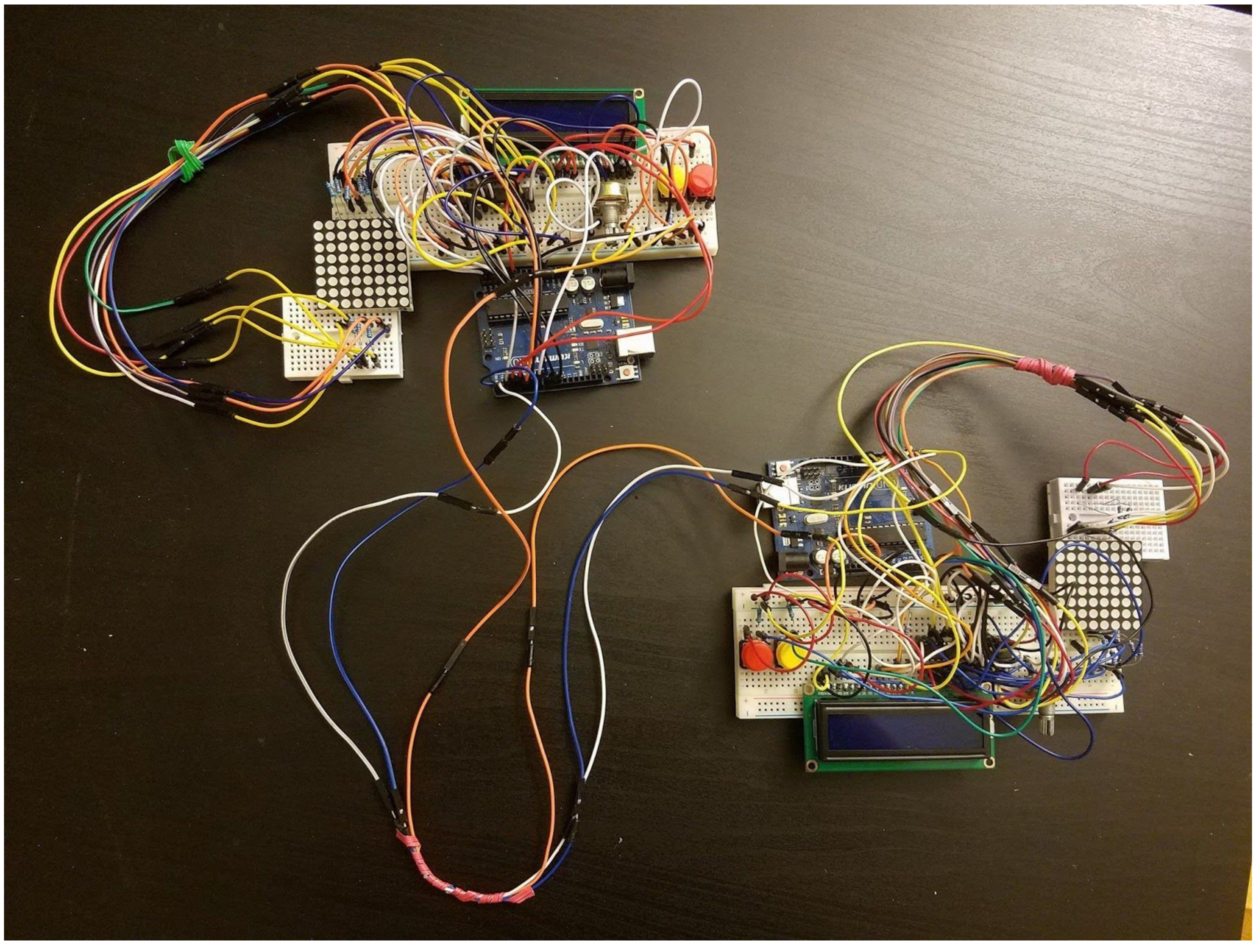
UIC CS362 Fall 2016

Final Project Report

Arduino Battleships

Alan Fan (alanfan1)

David Liang (dliang9)



Overhead view of our circuit
Photography courtesy of David Liang and Alan Fan

Welcome to Battleships!

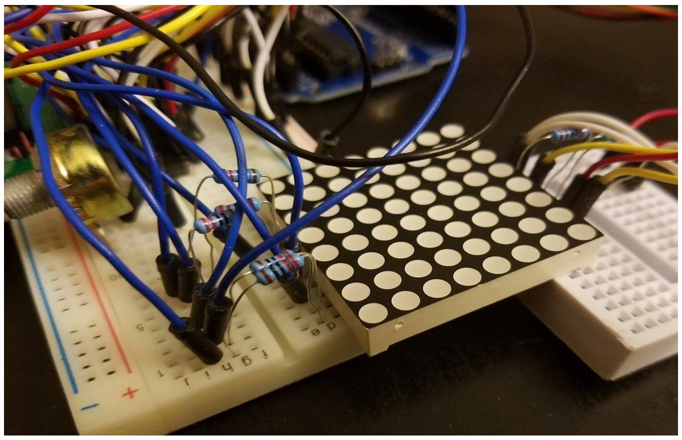
The game consists of 2 different phases:

1. Setup Phase

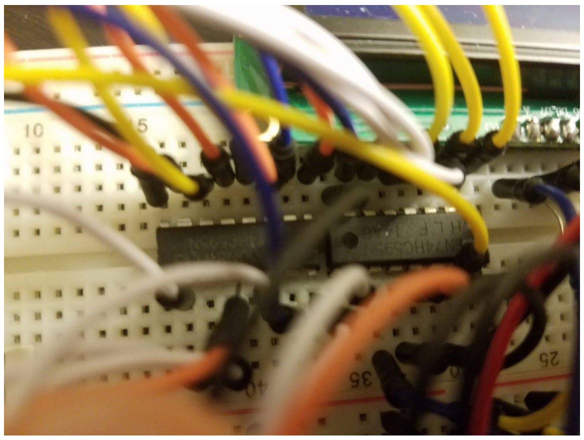
During the setup phase you are given the option of setting up a 2x1 ship, 3x1 ship, and a 4x1 ship in respective order. During that time you are given an option to give them a horizontal or vertical orientation with the press of the red button or yellow button respectively. After that you can press the red button to move them into a designated horizontal position and the yellow button to confirm. After that you can press the red button to move the ship vertically and yellow to confirm again. Once the setup for the 3 ships are complete, the LCD monitor will prompt “Waiting for opponent to finish setting up”. Once both of the players are done setting up the ships they are moved to the second stage of the game.

2. Playing Phase

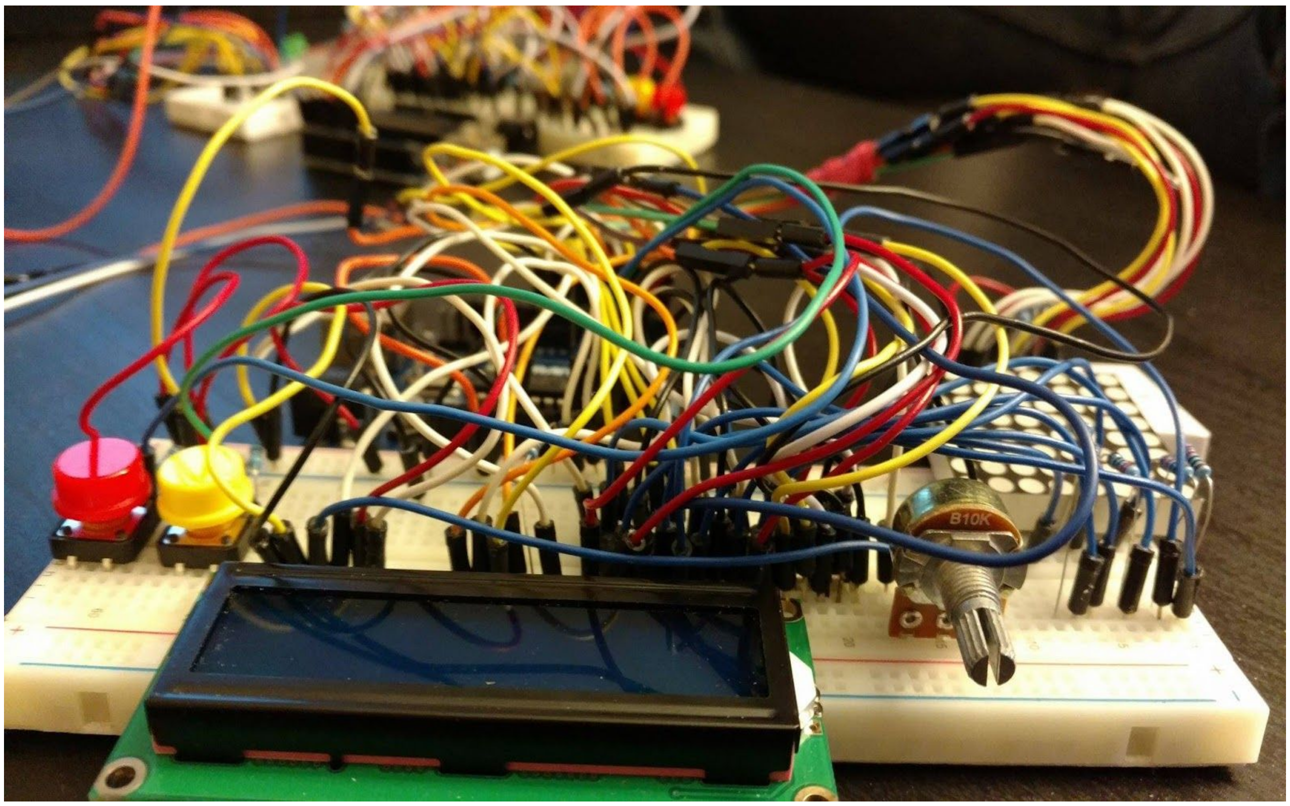
After all the ships have been set up by both players the players are free to play the game. The way moves are made are the same as the way that ships were set up. The game is turn based so it waits until both players have confirmed their moves and then once they are both performed the game proceeds. There are two different messages that the player will receive depending on whether they hit a target or missed a target. On a hit, the player will receive a message saying: “you have now hit a ship (number of ship spots left - 1) spots left to bomb” On the flipside the player that misses will have a “you have missed a ship and (number of ship spots left) spots left to bomb”. Once the game has concluded and one player has hit the maximum number of ship spots and in this case it was 9, the LCD monitor will display “You have won!” and exit appropriately.



8x8 LED Dot Matrix view of circuit



SN74HC595N Shift Registers view of circuit



16x2 Liquid Crystal Display (LCD) view of circuit

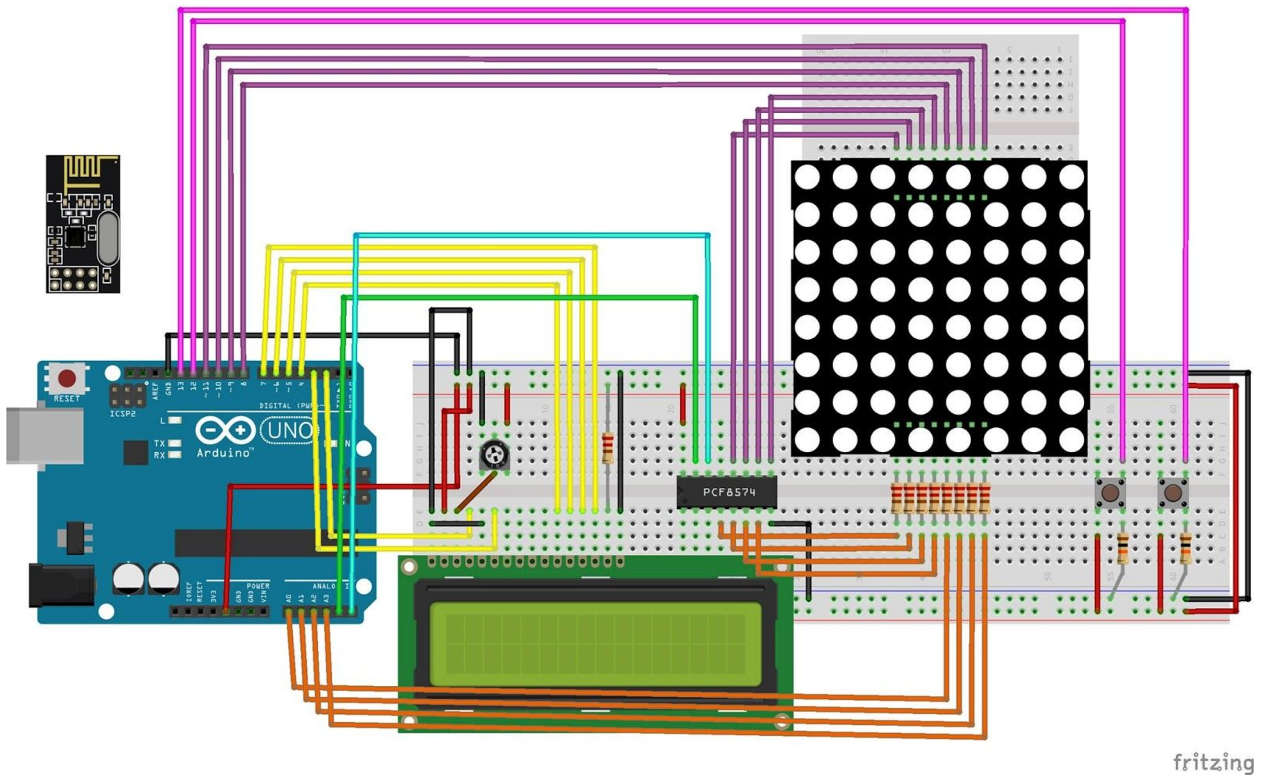
As you can probably tell this circuit utilizes numerous amounts of wires. We did not actually have enough between the two of us to complete the circuit and had to get more from an old Arduino kit.

The parts we used are:

- 2x liquid crystal display (LCD) 16x2 monitors
- 4x buttons
- 2 potentiometers
- 2x LED 8x8 dot matrices
- 4x 8 bit shift registers
- 4x 10kΩ resistors
- 18x 220Ω resistors
- many wires and female-to-female wire connectors

These were all provided in our Arduino Tinker Kits and we did not have to buy extra parts. However, as we will discuss in the next section, “Circuit Design Process”, if we were to build on this project, there are several electronics not included in the Arduino Tinker Kit that we would like to incorporate into our project.

Circuit Design Process

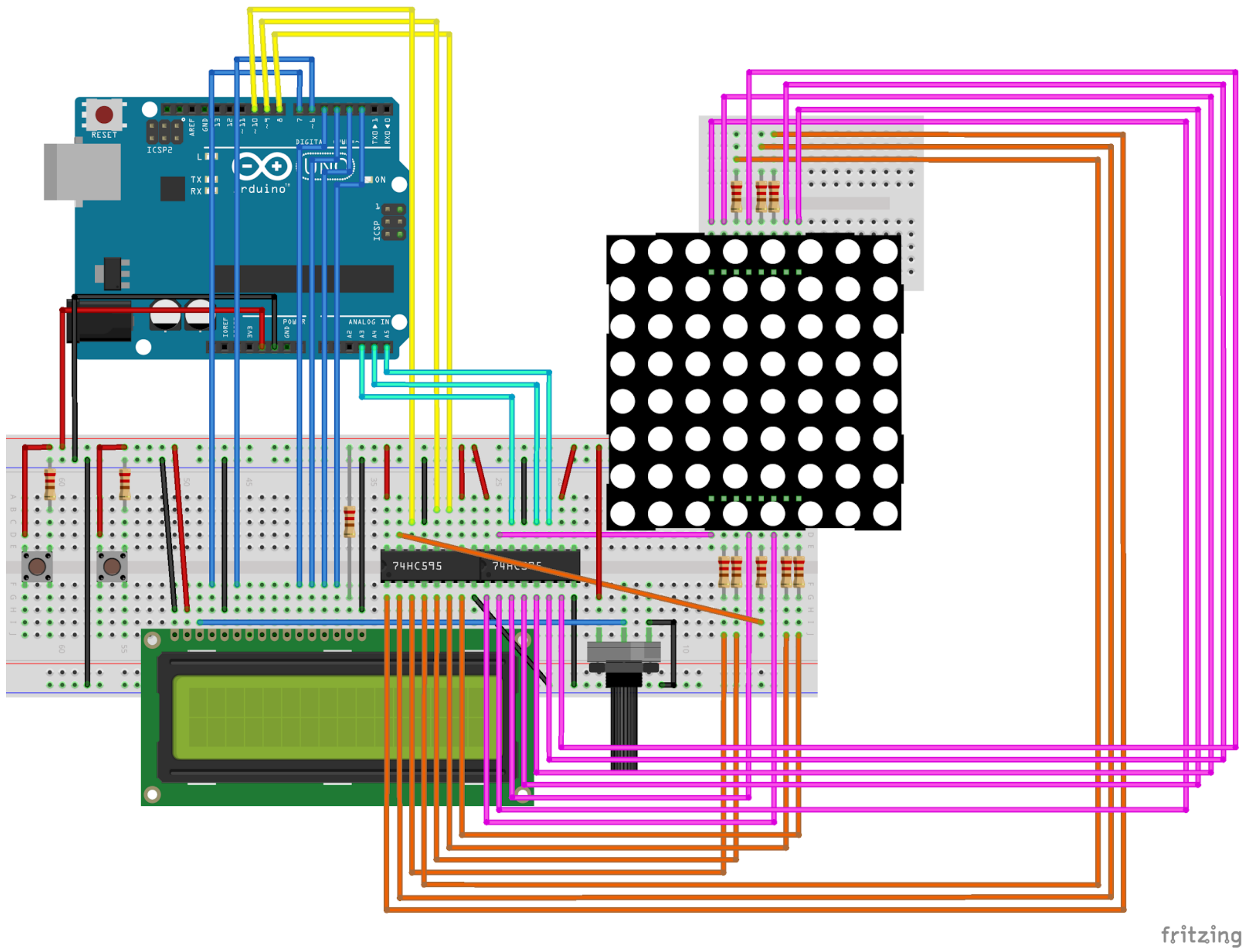


Our initial circuit design
Image produced using Fritzing

In our old design, we had planned on using a PCF8574 Arduino I/O Port Expander, which uses an I^{2}C protocol to control 8 digital pins of the LED dot matrix since the Arduino Uno does not have enough digital pins to connect all of our parts. Furthermore, we were planning on using a wireless receiver to communicate between our two Arduinos.

However, we realized that the part that came with our Arduino Tinker Kits were not the PCF8574 port expanders but SN74HC595N 8 bit shift registers. While the port expander would have taken only 2 ports to control, the shift register takes 3, a clock pin, a latch pin, and a data pin. As seen from the old circuit diagram, this would've meant that we would have to use two shift registers since we did not have any ports left or order online the port expander. However, given the time left to complete the project as well as the uncertainty of shipping during the Holiday Season, we chose to redo our circuitry with two shift registers instead (see below for final circuitry).

For the fear of shipping, we chose not to order a wireless receiver, but connected our Arduinos directly via a common ground, a wire from the TX port of one Arduino to the RX port of the other, and a wire from the RX port of the former to the TX port of the latter.

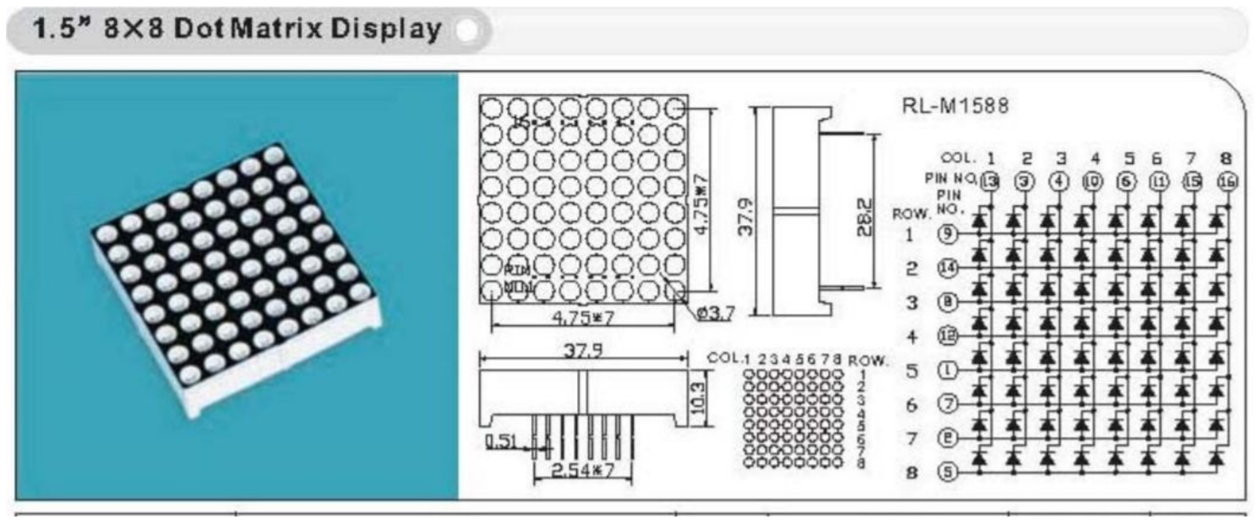


Our final circuit design
Image produced using Fritzing

In designing our circuit, we ran into yet another problem: the 8x8 LED dot matrix. At first, searching online yielded a multitude of different ways to connect various 8x8 LED dot matrices to an Arduino, and so we, as efficient students of Occam, chose the simplest as shown in our old circuit diagram. However, it became quickly apparent that such a connect was not the correct one for our dot matrix, and even it if were (by manipulating the software instead of the hardware), how would we code the program to be able to output the game board? And thus we set out on what would become an unnecessarily arduous journey to solve the mystery of our 8x8 LED dot matrix.

If Google fails, where do you turn? Obviously YouTube. logMaker360 posted a video titled "Write bytes to a [*sic*] 8x8 LED matrix Arduino" (https://youtu.be/tyAaW-Ts5T0) in which he explained how he manually tested his dot matrix using a 5V wire and a 3.3V wire to figure out which pins controlled which row or column. However, he failed to mention that if you connect the high (5V) to a column controlling pin and the low (3.3V) to a row controlling pin, you are sending current in reverse against the diode/LED, which is what happened to us. Thus, as any student learns in electrical engineering lab 101, we burned out some of the LED's on one of our dot matrix. May Doris rest in peace. If only we had a multimeter that has the diode knob to test diodes . . . Nevertheless, we were not daunted, and following logMaker360's tutorial as wel as another tutorial (http://layoftheland.net/archive/ART4645-2011/weeks7-12/LED_Matrix/How%20to%20Identify%20Pins%20on%20an%20LED%20Matrix.pdf), decided to add 220Ω resistors to all the pins and retested, and this time found the correct pin - row/column configurations.

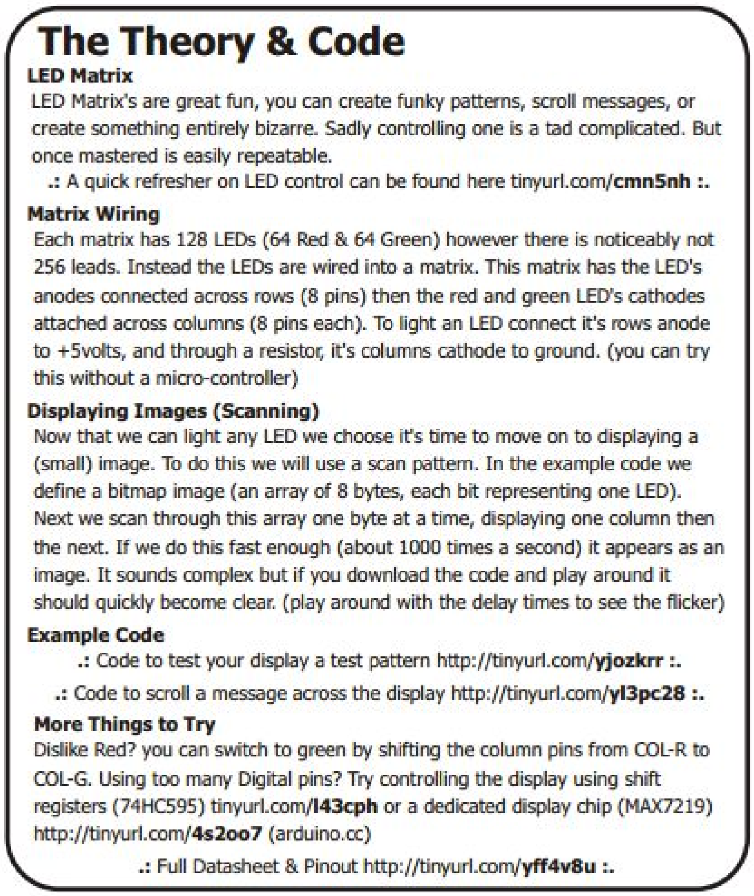
And then Alan was able to barely make out the faded product ID printed on the side of the matrix, and with some clever Googling (omitting the last character), he was able to actually find the product information sheet (see below).



Excerpt from Product Information Sheet for RL-M1588 (Red) Dot Matrices
http://www.auto-power-led.com/rimages/658/RL-M1588-SERIES.pdf

The circuit diagram for the dot matrix confirmed our personal testing of which pins control which row or column. It also shows the direction of the diodes, from which we learned to connect resistors to all of the pins controlling columns as they are the cathodes (current sinks). Now that the hardware problem was solved, the issue of software needed to be addressed.

Luckily for us, rscharff provided a tutorial on instructables.com (http://www.instructables.com/id/Beginner-tutorial-Controlling-LED-matrix-with-2-59/) for how to connect a LED dot matrix to an Arduino via two shift registers. Occam would be proud of us. We followed his design pattern and used one shift register to control all the column pins (the yellow wires controlling the shift register and the orange wires the column pins), and the other to control all the row pins (the cyan wires controlling the shift register and the magenta wires the column pins). The row scanning pattern we used was influenced by the excerpt below (http://oomlout.com/8X8M/8X8M-Guide.pdf), hence the 1ms delay in our printBoardOnLEDMatrix() function.



The Theory & Code

LED Matrix

LED Matrix's are great fun, you can create funky patterns, scroll messages, or create something entirely bizarre. Sadly controlling one is a tad complicated. But once mastered is easily repeatable.

.: A quick refresher on LED control can be found here tinyurl.com/**cmn5nh** :.

Matrix Wiring

Each matrix has 128 LEDs (64 Red & 64 Green) however there is noticeably not 256 leads. Instead the LEDs are wired into a matrix. This matrix has the LED's anodes connected across rows (8 pins) then the red and green LED's cathodes attached across columns (8 pins each). To light an LED connect it's rows anode to +5volts, and through a resistor, it's columns cathode to ground. (you can try this without a micro-controller)

Displaying Images (Scanning)

Now that we can light any LED we choose it's time to move on to displaying a (small) image. To do this we will use a scan pattern. In the example code we define a bitmap image (an array of 8 bytes, each bit representing one LED). Next we scan through this array one byte at a time, displaying one column then the next. If we do this fast enough (about 1000 times a second) it appears as an image. It sounds complex but if you download the code and play around it should quickly become clear. (play around with the delay times to see the flicker)

Example Code

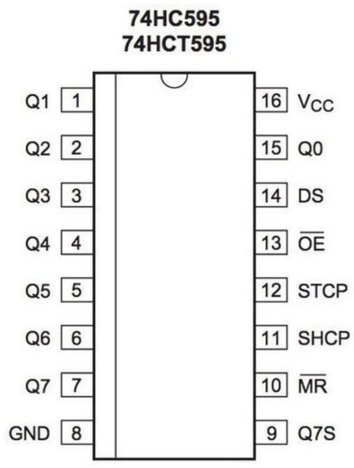
.: Code to test your display a test pattern http://tinyurl.com/**yjozkrr** :.

.: Code to scroll a message across the display http://tinyurl.com/**yl3pc28** :.

More Things to Try

Dislike Red? you can switch to green by shifting the column pins from COL-R to COL-G. Using too many Digital pins? Try controlling the display using shift registers (74HC595) tinyurl.com/**l43cph** or a dedicated display chip (MAX7219) http://tinyurl.com/**4s2oo7** (arduino.cc)

.: Full Datasheet & Pinout http://tinyurl.com/**yff4v8u** :.

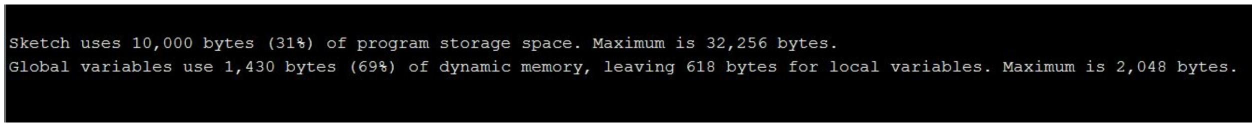


Excerpt from oomlout.com on LED Display

595 Shift Register diagram by rschaff

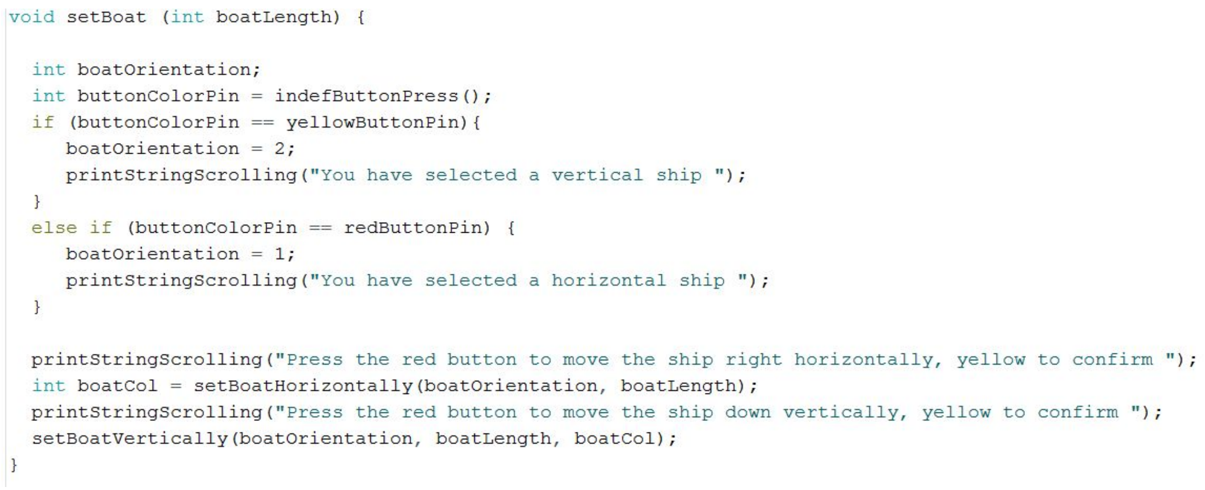
Software Design Process

We had to first create a way where users could set up their ships, and at first we created 3 functions, one that could set up a destroyer (2x1 size), cruiser (3x1 size), and battleship (4x1 size). After creating these three functions we tried to run the program, but early on we found out that the Arduino could not hold a program of that magnitude simply because of memory restrictions.



```
Sketch uses 10,000 bytes (31%) of program storage space. Maximum is 32,256 bytes.
Global variables use 1,430 bytes (69%) of dynamic memory, leaving 618 bytes for local variables. Maximum is 2,048 bytes.
```

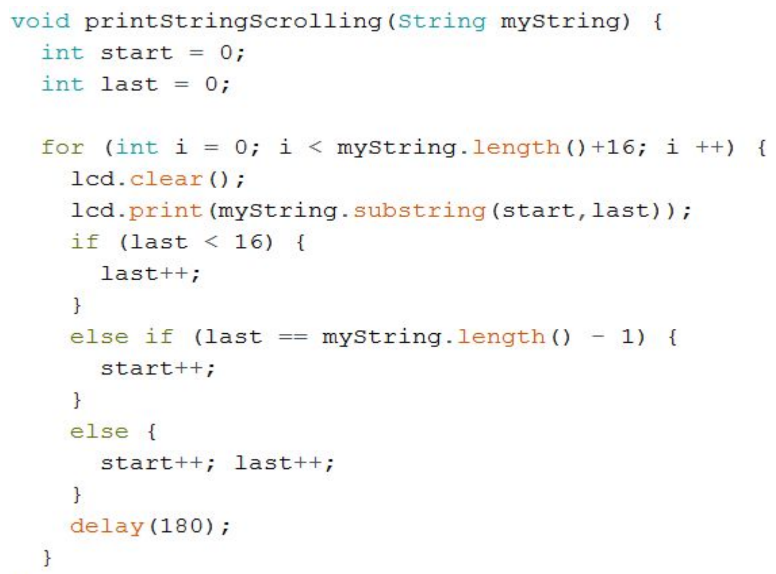
This is how much memory we got our program to use after a lot of refactoring of duplicate code, showing us the clear limitations of the Arduino, but also the importance of writing efficient code. The first step to refactoring the 3 functions we wrote to set up the ships was to create one function that would take in a boat length and that would set up the ship for any denoted length that was requested.



```
void setBoat (int boatLength) {

  int boatOrientation;
  int buttonColorPin = indefButtonPress();
  if (buttonColorPin == yellowButtonPin){
     boatOrientation = 2;
     printStringScrolling("You have selected a vertical ship ");
  }
  else if (buttonColorPin == redButtonPin) {
     boatOrientation = 1;
     printStringScrolling("You have selected a horizontal ship ");
  }

  printStringScrolling("Press the red button to move the ship right horizontally, yellow to confirm ");
  int boatCol = setBoatHorizontally(boatOrientation, boatLength);
  printStringScrolling("Press the red button to move the ship down vertically, yellow to confirm ");
  setBoatVertically(boatOrientation, boatLength, boatCol);
}
```

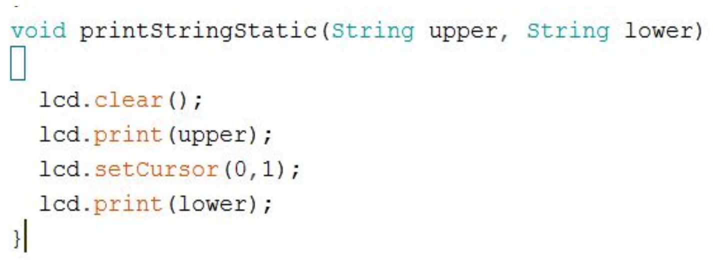
This is the function we decided upon to reduce the memory usage of having a program that was too long.

Some examples of refactored code/functions:



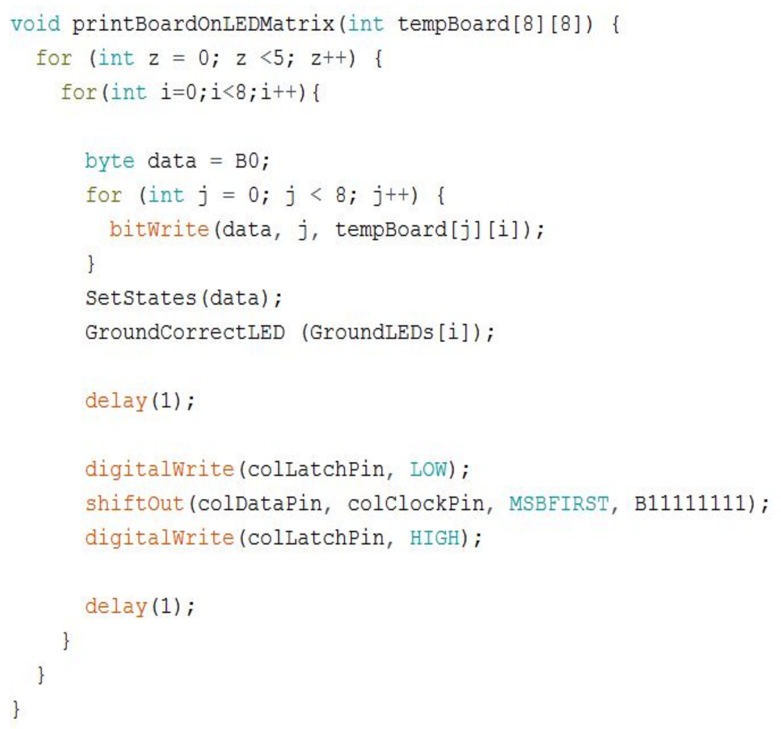
```
void printStringScrolling(String myString) {
  int start = 0;
  int last = 0;

  for (int i = 0; i < myString.length()+16; i ++) {
    lcd.clear();
    lcd.print(myString.substring(start,last));
    if (last < 16) {
      last++;
    }
    else if (last == myString.length() - 1) {
      start++;
    }
    else {
      start++; last++;
    }
    delay(180);
  }
```

This was the print function that we used to print out most of our messages in the game that were too long to fit within a 16x2 LED matrix so we had to have a scrolling text that would inform users of what their next move should be.



```
void printStringStatic(String upper, String lower)
{
  lcd.clear();
  lcd.print(upper);
  lcd.setCursor(0,1);
  lcd.print(lower);
}
```

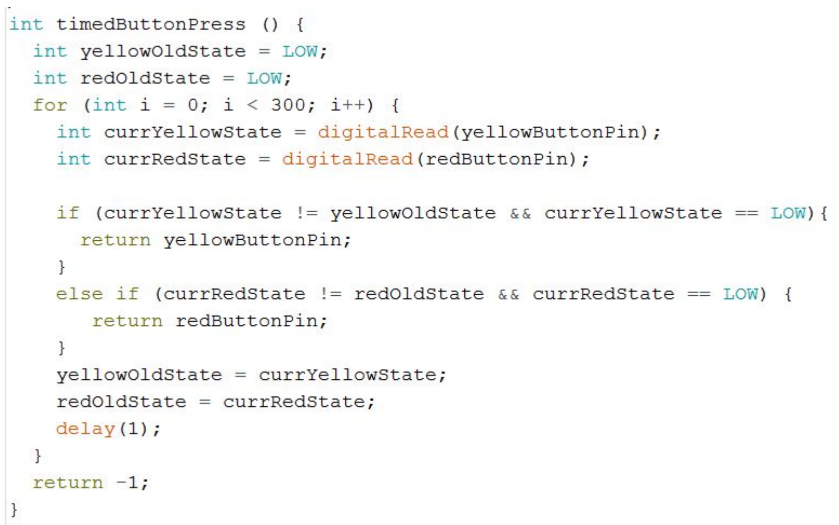
This is another print function where we would print static messages (messages that would not scroll and would be faster than scrolling messages).



```
void printBoardOnLEDMatrix(int tempBoard[8][8]) {
  for (int z = 0; z <5; z++) {
    for(int i=0;i<8;i++){

      byte data = B0;
      for (int j = 0; j < 8; j++) {
        bitWrite(data, j, tempBoard[j][i]);
      }
      SetStates(data);
      GroundCorrectLED (GroundLEDs[i]);

      delay(1);

      digitalWrite(colLatchPin, LOW);
      shiftOut(colDataPin, colClockPin, MSBFIRST, B11111111);
      digitalWrite(colLatchPin, HIGH);

      delay(1);
    }
  }
}
```

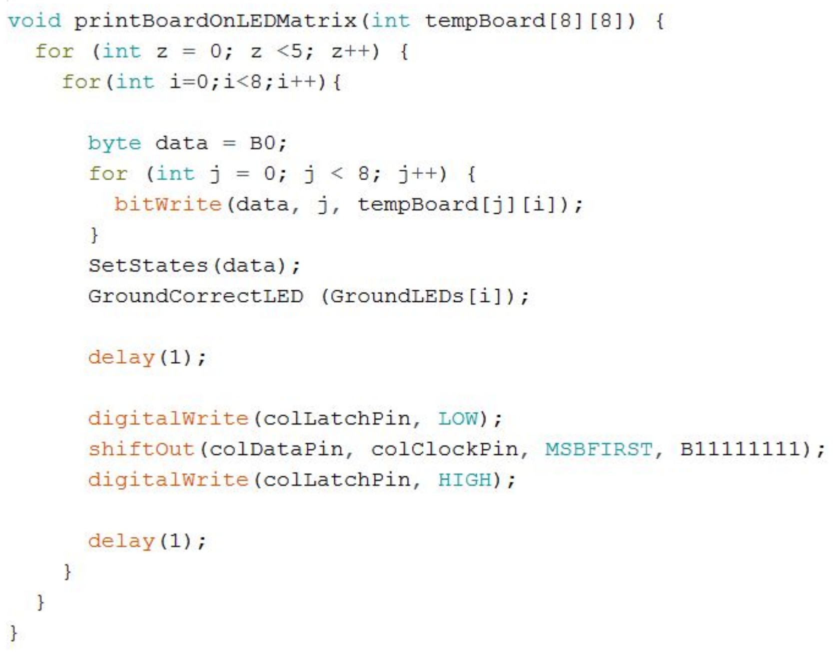
Here we have a function where we call every time we want to print out the LEDs on the LED matrix. We turn on 1 column at a time, and get the row data from the game board via bitWrite.

One of the issues we ran into when creating responsive buttons and 8x8 led matrices was that when the led matrix was printing and staying on the board the buttons would be unresponsive, and vice versa. We had to figure out a time interval where the LED matrix would be flattering to look at, but also responsive enough for the player to press on the buttons without much issue. We decided on two different delays in an attempt to cater to our users effectively.



```
int timedButtonPress () {
  int yellowOldState = LOW;
  int redOldState = LOW;
  for (int i = 0; i < 300; i++) {
    int currYellowState = digitalRead(yellowButtonPin);
    int currRedState = digitalRead(redButtonPin);

    if (currYellowState != yellowOldState && currYellowState == LOW){
      return yellowButtonPin;
    }
    else if (currRedState != redOldState && currRedState == LOW) {
       return redButtonPin;
    }
    yellowOldState = currYellowState;
    redOldState = currRedState;
    delay(1);
  }
  return -1;
}
```

Here we have the timed button press where there is a delay of 1ms in a for loop of 300 iterations



```
void printBoardOnLEDMatrix(int tempBoard[8][8]) {
  for (int z = 0; z <5; z++) {
    for(int i=0;i<8;i++){

      byte data = B0;
      for (int j = 0; j < 8; j++) {
        bitWrite(data, j, tempBoard[j][i]);
      }
      SetStates(data);
      GroundCorrectLED (GroundLEDs[i]);

      delay(1);

      digitalWrite(colLatchPin, LOW);
      shiftOut(colDataPin, colClockPin, MSBFIRST, B11111111);
      digitalWrite(colLatchPin, HIGH);

      delay(1);
    }
  }
}
```

Here is the LED board being printed onto the LED matrix and it is given 2 delays of 1 in a loop that loops 5 times until it is flickered off.

With these two delays we figured out a sweet spot for the alternating LED matrix and the button press of the user.

To communicate between the two Arduinos, firstly to work with the 32-byte buffer size of the Serial port, we sent the player's board as 8 bytes, where the nth byte encoded the nth row of the board. Then, the Arduino waits in a while loop until Serial.available() == 8. The procedure must be write first, then read otherwise deadlock would occur whereby neither Arduino can write till it has read which is dependent on a write (which is dependent on a read and so forth). Then, true to our UIC CS background, we used C/C++ bitwise operators (&, |, >>, and <<) to encode and decode our bytes; this is completely the wrong approach! In hindsight, we should have used the BitWrite and BitRead functions in the Arduino library.

The other major code adjustment we should implement in a future release would be controlling the print messages. Currently a major bottleneck in gameplay is displaying scrolling LCD messages. While we insist on having these messages as part of making our product user-friendly (e.g. some other Battleship projects by our classmates do not give instructions to their users and instead have a protocol visible only at the software level), we would like to add global flags so that the full message would only appear one their first iteration, and an abbreviated static message could display on proceeding iterations so that A) clarity of instruction is not lost and B) redundancy does not reduce efficiency.

A minor software alteration would be to implement another flag such that the blinking of the LED's correspond with hits/misses so the user can distinguish between the two. Additionally, storing additional information about the boats so that messages display when a specific boat is sunk would be a nice refinement.

In Conclusion

Battleships was a thoroughly fun project for both of us to implement: we are both avid gamers (often found late at night playing League of Legends or CS:GO) and it was a great opportunity to make a full-fledged game. We enjoyed considering the problem from a diverse array of perspectives: the hardware complexity, the software elegance and efficiency, the product user-friendliness, the depth of gameplay, and in general feasibility given we are but two students taking 300 level CS courses with limited time and resources. While we both initially felt very limited in our final project ideas by our limited experience with Arduino and the parts we had available to us, we both feel very proud of the product which we have created, and can say with certainty that as friends, this project is a beautiful memory that we will surely treasure. David will surely never cease to remind me (Alan) of how I destroyed one of our 8x8 LED matrices with raw current and no resistors, and Alan will never cease to tease me (David) of how I wrote unfactored code with so many side-effects that I overflowed dynamic memory.

Alan Fan

David Liang

Resources Used

Special thanks to Yoonha Kang and Danny Chen for allowing us to borrow their shift registers and dot matrixes.

1. Exceed Perseverance Electronic Industry Co., Ltd. “1.5” 8x8 Dot Matrix Display”. http://www.auto-power-led.com/rimages/658/RL-M1588-SERIES.pdf
2. oomlout. “LED Displays (8 x 8 LED Matrix)”. http://oomlout.com/8X8M/8X8M-Guide.pdf
3. rschaff. “Beginner tutorial: Controlling LED matrix with 2 595 shift registers and potmeter”. http://www.instructables.com/id/Beginner-tutorial-Controlling-LED-matrix-with-2-59
4. Author unlisted. “How to Use a Multimeter to Identify the Row and Column Pins of an LED Matrix”. http://layoftheland.net/archive/ART4645-2011/weeks7-12/LED_Matrix/How%20to%20Identify%20Pins%20on%20an%20LED%20Matrix.pdf