Anshul Aggarwal
Martin Kong
EE 3 Fall ‘16
Lab 1F

The Game of Snake

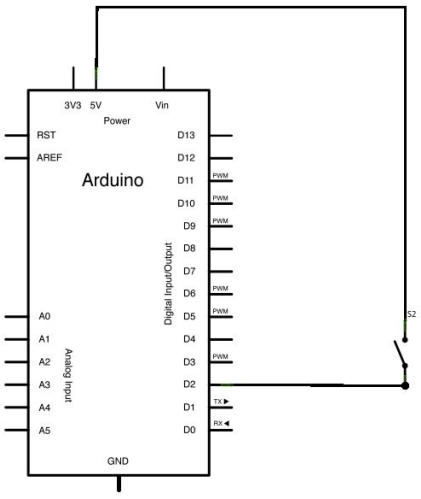
Introduction & Background

Our goal was to emulate a video game we played often as kids, Snake. Snake is a video game where you play as a hungry snake in search of more and more apples to feed on as you grow longer and longer. The player must direct the snake, taking care not to smash into the boundaries of the arena or into themselves, in their quest for more food. To implement this, we needed a matrix array of 8x8 LEDs as well as 2 buttons to control left and right movement, all of which we connected to an Arduino with wires and a breadboard.

To connect to the Arduino we used a direct implementation. This means that for every pin that needed driving we connected it by wire to a single pin on the microcontroller. We hooked up the two buttons (actually microswitches) directly from the Arduino’s grounded pin to the Analog 0 and 1 inputs. We then used code to create a display function that would quickly scan through each of the rows to turn on specific LED’s, so only the ones we wanted to be on turned on. The code for the game itself utilized an object-oriented focus that we first wrote in C++ then edited so that it would compatible with the Arduino. We combined this with the display function we created, and finally, we could to play the game.

Testing Methodology & Problems

Once we had everything connected to the Arduino, our biggest challenge was getting the buttons to work properly. Our buttons were really microswitches with Normally Open (NO) and Normally Closed (NC) terminals and a COM terminal[4]. We hooked up the COM terminals to the 5V and 3.3V outputs, and the NO terminals to Analog pins 0 and 1. The theory behind this was that we would have constant output going into the Analog pins that we could use for movement; if the switch was pressed it would complete the circuit so the Arduino would read a positive voltage, else it would read a negative one. However, when we started the buttons would only work half the time, then either stop working or fluctuate between output values. This broke our code as we could not accurately read in our inputs and thus could not play the game. The schematic of the old version that is incorrect is below [3i]:



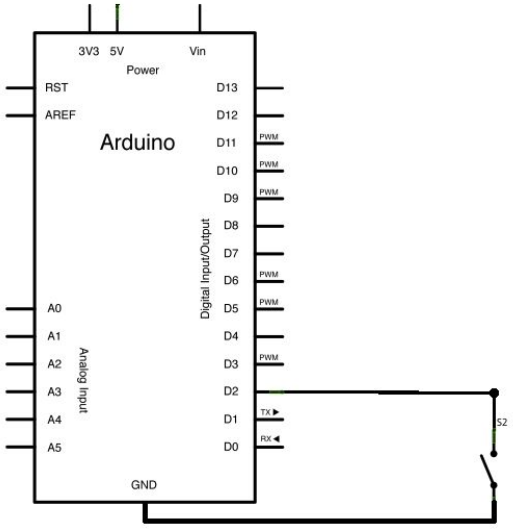
How We Designed The Test: The problem we were facing was an issue of data inputs. First, we knew that we needed to collect data that we were getting from the buttons. We set up the arduino code to print out the data being received from the pins to the output, using Arduino's Digital Read Serial. We isolated the wires to make sure there was no interference.

How We Conducted The Test: We allowed the Arduino to run a continuous loop where it checked the Sensor input and outputted the results to the screen. To start, we let the loop run by itself for few seconds. During this time the program printed a steady stream of zeros, which we used as the control for our experiment. Then as the loop ran, we would click one button, record the results, and wait; click the other button, record the results, and wait; then finally click both buttons and record the results. We repeated this methodology a few times to build a better image of what was happening with the switch.

How We Analyzed the Data: Ideally, each of our button presses would result in a single output of 1, which would briefly interrupt the flow of zeros. However we immediately noticed that when a button was pressed, the 1 signal would be sent for a much longer time than we had initially wanted. Even if the button had been pressed for a very very short time, the 1 signal would be lasting for multiple signals. Under some strange cases, we would get numbers other than 0 or 1 such as 14 or 15. From this we knew the problem must have been an issue with the output stability, as the output was fluctuating rather rapidly before holding steadily.

How We Interpreted The Data: After some research we discovered that this problem was due to a principle called contact bouncing [5], when metal parts of the switch making contact with each other can bounce apart before settling on an output state. Because of this, the button would appear to work occasionally then break down or no longer accept the data. Initially, to fix this we attempted to keep track of the time since the last button press, in order to obfuscate the seemingly repeated inputs as a singular one. However, this proved to still be an issue as the time was relatively variable, and pretty long so we could not accurately gauge the button. In the end we decided to switch up the circuit by connecting the microswitches' COM terminal to the ground pin GRD of the arduino, leaving the NO terminals connected to the analog input pins. This allowed us to make use of an INPUT_PULLUP mode for the arduino pins, meaning that the

pins would all initially be pulled up a voltage of 5 volts, determined by their internal resistance. Then when the switch was pressed and the circuit was completed, the voltage would drop to zero briefly because of the ground connection before going back to 5 volts. This new approach ended up giving us what we were looking for. We could now successfully obtain readings from the buttons that were both accurate and immediate, and successive button presses could be correctly read as intended. A rough schematic of our new setup is shown below [3i]:

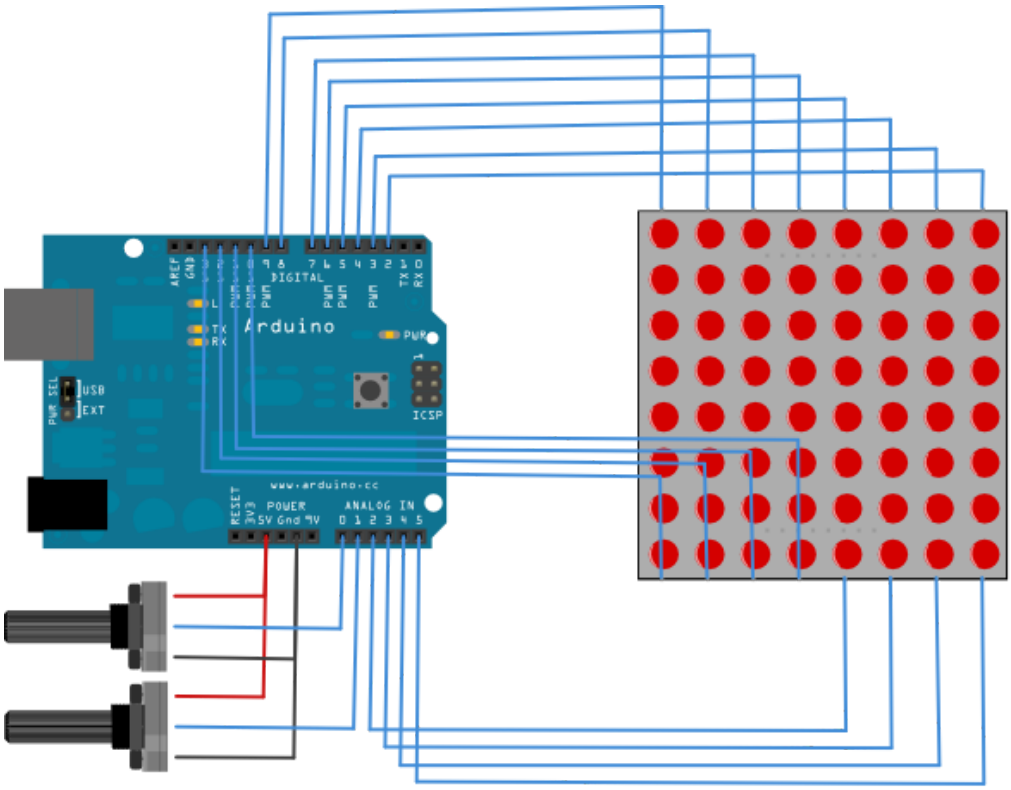


As you can see in the new circuit, we are able to utilize the internal resistance of the Arduino connected to the ground terminal to be able to more accurately see when the button is hit.

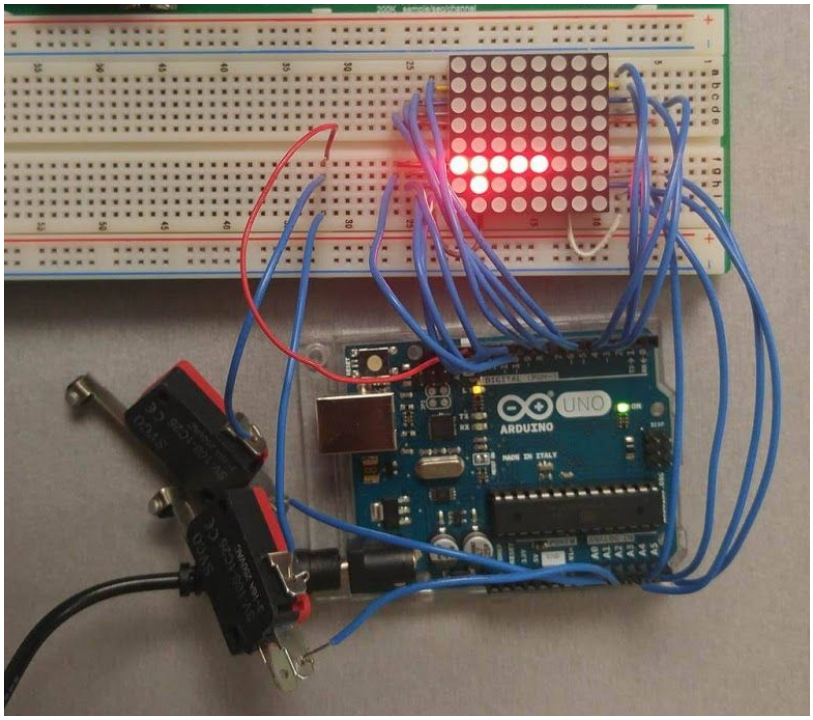
Results & Discussion

Our interaction with the data has been discussed in the previous portion. We did not need to use lab tools to analyze our circuit as we were able to solve our issues through code and through changing our implementation. Below we have attached an image of our final design:

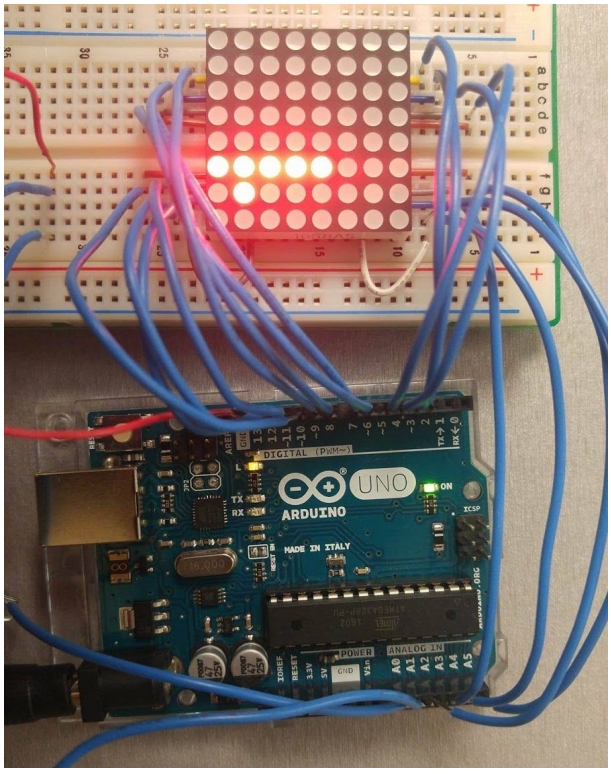
Schematic with Arduino and LED Matrix [1i]:



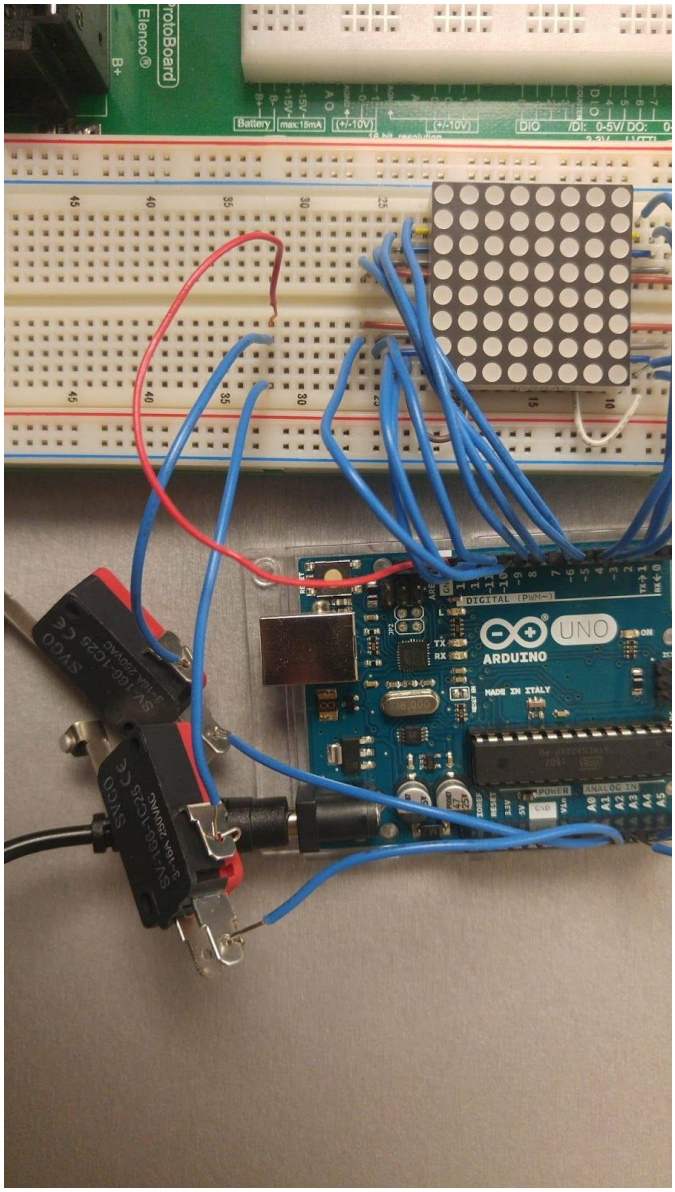
Overall Circuit:



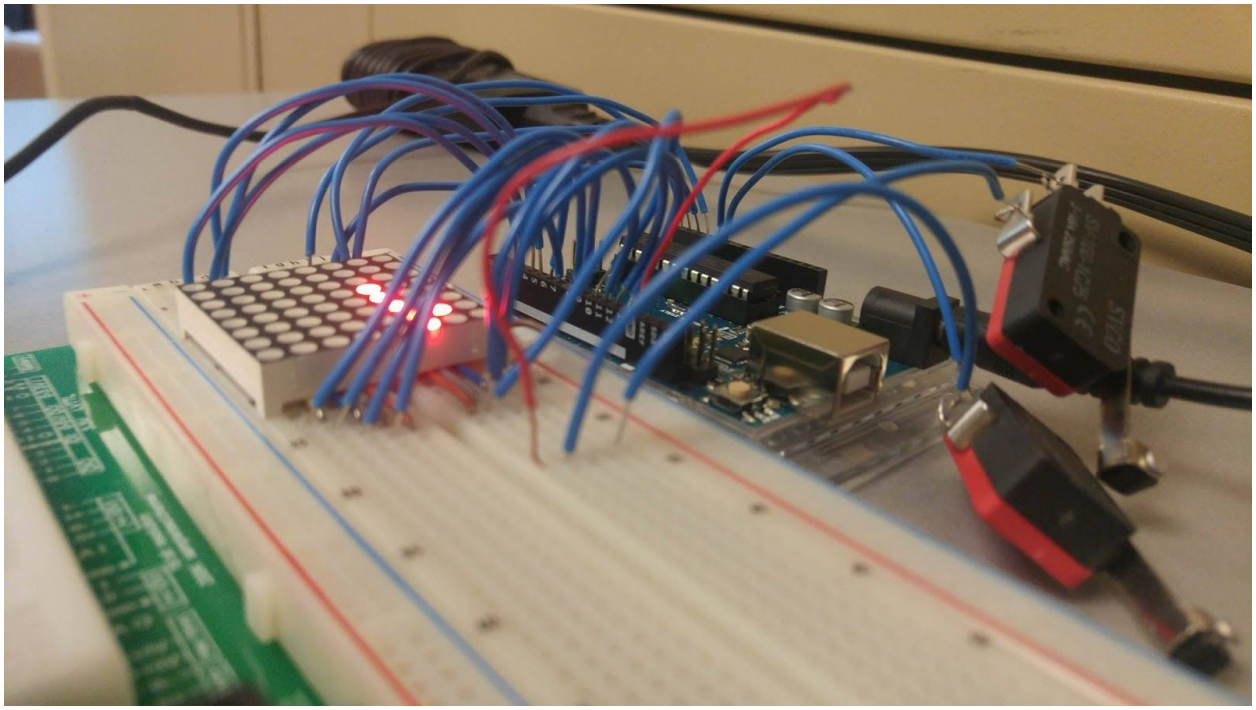
LED Matrix and Pins



Button Implementation

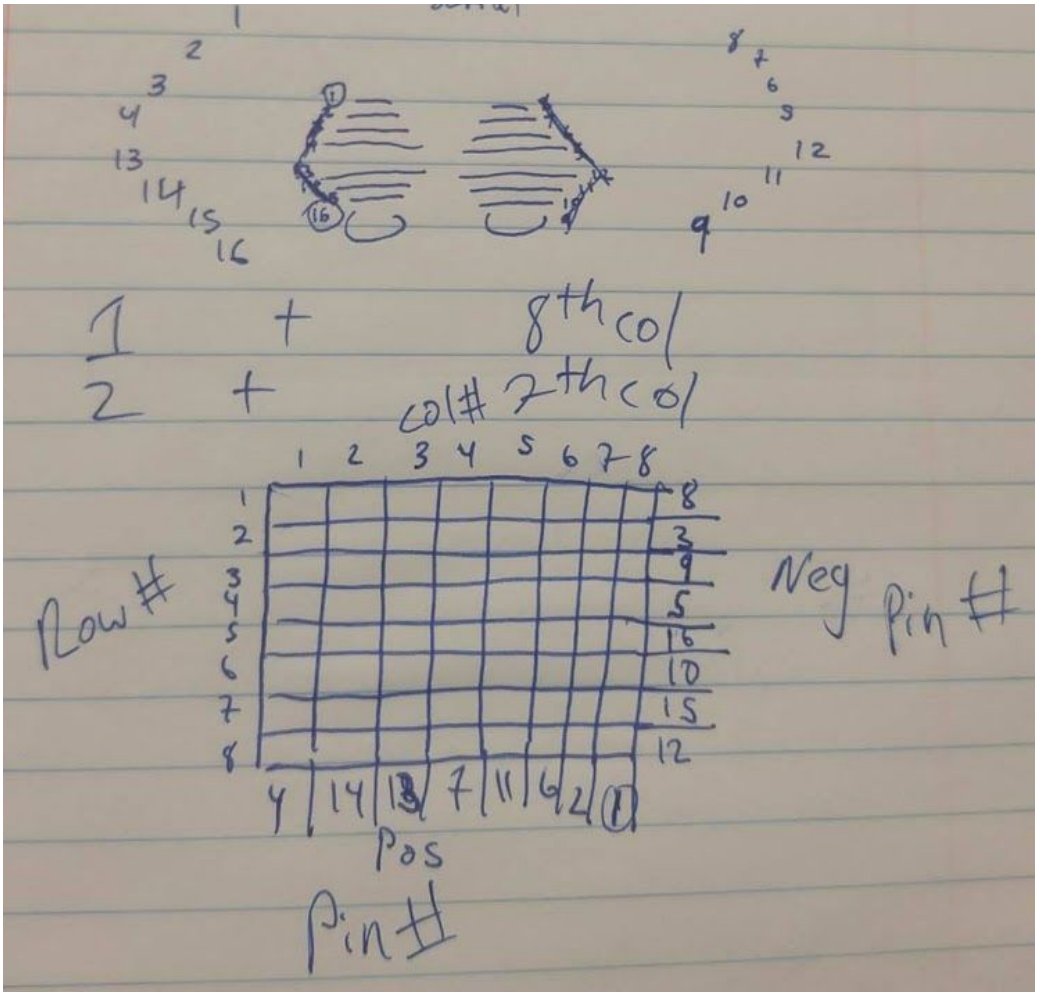

Elenco®
Battery
max 15mA
(+/-10V)
DIO
DIGITAL (PWM~)
UNO
ARDUINO
MADE IN ITALY
POWER
ANALOG IN

Side View



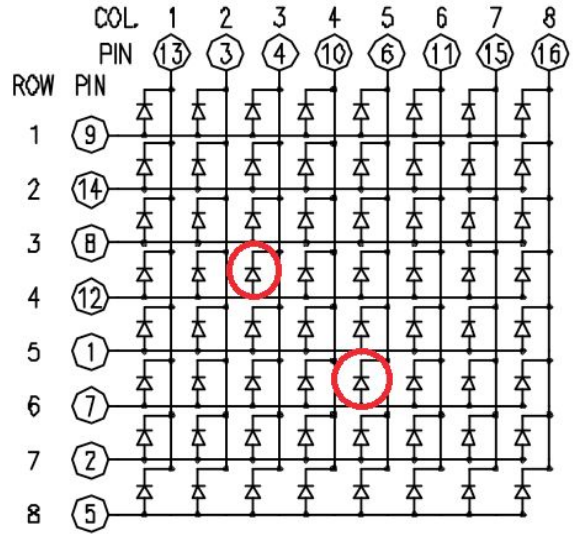
In addition to the button problems we had in the testing methodology section, we faced issues with determining which of the 16 pins on the LED matrix controlled which row or column. To figure this out, we took a digital multimeter and set it to diode test mode, then used the outputs to test each pin to see what column or row it would drive.We would hold the positive terminal to a singular pin, then sweep across each pin with the negative terminal and observe which rows and columns would turn on. If none did, we would switch the terminals and try again. As we did this with each pin, we were able to form a picture of the schematic of the matrix for which pins related to which columns or rows.

From our data, we created a chart like this one:

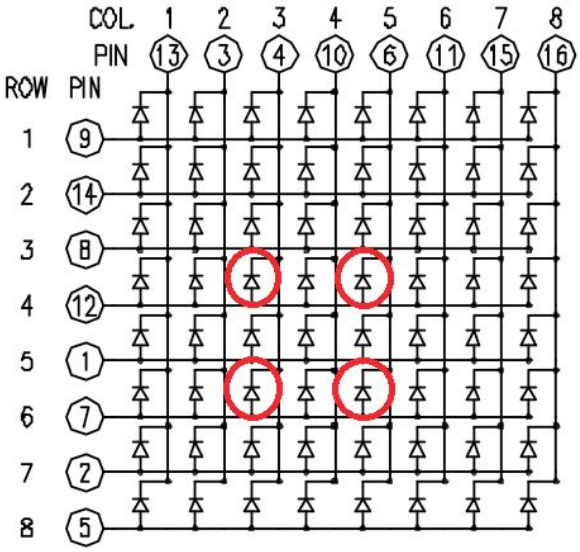


The top portion defines the pin number locations of the LED matrix, and the bottom defines the relationships between row/col and pin. With this information, we could accurately utilize the arduino to drive our matrix properly. We put wires from the matrix pins into the arduino in the proper order, using up 12 digital and 4 analog pins on the Arduino. Additionally, we found that to turn a specific light on, the corresponding column pin had to be set to HIGH and the corresponding row pin had to be set to LOW. We could now use the results of our analysis to drive the matrix.

Another challenge was displaying pixels independently. Though this may seem like an easy task at face value, it is much harder than initially anticipated. What we want is something that looks like this [2i]:



However, we were consistently getting something that looks like this [2i]:



The reason for this is that since the matrix has 16 pins, each associated with a row or column, it is very easy to turn a single LED on. If a row and 2 columns are to be turned on for 2 pixels, 2 LED's in that row will be turned on. However, if 2 rows and 2 columns are turned on, then 4 pixels will be turned on, rather than the anticipated 2. This causes a problem with the display as we see many more LED's than we had intended. After researching a bit, we ended up using a scanning approach that is used in displays today. Within our overall loop, we had an inner loop that scanned through every column very fast as we changed the row switches. For each column, we turned all other columns off and in that column, only turned on the necessary lights. Because the columns were being scanned faster than the human eye could see, we were able to create the illusion of a static image with this technique.

Conclusions & Future Work:

Our final design met our initial goal. We had set out to be able to play the Snake game on an LED matrix, and we managed to do just that. As it currently is, the game is playable until theoretically 63 apples (maximum amount of LED points on the screen is 64) and the buttons are responsive to having the snake change direction.

We learned a lot from this project. The first being how to test an LED matrix properly using a DMM and then connecting it to an arduino. We also learned how to interact heavily with the Arduino code base. We utilized a lot of external libraries and created our own to make the project run which we had never done before. On top of that, learning about button debouncing and learning how to correct it will be useful in future projects which will need buttons.

In the future, we would like to implement a larger LED system so the user has more space to play since right now the screen is a little small. We would test this in much the same way we implemented a singular matrix. On top of that, we want to be able to have a display for the score as well as right now the user has to count each of the portions of the snake to see their score.

Illustration Credits:

[1i] Arduino Row Column Scanning: https://www.arduino.cc/en/uploads/Tutorial/rowColScan_bb.png
[2i] Matrix Schematic: https://www.arduino.cc/en/uploads/Tutorial/LEDmatrix_schem.png
[3i] Button connection to Arduino: https://www.arduino.cc/en/uploads/Tutorial/button_sch.png
(We modified this slightly to show what our exact cicuit appeared to be, but used this as the base)

References:

[1] Queue Array Library for Arduino: http://playground.arduino.cc/Code/QueueArray
[2] Understanding Button Debouncing: https://www.arduino.cc/en/Tutorial/Debounce
[3] Further Debouncing Tutorial: https://programmingelectronics.com/tutorial-19-debouncing-a-button-with-arduino-old-version/
[4] Understanding a Microswitch: https://www.youtube.com/watch?v=q6nP1FjxAMU
[5] Understanding More About Switches: https://en.wikipedia.org/wiki/Switch
[6] Input Pullup mode: https://www.arduino.cc/en/Tutorial/InputPullupSerial
[7] Helpful Arduino Tutorials: http://playground.arduino.cc/Main/InterfacingWithHardware#Switches
[8] Switch Tutorial: https://www.arduino.cc/en/Tutorial/Switch
[9] Example Input Mode: https://www.arduino.cc/en/Tutorial/DigitalReadSerial
[10] Turning Analog Pins to Digital Pins: http://waihung.net/arduino-tip-turn-your-analog-pins-into-digital-io/
[11] Using Digital Read: https://www.arduino.cc/en/Reference/DigitalRead
[12] Row Column Scanning: https://www.arduino.cc/en/Tutorial/RowColumnScanning

[13] Using the Serial Output to view numbers:
https://programmingelectronics.com/using-the-print-function-with-arduino-part-1/
[14] Setting Up Blinking: https://www.arduino.cc/en/Tutorial/BlinkWithoutDelay