University of Washington
EE 299 Lab 3
Second Steps to Design

Mitchell Szeto
Bert Zhao
Feifan Qiao

TABLE OF CONTENTS

ABSTRACT **2**

INTRODUCTION **2**

DISCUSSION OF LAB 2

Design Specification 2

Hardware Implementation 3

Software Implementation 3

SUMMARY AND CONCLUSION **4**

APPENDICES **5**

ABSTRACT

In this lab, we create a variation of the game minesweeper. We create both a one player and two player game. Major parts of this design include player movement, board generation, and two player I2C. Both the hardware and software designs of this lab are outlined in this report.

INTRODUCTION

This third lab focuses on a more complex I2C problem, but only contains a high level description of the project. The main goal is to build a minesweeper like game that two Arduinos can play. But, a large part is learning how to design a project with minimal specifications. In comparison to lab two, the requirements are a lot more flexible which means that we need to learn how to go through the entire design process with only knowing the end product.

DISCUSSION OF LAB

Design Specification

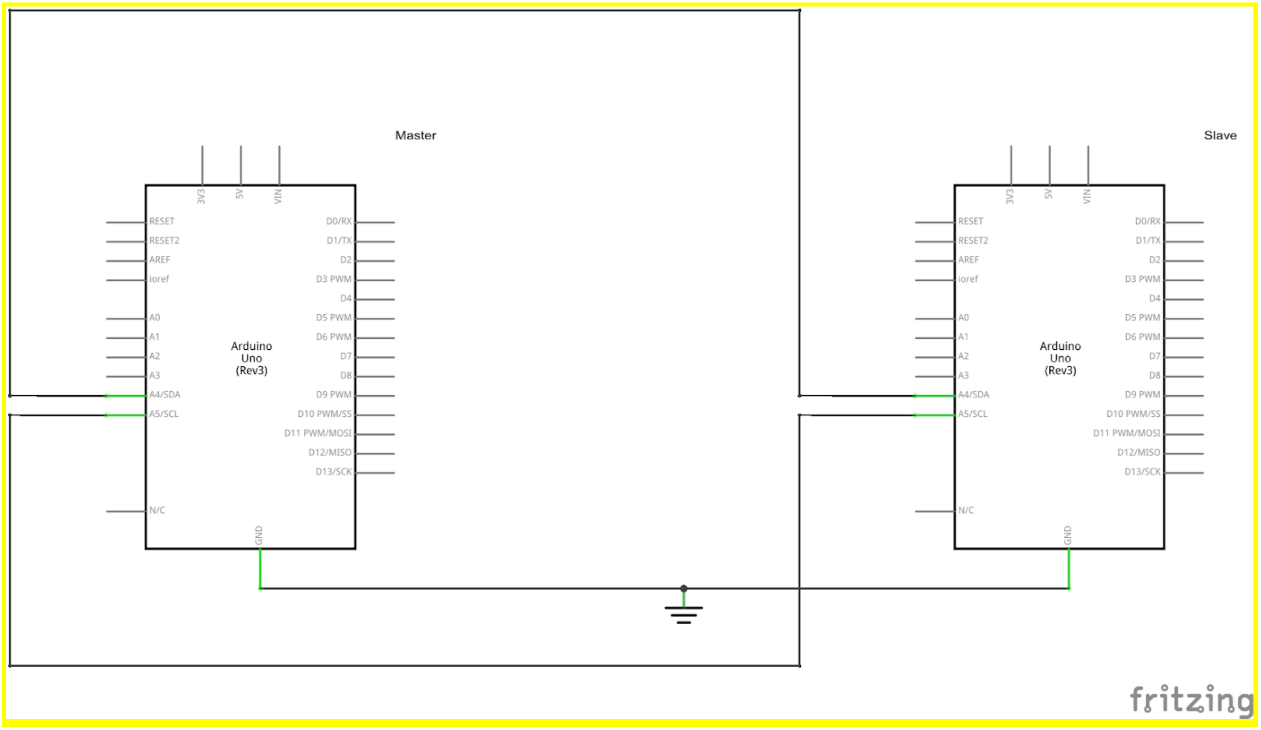
The goal of this lab is to build a minesweeper like game in Arduino. In this game, a player starts from one corner of the board and wins if the player successfully moves to the opposite corner without losing all of their lives. The player loses lives by stepping on bombs, which cannot be seen. The 4x4 board will be displayed through serial monitor. The player will be represented with an O, the hit bombs will be represented by X, and the rest of the board will be represented with *. Player movement will also be inputted through serial monitor. The user will enter in w, a, s, or d to move up, left, down, or right respectively. Many errors can occur in this process, so this program needs to contain error control to handle wrong and illegal user input. If the user enters in wrong/illegal input, the user will be notified in the serial monitor output.

The lab is split into two main parts. The first part consists of creating a single player game in which four bombs are randomly placed on the board. The second part builds on the game created previously to make a two player game. In this part, two Arduinos will communicate with each other through I2C. Architecture of the the two-player game is best described by option 2 in the demo sheet. Each of the device stores their own version of the board and player information. The advantage of this architecture is that the programs stored on each board retain the form of the original single-player game as much as possible. With minimum modifications, each individual board can be played in single-player mode. This architecture provides great versatility. When player 1 enters bomb location in device 1, that information is then delivered to device 2 and gets stored as game board for player 2, and vice versa. After both players have finished placing the bombs, they then take turns moving their locations on the game boards. When one player successfully reaches the other end of the board, the game prompts that player as the winner while displaying a defeat message for the other player. In a single player game, the user is given the option to start a new game.

Hardware Implementation

For part one, the hardware setup is very simple. A single Arduino Uno is connected to a computer through USB. We don’t need any other peripherals because the input and output is handed in the serial monitor.

For the second part, the two Arduinos, one as master device and the other one as slave device, are connected via analog pin 4 and 5 and communicate through the I^2C protocol. Both Arduinos use the same computer as ground.



Software Implementation

The single player game consists of four different parts: board management, bomb placement, user input, and display. The board is depicted through a 4x4 2-dimensional array of ints called matrix, each index of the matrix depicts a square on the board. The values within the matrix represent a different type of piece that could be on the board.

Value	Description	Printed Character
0	Empty Space	*
1	Hidden Bomb	*
2	Hit/exposed Bomb	X
3+	Player	O

The game uses these values for game logic and displaying the game on Serial Monitor. Each time the player moves, the values are adjusted and the game tells the user if they hit a bomb and lose a life, win the game, or lose the game.

To place random bombs on the board, we created a random seed that randomly generated a two numbers between 0 and 3. After that, a bomb is placed at that coordinate position on the board. If there is already a bomb at that position or the coordinate generated is a corner position, a new bomb position is generated.

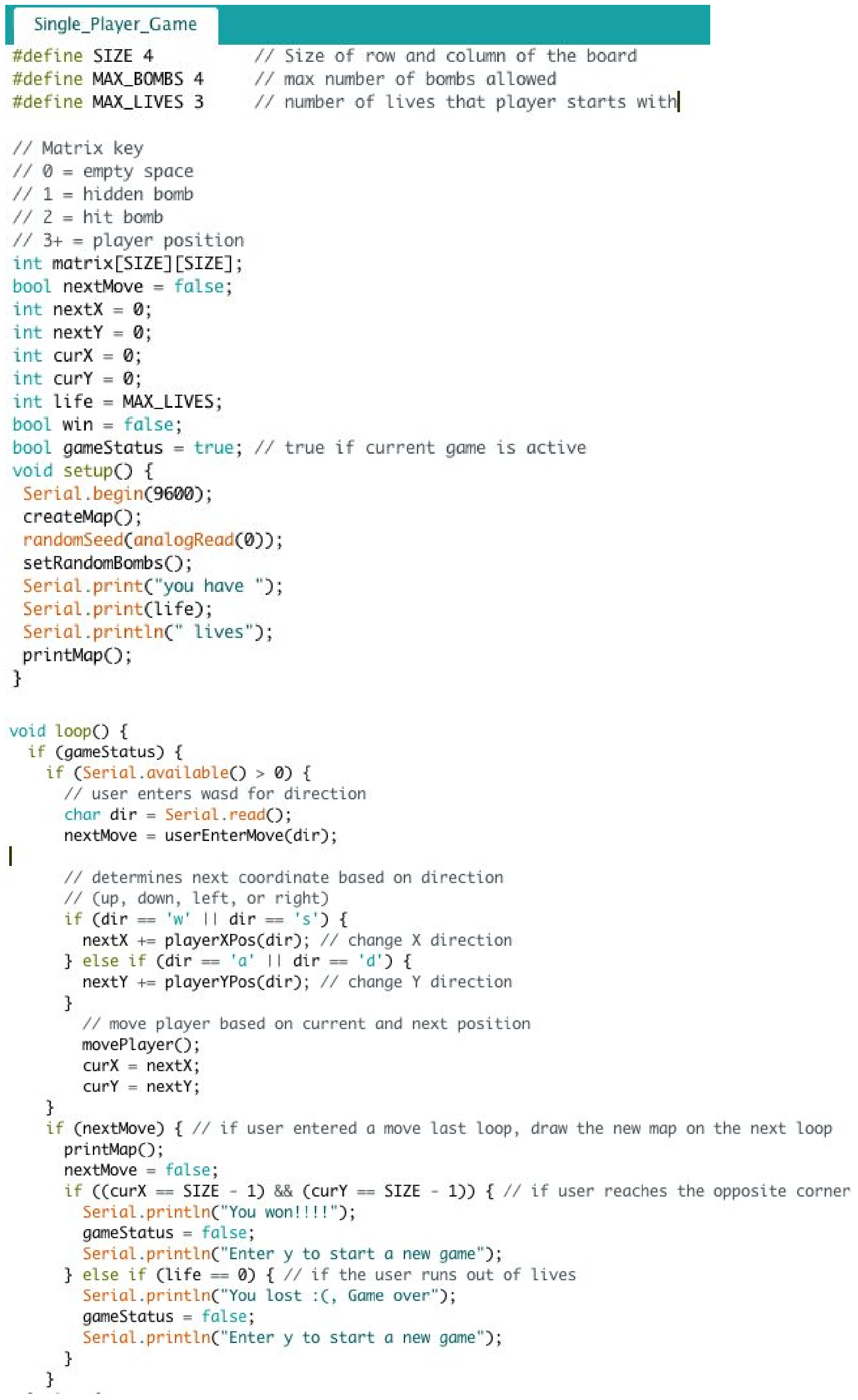
To control the player movements on the board, the user must enter w, a, s, or d for up, left, down, or right respectively. The user is free to move to any space directly adjacent to their current position. The only exception is if the player on on the edge, they will not be allowed to move off the board. The game starts an initializes the position of the player in the top left corner at (0, 0). The game is won if the player can successfully move to the bottom right corner at (3, 3).

Two-player mode inherits a lot of the control implementations from the single-player mode. Both player uses w, a, s, d to control their movement in serial monitor. There are two major differences between the two modes. The most significant difference is that bombs in multiplayer mode is placed by actual players, not random computer-generated numbers compared to the single-player mode. Before the games starts, players are prompted by the game to enter the coordinates of all the bombs. The coordinate input starts with a left arrow, followed by the row number, then a comma, followed by the column number, and finally finished by a right arrow. The coordinates of the bombs are stored in a 2-dimensional array and transferred to the other device via I2C. The other way that this is different from the single-player mode is that players in multiplayer mode take turns moving their location on board. While one player is making their move, the other player is prohibited from moving at the same time.

SUMMARY AND CONCLUSION

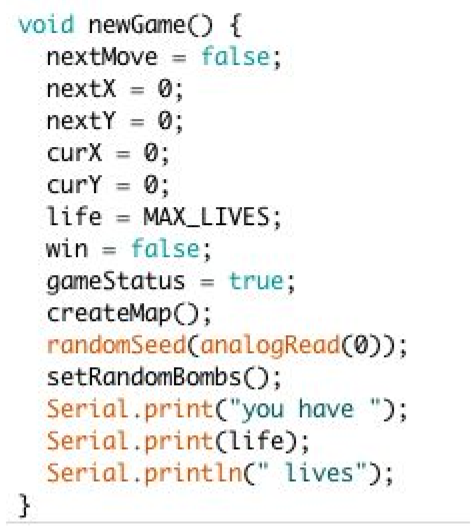
Overall, this lab was to build a one player and two player game of an adapted version of minesweeper. This lab focuses primarily on the serial monitor and I2C for the two player component. We also introduce 2-dimensional matrices for the implementation of the minefield. This is also the first lab where we work with random number generation since that is needed for the bomb placement in single player. We work with more complex outputs to the serial monitor and a larger variety of data sent between the two Arduinos. The skills used in this lab will be very useful for the final project.

APPENDICES

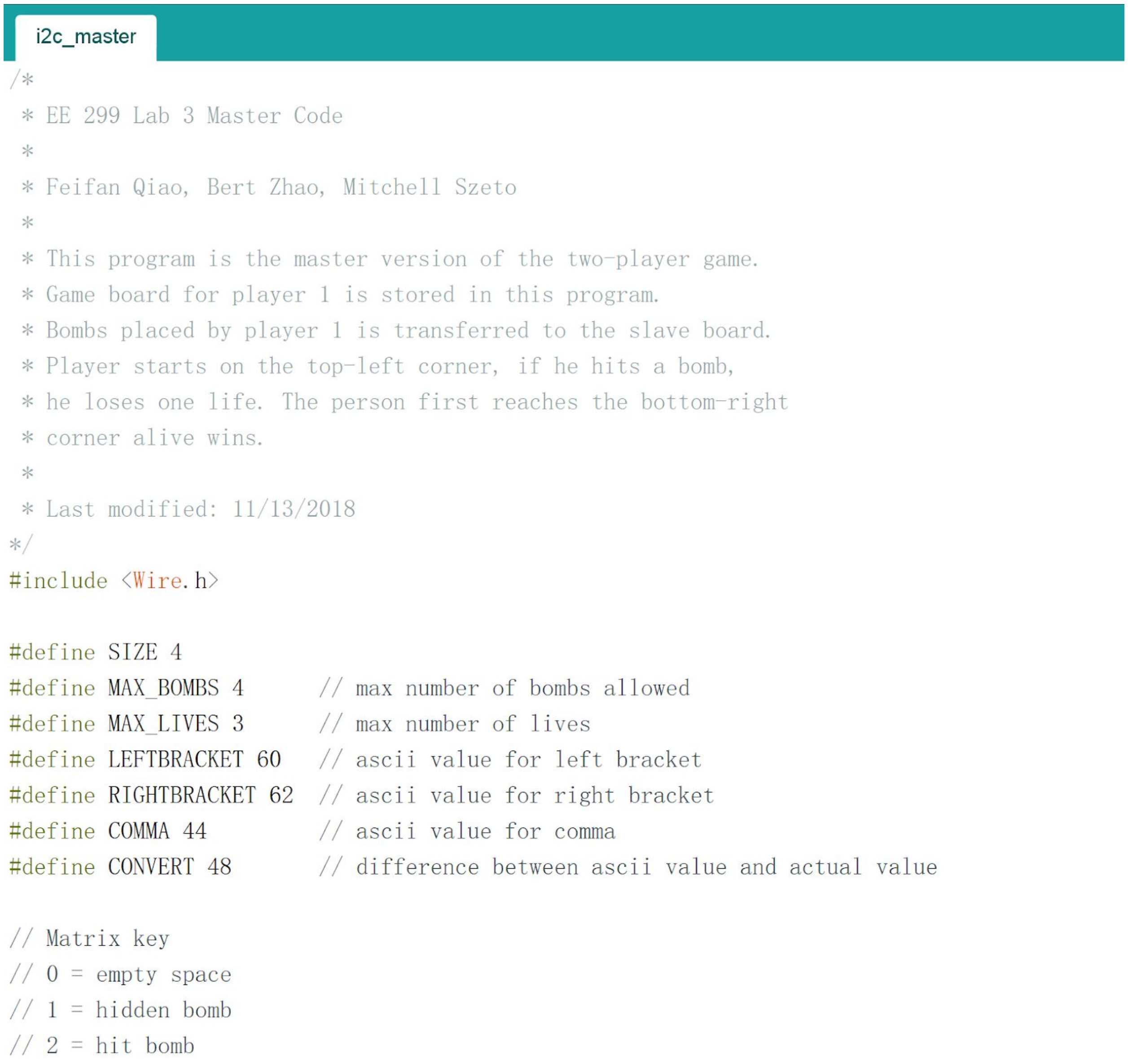

Single_Player_Game

```
#define SIZE 4          // Size of row and column of the board
#define MAX_BOMBS 4     // max number of bombs allowed
#define MAX_LIVES 3     // number of lives that player starts with

// Matrix key
// 0 = empty space
// 1 = hidden bomb
// 2 = hit bomb
// 3+ = player position
int matrix[SIZE][SIZE];
bool nextMove = false;
int nextX = 0;
int nextY = 0;
int curX = 0;
int curY = 0;
int life = MAX_LIVES;
bool win = false;
bool gameStatus = true; // true if current game is active
void setup() {
 Serial.begin(9600);
 createMap();
 randomSeed(analogRead(0));
 setRandomBombs();
 Serial.print("you have ");
 Serial.print(life);
 Serial.println(" lives");
 printMap();
}

void loop() {
  if (gameStatus) {
    if (Serial.available() > 0) {
      // user enters wasd for direction
      char dir = Serial.read();
      nextMove = userEnterMove(dir);

      // determines next coordinate based on direction
      // (up, down, left, or right)
      if (dir == 'w' || dir == 's') {
        nextX += playerXPos(dir); // change X direction
      } else if (dir == 'a' || dir == 'd') {
        nextY += playerYPos(dir); // change Y direction
      }
        // move player based on current and next position
        movePlayer();
        curX = nextX;
        curY = nextY;
    }
    if (nextMove) { // if user entered a move last loop, draw the new map on the next loop
      printMap();
      nextMove = false;
      if ((curX == SIZE - 1) && (curY == SIZE - 1)) { // if user reaches the opposite corner
        Serial.println("You won!!!!");
        gameStatus = false;
        Serial.println("Enter y to start a new game");
      } else if (life == 0) { // if the user runs out of lives
        Serial.println("You lost :(, Game over");
        gameStatus = false;
        Serial.println("Enter y to start a new game");
      }
    }
  } else {
     if (Serial.available() > 0) {
       char c = Serial.read();
       if (c == 'y') {
        newGame();
       }
     }
  }
}

// creats a new map with all empty spaces except for the player at the top left corner
void createMap() {
  for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
      if (i == 0 && j == 0) {
        matrix[i][j] = 3;
      } else {
        matrix[i][j] = 0;
      }
    }
  }
}


// Handles player movement
void movePlayer() {
  if (matrix[nextX][nextY] == 1) { // if player hits a hidden bomb
    life--;
    Serial.println("OH NO! You hit a bomb!");
    Serial.print("You have ");
    Serial.print(life);
    Serial.println(" lives remaining");
  }
  if (matrix[curX][curY] == 4) {
    matrix[curX][curY] += 1;
  }
  matrix[nextX][nextY] += 3;
  matrix[curX][curY] -= 3;
}

// Prints the current status of the map to the console
void printMap() {
  for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
      if (matrix[i][j] <= 1) {
        Serial.print("*");
      } else if (matrix[i][j] == 2) {
        Serial.print("x");
      } else if (matrix[i][j] >= 3) {
        Serial.print("o");
      }
    }
    Serial.println();
  }
  Serial.println();
}
```

```
bool userEnterMove(int c) {
  if (c == 10) {
    return true;
  } else {
    return false;
  }
}

// Returns the direction the user moves in the X direction
int playerXPos(char dir) {
  if ((dir == 'w') && (curX > 0)) {
    return -1;
  } else if ((dir == 's') && (curX < SIZE - 1)) {
    return 1;
  } else {
    return 0;
  }
}

// Returns the direction the user moves in the Y direction
int playerYPos(char dir) {
  if ((dir == 'a') && (curY > 0)) {
    return -1;
  } else if ((dir == 'd') && (curY < SIZE - 1)) {
    return 1;
  } else {
    return 0;
  }
}


// randomly sets MAX_BOMBS on the board
void setRandomBombs() {
  int numBombs = 0;
  while (numBombs < MAX_BOMBS) {
    if (setBombs(random(0, SIZE), random(0, SIZE))){
      numBombs++;
    }
  }
}

// row: row of board
// col: column of board
// tries to set the given position as a bomb
// returns true if successful and false if not successful
bool setBombs(int row, int col) {
  if (matrix[row][col] == 1 || isCorner(row, col)) {
    return false;
  } else {
    matrix[row][col] = 1;
    return true;
  }
}

// returns true if row and col make a corner
// otherwise, returns false
bool isCorner(int row, int col) {
  return (row == 0 && col == 0) || (row == 0 && col == SIZE - 1) || (row == SIZE - 1 && col == 0) || (row == SIZE -1 && col == SIZE - 1);
}
```



```
void newGame() {
  nextMove = false;
  nextX = 0;
  nextY = 0;
  curX = 0;
  curY = 0;
  life = MAX_LIVES;
  win = false;
  gameStatus = true;
  createMap();
  randomSeed(analogRead(0));
  setRandomBombs();
  Serial.print("you have ");
  Serial.print(life);
  Serial.println(" lives");
}
```


i2c_master

```
/*
 * EE 299 Lab 3 Master Code
 *
 * Feifan Qiao, Bert Zhao, Mitchell Szeto
 *
 * This program is the master version of the two-player game.
 * Game board for player 1 is stored in this program.
 * Bombs placed by player 1 is transferred to the slave board.
 * Player starts on the top-left corner, if he hits a bomb,
 * he loses one life. The person first reaches the bottom-right
 * corner alive wins.
 *
 * Last modified: 11/13/2018
*/
#include <Wire.h>

#define SIZE 4
#define MAX_BOMBS 4       // max number of bombs allowed
#define MAX_LIVES 3       // max number of lives
#define LEFTBRACKET 60    // ascii value for left bracket
#define RIGHTBRACKET 62   // ascii value for right bracket
#define COMMA 44          // ascii value for comma
#define CONVERT 48        // difference between ascii value and actual value

// Matrix key
// 0 = empty space
// 1 = hidden bomb
// 2 = hit bomb
// 3+ = player position
int matrix[SIZE][SIZE];
int matrix2[SIZE][SIZE];  // temporily stores the bomb locations
bool nextMove = false;
int limit = 4;
int nextX = 0;
int nextY = 0;
int curX = 0;
int curY = 0;
int life = MAX_LIVES;
int bIndex = 0;
int prevData = 0;
int row = -1;
int col = -1;
int x = 0;
int z = 0;
bool status1First = true;
bool status2First = true;
// 0--player 1 setting up bombs, 1--player 2 setting up bombs,
// 2--current game active, 3--current game not active
int gameStatus = 0;
```

```
void setup() {
 Serial.begin(9600);
 Wire.begin();
 Serial.println("Player 1 now setting bombs");
 Serial.println("Enter in the form <row,col>");
 Serial.println("Please use zero indexing");
}

void loop() {
  if (gameStatus == 0) {                 // Player 1 is now setting bombs
    if (Serial.available() > 0) {
      int data = Serial.read();
      if (prevData == LEFTBRACKET) {
        row = data - CONVERT;
      } else if (prevData == COMMA) {
        col = data - CONVERT;
      }
      if (data == RIGHTBRACKET) {
        matrix2[row][col] = 1;
        if (setBombs(row, col)) {
          Serial.print("Row: ");
          Serial.println(row);
          Serial.print("Col: ");
          Serial.println(col);
          Serial.println();
          bIndex++;
        bIndex++;
        Wire.beginTransmission(4);
        Wire.write(8);                // command for giving out player 1 bomb location
        Wire.write(row);
        Wire.write(col);
        Wire.endTransmission(4);
      }
      row = -1;
      col = -1;
    }
    prevData = data;
  }
  if (bIndex == (MAX_BOMBS)) {
    gameStatus = 1;
  }
} else if (gameStatus == 1) {          // Player 2 is now setting bombs
  if (status1First) {
    Wire.beginTransmission(4);
    Wire.write(9);                     // 9 is the command to signal player 2 setting bomb
    Wire.endTransmission(4);
    status1First = false;
    delay(20000);                      // delay 20s, wait for player 2 to finish inputting bombs
    for (int i = 0; i < 16; i++) {
      Wire.requestFrom(4,1);
      while (Wire.available()) {
        matrix[x][z] = Wire.read();     // x represents row while z represents column
        z++;
        if (z == SIZE) {
          z = 0;
          x++;
        }
      }
    }
    gameStatus = 2;
  }
} else if (gameStatus == 2) {         // game is actively played
  if (status2First) {                 // this part is only run one time
    createMap();
    Serial.print("you have ");
    Serial.print(life);
    Serial.println(" lives");
    status2First = false;
    Wire.beginTransmission(4);
    Wire.write(7);                    // command for gameStatus 2
    Wire.endTransmission(4);
  }
  if (Serial.available() > 0) {
    // user enters wasd for direction
    char c = Serial.read();
    nextMove = userEnterMove(c);
    char dir = moveCmd(c);
  // determines next coordinate based on direction
  // (up, down, left, or right)
  if (dir == 'u' || dir == 'd') {
    nextX += playerXPos(dir); // change X direction
  } else if (dir == 'l' || dir == 'r') {
    nextY += playerYPos(dir); // change Y direction
  }
    // move player based on current and next position
    movePlayer();
    curX = nextX;
    curY = nextY;
}
if (nextMove) {                                    // if user entered a move last loop, draw the new map on the next loop
  printMap();
  nextMove = false;
  if ((curX == SIZE - 1) && (curY == SIZE - 1)) { // if user reaches the opposite corner
    Serial.println("Player 1 won");
    gameStatus = 3;
    Wire.beginTransmission(4);
    Wire.write(6);                               // command for end of game situation
    Wire.write('w');                             // signals a victory for player 1
    Wire.endTransmission(4);
  } else if (life == 0) {                        // if the user runs out of lives
    Serial.println("game over");
    Wire.beginTransmission(4);
    Wire.write(6);                               // command for end of game situation
    Wire.write('l');                             // signals a defeat for player 1
    Wire.endTransmission(4);
```

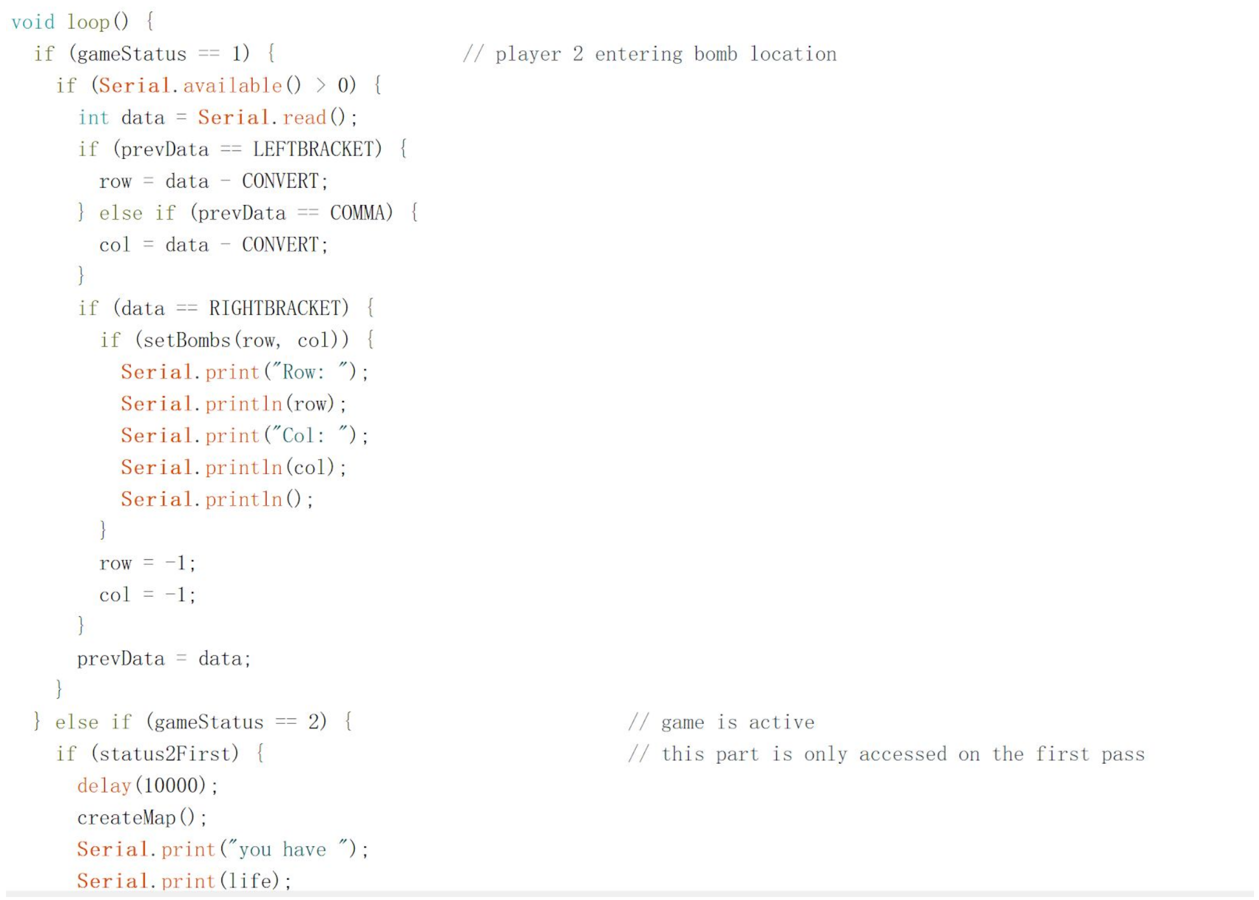
```
        gameStatus = 3;
      }
    }
  }
}

// creats a new map with the player at the top left corner
void createMap() {
  matrix[0][0] = 3;
}

// Handles player movement
void movePlayer() {
  if (matrix[nextX][nextY] == 1) {          // if player hits a hidden bomb
    life--;
    Serial.print("you have ");
    Serial.print(life);
    Serial.println(" lives remaining");
  }
  if (matrix[curX][curY] == 4) {
    matrix[curX][curY] += 1;
  }
  matrix[nextX][nextY] += 3;
  matrix[curX][curY] -= 3;
}
```

```
// Prints the current status of the map to the console
void printMap() {
  for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
      if (matrix[i][j] <= 1) {
        Serial.print("*");
      } else if (matrix[i][j] == 2) {
        Serial.print("x");
      } else if (matrix[i][j] >= 3) {
        Serial.print("o");
      }
    }
    Serial.println();
  }
}

bool userEnterMove(int c) {
  if (c == 10) {
    Serial.println("Player 2's turn");
    return true;
  } else {
    return false;
  }
}
```

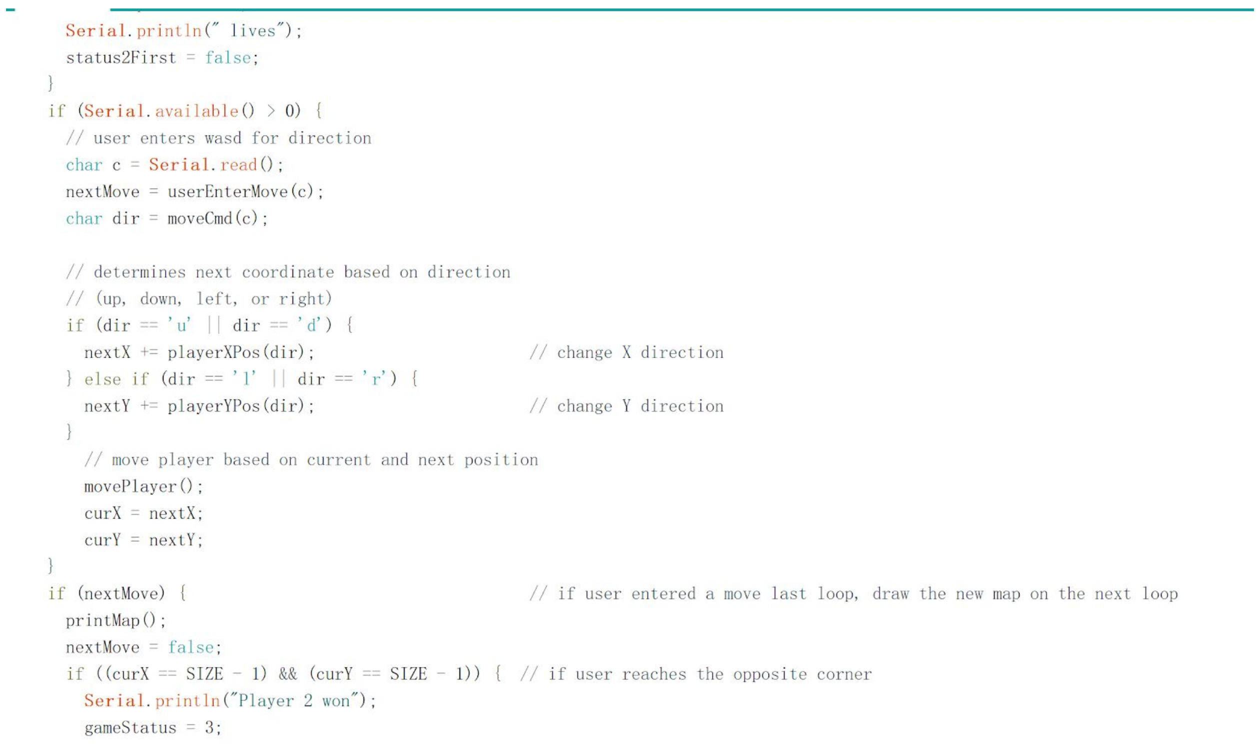
```
// Prints direction user moved to console
char moveCmd(char c) {
  char moveDirection = ' ';
  if (c == 'w') {
    moveDirection = 'u';
    Serial.println("up");
  } else if (c == 'a') {
    moveDirection = 'l';
    Serial.println("left");
  } else if (c == 's') {
    moveDirection = 'd';
    Serial.println("down");
  } else if (c == 'd') {
    moveDirection = 'r';
    Serial.println("right");
  }
  return moveDirection;
}

// Returns the direction the user moves in the X direction
int playerXPos(char dir) {
  if ((dir == 'u') && (curX > 0)) {
    return -1;
  } else if ((dir == 'd') && (curX < SIZE - 1)) {
    return 1;
  } else {
    return 0;
  }
}

// Returns the direction the user moves in the Y direction
int playerYPos(char dir) {
  if ((dir == 'l') && (curY > 0)) {
    return -1;
  } else if ((dir == 'r') && (curY < SIZE - 1)) {
    return 1;
  } else {
    return 0;
  }
}

// randomly sets MAX_BOMBS on the board
void setRandomBombs() {
  int numBombs = 0;
  while (numBombs < MAX_BOMBS) {
    if (setBombs(random(0, SIZE), random(0, SIZE))){
      numBombs++;
    }
  }
}

// row: row of board
// col: column of board
// tries to set the given position as a bomb

bool setBombs(int row, int col) {
  if (matrix[row][col] == 1 || isCorner(row, col)) {
    return false;
  } else {
    matrix[row][col] = 1;
    return true;
  }
}

// returns true if row and col make a corner
// otherwise, returns false
bool isCorner(int row, int col) {
  return (row == 0 && col == 0) || (row == 0 && col == SIZE - 1) || (row == SIZE - 1 && col == 0) || (row == SIZE -1 && col == SIZE - 1);
}
```

i2c_slave

```
/*
 * EE 299 Lab 3 Slave Code
 *
 * Feifan Qiao, Bert Zhao, Mitchell Szeto
 *
 * This program is the slave version of the two-player game.
 * Game board for player 2 is stored in this program.
 * Bombs placed by player 2 is transferred to the master board.
 * Player starts on the top-left corner, if he hits a bomb,
 * he loses one life. The person first reaches the bottom-right
 * corner alive wins.
 *
 * Last modified: 11/13/2018
*/
#include <Wire.h>

#define SIZE 4
#define MAX_BOMBS 4          // max number of bombs allowed
#define MAX_LIVES 3          // mas number of lives
#define LEFTBRACKET 60       // ascii value for left bracket
#define RIGHTBRACKET 62      // ascii value for right bracket
#define COMMA 44             // ascii value for comma
#define CONVERT 48           // difference between ascii value and actual value

// Matrix key
// 0 = empty space
// 1 = hidden bomb
// 2 = hit bomb
// 3+ = player position
int matrix[SIZE][SIZE];
int matrixTemp[SIZE][SIZE];   // temporily stores the bomb location
bool nextMove = false;
int limit = 5;
int nextX = 0;
int nextY = 0;
int curX = 0;
int curY = 0;
int prevData = 0;
int row = -1;
int col = -1;
int i = 0;
int j = 0;
int life = MAX_LIVES;
bool status2First = true;
// 0--player 1 setting up bombs, 1--player 2 setting up bombs,
// 2--current game active, 3--current game not active
int gameStatus = 0;

void setup() {
  Serial.begin(9600);
  Wire.begin(4);
  Wire.onReceive(receiveEvent);
  Wire.onRequest(requestEvent);
}
```



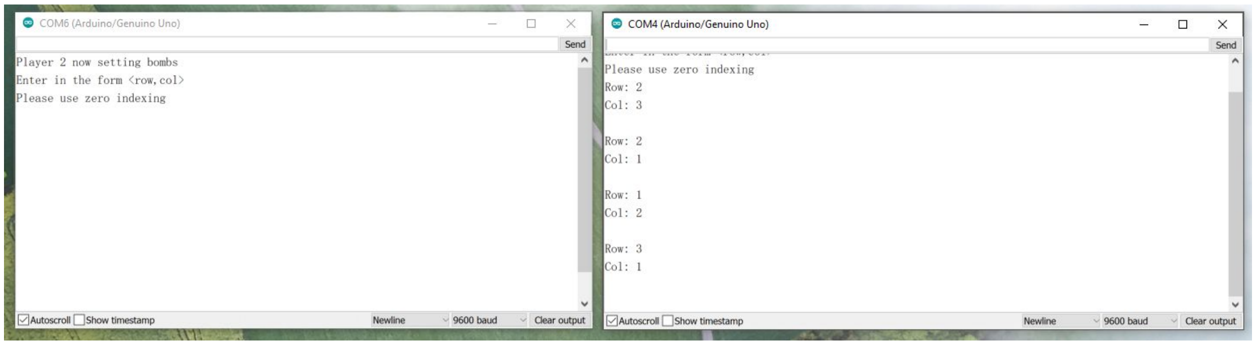
```cpp
void loop() {
  if (gameStatus == 1) {                 // player 2 entering bomb location
    if (Serial.available() > 0) {
      int data = Serial.read();
      if (prevData == LEFTBRACKET) {
        row = data - CONVERT;
      } else if (prevData == COMMA) {
        col = data - CONVERT;
      }
      if (data == RIGHTBRACKET) {
        if (setBombs(row, col)) {
          Serial.print("Row: ");
          Serial.println(row);
          Serial.print("Col: ");
          Serial.println(col);
          Serial.println();
        }
        row = -1;
        col = -1;
      }
      prevData = data;
    }
  } else if (gameStatus == 2) {                 // game is active
    if (status2First) {                         // this part is only accessed on the first pass
      delay(10000);
      createMap();
      Serial.print("you have ");
      Serial.print(life);
```



```cpp
      Serial.println(" lives");
      status2First = false;
    }
    if (Serial.available() > 0) {
      // user enters wasd for direction
      char c = Serial.read();
      nextMove = userEnterMove(c);
      char dir = moveCmd(c);

      // determines next coordinate based on direction
      // (up, down, left, or right)
      if (dir == 'u' || dir == 'd') {
        nextX += playerXPos(dir);                       // change X direction
      } else if (dir == 'l' || dir == 'r') {
        nextY += playerYPos(dir);                       // change Y direction
      }
        // move player based on current and next position
        movePlayer();
        curX = nextX;
        curY = nextY;
    }
    if (nextMove) {                                     // if user entered a move last loop, draw the new map on the next loop
      printMap();
      nextMove = false;
      if ((curX == SIZE - 1) && (curY == SIZE - 1)) {  // if user reaches the opposite corner
        Serial.println("Player 2 won");
        gameStatus = 3;
      }
    }
  }
}

// this function is called when the slave receives input
// from the master.
void receiveEvent(int howMany) {
  int data = Wire.read();
  // command to set bomb
  if (data == 9) {
    gameStatus = 1;
    Serial.println("Player 2 now setting bombs");
    Serial.println("Enter in the form <row,col>");
    Serial.println("Please use zero indexing");
  } else if (data == 8) {  // receiving bomb location from player 1
    int temp1 = Wire.read();
    int temp2 = Wire.read();
    matrix[temp1][temp2] = 1;
  } else if (data == 7) {
    gameStatus = 2;
  } else if (data == 6) {
    int temp = Wire.read();
    if (temp == 108) {
      Serial.println("Player 2 won ");
    } else if (temp == 119) {
      Serial.println("Player 1 won ");
    }
    gameStatus = 3;
  }
}

// sends the maxtrix containing bomb location entered by
// player 2 to player 1 upon request
void requestEvent() {
  Wire.write(matrixTemp[i][j]);
  j++;
  if (j == SIZE) {
    j = 0;
    i++;
  }
}

// creats a new map with the player at the top left corner
void createMap() {
  matrix[0][0] = 3;
}

// Handles player movement
void movePlayer() {
  if (matrix[nextX][nextY] == 1) {          // if player hits a hidden bomb
    life--;
    Serial.print("you have ");
    Serial.print(life);
    Serial.println(" lives remaining");
  }
  if (matrix[curX][curY] == 4) {
    matrix[curX][curY] += 1;
  }
  matrix[nextX][nextY] += 3;
  matrix[curX][curY] -= 3;
}

// Prints the current status of the map to the console
void printMap() {
  for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
      if (matrix[i][j] <= 1) {
        Serial.print("*");
      } else if (matrix[i][j] == 2) {
        Serial.print("x");
      } else if (matrix[i][j] >= 3) {
        Serial.print("o");
      }
    }
    Serial.println();
  }
}
```

```
// returns trye if the user enters a movement
bool userEnterMove(int c) {
  if (c == 10) {
    Serial.println("Player 1's turn");
    return true;
  } else {
    return false;
  }
}

// Prints direction user moved to console
char moveCmd(char c) {
  char moveDirection = ' ';
  if (c == 'w') {
    moveDirection = 'u';
    Serial.println("up");
  } else if (c == 'a') {
    moveDirection = 'l';
    Serial.println("left");
  } else if (c == 's') {
    moveDirection = 'd';
    Serial.println("down");
  } else if (c == 'd') {
    moveDirection = 'r';
    Serial.println("right");
  }
  return moveDirection;
}
```

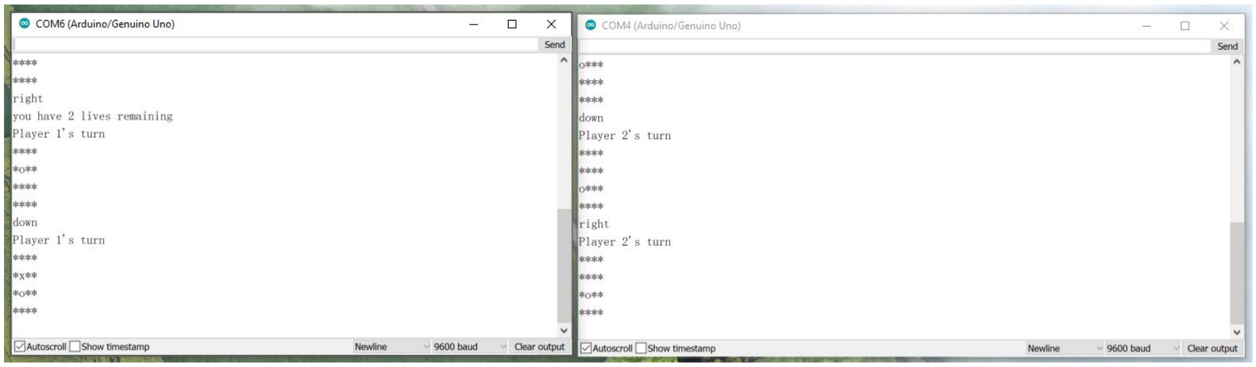
```
// Returns the direction the user moves in the X direction
int playerXPos(char dir) {
  if ((dir == 'u') && (curX > 0)) {
    return -1;
  } else if ((dir == 'd') && (curX < SIZE - 1)) {
    return 1;
  } else {
    return 0;
  }
}

// Returns the direction the user moves in the Y direction
int playerYPos(char dir) {
  if ((dir == 'l') && (curY > 0)) {
    return -1;
  } else if ((dir == 'r') && (curY < SIZE - 1)) {
    return 1;
  } else {
    return 0;
  }
}

// row: row of board
// col: column of board
// tries to set the given position as a bomb
// returns true if successful and false if not successful

bool setBombs(int row, int col) {
  if (matrixTemp[row][col] == 1 || isCorner(row, col)) {
    return false;
  } else {
    matrixTemp[row][col] = 1;
    return true;
  }
}

// returns true if row and col make a corner
// otherwise, returns false
bool isCorner(int row, int col) {
  return (row == 0 && col == 0) || (row == 0 && col == SIZE - 1) || (row == SIZE - 1 && col == 0) || (row == SIZE -1 && col == SIZE - 1);
}
```


COM6 (Arduino/Genuino Uno)

```
Player 2 now setting bombs
Enter in the form <row,col>
Please use zero indexing
```

COM4 (Arduino/Genuino Uno)

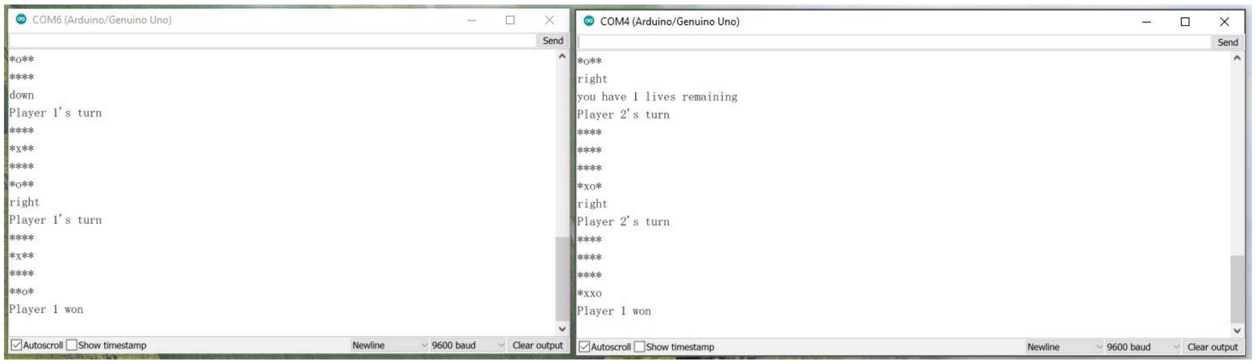
```
Please use zero indexing
Row: 2
Col: 3

Row: 2
Col: 1

Row: 1
Col: 2

Row: 3
Col: 1
```


COM6 (Arduino/Genuino Uno)

```
****
****
right
you have 2 lives remaining
Player 1's turn
****
*o**
****
****
down
Player 1's turn
****
*x**
*o**
****
```

COM4 (Arduino/Genuino Uno)

```
o***
****
****
down
Player 2's turn
****
****
o***
****
right
Player 2's turn
****
****
*o**
****
```


COM6 (Arduino/Genuino Uno)

```
*o**
****
down
Player 1's turn
****
*x**
****
*o**
right
Player 1's turn
****
*x**
****
**o*
Player 1 won
```

COM4 (Arduino/Genuino Uno)

```
*o**
right
you have 1 lives remaining
Player 2's turn
****
****
****
*xo*
right
Player 2's turn
****
****
****
*xxo
Player 1 won
```

(Serial monitor on the right is master device/player 1, serial monitor on the left is slave device/player 2)