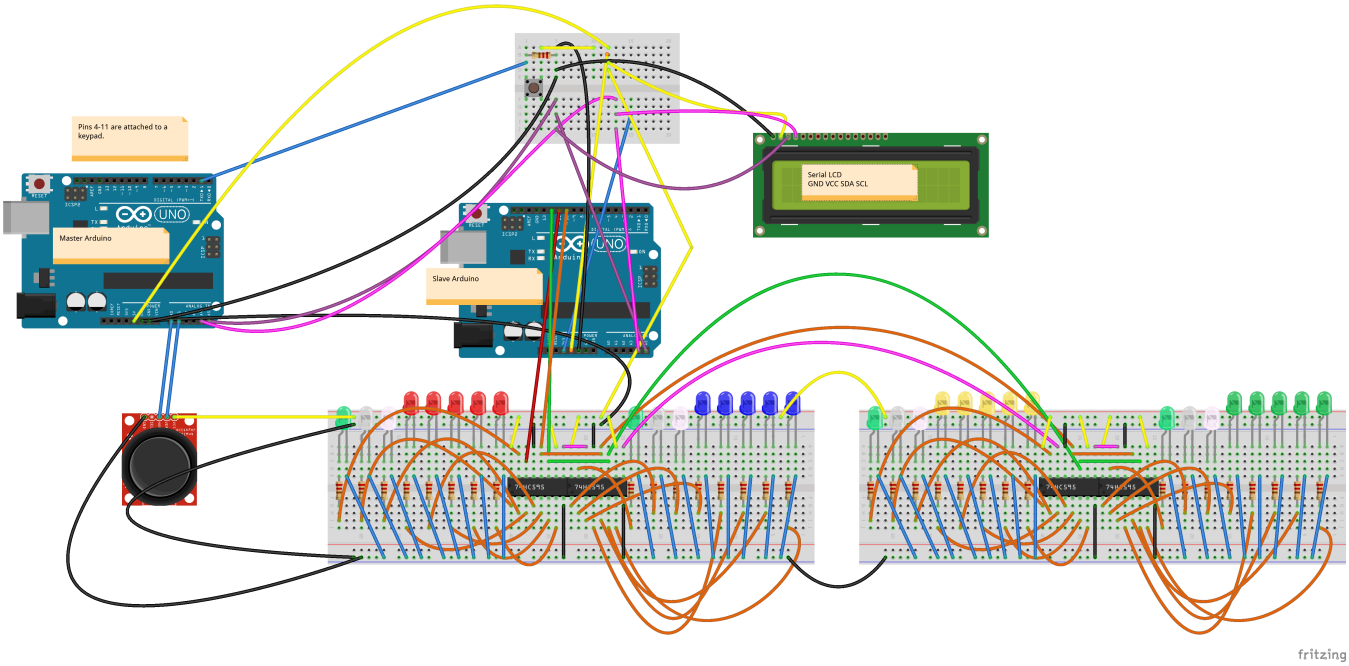
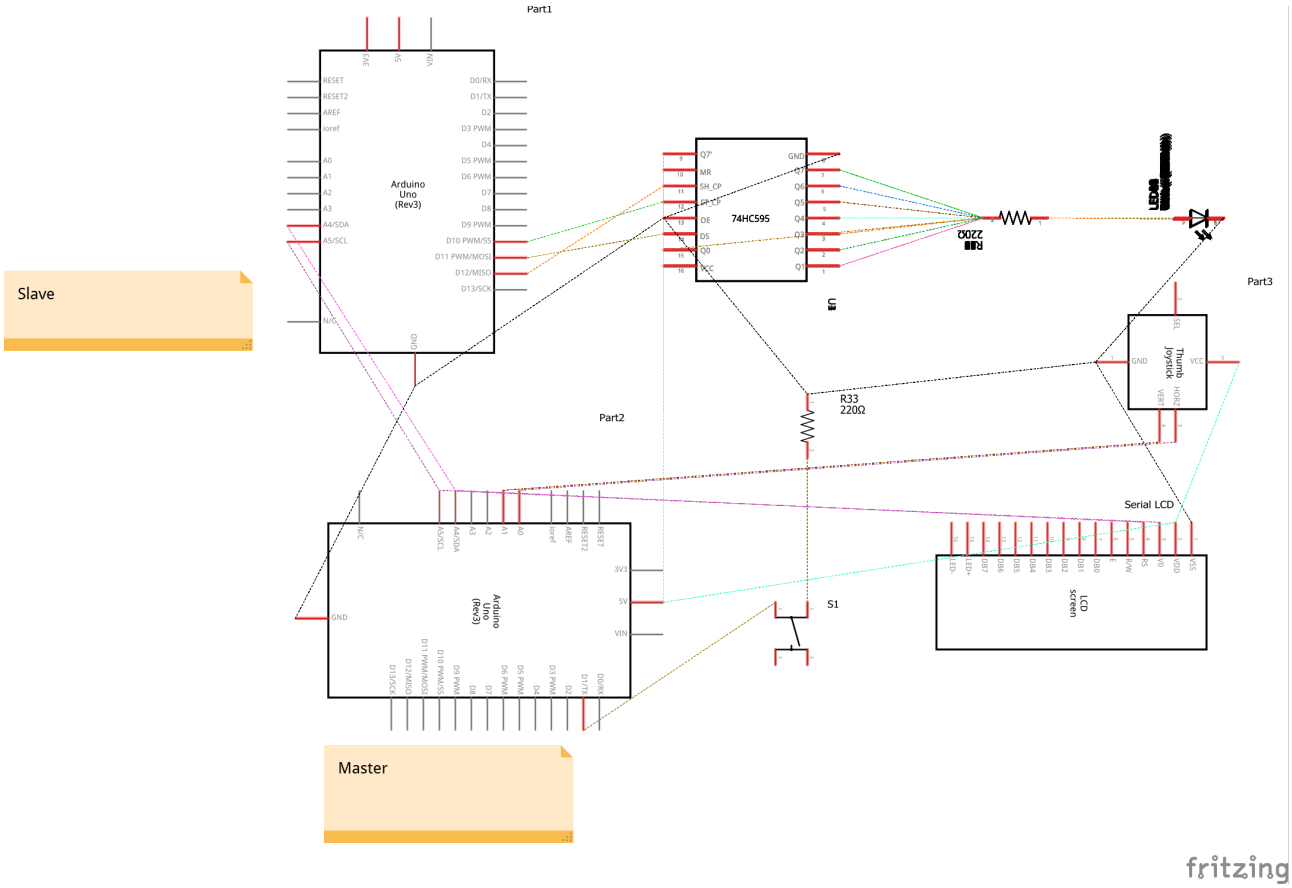
Arrian Chi
Robotics
Professor Mortezaie
22 May 2019

Final Project Report on Sorry Game

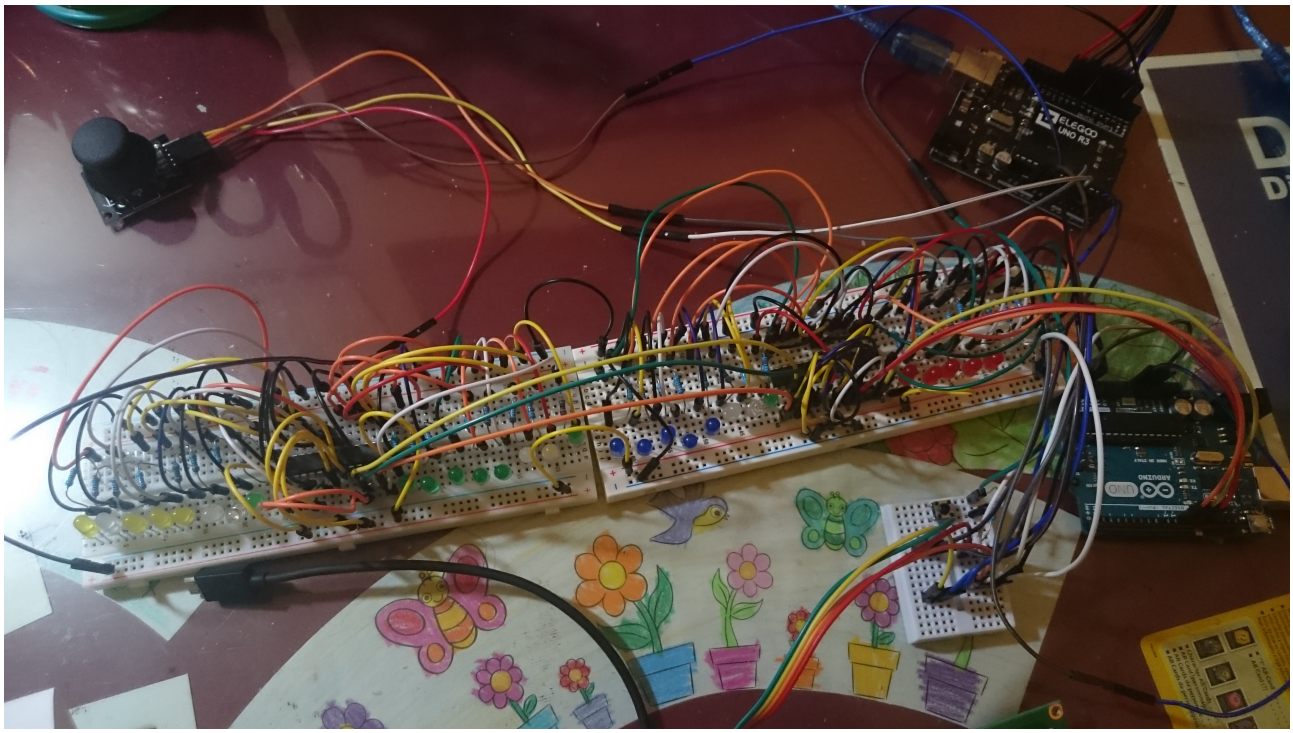
Abstract - **The idea of choosing to make a sorry game for my final project came to me when I was thinking about incorporating as many parts as possible. I wanted to use multiple LEDs to display something as well as recieve input from the user. This is how I came up with the concept of creating Sorry. My output composes of LEDS and shift registers that are controlled by a slave arduino and a serial LCD. Due to limited supplies and hardware limitations, I used 32 LEDs to represent 32 spaces of the board, and 12 of them are homes for the pawns. My input composes of a button, joystick and keypad all controlled by a master arduino. I will explain how my project is operated below.**



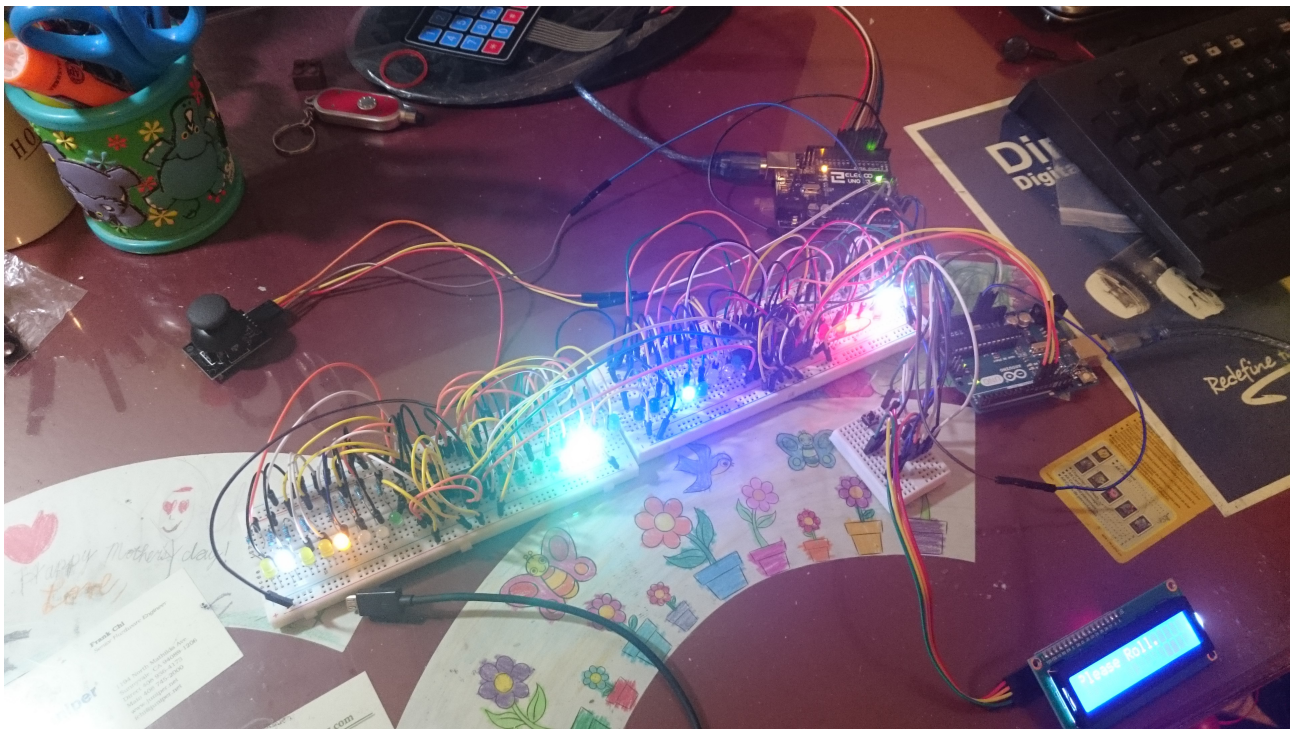
A visual of the project



A schematic diagram of my project.



The game in real life



The game in action!

Code will be supplied at the end of the report.

In the code, I have divided the problem into many functions. However, it is best to divide my code into 4 parts. But before explaining the program, it is best to show what libraries I used and why I used them. First, I included the KeyPad library. This was needed to retrieve keypad input in the beginning of the program. It allows the program to map chars in a char array to keys on the keypad. Next, I included the LiquidCrystal_i2c library. This is needed to allow i2c communication between the master arduino and the serial LCD. Then my program can display prompts and allow the user to interface with the system. Finally, I included the Wire library. This is required for the master arduino to communicate with the slave arduino. The slave arduino is responsible for displaying the board state and is the way the system communicates with the players.

The first part of my program is to get the number of players and names of each player. For this, I used a button that I have attached to interrupt. The user enters the number of players into the keypad and if valid, the LCD prompts the user to enter a name for each player. I have set the keypad to be able to switch keypad mappings when A, B, C are pressed. In this way, I allow the user to type in all letters and numbers. I also included a backspace option when the user presses D. This allows the user to change their entry if there are any mistakes. When the user types, the character is stored in a buffer and displayed to the lcd. A counter is incremented everytime the user types. The user cannot type forever because I had set a character limit for the name for display purposes. For backspacing, I replaced the current character with a space and decremented the counter. In this way, user can enter the

name properly. Once the name is submitted, I had included a function to remove the trailing whitespace from the name before being stored into the names array. The program moves to the next player until all player's names are retrieved.

The second part of my master code is getting the roll and the pawn from the user. By now, the game has already started. Each person starts out by pressing the button to roll. Once the number is determined, the player can select the pawn they would like to move by moving the joystick in the x direction. While selection, the user may see the current state of the board by moving the joystick in the y direction. This can be done because I used a while loop for this step. At every pass thorugh the loop, the program read from the joystick and decides if the joystick has moved. After the decision, the program updates the board to its necessary state.

After the selection of the pawns, the program computes the board's state. I allowed the pawn to move one space at a time in order to animate the movement. This proved to be helpful because I was also able to check if the pawn would possibly land in the homes of other player's pawns and make necessary steps to prevent this. In order to update the board, the program updates the position of the pawn in the pawn position array. It checks if the position is within the range of the homes of the players and also changes the value if the pawn passes the edge of the board. The program loops through the array and updates a long int by setting the bit at each position to one. Then it separates this value into 4 bytes and places it in a display array, whose values are sent to the slave arduino. Upon recieving input, the slave arduino updates the states of the shift registers and pushes these bytes onto them. Because I daisychained the shift registers, the leds show the binary pattern of the original long int. By doing this, at every pass of the loop, the program can animate the movement even when the pawn has to pass multiple homes and pass the edge all on the same turn.

Finally, at the end of the turn, there are checks made by the program. First, the program checks for a sorry. The program iterates through all the positions of the other player's pawn's positions and reports a sorry if there is one. Looping through the array like this allows the program to catch multiple sorrys in one turn. The old pawn is moved back home and is required to make another journey across the board. The next and most important step is to check for a win. Since it is only possible to win if it is your turn, the program only checks the pawn positions of the current player and whether the pawn has passed the edge of the board. If the program counts that all pawns have done so, the win is shown and the program is stuck in an infinite loop. The only way to get out of the loop is to restart the program.

All in all, this project taught me many new things. To start, I learned more about c and its capabilities. I learned about pointers and how to use them to loop through an array. This can be seen in the function trimWhitespace. I also learned that like other languages, C also has a switch-case structure. I used this when I recieved the number of players because it is better than a if else structure in terms of length. Another thing I learned is that I can manipulate bits in any type of variable and change it into the

value I want. This is seen in the function updateBoard() where I set the bits of the long at the pawn positions to one and divided the long into 4 bytes that are used to display the board. Not only did I learn software, I learned how to wire hardware. For instance, I learned that I can daisy chain many shift registers together to reduce the number of digital pins I use from 16 to 4. I also learned how to connect components for i2c communication. This is essential for the program because I had to save pins and separate my inputs and outputs. Even though I learned mcu from this project, there were some things I hope to improve in this project. One is to find a way to make organize the LEDs. With the mess of wires on my hardware, it is hard for players and people to see what is going on the board and find where the button is. I would also like to switch my LEDs into Rgb LEDs. It is very confusing to see different colored LEDs light up. With my current hardware, it is hard to keep track of whose LED is which when another player is selecting the pawn. Finally, I believe my software algorithms are inefficient. Though it is a nice approach, the moveOne() function always loops twice for each roll. The big O of this is O(NM) where N is the number of players and M is the number of pawns. This is bad because if the number of players and pawns were to both increase, the program would need to run for more than a couple seconds in between each move. In addition to that, only one pawn is moving, so there is no real need to update the entire board. So in the end, I believe I have learned much about software and hardware when doing this project. But I believe I can do better by improving this project.

Attached below is my code:

ProjectMaster.ino

```
#include <LiquidCrystal_I2C.h>
#include <Wire.h>
#include <Keypad.h>

// Keypad constants
const byte ROWS = 4;
const byte COLS = 4;
char hexaKeys1[ROWS][COLS] = {
  {'1', '2', '3', 'a'},
  {'4', '5', '6', 'b'},
  {'7', '8', '9', 'c'},
  {'Y', '0', 'Z', 'd'}
};

char hexaKeys2[ROWS][COLS] = {
  {'A', 'B', 'C', 'a'},
  {'D', 'E', 'F', 'b'},
  {'G', 'H', 'I', 'c'},
```

```
  {'J', 'K', 'L', 'd'}
};

char hexaKeys3[ROWS][COLS] = {
  {'M', 'N', 'O', 'a'},
  {'P', 'Q', 'R', 'b'},
  {'S', 'T', 'U', 'c'},
  {'V', 'W', 'X', 'd'}
};

// Digital Pins
byte rowPins[ROWS] = {11, 10, 9, 8};
byte colPins[COLS] = {7, 6, 5, 4};
int latchPin = 10;
int clockPin = 12;
int dataPin = 11;
int sw = 2;

// Analog Pins
int xcoord = 0;
int ycoord = 1;

// Initializations
Keypad customKeypad = Keypad(makeKeymap(hexaKeys1), rowPins, colPins,
ROWS, COLS);
LiquidCrystal_I2C lcd(0x27, 16, 2);
// For the following array values, they correspond to red, blue, green, yellow
char buf[10] = "         ";
char * names[4] = {};
int pawn_homes[4] = {2, 10, 18, 26};
int pawn_pos[4][3] = {{},{},{},{}}; // row 1: red, row 2: blue, etc...
bool pawn_pass[4][3] = {{false, false, false},
                        {false, false, false},
                        {false, false, false},
                        {false, false, false}};

// Helper variables
int numplayers = 0;
int numPawns = 3;
int charcount = 0;

int xnorm;
int ynorm;
long lint = 0;
word high_b = 0;
```

```
word low_b = 0;
byte* leds[4];

boolean waitingForInput = false;
boolean switchOn = false;

int curr_player = 0;
int curr_pawn = 0;
int currRoll = 0;
long boardstate = 0;

// Adds characters to LCD and buffer
void addToLine1(char customKey)
{
   buf[charcount] = customKey;
   lcd.setCursor(0, 1);
   lcd.print(buf);
   if (charcount != 9) {charcount++;};
}

// Strips trailing whitespace
void trimWhitespace()
{
  char *end;

  // Trim trailing space using a pointer
  end = buf + 8;
  while(end > buf && isspace((unsigned char)*end)) end--;

  // Write new null terminator character
  end[1] = '\0';
}

// Backspaces all characters
void backSpace()
{
   if (charcount != 0) {charcount--;};
   buf[charcount] = ' ';
   lcd.setCursor(0, 1);
   lcd.print(buf);
}

// Turns switch on if waiting for reply
void turnSwitchOn()
{
```

```
  if (waitingForInput){ switchOn = true;}
}

// Gets number of players
void getPlayers()
{
  while (true)
  {
    char customKey = customKeypad.getKey();
    if (customKey)
      {
      switch (customKey)
      {
        case '2':
          numplayers = 2;
          break;
        case '3':
          numplayers = 3;
          break;
        case '4':
          numplayers = 4;
          break;
        default:
          lcd.setCursor(0, 0);
          lcd.print("Invalid Number.  ");
          lcd.setCursor(0, 1);
          lcd.print("Try Again");
          delay(3000);
          break;
      }
    }
    if (numplayers != 0)
    {
      break;
    }
  }
}

// Gets names of players
void getNames()
{
 for (int i = 0; i < numplayers; i++)
 {
    while (true)
    {
```

```
      waitingForInput = true;
      if (switchOn == true)
      {
       trimWhitespace();
       names[i] = strdup(buf);
       memset(buf, ' ', 9*sizeof(char));
       lcd.setCursor(0, 1);
       lcd.print(buf);
       charcount = 0;
       switchOn = false;
       waitingForInput = false;
       break;
      }
      char customKey = customKeypad.getKey();
      if (customKey)
      {
       if (customKey == 'a') { customKeypad.begin(makeKeymap(hexaKeys1)); }
       else if (customKey == 'b') { customKeypad.begin(makeKeymap(hexaKeys2)); }
       else if (customKey == 'c') { customKeypad.begin(makeKeymap(hexaKeys3)); }
       else if (customKey == 'd') { backSpace(); }
       else
       {
        if (charcount < 9)
        {
         addToLine1(customKey);
        }
       }
      }
    }
  }
}

// Sends values in led array to slave arduino
void sendLEDArray()
{
  Wire.beginTransmission(9);
  for (int i = 0; i < 4; i++)
  {
    Wire.write((byte)leds[i]);
  }
  Wire.endTransmission();
}

// Gets Roll from current player
void getRoll()
```

```
{
  lcd.clear();
  lcd.print("Please Roll.");
  waitingForInput = true;
  while (true)
  {
    delay(100);
    if (switchOn)
    {
      currRoll = (int) random(1,13);
      switchOn = false;
      break;
    }
  }
  waitingForInput = false;
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("You rolled a ");
  lcd.print(currRoll);
  delay(2000);
}

// Initialize pawns
void initializeBoard()
{
  for (int i = 0; i < numplayers; i++)
  {
    for (int j = 0; j < numPawns; j++)
    {
      pawn_pos[i][j] = pawn_homes[i];
    }
  }
  // set lint
  updateBoard();
  sendLEDArray();
}

// Display Board for debugging purposes
void DisplayBoard()
{
  for (int i = 0; i < numplayers; i++)
  {
    for (int j = 0; j < numPawns; j++)
    {
      Serial.print(pawn_pos[i][j]);
```

```
      Serial.print(" ");
    }
    Serial.println();
  }
}

// Updates lint for player
void updateForPlayer(int player)
{
  for (int j = 0; j < numPawns; j++)
  {
    bitSet(lint, pawn_pos[player][j]);
  }
}

// Updates lint for all players
void updateBoard()
{
  lint = 0;
  for (int i = 0; i < numplayers; i++)
  {
    updateForPlayer(i);
  }
  updateLightArray();
}

// Update Led Array
void updateLightArray()
{
  int i = 0;
  high_b = lint >> 16;
  low_b = (word) lint;
  leds[i] = highByte(high_b);
  i++;
  leds[i]= lowByte(high_b);
  i++;
  leds[i] = highByte(low_b);
  i++;
  leds[i]= lowByte(low_b);
}

// Sets the lint for player to 0
void setPlayersToZero(int player)
{
  for (int j = 0; j < numPawns; j++)
```

```
  {
   bitClear(lint, pawn_pos[player][j]);
  }
}

// Select Pawn
void getSelectedpawn()
{
 curr_pawn =  0;
 lint = 0;

 updateForPlayer(curr_player);
 updateLightArray();
 sendLEDArray();
 waitingForInput = true;
 lcd.setCursor(0,1);

 while (true)
 {

  int x = analogRead(xcoord);
  int y = analogRead(ycoord);

  if (x > xnorm + 60)
  {
   updateForPlayer(curr_player);
   curr_pawn = (curr_pawn + 1) % numPawns;
  }
  else if (x < xnorm - 60)
  {
   updateForPlayer(curr_player);
   if (curr_pawn == 0) curr_pawn = numPawns-1;
   else curr_pawn = (curr_pawn - 1) % numPawns;
  }
  lcd.setCursor(0,1);
  lcd.print("Select a pawn: ");
  lcd.setCursor(15,1);
  lcd.print(curr_pawn);

  //Allow seeing boardstate
  if (y > ynorm + 60 or y < ynorm - 60)
  {
   updateBoard();
   sendLEDArray();
  }
```

```
    else
    {
      for (int i = 0; i < 4; i++)
      {
        if (i == curr_player) continue;
        setPlayersToZero(i);
      }
      //Toggle pawn's light
      bitWrite(lint, pawn_pos[curr_player][curr_pawn], not bitRead(lint,
pawn_pos[curr_player][curr_pawn]));
      updateLightArray();
      sendLEDArray();
    }

    if (switchOn)
    {
      switchOn = false;
      if (pawn_pos[curr_player][curr_pawn] == pawn_homes[curr_player] and
pawn_pass[curr_player][curr_pawn])
      {
        lcd.setCursor(0,1);
        lcd.print("Invalid Pawn    ");
        delay(2000);
      }
      else
      {
        waitingForInput = false;
        break;
      }
    }
    delay(250);
  }
}

// Curr pawn moves one space
bool moveOne()
{
  int test_space = pawn_pos[curr_player][curr_pawn] + 1;
  // Check if pawn is going to go over
  if (test_space >= 32)
  {
    test_space -= 32;
    pawn_pass[curr_player][curr_pawn] = true;
  }
  // Check if pawn is going to be at a home
```

```
  for (int i = 0; i < 4; i++)
  {
   if (pawn_homes[i] - 2 <= test_space  and test_space <= pawn_homes[i])
   {
    if (i == curr_player)
    {
     if (test_space == pawn_homes[curr_player])
     {
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print(names[curr_player]);
      lcd.print(" ");
      lcd.print(curr_pawn);
      lcd.setCursor(0,1);
      lcd.print("went home.");
      pawn_pos[curr_player][curr_pawn] = test_space;
      updateBoard();
      sendLEDArray();
      delay(2000);
      return true;
     }
    }
    else
    {
     test_space = pawn_homes[i] + 1;
    }
   }
  }
  // Update everything
  pawn_pos[curr_player][curr_pawn] = test_space;
  updateBoard();
  sendLEDArray();
  return false;
}

void setup() {
  Wire.begin();
  Serial.begin(9600);
  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.noAutoscroll();
  customKeypad.setDebounceTime(100);
  attachInterrupt(digitalPinToInterrupt(sw), turnSwitchOn, FALLING);
  xnorm = analogRead(xcoord);
```

```
  ynorm = analogRead(ycoord);

  // Start receiving input
  lcd.print("How many players?");
  getPlayers();
  initializeBoard();

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Enter your name: ");
  getNames();
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.println("GAME START      ");
  delay(5000);
}

void loop() {
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(names[curr_player]);
  lcd.print("'s Turn");
  delay(1750);
  getRoll();
  getSelectedpawn();

  // Move the pawn step by step
  for (int j = 0; j < currRoll; j++)
  {
    // Stop when the roll is done or the pawn reaches home
    bool pawnStop = moveOne();
    delay(500);
    if (pawnStop) break;
  }

  // Check if there is a sorry
  for (int i = 0; i < numplayers; i++)
  {
    if (i == curr_player)
    {
      continue;
    }
    for (int j = 0; j < numPawns; j++)
    {
      if (pawn_pos[curr_player][curr_pawn] == pawn_pos[i][j])
```

```
      {
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("SORRY!!");
        delay(5000);
        lcd.clear();
        lcd.print(names[curr_player]);
        lcd.print(" ");
        lcd.print(curr_pawn);
        lcd.print(" bmps");
        lcd.setCursor(0,1);
        lcd.print(names[i]);
        lcd.print(" ");
        lcd.print(j);
        delay(3000);
        pawn_pos[i][j] = pawn_homes[i];
        updateBoard();
        sendLEDArray();
      }
    }
  }

  // Check if the player has won
  int homeCount = 0;
  for (int i = 0; i < numPawns; i++)
  {
    if (pawn_pos[curr_player][i] == pawn_homes[curr_player] and
pawn_pass[curr_player][i])
    {
      homeCount++;
    }
  }

  if (homeCount == numPawns)
  {
    lcd.clear();
    while (true)
    {
      lcd.setCursor(0,0);
      lcd.print(names[curr_player]);
      lcd.print(" Wins!");
    }
  }
  curr_player = (curr_player + 1) % numplayers;
}
```

ProjectSlave.ino

```
#include <Wire.h>

int latchPin = 10;
int clockPin = 12;
int dataPin = 11;

byte* leds[4];

void updateShiftRegister()
{
  digitalWrite(latchPin, LOW);
  for (int i = 0; i < 4; i++)
  { shiftOut(dataPin, clockPin, MSBFIRST, leds[i]);}
  digitalWrite(latchPin, HIGH);
}

void receiveEvent(int bytes)
{
  for (int i = 0; i < bytes; i++)
  {
    leds[i] = Wire.read();
  }
  updateShiftRegister();
}

void setup() {
  Wire.begin(9);
  Wire.onReceive(receiveEvent);
  Serial.begin(9600);
  pinMode(latchPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
}
void loop()
{
  ;
}
```