry to inform the Arduino when a paddle is at a specific location such that the code can determine if a ball is to bounce back upon being hit or if the game is over because the paddle missed the ball (i.e. the paddle was not in the same row as the ball at the time the ball reached the last column in the field). Due to time constraints we were unable to incorporate this aspect of the project, but were able to prototype illuminating four different LEDs with a single potentiometer.

We tested the range of the potentiometer by sampling output with the Arduino, and found it to range from 0 to 1023. By assigning output to light the LEDs to specific pins and to allow each port a range of function within the 1024 values (i.e 0 to 255 lights LED #1, 256 to 511 lights LED#2, and so on), we created a paddle controlled by a potentiometer. The paddle can be scaled up to 8 LEDs and we could insert code into the program to check is the paddle being lit up is in the same row as the ball on the playing field when the ball reaches the appropriate end columns. If the paddle is in the same row as ball, the ball would be made to bounce back into the field, and if the paddle is not in the same row, then the game is over.