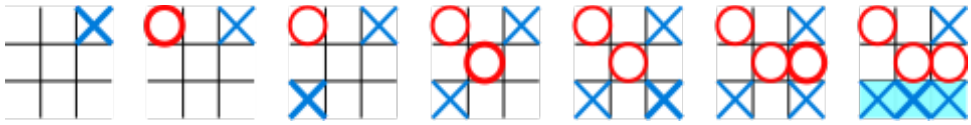
Tic Tac Toe
PROJECT
Report

Student: Dlo Bagari

Tic-tac-toe(also known as **noughts and crosses** or **Xs and Os**) is a paper-and-pencil game for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.

The following example game is won by the first player, X:



This project is about building a tic-tac-tie game with microController Arduino in a group of Three Students.
All the work is the creation of our team, each of us have a deep underistanding of the whole project not just their parts.

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board.

Parts Required:

- Arduino Mega
- 9 green Diffused 5mm LED.
- 9 red Diffused 5mm LED.
- 6 220Ω Resistors.
- 1 Breadboard.
- 16X2 LCD.
- IR Sensor.
- IR Remote.
- Breadboard wires.

Using only 9 pins of the microcontroller to turn on 18 LEDs and make use of persistence of vision to maintain the visual state of the display, micorcontoller store each player’s choices , and if one of the player won the game , micorcontroller notifies the processing application via Serial port and display some LEDs.

The game can be played by two methods the IR remote or by mouse click on the processing application.

For each player we used an integer 16bit to store all information about the player, when player press a button of IR Remote or click on a cell in processing application, the corresponding bit will be set on the player's storage.

Player 1 storage: 16 bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Unused Bits

Player 2 storage: 16 bits

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Unused Bits

assume player 1 pressed button number 3 on IR remote or clicked on cell number 3 on processing application , the player storage will be update as following:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Unused Bits playerStorage = playerStorage | B100

on each update in the player storage microcontroller checks if there is win case matched or if all cells been reserved .

Win cases:
comparing the bits in the player storage to one of the wincases, if wincase bits match the corresponding bits in the player storage, microcontroller declare the player as winner.

Example:
Player 1 storage: 16 bits:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	1	0	1	0	1	0	0

Unused Bits

one case of the win case is :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0

Unused Bits

using AND bit maniplation between the player storage and the win case , if the result is equal to the win case , player declared as winner.
If (playerStorge & wincase == wincase)
we have a winner

send notification

Draw case , where all cells are reserved:
In the draw case the game is over with no winner.

Player 1 storage: 16 bits:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1

Unused Bits

Player 2 storage: 16 bits:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	0

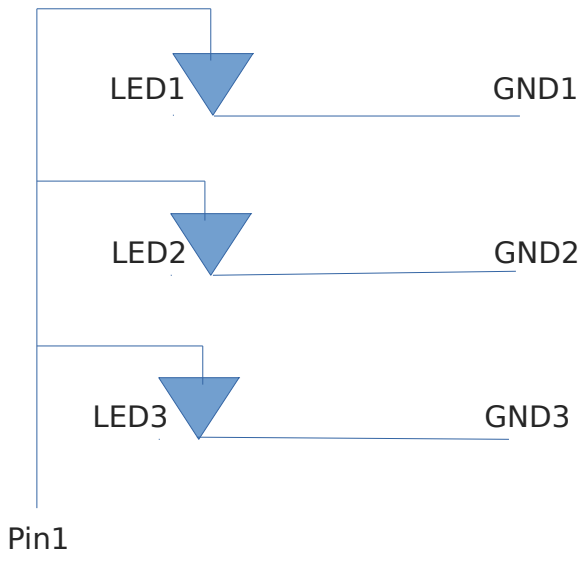
Unused Bits

using OR bit maniplation between player 1 storage and player 2 storage if the result is equal to B000000111111111 (0x1ff) the notification of draw is raised.

If (player1Stoage | player2Storage == 0x1ff)
we have draw case
send notification

tic tac toe circut:
the circut is designed to minimize the number of microcontroller pins needed
we used only 9 pins instead of 18 pins + GND.

We connect every 3 LED to one pin where pin value is HIGH LED will be On.



All ground (GND1, GND2, GND3) are HIGH by default, to turn on LED1 micorcontroller sets pin1 HIGH and sets GND1 LOW.
The following table explain how to display each LED.

GND	Pin1 value	Pin2 value	Pin3 value	Pin4 value	Pin5 value	Pin6 value

	100000	010000	001000	000100	000010	000001
1 1 0	LED1	LED1	LED2	LED2	LED3	LED3
1 0 1	LED4	LED4	LED5	LED5	LED6	LED6
0 1 1	LED7	LED7	LED8	LED8	LED9	LED9

Desgin and Analagis the circut:
Port registers allow for lower-level and faster manipulation of the i/o pins of the microcontroller on an Arduino board. The chips used on the Arduino board have three ports:

B (digital pin 8 to 13)

C (analog input pins)

D (digital pins 0 to 7)

Each port is controlled by three registers, which are also defined variables in the arduino language. The DDR register, determines whether the pin is an INPUT or OUTPUT. The PORT register controls whether the pin is HIGH or LOW, and the PIN register reads the state of INPUT pins set to input with pinMode()

DDR and PORT registers may be both written to, and read. PIN registers correspond to the state of inputs and may only be read.

We used DDRB the port B data Direction register to define pin 8 to 13 as output pins

```
DDRB |= B00111111; // set digital pins 8 to 13 as output pins
```

by default values of pin 8 to 13 must be LOW , to do that we write into PORTB the value :

```
PORTB |= B00000000; // set digital pins 8 to 13 LOW.
```

For controling the GND pins Value we mapped them to DDRC, making digital pin A0, A1, A2 on Port register C as input pins and set their value to be HIGH by default.

```
DDRC |= B000111; // set digital pin A0, A1, A2 as input pins.
PORTC |= B000111; // set digital pin A0, A1, A2 HIGH
```

we used A5 pin as input pin for IR Sensor.

we used Port Registers D as requried pins to display LCD.
The following is the diagram of the circut:

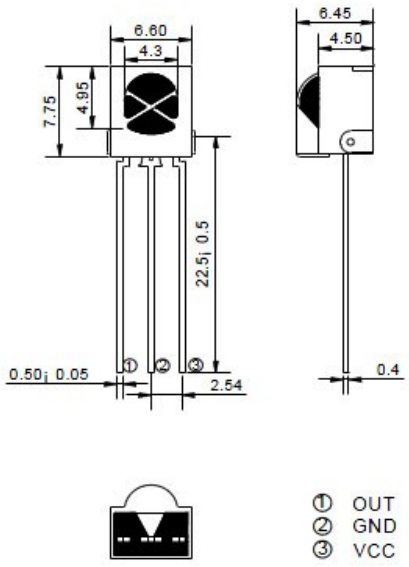
we use the resistor in series wirh LEDS to limit the current that can flow through the LEDS this is called a current limiting resistor, the current flow through components connected in series in the same in each component.

$V_S = V_R + V_F$

calculate the resister needed:

$R = (V_S - V_F) / I$; where I represents a safe LED current .

We use IR Sensor to receive a signal from IR Remote and forword the value of the signal to pin A5

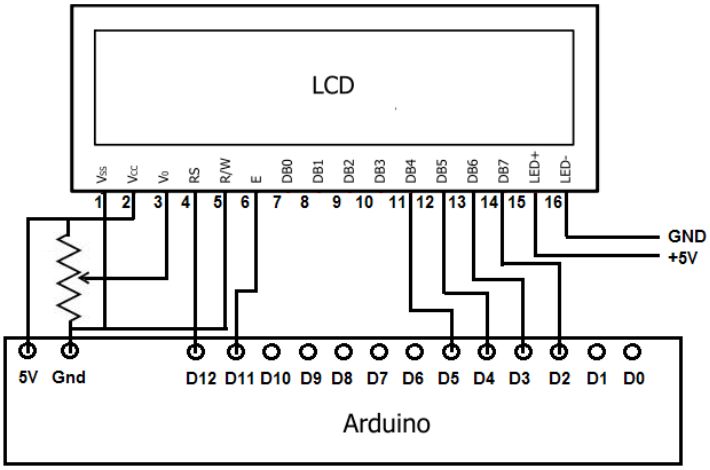


we use LCD display to display which player has to play next and to display the status of the game when the game is over.

The LCDs have a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:

- A register select (RS) pin that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.
- A Read/Write (R/W) pin that selects reading mode or writing mode
- An Enable pin that enables writing to the registers
- 8 data pins (D0 -D7). The states of these pins (high or low) are the bits that you're writing to a register when you write, or the values you're reading when you read.

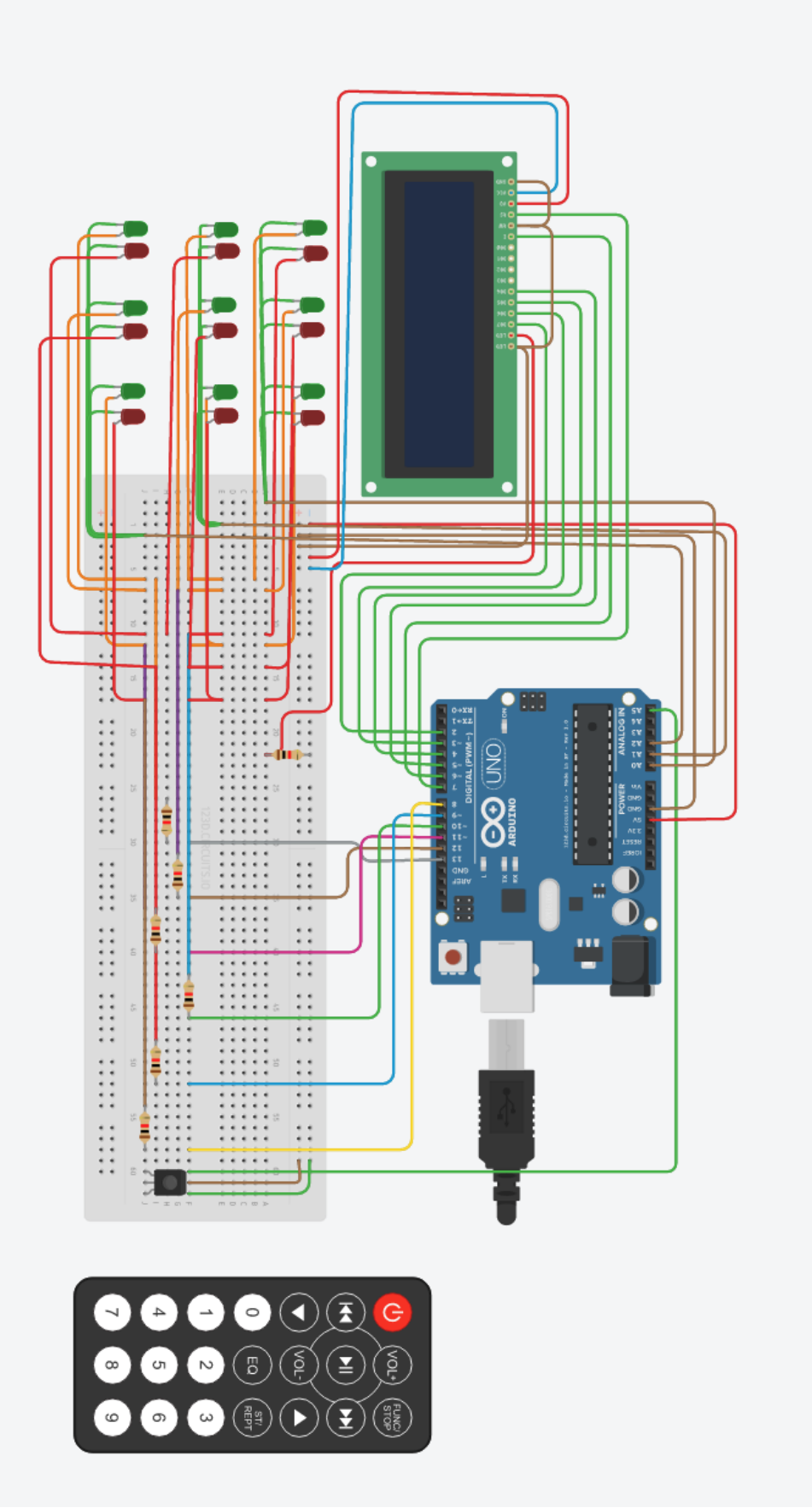
There's also a display constrast pin (Vo), power supply pins (+5V and Gnd) and LED Backlight (Bklt+ and Bklt-) pins that you can use to power the LCD, control the display contrast, and turn on and off the LED backlight, respectively.



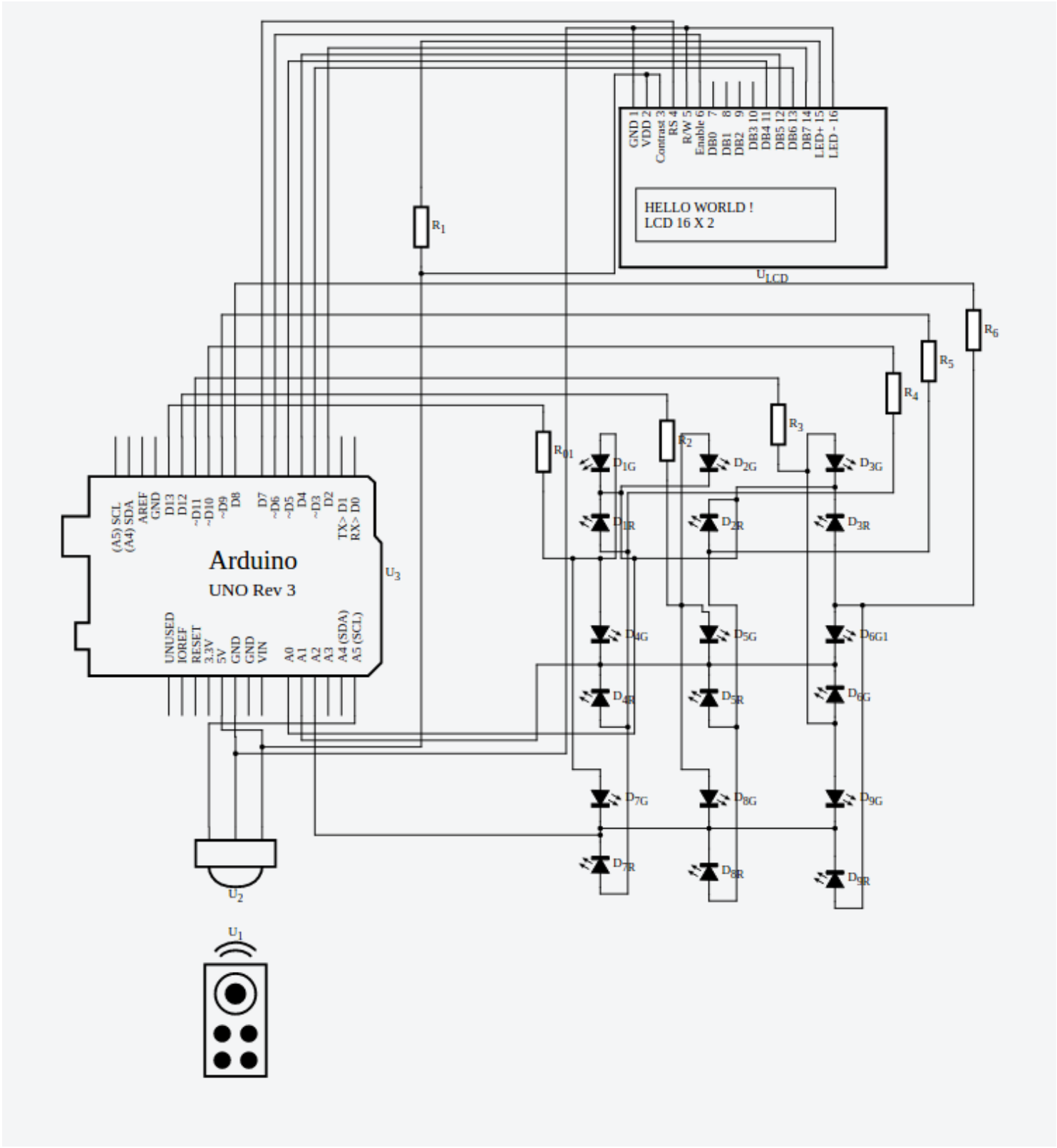
In our circut we to wire LCD screen to the board we connect the following pins:

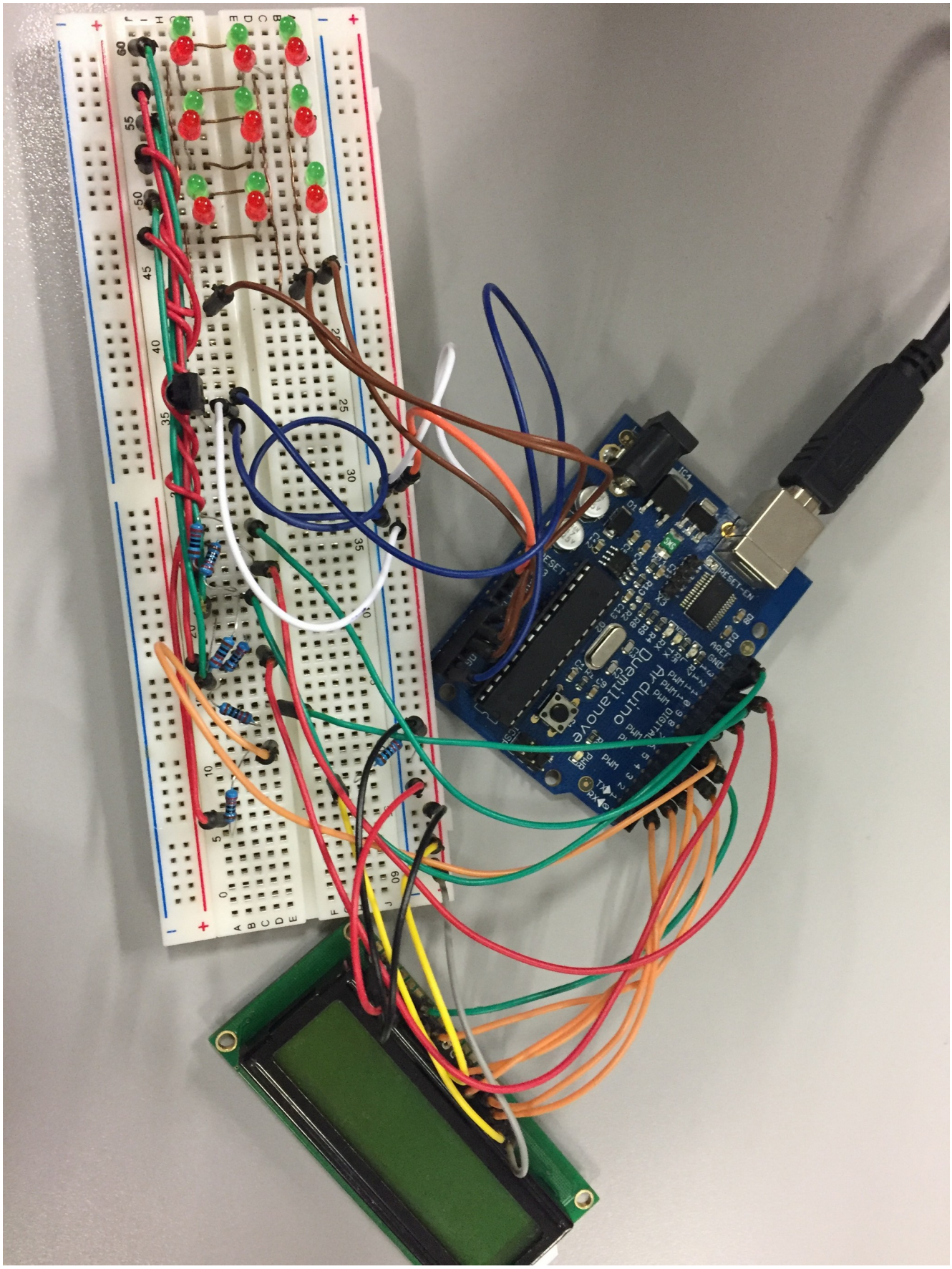
- LCD RS pin to digital pin 7
- LCD Enable pin to digital pin 6
- LCD D4 pin to digital pin 5
- LCD D5 pin to digital pin 4
- LCD D6 pin to digital pin 3
- LCD D7 pin to digital pin 2

Circuit:



Schematic:


GND 1
VDD 2
Contrast 3
RS 4
R/W 5
Enable 6
DB0 7
DB1 8
DB2 9
DB3 10
DB4 11
DB5 12
DB6 13
DB7 14
LED+ 15
LED - 16
HELLO WORLD !
LCD 16 X 2
U_LCD
R1
R2
R3
R4
R5
R6
R01
D1G
D1R
D2G
D2R
D3G
D3R
D4G
D4R
D5G
D5R
D6G1
D6G
D7G
D7R
D8G
D8R
D9G
D9R
Arduino
UNO Rev 3
U3
U2
U1



playing game as stand alone :
players can play ther game with out need to connect the microcontroller to computer , to power microcontroller 5 volt battary needed.

Playing on computer:
players can play the game with processing application which is designed to control micorcontroller via Serial.

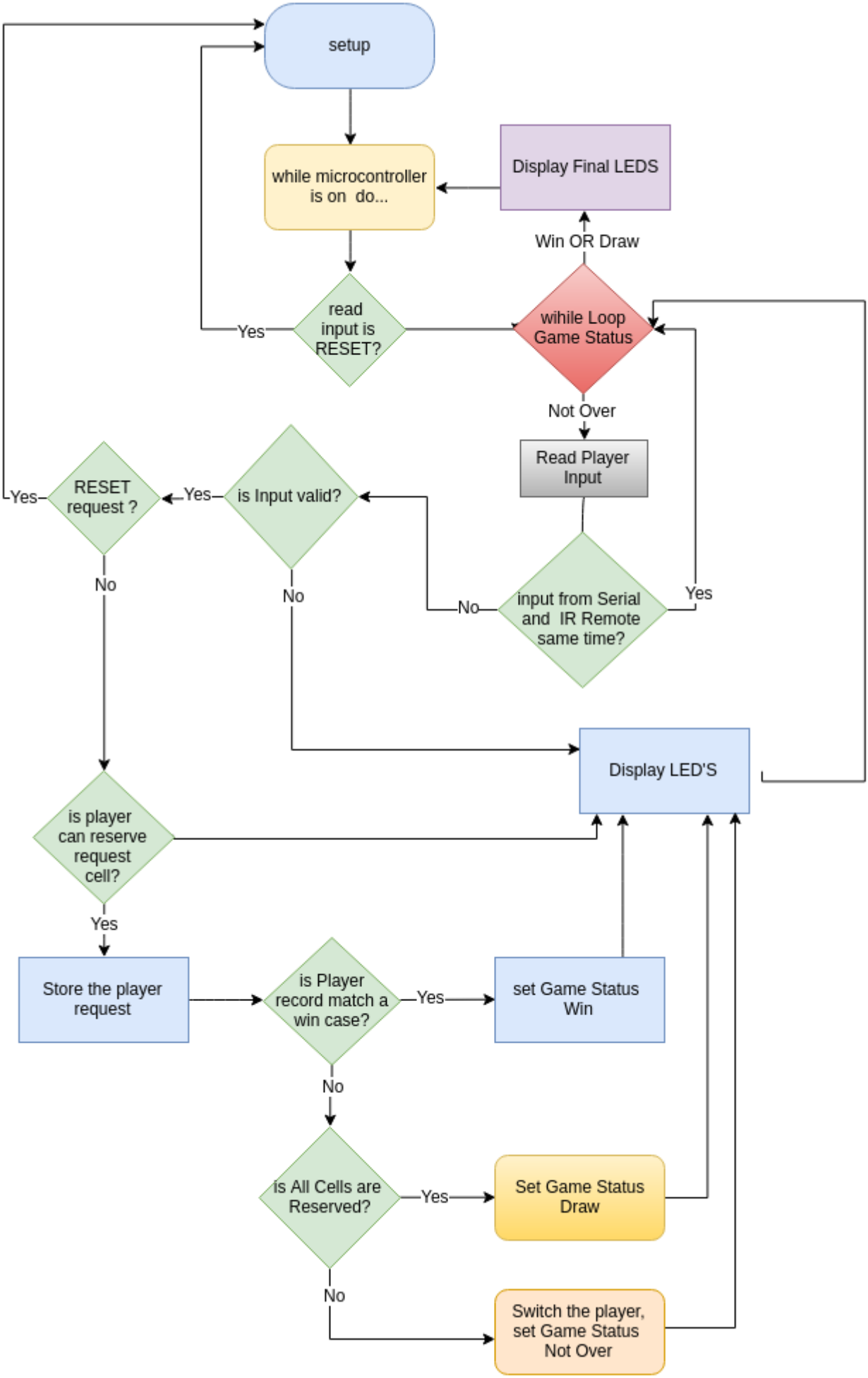
we include the IR Remote Library to enable the complied code in microcontroller to read player input from IR remote :

```
int RECV_PIN = A5; // the output pin for IR Sensor
IRrecv irrecv(RECV_PIN); // RECV Object
```

the function byte OnButtonPressed(unsigned long button Value, byte player) will be called for each player input takes two paramaters the player input and the player number, if player input comes from IR Remote the getButtonInfo() function will be called to converts player input to a single digit.

```
//convert player input to coresspinding cell number and reserve that cell
byte OnButtonPressed(unsigned long buttonValue, byte player) {
    //if player's input comes from IR Remote the method getButtonInfo will be called
    byte button = (buttonValue > 9 )? getButtonInfo(buttonValue) : buttonValue;
    if (button) {
       int one = 1;
       int wantedBit = one << (button - 1);// which bit the player wants to reserve

      //if both player have not reserved wantedBit, the player can reserve that bit
      if (!((reserved[0] & wantedBit) || (reserved[1] & wantedBit))) {
          reserved[player - 1] |= wantedBit; //write wantedBit to player's storage.
        // notify Serial which cell is reserved by which player.
        (player == 1)? Serial.write(button) : Serial.write(button + 10);
        return 1;
      }
  }
 return 0;
}
```


setup
while microcontroller is on do...
Display Final LEDS
Win OR Draw
read input is RESET?
Yes
wihile Loop Game Status
Not Over
Read Player Input
RESET request ?
Yes
is Input valid?
Yes
No
input from Serial and IR Remote same time?
No
Yes
No
Display LED'S
is player can reserve request cell?
Yes
Store the player request
is Player record match a win case?
Yes
set Game Status Win
No
is All Cells are Reserved?
Yes
Set Game Status Draw
No
Switch the player, set Game Status Not Over

code:

```
#include <IRremote.h>
#include <LiquidCrystal.h>
#define P1TURN "O TURN "
#define P2TURN "X TURN "
#define DRAW   "GAME OVER DRAW"
#define BAUD_RATE 9600
#define P1SHIFT 4
#define P2SHIFT 5
#define P1 0
#define P2 1

int RECV_PIN = A5;
IRrecv irrecv(RECV_PIN);
decode_results results;
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

byte ground[3]   = {B110, B101, B011};//ground cases
int reserved[2] = {0, 0};
int winCases[8] = {0x7, 0x38, 0x1c0, 0x49, 0x92, 0x124, 0x111, 0x54}; //wincases
byte whoIsTurn = 1; // which player has to play in next rotation
char statuss[16];// string to save the status of the game when the game is over
byte gameOver = 0;
unsigned long newInput = 0;

//declare reset function @ address 0
void (* resetFunc)(void) = 0;//The function starts to execute at addres 0


void setup() {
  DDRB  |= B00111111;//set all pins in DDRB as output
  PORTB |= B00000000;
  DDRC  = PORTC = B000111; //enable pull ups on pins 1, 2, 3
  Serial.begin(BAUD_RATE); // start Serial
  irrecv.enableIRIn(); // Start the receiver
  lcd.begin(16, 2);
}

//reseting the micorColtoller and Processing
void resetProcessing() {
    PORTC = B000000;
    Serial.write(61); //set Processing backround color to red
    PORTB = B010101; //display red LEDS
    delay(300);
    PORTB = B101010;// display green LEDS
    Serial.write(62);//set Processing backgound color to green
```

```
    delay(300);
    PORTB = B111111;// display all LEDS
    Serial.write(60);//set Processing backgound color to white
    delay(200);
    resetFunc();

}

//convert player input to a singl digit
byte getButtonInfo(unsigned long buttonValue){
  switch(buttonValue) {
     case 0xff30cf : return 1;
    case 0xff18e7 : return 2;
    case 0xff7a85 : return 3;
    case 0xff10ef : return 4;
    case 0xff38c7 : return 5;
    case 0xff5aa5 : return 6;
    case 0xff42bd : return 7;
    case 0xff4ab5 : return 8;
    case 0xff52ad : return 9;
    case 10:
    case 0xffa25d : resetProcessing();  //this case is to reset the microContoller and Processing
  }
  return 0;
}

//convert player input to coresspinding cell number and reserve that cell
byte OnButtonPressed(unsigned long buttonValue, byte player) {
   //if player's input comes from IR Remote the method getButtonInfo will be called
   byte button = (buttonValue > 9 )? getButtonInfo(buttonValue) : buttonValue;
   if (button) {
     int one = 1;
     int wantedBit = one << (button - 1);// which bit the player wants to reserve

     //if both player have not reserved wantedBit, the player can reserve that bit
     if (!((reserved[0] & wantedBit) || (reserved[1] & wantedBit))) {
       reserved[player - 1] |= wantedBit; //write wantedBit to player's storage.
       // notify Serial which cell is reserved by which player.
       (player == 1)? Serial.write(button) : Serial.write(button + 10);
       return 1;
     }
   }
  return 0;
}

//compare the player's storage with each win case, if there is match
//send a notification to Serial
byte isWon(byte player) {
  for (byte i = 0; i < 8 ; i++) {
```

```
    if ((reserved[player] & winCases[i]) == winCases[i]) {
      reserved[player] = winCases[i];
      reserved[whoIsTurn % 2] = 0;// reset the storage of the loser player
      Serial.write(200 + i); //write to Serial the win case
      return 1;
    }
  }
  return 0;
}


byte isFull() {
  //return Non-zero Value if all cells are reserved and the board is full
  return ((reserved[0] | reserved[1]) == 0x1ff);
}

void oneLedOn(byte player, byte targetBit, byte shiftAmount, byte _delay) {
        byte bitOn = (reserved[player] >> targetBit) & B1;// read the targetBit from player's
storage
        if(bitOn){
         byte col = targetBit % 3; //targetBit is in which column
         // PORTB = 1 << ( shiftAmount - ( col * 2)) ;
         PORTB = 1 << (shiftAmount - (col << 1)); // make the corresponding pin high in PORTB
         //targetBit / 3 : targetBit in which row
         PORTC = ground[targetBit / 3]; //make the corresponding pin low in PORTC
         delay(_delay);
         PORTB = B00000000; // reset PORTB
         PORTC = B000111; //reset PORTC
        }
}

void ledsOn(byte _delay) {
    //for each bit in each player's storage if the bit is on display the corresponding LED on
    for ( byte i = 0 ; i < 9; i++) {
        oneLedOn(P1, i, P1SHIFT, _delay); // for player 1
        oneLedOn(P2, i, P2SHIFT, _delay); // for player 2
  }
}



void loop() {
   //listen for reset request
   if (irrecv.decode(&results) || Serial.available()) {
     if (results.value == 0xffa25d || Serial.read() == 10) {
        resetProcessing();
     }
     irrecv.resume();
   }
   lcd.setCursor(0, 0);//set coursor at row 0 colum 0
```

```
   lcd.print(statuss);//display the game status on LCD
   while(whoIsTurn) {
     newInput = 0;
     lcd.setCursor(0, 0);//set coursor at row 0 colum 0
     (whoIsTurn == 1)? lcd.print(P1TURN):lcd.print(P2TURN); //display on LCD which layer has
to play
     //if player input from both input methods then ignore this request
     if ( irrecv.decode(&results) && Serial.available()) {
       //empty the serial buffer
        while (Serial.available()) {
          Serial.read();
        }
       results.value = 0;
       irrecv.resume();
       continue;
     }
     //player input from IR Remote
     else if (irrecv.decode(&results)) {
         newInput = results.value;
         irrecv.resume();
     }

     //player input from mouseclick on Porcessing
     else if( Serial.available()) {
       newInput = Serial.read();
     }

     if (newInput && OnButtonPressed(newInput, whoIsTurn) ) {
       if (isWon(whoIsTurn - 1)) {
         // status = which player has won the game
         (whoIsTurn == 1)? sprintf(statuss, "%s won !", "O"): sprintf(statuss, "%s won !", "X");
         whoIsTurn = 0;
       }
       else if (isFull()) {
         // status = draw
         sprintf(statuss, "%s", DRAW);
         reserved[0] = reserved[1] = 0x1ff;
         whoIsTurn = 0;
       }
       else{
       (whoIsTurn == 1)? whoIsTurn = 2 :  whoIsTurn = 1;
       }
      }
    ledsOn(1);
   }
  ledsOn(15);
}
```

processing :

```
import processing.serial.*;
Serial myPort;
PImage imgO;
PImage imgX;
PFont f;

void redraw(int c){
  switch(c){
    case 60: background(255, 255, 255);
             break;
    case 61: background(255,0,0);
             break;
    case 62: background(0,255,0);
             break;
  }

  stroke(128,128,128);
  strokeWeight(4);
  line(200,50,200,360);
  line(400,50,400,360);
  line(100,150,500,150);
  line(100,260,500,260);
}

void setup() {
  myPort = new Serial(this, Serial.list()[0], 9600);
  size(640, 400);
  redraw(60);
  imgO = loadImage("o.gif");
  imgX = loadImage("x.gif");
  f = createFont("Arial",16,true);

}

int fillSquare(int s){
  switch (s){
     case 1: image(imgO,100,45);  break;
     case 11: image(imgX,100,45); break;
     case 2: image(imgO,260,45);  break;
     case 12: image(imgX,260,45); break;
     case 3: image(imgO,420,45);  break;
     case 13: image(imgX,420,45); break;
     case 4: image(imgO,100,155); break;
     case 14: image(imgX,100,155);break;
```

```
    case 5:  image(imgO,260,155);break;
    case 15: image(imgX,260,155);break;
    case 6: image(imgO,420,155); break;
    case 16: image(imgX,420,155);break;
    case 7: image(imgO,100,265); break;
    case 17: image(imgX,100,265);break;
    case 8: image(imgO,260,265); break;
    case 18: image(imgX,260,265);break;
    case 9:  image(imgO,420,265);break;
    case 19: image(imgX,420,265);break;
    case 60:
    case 61:
    case 62: redraw(s);
  }
  return s;
}

void win(int w){
  stroke(0,0, 0);
  strokeWeight(10);
  switch(w){
    case 200: line(110,100,495,100); /* ---  */
              return;
    case 201: line(110,210,495,210); /* ---   */
              return;
    case 202: line(110,320,495,320); /* ___   */
              return;
    case 203: line(140,90,140,320); /* |     */
              return;
    case 204: line(300, 90,300,320); /*  |    */
              return;
    case 205: line(460, 90,460,320); /*   |   */
              return;
    case 206: line(140, 90,470,320); /* \     */
              return;
    case 207: line(140,320,450,90); /* /     */
              return;
  }
}

void draw(){
  textFont(f,16);
  fill(0);
  text("RESET",550,30);
  while (myPort.available()>0){
    int val = myPort.read();
    win(fillSquare(val));
    println(val);
  }
```

```
}

void mouseClicked() {
  if (mouseX > 100 && mouseX < 190 && mouseY > 60 && mouseY < 140) {
   myPort.write(1);
  }
  else if (mouseX > 210 && mouseX < 390 && mouseY > 60 && mouseY < 140) {
   myPort.write(2);
  }
 else if (mouseX > 410 && mouseX < 500 && mouseY > 60 && mouseY < 140) {
   myPort.write(3);
  }
  else if (mouseX > 100 && mouseX < 190 && mouseY > 160 && mouseY < 250) {
   myPort.write(4);
  }
  else if (mouseX > 210 && mouseX < 390 && mouseY > 160 && mouseY < 250) {
   myPort.write(5);
  }
 else if (mouseX > 410 && mouseX < 500 && mouseY > 160 && mouseY < 250) {
   myPort.write(6);
  }
  else if (mouseX > 100 && mouseX < 190 && mouseY > 275 && mouseY < 400) {
   myPort.write(7);
  }
  else if (mouseX > 210 && mouseX < 390 && mouseY > 275 && mouseY < 400) {
   myPort.write(8);
  }
 else if (mouseX > 410 && mouseX < 500 && mouseY > 275 && mouseY < 400) {
   myPort.write(9);
  }
  else if (mouseX > 550 && mouseX < 600 && mouseY > 20 && mouseY < 35) {
   myPort.write(10);
  }

}
```