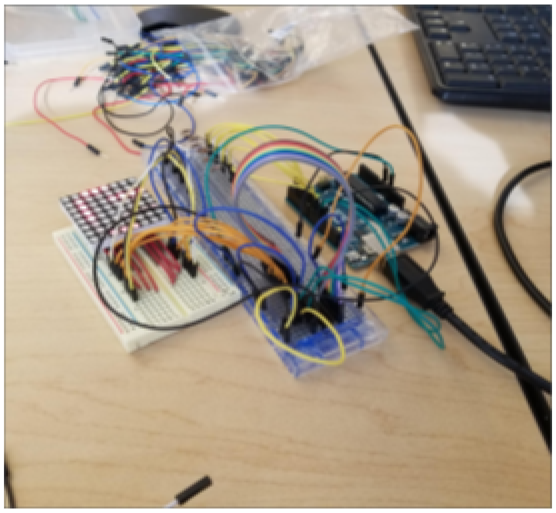
Anthony DePaul, Kang Lee December 11, 2017

Arduino Battleship

Introduction

Our goal for this project was to replicate the popular board game Battleship using two communicating Arduinos. To do so, we needed to be able to complete three main tasks: control an 8x8 LED matrix, get player input through buttons, and have the two Arduinos communicate to one another. Once we had these hardware core pieces, we could program the remaining mechanics of the Battleship game. These mechanics included: keeping track of the players' ships' statuses, spawning random ships, and taking turns with the other player. Each player would get their own Arduino paired with an LED matrix. The Arduino would keep track of and perform all the game mechanics. The LED matrix would display the ships, shots guessed, and shots hits to the player. The three buttons, x, y, and enter, would be used to control where the player wanted to take a shot at the enemy. The enter button would also double as a toggle between the player's ships and the player's guesses while waiting for the other player to take their turn. Finally, there would have to be communication between the two Arduinos so that they could properly know the enemy's shots locations or when the player's shots hit.

Hardware



8x8 LED Matrix

We started off with trying to control an 8x8 LED matrix. Initially, we tried to use the matrix and shift register given to us by the school, but it ended up getting extremely messy. To do this, the shift register would convert a serial signal sent from the Arduino into a parallel one. Each row and column would be sent a signal in quick succession to produce a desired image on the matrix. Since this method requires a lot of wires, there was a lot of room for error. By the time that we were finally able to get one of the LED matrices working this way, our simplified solution arrived. We found 8x8 LED matrices online (goo.gl/h8AqzT) that significantly reduced the amount of wires. Instead of needing to wire the matrix's columns into resistors, those resistors into the shift register, the shift register into the Arduino, and the matrix's rows into the Arduino, all we had to do was route five wires from the Arduino to this module.

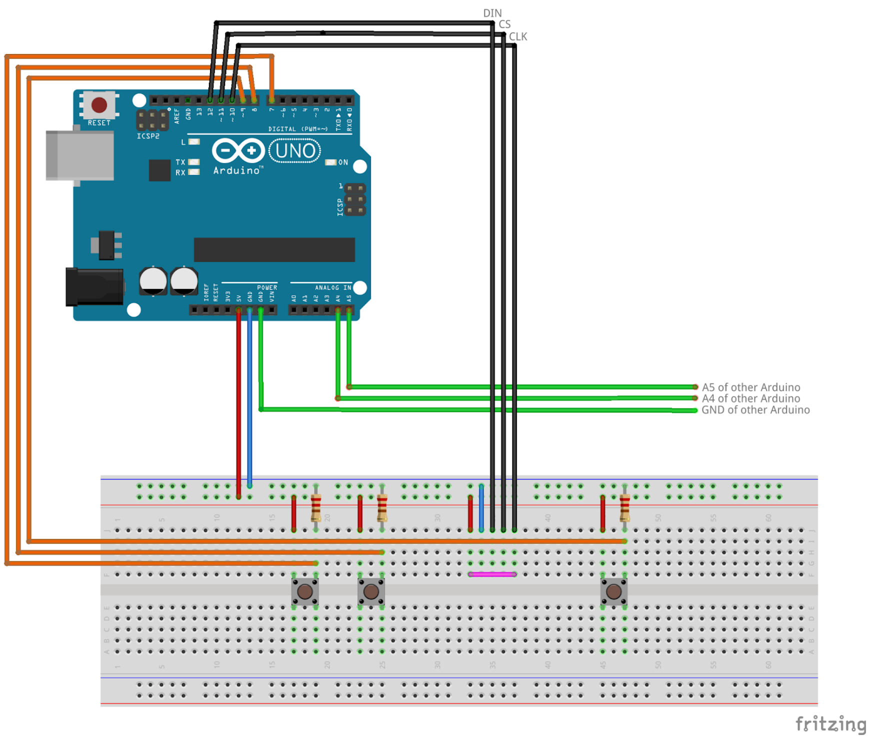
Once we had our 8x8 LED matrices, we had to be able to control them. We were lucky enough to find a guide on the internet (https://goo.gl/pSAMZo) that told us exactly what we needed. More specifically, from this website, downloading LedControlMS.zip will give you some C++ libraries that make it easy to interface with the LED matrix. To use their library, we had to import it into our Arduino code, run the constructor `LedControl(<DIN pin #>,<CLK pin #>, <CS pin #>, <# devices>)`, and call `setLed(<module #>, <column>, <row>, <value>)` whenever we wanted to set a particular LED. Now we had what we needed to control the display for our Battleship game.

Buttons

The next step was wiring up the buttons. Arduino has example code showing how to program a button to be used as input as well as a tutorial on how to wire one (https://goo.gl/pGA9Ht). Using their wiring method, the Arduino understands the input because the button and resistor creates a change in current that it can detect. Programming the buttons was simple as well. The pinmode of the pins the buttons used had to be set to input. Then, to detect when a button was being pressed we would use `digitalRead(<pin #>)`.

Communication

To let the two Arduinos to communicate with one another, we used I2C which is a serial communications protocol. We learned how to do this in class so we did not have to do any research. The Arduino comes with a library called Wire which allows us to communicate through I2C. The wiring consisted of attaching the two Arduinos by two analog pins and ground. To send a message over the wire, we would begin the transmission with: `Wire.beginTransmission(<device #>)`, write to the transmission with: `Wire.write(<data>)`, and end the transmission with: `Wire.endTransmission()`. To receive a message from the wire, we set a desired function to be run when a transmission is received using: `Wire.onReceive(<function>)`. Once that event function is ran, we can read the bytes from the message using: `Wire.read()`.



Software

Assembly Tables

Now that we had our core hardware components, we could move on to programming our game. First, we had to create a table in assembly to keep track of the players' ships' statuses. To do this, three tables were created: tablex, tabley, and tablehit. Each were created with 12 initial values. Twelve was used because players would start off with four ships. One ship would be 4 pixels long, two were 3, and one was 2. The tables would all work in parallel where the first value of each table would all represent the same ship pixel. Then, the table functions were made, writeShip, readShipX, readShipY, and readShipHit. First, to read from the tables, we have to load from them. Using: `ldi r30,lo8(<table>)` and `ldi r31,hi8(<table>)`, we can load the location of a specific table. From there, we can add to it to get the index we want using: `add r30,r24` (r24 being the argument) and `adc r31,r1`. Finally, we use: `ld r24,z` to get the byte from location z and return from there. For the writeShip method, we changed `ld r24,z` to `st z,r22` to store the passed in argument into the table instead of retrieving from it. Also, instead of doing it once, we do this three times as all three arguments are passed instead of one.

Spawning Random Ships

Next, we decided to spawn the ships randomly as choosing the locations and directions for each would be challenging with our limited controls. To begin, a common technique used to ensure random numbers on an Arduino is to use the small amount of noise from one of the analog pins as a random seed. From there, a loop is run four times, once for each ship. For the first ship, shipSize equals 4, the third and second is 3, and the last is 2. This is how the quantity and size of the ships is decided. From there, a while loop repeatedly tries random locations and directions to place the ship until it can. It does this once for each ship.

Communication

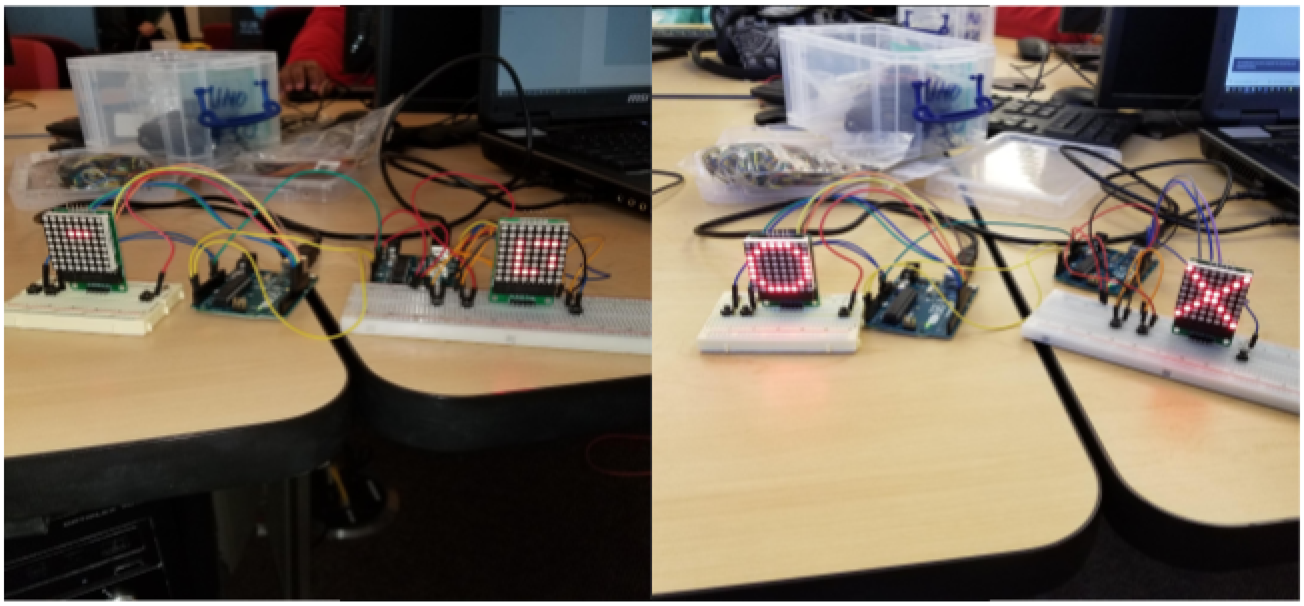
Another step in the programming was to have them properly communicate. The two Arduinos take turns waiting. If the Arduino is waiting, that means the other arduino is taking their turn to guess. Once that Arduino takes a guess, it will transmit its cursor's x coordinate and y coordinate to the other. The waiting Arduino will receive these two bytes and check to see if those two coordinates had a ship. If it did, it will write that ship as hit. Once it does this, a response value will either be set to 1 or 2. 0 represents no response, 1 represents a miss, and 2 represents a hit. The main loop will notice that the response value is no longer 0 and send hit or miss back to the other Arduino. Also, waiting will be set to false so that the player can take a turn. Finally, when that Arduino gets its response it will update its guess 2D array to a hit or a miss. The Arduinos are able to tell the difference from a received message by the length of the message. The initial guess message contains two bytes, while the response only has one.

My Contribution

I mainly worked on the software of the project. Kang did a lot of work on the hardware, specifically getting the LED matrix to work. I programmed the assembly table to be used to store battleship data. To do so, I looked at the table code from a previous lab and modified it to fit our needs. The lab only had one table while ours needed three. Also, our table needed to have the ability to write to it. I also wrote much of the Arduino code that took care of main game mechanics.

Conclusion

In the end, the Arduino Battleship project was a success. We were able to accomplish many different things. In regards to hardware, we were able to use the 8x8 LED matrices to display how we wanted them to, we were able to get player input through buttons, and have the two Arduinos communicate. All this knowledge was completely new to us and we were able to use it to accomplish one bigger goal. We were also able to create a table using Assembly, a very low level programming language. By doing so, we could understand much more of how the Arduino processes and represents bytes. Using all of the above, we could program the additional mechanics to make the Battleship game function as intended. Spawning random ships and having them properly communicate their data with one another was part of the process to make the finalized game. Finally, after all the pieces were put together, we are very satisfied with how it all fit into place.



Code

battleshiptable.h

```
// writes a ship to the table (index,x,y,hit)
extern "C" void writeShip(byte, byte, byte, byte);
extern "C" byte readShipX(byte);       // returns specified ship x
extern "C" byte readShipY(byte);       // returns specified ship y
extern "C" byte readShipHit(byte);     // returns specified ship hit
```

battleshiptable.S

```
.file "battleshiptable.S"
.data

; tables storing 12 ships' data
tablex:
      .byte 100,100,100,100,100,100,100,100,100,100,100,100
tabley:
      .byte 100,100,100,100,100,100,100,100,100,100,100,100
tablehit:
      .byte 0,0,0,0,0,0,0,0,0,0,0,0

.global writeShip
.global readShipX
.global readShipY
.global readShipHit

.text

; write ship data
; r24=index, r22=x, r20=y, r18=hit
writeShip:
      ; set x value
      ldi r30,lo8(tablex)
      ldi r31,hi8(tablex)
      add r30,24
      adc r31,r1
      st z, r22
      ; set y value
      ldi r30,lo8(tabley)
      ldi r31,hi8(tabley)

      add r30,24
      adc r31,r1
      st z, r20

      ; set x value
      ldi r30,lo8(tablehit)
      ldi r31,hi8(tablehit)

      add r30,24
      adc r31,r1
      st z, r18
      ret

; retrun ship X
readShipX:
      ldi r30,lo8(tablex)
      ldi r31,hi8(tablex)

      add r30,r24
      adc r31,r1

      ld r24,z
      ret

; return ship Y
readShipY:
      ldi r30,lo8(tabley)
      ldi r31,hi8(tabley)

      add r30,r24
      adc r31,r1

      ld r24,z
      ret

; return ship hit
readShipHit:
      ldi r30,lo8(tablehit)
      ldi r31,hi8(tablehit)
```

```
     add r30,r24
     adc r31,r1

     ld r24,z
     ret
```

battleship.ino

```
#include <battleshiptable.h>
#include <Wire.h>
#include "LedControlMS.h"

// used to control leds
LedControl lc=LedControl(12,11,10, 1);

// pins for buttons
const byte xButt = 7;
const byte yButt = 8;
const byte entButt = 9;

// if this arduino is waiting for the other player
boolean waiting = false;
// display guesses if true, display ships if false
boolean displayToggle = false;
// alternates between true and false
boolean blinker = true;
byte blinkerCounter = 0;
// alternates between true and false at half the rate
boolean slowBlinker = true;
byte slowBlinkerCounter = 0;

// button values when button is held
boolean entHeld = false;
boolean xHeld = false;
boolean yHeld = false;
// button values when button is initially pressed
boolean entPressed = false;
boolean xPressed = false;
boolean yPressed = false;

// location of guess cursor
byte cursorX = 3;
byte cursorY = 3;

// all the guesses locations 0=not guessed, 1=guessed, 2=hit
byte guesses[8][8];
// the last guessed location
byte lastGuessX;
byte lastGuessY;
byte response = 0;  // 0=no response, 1=miss, 2=hit

void setup() {

  pinMode(xButt, INPUT);
  pinMode(yButt, INPUT);
  pinMode(entButt, INPUT);

  Wire.begin(8);
  Wire.onReceive(receiveTurnEvent);

  Serial.begin (9600);
  Serial.println("Setup");

    lc.shutdown(0,false);
  // Set the brightness to a medium values
  lc.setIntensity(0,8);
  // and clear the display
  lc.clearDisplay(0);

  placeShips();

  Serial.println("Ship locations");
  for (byte i=0; i<12; i++) {
    Serial.print(readShipX(i));
    Serial.print(" ");
    Serial.println(readShipY(i));
  }

  startingAnimation();
}


void loop() {

  boolean pixels[8][8] = {{0,0,0,0,0,0,0,0},
                              {0,0,0,0,0,0,0,0},
                              {0,0,0,0,0,0,0,0},
                              {0,0,0,0,0,0,0,0},
                              {0,0,0,0,0,0,0,0},
                              {0,0,0,0,0,0,0,0},
                              {0,0,0,0,0,0,0,0},
                              {0,0,0,0,0,0,0,0}};

  boolean a = 1;

  manageButtons();

  if (response != 0) {
    Wire.beginTransmission(8);
    Wire.write(response);
    Wire.endTransmission();

    response = 0;
  }

  // lost detection
  boolean lost = true;
  for (byte i = 0; i<12; i++)
    if (readShipHit(i) == 0)
      lost = false;
  if (lost) {
    boolean cross[8][8] = {{1,0,0,0,0,0,0,1},
                           {0,1,0,0,0,0,1,0},
                           {0,0,1,0,0,1,0,0},
                           {0,0,0,1,1,0,0,0},
                           {0,0,0,1,1,0,0,0},
                           {0,0,1,0,0,1,0,0},
                           {0,1,0,0,0,0,1,0},
                           {1,0,0,0,0,0,0,1}};
    updateDisplay(cross);
    delay(1000);
    return;
  }

  // win detection
  byte hitCnt = 0;
  for (byte row=0; row<8; row++)
    for (byte col=0; col<8; col++)
```

```
      if (guesses[row][col] == 2)
        hitCnt++;
  if (hitCnt >= 12) {
    boolean circle[8][8] = {{0,0,1,1,1,1,0,0},
                            {0,1,0,0,0,0,1,0},
                            {1,0,0,0,0,0,0,1},
                            {1,0,0,0,0,0,0,1},
                            {1,0,0,0,0,0,0,1},
                            {1,0,0,0,0,0,0,1},
                            {0,1,0,0,0,0,1,0},
                            {0,0,1,1,1,1,0,0}};
    updateDisplay(circle);
    delay(1000);
    return;
  }

  // my turn
  if (!waiting) {
    displayGuesses(pixels);
    pixels[cursorY][cursorX] = slowBlinker;

    if (xPressed)
      cursorX = ++cursorX%8;
    if (yPressed)
      cursorY = ++cursorY%8;
    if (entPressed) {
      guesses[cursorY][cursorX] = 1;
      lastGuessX = cursorX;
      lastGuessY = cursorY;

      Wire.beginTransmission(8);
      Wire.write(cursorX);
      Wire.write(cursorY);
      Wire.endTransmission();

      displayToggle = false;
      waiting = true;
    }
  }
  // others turn
  else {
    // ent is toggle displays
    if (entPressed)
```

```
      displayToggle = !displayToggle;

    if (!displayToggle) {
      displayShips(pixels);
    }
    else {
      displayGuesses(pixels);
    }
  }

  // blinkers
  blinkerCounter++;
  if (blinkerCounter >= 2) {
    blinkerCounter = 0;
    blinker = !blinker;
  }
  slowBlinkerCounter++;
  if (slowBlinkerCounter >= 4) {
    slowBlinkerCounter = 0;
    slowBlinker = !slowBlinker;
  }

  updateDisplay(pixels);
  delay(50);
}

void placeShips() {
  randomSeed(analogRead(0));
  byte shipCnt = 0;

  for (byte i=0; i<4; i++) {

    // 1st ship is 4 long, 3rd and 2nd is 3, 4th is 2
    byte shipSize;
    if (i==0)
      shipSize = 4;
    else if(i==1 || i==2)
      shipSize = 3;
    else
      shipSize = 2;

    // place the ship in a random location
    boolean placed = false;
    while (!placed) {
      byte x = random(8);
      byte y = random(8);
      byte dir = random(4);

      if (canPlace(x,y,dir,shipSize)) {
        // place ship
        for (byte j=0; j<shipSize; j++) {
          // up
          if (dir==0)
            writeShip(shipCnt++, x, y+j, false);
          // right
          if (dir==1)
            writeShip(shipCnt++, x+j, y, false);
          // down
          if (dir==2)
            writeShip(shipCnt++, x, y-j, false);
          // left
          if (dir==3)
            writeShip(shipCnt++, x-j, y, false);
        }

        placed = true;
      }
    }
  }
}

boolean canPlace(byte x, byte y, byte dir, byte shipSize) {
  // up
  if (dir==0) {
    // check if falls off display
    if (y+shipSize > 7)
      return false;

    // check if collides with another ship
    for (byte i=0; i<shipSize; i++)
      if (isShip(x, y+i))
        return false;
  }

  // right
  else if (dir==1) {
    if (x+shipSize > 7)
      return false;

    for (byte i=0; i<shipSize; i++)
      if (isShip(x+i, y))
        return false;
  }

  // down
  else if (dir==2) {
    if (y-shipSize < 0)
      return false;

    for (byte i=0; i<shipSize; i++)
      if (isShip(x, y-i))
        return false;
  }

  // left
  else {
    if (x-shipSize < 0)
      return false;

    for (byte i=0; i<shipSize; i++)
      if (isShip(x-i, y))
        return false;
  }

  return true;
}

boolean isShip(byte x, byte y) {
  for (byte i=0; i<12; i++)
    if (readShipX(i)==x && readShipY(i)==y)
      return true;
  return false;
}

void displayShips(boolean pixels[8][8]) {
  for (byte i=0; i<12; i++) {
    if (readShipHit(i))
      pixels[readShipY(i)][readShipX(i)] = blinker;
    else
```

```
      pixels[readShipY(i)][readShipX(i)] = true;
    }
}

void startingAnimation() {

  for (int x=3; x>=0; x--) {
    boolean pixels[8][8] = {{0,0,0,0,0,0,0,0},
                            {0,0,0,0,0,0,0,0},
                            {0,0,0,0,0,0,0,0},
                            {0,0,0,0,0,0,0,0},
                            {0,0,0,0,0,0,0,0},
                            {0,0,0,0,0,0,0,0},
                            {0,0,0,0,0,0,0,0},
                            {0,0,0,0,0,0,0,0}};

    for (byte y=0; y<8; y++)
      pixels[y][x] = true;
    for (byte y=0; y<8; y++)
      pixels[y][7-x] = true;

    updateDisplay(pixels);
    delay(100);
  }
}

void manageButtons() {
  if (!entHeld && digitalRead(entButt)) {
    entHeld = true;
    entPressed = true;
  } else if (digitalRead(entButt)) {
    entHeld = true;
    entPressed = false;
  } else {
    entHeld = false;
    entPressed = false;
  }

  if (!xHeld && digitalRead(xButt)) {
    xHeld = true;
    xPressed = true;
  } else if (digitalRead(xButt)) {
    xHeld = true;
    xPressed = false;
  } else {
    xHeld = false;
    xPressed = false;
  }

  if (!yHeld && digitalRead(yButt)) {
    yHeld = true;
    yPressed = true;
  } else if (digitalRead(yButt)) {
    yHeld = true;
    yPressed = false;
  } else {
    yHeld = false;
    yPressed = false;
  }
}

void displayGuesses(boolean pixels[8][8]) {
  for (byte row=0; row<8; row++) {
    for (byte col=0; col<8; col++) {
      if (guesses[row][col] == 1)
        pixels[row][col] = true;
      if (guesses[row][col] == 2)
        pixels[row][col] = blinker;
    }
  }
}

void receiveTurnEvent(int bytes) {

  if (bytes == 2) {
    byte x = Wire.read();
    byte y = Wire.read();

    response = 1;

    for (byte i=0; i<12; i++) {
      // hit
      if (readShipX(i)==x && readShipY(i)==y) {
        writeShip(i,x,y,true);
        response = 2;
      }
    }
    waiting = false;
  }
  else if (bytes == 1)
    if (Wire.read() == 2)
      guesses[lastGuessY][lastGuessX] = 2;
}

void updateDisplay(boolean pixels[8][8]) {
  for (byte row=0; row<8; row++)
    for (byte col=0; col<8; col++)
      lc.setLed(0,col,row,pixels[row][col]);
}
```