UNIVERSITATEA POLITEHNICA TIMIȘOARA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

SISTEME INCORPORATE

BATTLESHIP GAME

ARDUINO

FLOROIU IOANA-ALEXANDRA

DUDUMAN RAUL-ANDREI

GRUPA 2.2 CTI RO

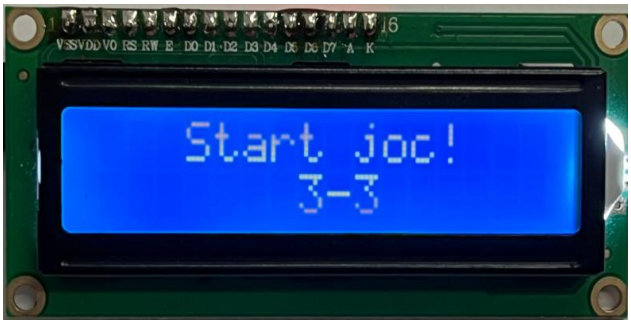
ANUL III

Implementarea jocului Battleship

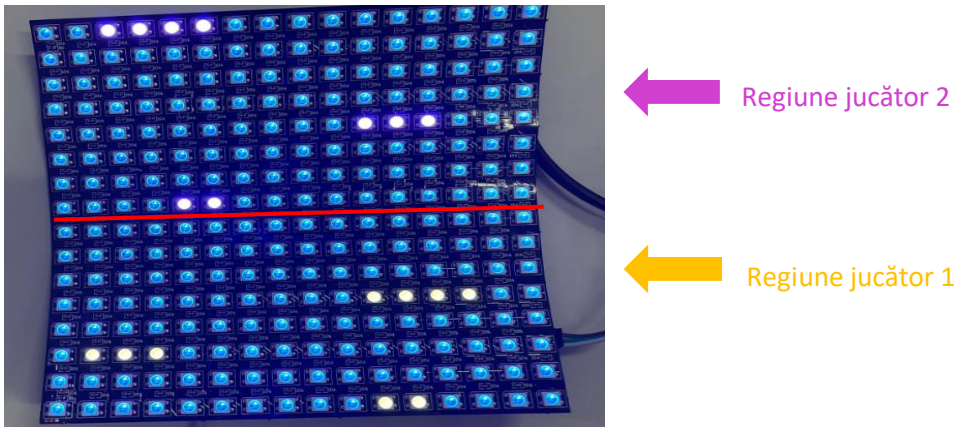
Acest proiect își propune să creeze jocul Battleship, joc de luptă navală ce se poate desfășura între 2 jucători.

Caracteristici:

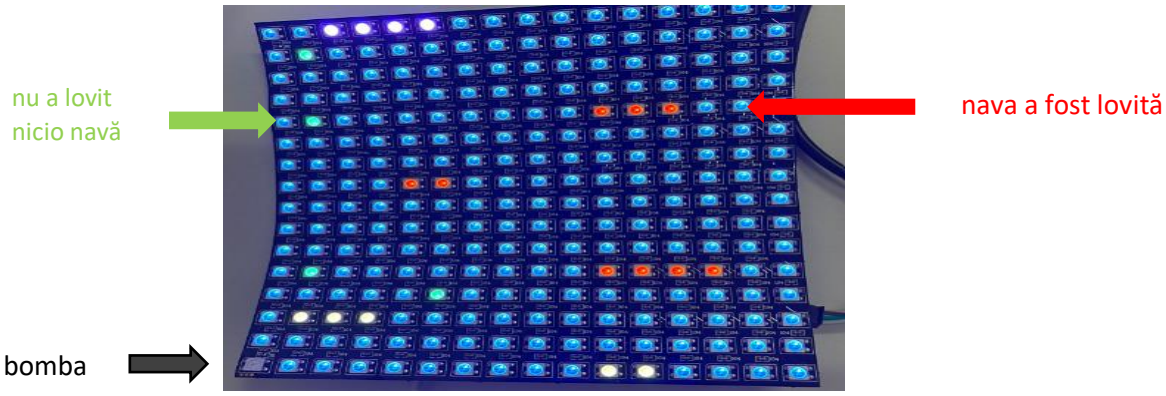
- Jocul implică doi jucători, fiecare având posibilitatea de a plasa în locul dorit câte 3 nave;



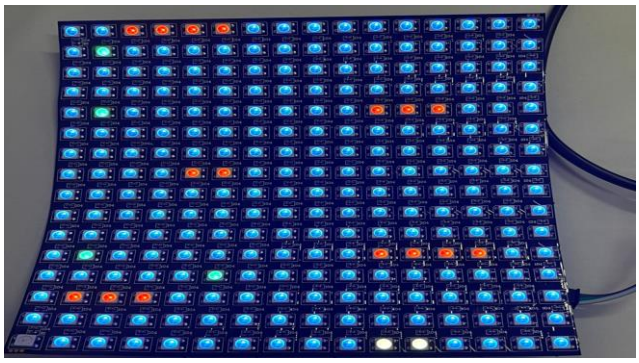
- Matricea de dimensiune 16x16 va fi divizată în două regiuni egale de 16x8, fiecare regiune corespunzând unui jucător;
- Există trei tipuri de nave în joc: una de dimensiune 2, una de dimensiune 3 și una de dimensiune 4;
- Navele primului jucător vor fi galbene, în timp ce navele celui de-al doilea jucător vor fi mov;
- Jucătorul poate alege poziția de lovire folosind un joystick pentru direcțiile stânga, dreapta, sus și jos, iar confirmarea poziției se va face prin apăsarea butonului de pe joystick;



- După ce o navă a fost lovită și scufundată, led-urile asociate acesteia se vor aprinde în culoarea roșie și vor rămâne aprinse pe toată durata jocului;
- Pozițiile de pe matrice unde s-a încercat aruncarea unei bombe care nu a lovit nicio navă, vor rămâne aprinse pe toată durata jocului în culoarea verde;



- Jocul se va încheia atunci când toate navele unui jucător vor fi scufundate și vor fi marcate cu culoarea roșie. Scorul va fi afișat pe un ecran LCD împreună cu un mesaj sugestiv;



Toate navele jucătorului 2 au fost atacate, ceea ce înseamnă că jucătorul 1 a câștigat.

Jocul s-a încheiat

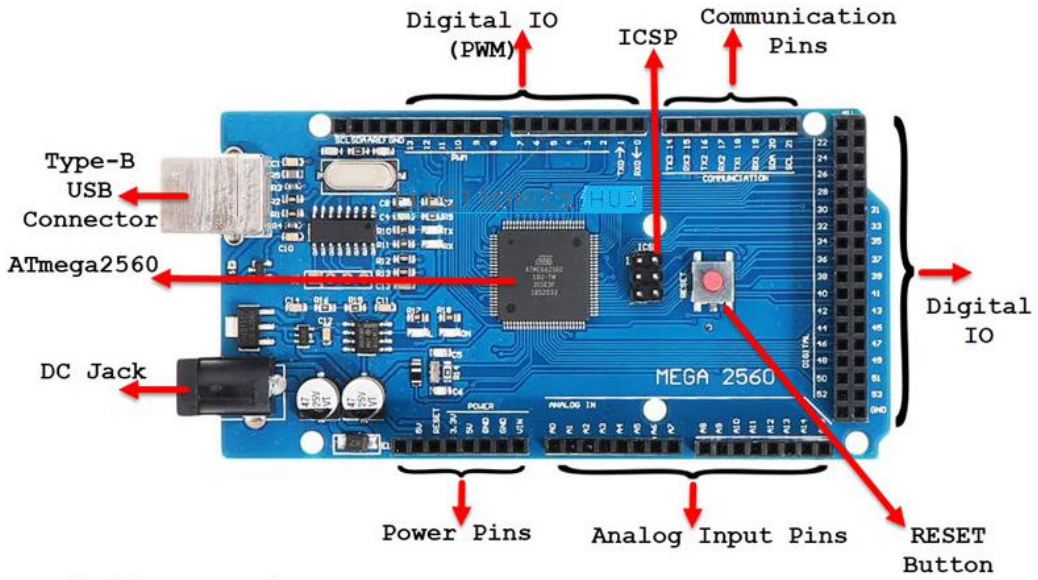
Arduino Mega 2560

Arduino Mega utilizează microcontrolerul ATmega2560, un MCU bazat pe arhitectura AVR cu 8 biți dezvoltată de ATMEL.

Placa dispune de 54 de pini de intrare/ieșire digitală (din care 15 pot funcționa ca ieșiri PWM), 16 intrări analogice, 4 UART-uri (porturi seriale hardware), un oscilator cu cristal de 16 MHz, un port USB, o mufă de alimentare, un antet ICSP și un buton de resetare. Este concepută și dezvoltată pentru a furniza un număr extins de linii IO (atat digitale, cât și analogice), o capacitate mai mare de memorie flash și o memorie RAM mai generoasă în comparație cu placa Arduino UNO.

Aspectul plăcuței Arduino Mega

Spre deosebire de Arduino Nano, toate componentele sunt plasate pe partea superioară a PCB-ului. Există un conector USB de tip B situat pe marginea scurtă din stânga a plăcii, care este folosit atât pentru alimentarea plăcii, cât și pentru programarea microcontrolerului. În plus, se găsește o mufă DC de 2,1 mm pentru a furniza alimentare externă.



Aspectul Arduino Mega Board

Specificații tehnice

MCU	ATmega2560
Arhitectură	AVR
Tensiune de operare	5V
Tensiune de intrare	6V – 20V (limită) 7V – 12V (recomandat)
Viteza ceasului	16 MHz
Memorie flash	256 KB (din care 8 KB sunt folosite de bootloader)
SRAM	8 KB
EEPROM	4 KB
Pini IO digitale	54 (dintre care 15 pot produce PWM)
Pini de intrare analogică	16

ATmega2560 oferă trei tipuri distincte de memorie disponibile:

- 256 KB de memorie flash
- 8 KB de SRAM
- 4 KB de EEPROM

Există 54 de pini (de la D0 la D53) care sunt pini IO digitali reali, ce pot fi configurați conform necesităților aplicației folosind funcțiile pinMode(), digitalWrite() și digitalRead(). Toți acești pini digitali IO sunt capabili să furnizeze sau să consume un curent de până la 20mA (cu o limită maximă de 40mA). O caracteristică suplimentară a pinilor Digital IO este disponibilitatea rezistenței interne de pull-up (care nu este conectată implicit). Valoarea rezistenței interne de pull-up se încadrează în intervalul de la 20KΩ la 50KΩ.

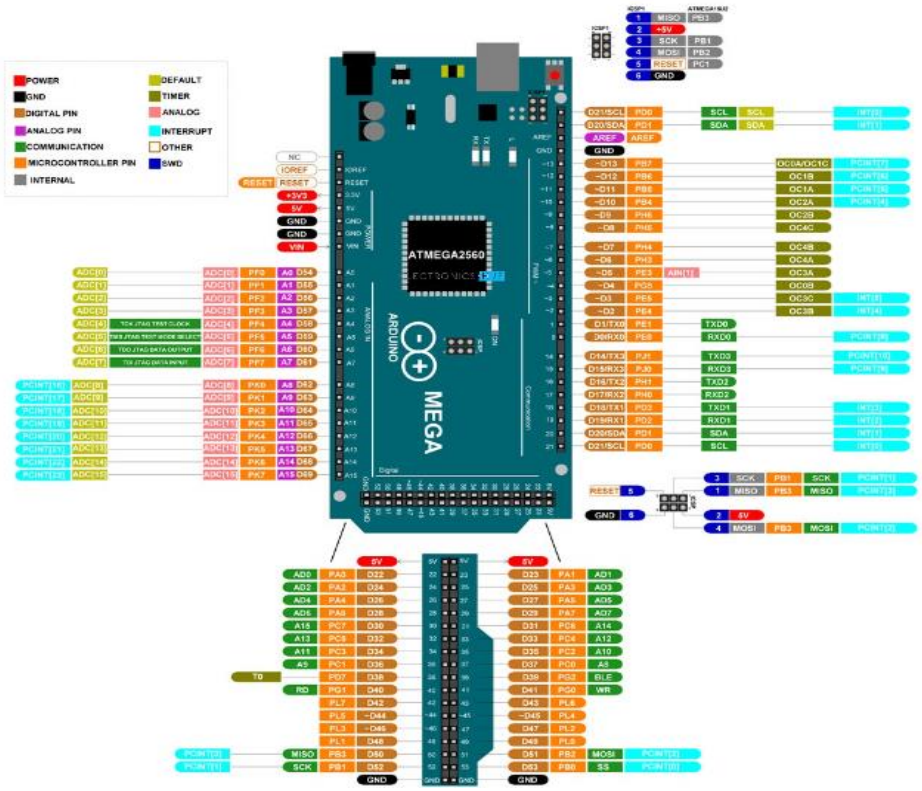
De asemenea, există 16 pini de intrare analogică (de la A0 la A15). Toți acești pini de intrare analogică oferă o caracteristică ADC cu rezoluție de 10 biți, ce poate fi citită folosind funcția analogRead().

Pinii IO digitali 2 - 13 și 44 - 46 sunt capabili să genereze semnale PWM pe 8 biți.

Arduino Mega acceptă trei tipuri diferite de interfețe de comunicare:

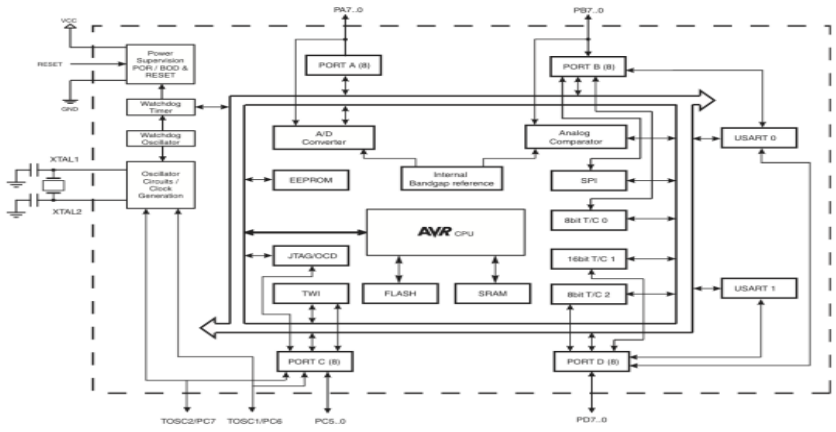
- Serial
- I2C
- SPI

Diagrama de conectare a Arduino Mega 2560



Arhitectura generala a unui microcontroller AVR

Arhitectura generală a unui microcontroler AVR constă într-un nucleu central de procesare (CPU) RISC (Reduced Instruction Set Computing), un set de registre de lucru, o memorie de program, o memorie de date, multiple module periferice de intrare/ieșire și alte funcționalități integrate. CPU-ul RISC este proiectat pentru a executa instrucțiuni simple și eficiente într-un timp minim, iar registrele de lucru sunt utilizate pentru a stoca temporar date și adrese. Memoria program conține codul executabil al aplicației, iar memoria de date este utilizată pentru a stoca variabile și alte date temporare.



Matrice LED RGB

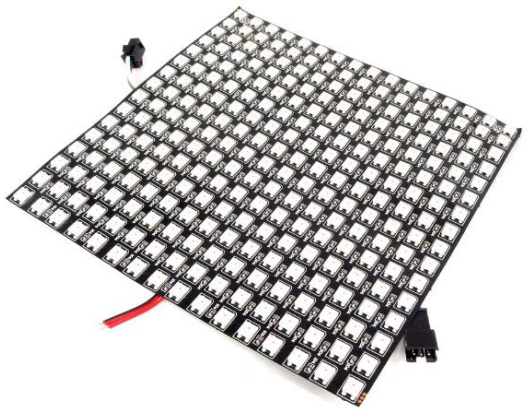
Matricea de culoare cu LED-uri RGB WS2812B adresabile individual. Are o rezoluție de 16x16 și oferă posibilitatea de a adăuga module suplimentare pentru a extinde suprafața de afișare. LED-urile sunt montate pe o placă flexibilă, ceea ce permite montarea modulelor pe substraturi rotunjite.

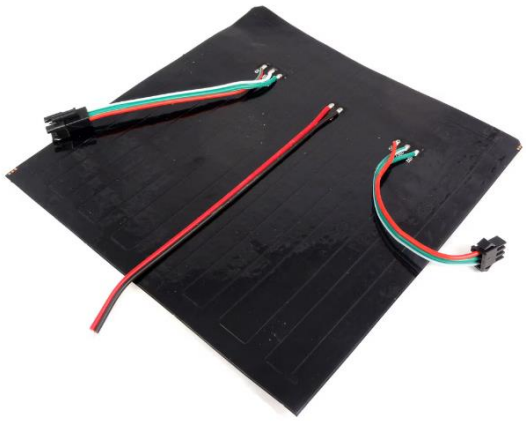
Este deplin compatibil cu biblioteca NeoPixel și necesită o sursă de alimentare de 5V și un curent de 15A pentru a asigura consumul maxim atunci când toate LED-urile sunt aprinse în alb.

Specificații:

- tensiune de alimentare 5V;
- rezolutie 16x16, 256 LED;
- tipul de LED: WS2812B;
- consum de curent 60mA pentru 1 dioda, max. 15A;
- dimensiuni 160 mm x 160 mm;

Aspectul matricei 16x16 RGB cu LED-uri WS2812B NeoPixel





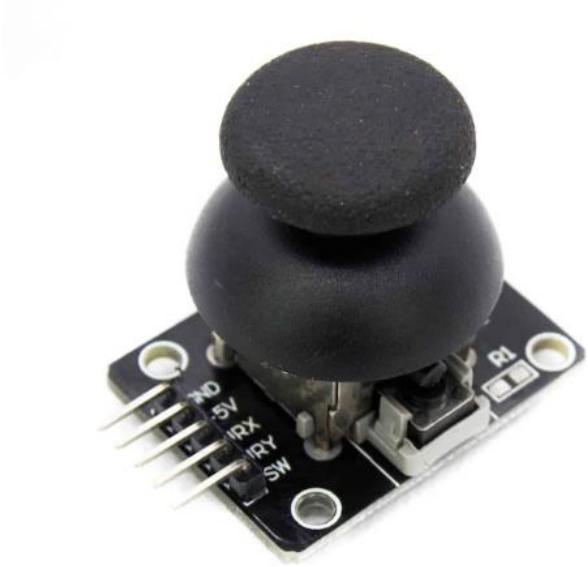
Joystick

Modulul joystick analogic cu 3 axe pentru Arduino este un dispozitiv de intrare care permite utilizatorilor să controleze mișcările în trei direcții: pe axa X, axa Y și axa Z (cunoscută și sub numele de buton de apăsare). Acest modul include două potențiometre pentru axe X și Y, care măsoară poziția joystick-ului în aceste direcții, cât și un comutator de apăsare pentru axa Z.

Modulul se conectează la placa Arduino folosind pini analogici pentru citirea valorilor analogice generate de potențiometre și pini digitali pentru citirea stării comutatorului de apăsare. Aceste date sunt utilizate apoi în program pentru a controla diverse acțiuni, cum ar fi deplasarea navelor pe matricea de leduri, deplasarea bombelor, cât si plasarea acestora în momentul în care butonul este apăsat. Este versatil și ușor de integrat, fiind compatibil cu majoritatea plăcilor Arduino.

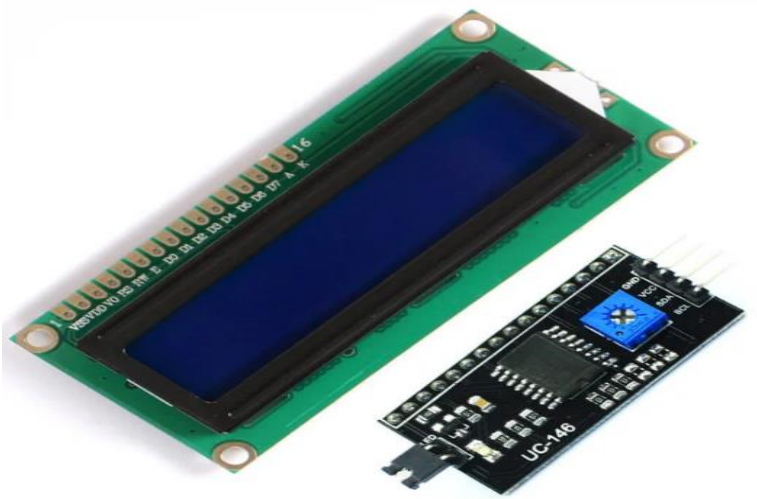
Acesta are dimensiunile de 34 x 27 mm, putând fi folosit în aplicații cu spațiu limitat. Înălțime este de 32 mm, oferind suficient spațiu pentru mișcarea confortabilă a joystick-ului.

De asemenea, necesită o sursă de alimentare de 5V pentru a funcționa corect. Aceasta poate fi furnizată direct de la placa Arduino sau de la o altă sursă de alimentare externă cu aceeași tensiune. Asigurarea unei alimentări stabile de 5V este esențială pentru funcționarea corectă a modulului și pentru evitarea deteriorării componentelor.



LCD

Pentru a facilita afișarea informațiilor pe un ecran LCD în cadrul acestui proiect, am utilizat un modul LCD cu interfață I2C. Acest modul permite comunicarea între Arduino Mega 2560 și LCD folosind protocolul I2C, ceea ce simplifică realizarea conexiunii și controlul.



Interfața I2C (Inter-Integrated Circuit)

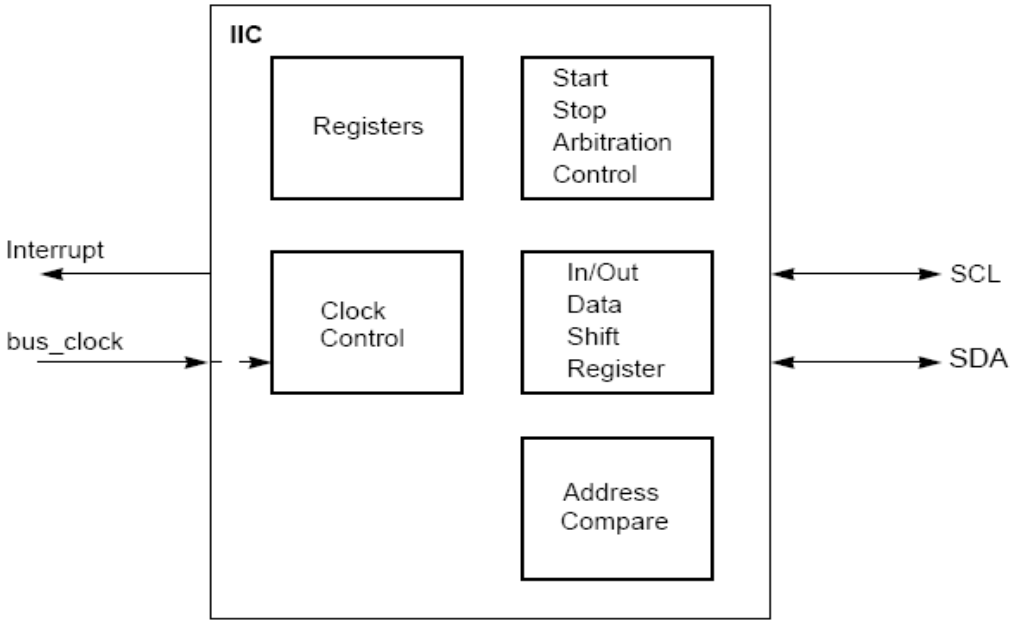
Este o interfață serială sincronă, multimaster. Ratele de transfer I2C sunt de 100 și 400 de kilobiți pe secundă (kbps), facilitând transferul eficient al datelor între dispozitive. Utilizează două linii de comunicație: SDA (Serial Data), pentru transmiterea datelor și SCL (Serial Clock) pentru sincronizare. Numărul de terminale care se pot conecta la aceste linii este limitat doar de capacitatea maximă a acestora, care este de 400pF.

Frecvența tactului poate avea 256 de valori diferite, oferind flexibilitate în ajustarea vitezei de transfer în funcție de cerințele aplicației. De asemenea, un alt aspect important al interfeței I2C este capacitatea de a genera cereri de întrerupere la transferul fiecărui octet de date. Această capacitate permite dispozitivelor conectate să gestioneze eficient datele primite și să reacționeze prompt.

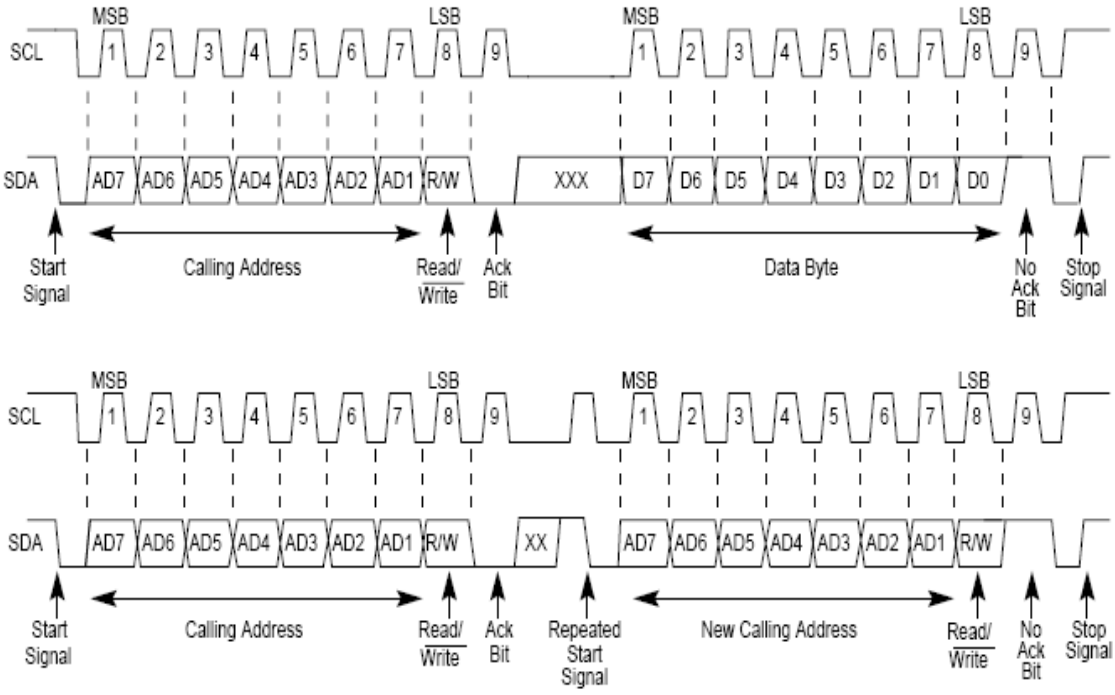
În comparație cu alte metode de comunicare serială, interfața I2C utilizează o magistrală de tip "open drain". Aceasta înseamnă că dispozitivele conectate la magistrală pot trage linia de semnal la nivelul logic 0, dar nu pot să o conducă la nivelul logic 1. Astfel, se elimină problema cunoscută sub numele de "bus contention", în care două sau mai multe dispozitive încearcă să tragă aceeași linie la nivelul logic "high" și "low" în același timp, ceea ce ar putea duce la distrugerea componentelor.

În codul Arduino, am utilizat biblioteca "LiquidCrystal_I2C.h" pentru a comunica cu LCD-ul prin intermediul modulului I2C.

Schema bloc:



Protocolul IIC:



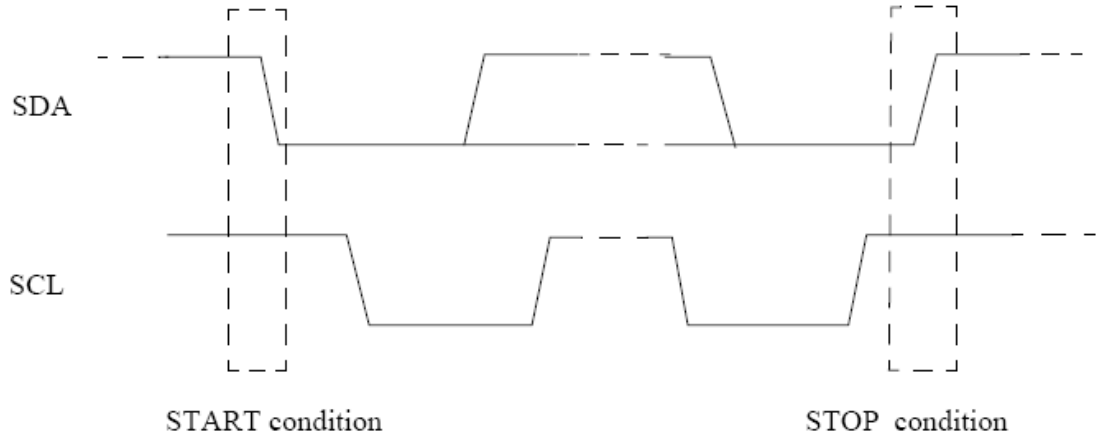
START şi STOP

Condiția de start în protocolul I2C este esențială pentru inițierea unei transmisii de date între dispozitivele conectate la magistrală. Dispozitivul master (care inițiază comunicarea) lasă linia de ceas (SCL) în stare logică "high" și trage linia de date (SDA) în stare logică "low". Acest lucru semnalizează tuturor dispozitivelor slave care primesc comenzi că o transmisie de date este pe cale să înceapă și că trebuie să fie pregătite pentru a primi sau pentru a transmite date.

În situația în care există două sau mai multe dispozitive master care doresc să preia controlul magistralei I2C în același timp, se utilizează un mecanism de arbitraj pentru a determina care dispozitiv va prelua controlul: dispozitivul care trage primul linia SDA în stare logică "low" va câștiga arbitrajul și va obține controlul magistralei.

Condiția de stop în protocolul I2C este folosită pentru a indica sfârșitul unei transmisii de date între dispozitivele conectate la magistrală. După ce toate cadrele de date au fost trimise de către dispozitivul master către dispozitivele slave sau invers, dispozitivul master generează o condiție de oprire pentru a încheia transmisia.

Condiția de oprire este definită de o tranziție de la nivel logic "low" la nivel logic "high" pe linia de date (SDA), urmată de o tranziție similară pe linia de ceas (SCL), cu SCL rămânând în continuare la nivel logic "high". Această secvență de tranziții indică dispozitivelor conectate că transmisia s-a încheiat și că magistrala a fost eliberată pentru alte comunicații.

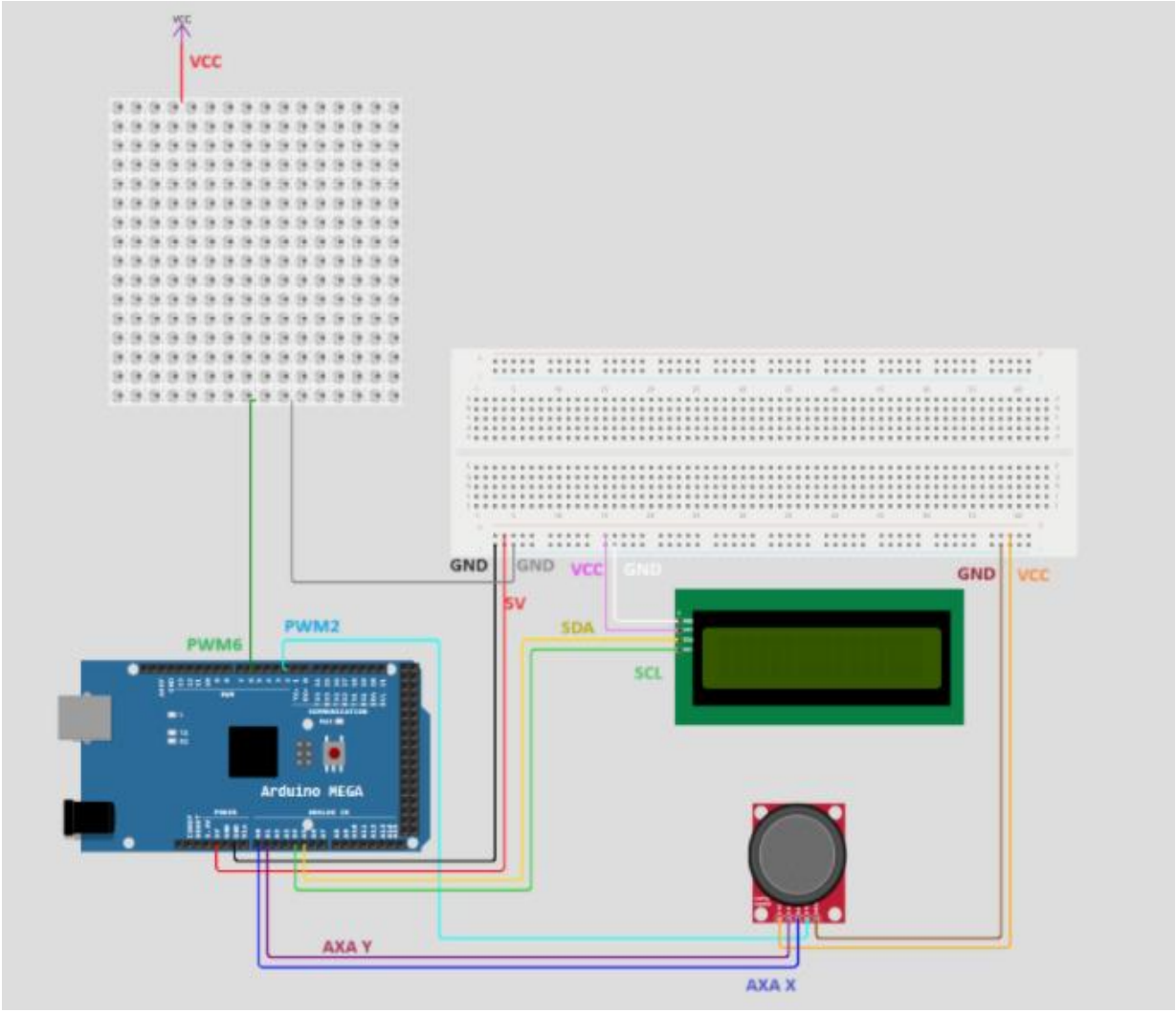


PWM (Pulse Width Modulation)

PWM este o tehnică utilizată pentru a varia în mod controlat tensiunea dată unui dispozitiv electronic. Această metodă schimbă foarte rapid tensiunea oferită dispozitivului respectiv din ON în OFF și invers (treceri rapide din HIGH în LOW, de exemplu 5V - 0V). Raportul dintre perioada de timp corespunzătoare valorii ON și perioada totală dintr-un ciclu ON- Internal OFF se numește factor de umplere (duty cycle) și reprezintă, în medie, tensiunea pe care o va primi dispozitivul electronic.

Prin urmare, circuitele analogice pot fi controlate dintr-un mediu digital.

Schema



Cod sursă

```
#include <Adafruit_NeoPixel.h>
#include <Wire.h> // Biblioteca pentru comunicare I2C
#include <LiquidCrystal_I2C.h> // Biblioteca pentru ecrane LCD I2C
#include <LCD-I2C.h>

#define LCD_ADDRESS 0x27 // Adresa I2C a ecranului LCD
#define LCD_COLS 16      // Numărul de coloane al ecranului LCD
#define LCD_ROWS 2       // Numărul de rânduri al ecranului LCD

LiquidCrystal_I2C lcd(LCD_ADDRESS, LCD_COLS, LCD_ROWS); //activam LCD ul

#define LED_PIN 6    //pin de date pentru matricea de led-uri
#define LED_COUNT 256 // cate led-uri sunt in matrice

#define JOY_X A0    // pin de date pentru joystick axa X
#define JOY_Y A1    // pin de date pentru joystick axa Y
#define JOY_BTN 2   // pin de date pentru joystick button

Adafruit_NeoPixel strip(LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800); // setam matricea de  led-uri

int shipsPlayer1[9] ={400, 400, 400, 400, 400, 400, 400, 400, 400};  //vector cu pozitii nave jucator 1
int shipsPlayer2[9];                                  //vector cu pozitii nave jucator 2

int redPositionPlayer1[9];                            //vector cu pozitii nave atacate jucator 1
int greenPositionPlayer1[119];                        //vector cu pozitii bombe ratate jucator 1

int redPositionPlayer2[9];                            //vector cu pozitii nave atacate jucator 1
int greenPositionPlayer2[119];                        //vector cu pozitii bombe ratate jucator 1

int indexPlayer1 = 0;                                 //index nave jucator 1
int indexPlayer2 = 0;                                 //index nave jucator 2

int indexRedPlayer1 = 0;                              //index nave atacate jucator 1
int indexRedPlayer2 = 0;                              //index nave atacate jucator 2

int indexGreenPlayer1 = 0;                            //index bombe ratate jucator 1
int indexGreenPlayer2 = 0;                            //index bombe ratate jucator 2

int player1_ships = 3;                                //cate are de plasat jucator 1
int player2_ships = 3;                                //cate are de plasat jucator 2

int remain_player1_ships = 3;                         //cate barci mai are jucator 1
int remain_player2_ships = 3;                         //cate barci mai are jucator 2

int current_ship_size = 1;                            // Dimensiunea bărcii curente pe care jucătorul o plasează
int current_ship_position_x = 0;                      // Poziția X curentă a bărcii
int current_ship_position_y = 0;                      // Poziția Y curentă a bărcii

int current_ship_size_2 = 1;                          // Dimensiunea bărcii curente pe care jucătorul o plasează
int current_ship_position_x_2 = 0;                    // Poziția X curentă a bărcii
int current_ship_position_y_2 = 0;                    // Poziția Y curentă a bărcii

int player1_board[16][8] = {                          //matricea ce contine led-urile jucatorului 1
  0, 31, 32, 63, 64, 95, 96, 127,
  1, 30, 33, 62, 65, 94, 97, 126,
  2, 29, 34, 61, 66, 93, 98, 125,
  3, 28, 35, 60, 67, 92, 99, 124,
```

```
  4, 27, 36, 59, 68, 91, 100, 123,
  5, 26, 37, 58, 69, 90, 101, 122,
  6, 25, 38, 57, 70, 89, 102, 121,
  7, 24, 39, 56, 71, 88, 103, 120,
  8, 23, 40, 55, 72, 87, 104, 119,
  9, 22, 41, 54, 73, 86, 105, 118,
  10,21, 42, 53, 74, 85, 106, 117,
  11,20, 43, 52, 75, 84, 107, 116,
  12,19, 44, 51, 76, 83, 108, 115,
  13,18, 45, 50, 77, 82, 109, 114,
  14,17, 46, 49, 78, 81, 110, 113,
  15,16, 47, 48, 79, 80, 111, 112
};

int player2_board[16][8] = {                    //matricea ce contine led-urile jucatorului 2
  128, 159, 160, 191, 192, 223, 224, 255,
  129, 158, 161, 190, 193, 222, 225, 254,
  130, 157, 162, 189, 194, 221, 226, 253,
  131, 156, 163, 188, 195, 220, 227, 252,
  132, 155, 164, 187, 196, 219, 228, 251,
  133, 154, 165, 186, 197, 218, 229, 250,
  134, 153, 166, 185, 198, 217, 230, 249,
  135, 152, 167, 184, 199, 216, 231, 248,
  136, 151, 168, 183, 200, 215, 232, 247,
  137, 150, 169, 182, 201, 214, 233, 246,
  138, 149, 170, 181, 202, 213, 234, 245,
  139, 148, 171, 180, 203, 212, 235, 244,
  140, 147, 172, 179, 204, 211, 236, 243,
  141, 146, 173, 178, 205, 210, 237, 242,
  142, 145, 174, 177, 206, 209, 238, 241,
  143, 144, 175, 176, 207, 208, 239, 240
};
                                        //culorile folosite
uint32_t color_blue = strip.Color(0, 0, 5);
uint32_t color_yellow = strip.Color(5, 5, 0);
uint32_t color_purple = strip.Color(20, 5, 45);
uint32_t color_red = strip.Color(5, 0, 0);
uint32_t color_green = strip.Color(0, 5, 0);
uint32_t color_black = strip.Color(0, 0, 0);

void fillBackground(int player[16][8], uint32_t color) {        //functie ce seteaza plansa unui jucator in albastru
   for (int i = 0; i < 16; i++) {
     for(int j=0; j<8;j++){
       strip.setPixelColor(player[i][j], color); // Setam led cu led plansa unui jucator in albastru
     }

   }
   strip.show();
}

void changeColorPlayer2(int x, int y, uint32_t color) {         //functie ce modifica culoarea unui led pentru
jucator 2
  strip.setPixelColor(player2_board[x][y], color);//setam in pozitia specificata pe plansa jucatorului 2 cu
culoarea oferita ca parametru
  strip.show();
}

void changeColorPlayer1(int x, int y, uint32_t color) {         //functie ce modifica culoarea unui led pentru
jucator 1
```

```
  strip.setPixelColor(player1_board[x][y], color);//setam in pozitia specificata pe plansa jucatorului 1 cu
culoarea oferita ca parametru
  strip.show();
}

void placeShip_player2(int x, int y, int size, uint32_t color) {   //functie pentru plasarea unei nave pentru jucator
2
  for(int i=0; i<=size; i++){
    strip.setPixelColor(player2_board[x+i][y], color);//setam de la pozitia x pana la x+i led-urile in culoarea
oferita ca parametru
  }
}

void placeBombPlayer1(int bomb_x, int bomb_y){                  //functie de plasare a unei bombe pentru jucator
1
  int xValue, yValue, buttonState;    //variabile de stocare a starii joystick-ului
  buttonState = digitalRead(JOY_BTN); //citim starea butonului
  int flagButton = 0; //flag daca avem butonul apasat
  int flagOver = 0;   //flag daca bomba se afla deasupra unei nave
  buttonState ? flagButton = 1 : flagButton = 0;  //setam flag-ul de buton
  changeColorPlayer2(bomb_x, bomb_y, color_black);//pornim prima bomba
  delay(100);
   while(flagButton == 1){ //cat timp nu apasam butonul
    // Actualizam valorile citite de joystick
    xValue = analogRead(JOY_X);
    yValue = analogRead(JOY_Y);
    buttonState = digitalRead(JOY_BTN);
    buttonState ? flagButton = 1 : flagButton = 0;
    delay(100);
     for(int i=0; i<9; i++){ // parcurgem vectorul de nave
      int flagRed = 0; // pt rosu
      for(int j=0; j<9; j++){ // parcurgem navele lovite

          if(player2_board[bomb_x][bomb_y] == redPositionPlayer2[j]){
             flagRed = 1; //setam flag ul pentru rosu daca ne aflam cu bomba peste un led deja rosu
          }}

          if(player2_board[bomb_x][bomb_y] == shipsPlayer2[i] && flagRed == 0){
            flagOver = 1;//setam flag ul pentru buton daca ne aflam cu bomba peste un led roz
            break;
          }else{
            flagOver = 0;
        }


    }
    if ((xValue == 0) && (yValue >= 400 && yValue <= 600)){// Deplasare la stânga
      if((bomb_x - 1) >= 0){ // daca putem muta matricea la stanga
        int flagBlue = 1; // flag pentru blue
        if(flagOver == 1){ // daca suntem peste o nava, dar nu apasam butonul, ne deplasam la stanga cu led ul
          changeColorPlayer2(bomb_x, bomb_y, color_purple);
          changeColorPlayer2(bomb_x-1, bomb_y, color_black);
        }else{
          for(int i=0; i<=indexRedPlayer2; i++){
            if(player2_board[bomb_x][bomb_y] == redPositionPlayer2[i]){
              changeColorPlayer2(bomb_x, bomb_y, color_red);
              changeColorPlayer2(bomb_x-1, bomb_y, color_black);
              flagBlue=0;
              break;
            }
```

```
    }
    for(int i=0; i<=indexGreenPlayer2; i++){
     if(player2_board[bomb_x][bomb_y] == greenPositionPlayer2[i]){
      changeColorPlayer2(bomb_x, bomb_y, color_green);
      changeColorPlayer2(bomb_x-1, bomb_y, color_black);
      flagBlue=0;
      break;
     }
    }
    if(flagBlue){
     changeColorPlayer2(bomb_x, bomb_y, color_blue);
     changeColorPlayer2(bomb_x-1, bomb_y, color_black);
    }
   }

   delay(100);
   bomb_x = bomb_x - 1;
  }
 }

 if ((xValue >= 1000 && xValue <= 1023) && (yValue >= 400 && yValue <= 600)) { // Deplasare la dreapta
  if((bomb_x + 1) < 15){
   int flagBlue=1;
   if(flagOver == 1){ // daca suntem peste o nava, dar nu apasam butonul, ne deplasam la dreapta cu led ul
    changeColorPlayer2(bomb_x, bomb_y, color_purple);
    changeColorPlayer2(bomb_x+1, bomb_y, color_black);
   }else{
    for(int i=0; i<=indexRedPlayer2; i++){
     if(player2_board[bomb_x][bomb_y] == redPositionPlayer2[i]){
      changeColorPlayer2(bomb_x, bomb_y, color_red);
      changeColorPlayer2(bomb_x+1, bomb_y, color_black);
      flagBlue=0;
      break;
     }
    }
    for(int i=0; i<=indexGreenPlayer2; i++){
     if(player2_board[bomb_x][bomb_y] == greenPositionPlayer2[i]){
      changeColorPlayer2(bomb_x, bomb_y, color_green);
      changeColorPlayer2(bomb_x+1, bomb_y, color_black);
      flagBlue=0;
      break;
     }
    }
    if(flagBlue){
     changeColorPlayer2(bomb_x, bomb_y, color_blue);
     changeColorPlayer2(bomb_x+1, bomb_y, color_black);
    }
   }
   delay(100);
   bomb_x = bomb_x + 1;
  }
 }

 if ((yValue == 0) && (xValue >= 400 && xValue <= 600)) {// Deplasare în sus
  if((bomb_y + 1) <= 7){ // daca suntem peste o nava, dar nu apasam butonul, ne deplasam in sus cu led ul
   int flagBlue=1;
   if(flagOver == 1){
    changeColorPlayer2(bomb_x, bomb_y, color_purple);
    changeColorPlayer2(bomb_x, bomb_y+1, color_black);
   }else{
```

```
    for(int i=0; i<=indexRedPlayer2; i++){
     if(player2_board[bomb_x][bomb_y] == redPositionPlayer2[i]){
      changeColorPlayer2(bomb_x, bomb_y, color_red);
      changeColorPlayer2(bomb_x, bomb_y+1, color_black);
      flagBlue=0;
      break;
     }
    }
    for(int i=0; i<=indexGreenPlayer2; i++){
     if(player2_board[bomb_x][bomb_y] == greenPositionPlayer2[i]){
      changeColorPlayer2(bomb_x, bomb_y, color_green);
      changeColorPlayer2(bomb_x, bomb_y+1, color_black);
      flagBlue=0;
      break;
     }
    }
    if(flagBlue){
     changeColorPlayer2(bomb_x, bomb_y, color_blue);
     changeColorPlayer2(bomb_x, bomb_y+1, color_black);
    }
   }
   delay(100);
   bomb_y = bomb_y + 1;
  }
 }
 if ((yValue >= 1000 && yValue <= 1023) && (xValue >= 400 && xValue <= 600)) {// Deplasare în jos
  if((bomb_y - 1) >= 0){
   int flagBlue=1;
   if(flagOver == 1){ // daca suntem peste o nava, dar nu apasam butonul, ne deplasam in jos cu led ul
    changeColorPlayer2(bomb_x, bomb_y, color_purple);
    changeColorPlayer2(bomb_x, bomb_y-1, color_black);
   }else{
    for(int i=0; i<=indexRedPlayer2; i++){
     if(player2_board[bomb_x][bomb_y] == redPositionPlayer2[i]){
      changeColorPlayer2(bomb_x, bomb_y, color_red);
      changeColorPlayer2(bomb_x, bomb_y-1, color_black);
      flagBlue=0;
      break;
     }
    }
    for(int i=0; i<=indexGreenPlayer2; i++){
     if(player2_board[bomb_x][bomb_y] == greenPositionPlayer2[i]){
      changeColorPlayer2(bomb_x, bomb_y, color_green);
      changeColorPlayer2(bomb_x, bomb_y-1, color_black);
      flagBlue=0;
      break;
     }
    }
    if(flagBlue){
     changeColorPlayer2(bomb_x, bomb_y, color_blue);
     changeColorPlayer2(bomb_x, bomb_y-1, color_black);
    }

   }
   delay(100);
   bomb_y = bomb_y - 1;
  }
 }
 buttonState = digitalRead(JOY_BTN);
 buttonState ? flagButton = 1 : flagButton = 0;
```

```
   delay(100);

  }
  int position_bomb =20;

   if(flagButton == 0){ //daca apasam butonul
    for(int i=0; i<indexGreenPlayer2; i++){ //green
     if(player2_board[bomb_x][bomb_y] == greenPositionPlayer2[i]){
      changeColorPlayer2(bomb_x, bomb_y, color_green);
      flagExitPressed = 1;
     }
    }
    for(int i=0; i<indexRedPlayer2; i++){ //red
     if(player2_board[bomb_x][bomb_y] == redPositionPlayer2[i]){
      changeColorPlayer2(bomb_x, bomb_y, color_red);
      flagExitPressed = 1;
     }
    }
    if(flagExitPressed == 0){
     for(int i=0;i<9;i++){
      if(player2_board[bomb_x][bomb_y] == shipsPlayer2[i]){
       position_bomb = i;
       remain_player2_ships --;
       break;
      }
     }
    if(position_bomb > 9){// daca am ratat o nava
     changeColorPlayer2(bomb_x, bomb_y, color_green);
     greenPositionPlayer2[indexGreenPlayer2] = player2_board[bomb_x][bomb_y];
     indexGreenPlayer2++;
    }
    if(position_bomb == 0 || position_bomb == 1){ //daca lovim nava 1
     for (int i = 0; i < 16; i++) {
      for(int j=0; j<8;j++){
       if(player2_board[i][j] == shipsPlayer2[0] || player2_board[i][j] == shipsPlayer2[1]){
        changeColorPlayer2(i,j,color_red);
        redPositionPlayer2[indexRedPlayer2] = player2_board[i][j];
        indexRedPlayer2++;
       }
      }

     }
    }

    if(position_bomb == 2 || position_bomb == 3|| position_bomb == 4){//daca lovim nava 2
     for (int i = 0; i < 16; i++) {
      for(int j=0; j<8;j++){
       if(player2_board[i][j] == shipsPlayer2[2] || player2_board[i][j] == shipsPlayer2[3]|| player2_board[i][j] ==
shipsPlayer2[4]){
        changeColorPlayer2(i,j,color_red);
        redPositionPlayer2[indexRedPlayer2] = player2_board[i][j];
        indexRedPlayer2++;
       }
      }
     }
    }

    if(position_bomb == 5 || position_bomb == 6 || position_bomb == 7|| position_bomb == 8){//daca lovim
nava 3
     for (int i = 0; i < 16; i++) {
```

```
        for(int j=0; j<8;j++){
         if(player2_board[i][j] == shipsPlayer2[5] || player2_board[i][j] == shipsPlayer2[6]|| player2_board[i][j] ==
shipsPlayer2[7]||player2_board[i][j] == shipsPlayer2[8]){
            changeColorPlayer2(i,j,color_red);
            redPositionPlayer2[indexRedPlayer2] = player2_board[i][j];
            indexRedPlayer2++;
          }
         }

       }
      }
      }
     delay(200);
    }
  return;
}

void placeBombPlayer2(int bomb_x, int bomb_y){                  //functie de plasare a unei bombe pentru jucator
2
  //logica similara cu functia de la jucatorul 1
  int xValue, yValue, buttonState;  //Actualizam starea joystick-ului
  buttonState = digitalRead(JOY_BTN);
  int flagButton = 0;
  int flagOver = 0;
  buttonState ? flagButton = 1 : flagButton = 0; // setam flag pentru buton
  changeColorPlayer1(bomb_x, bomb_y, color_black);//pornim cu prima bomba
  delay(100);
   while(flagButton == 1){
    // Actualizarea valorilor citite ale joystickului
    xValue = analogRead(JOY_X);
    yValue = analogRead(JOY_Y);
    buttonState = digitalRead(JOY_BTN);
    buttonState ? flagButton = 1 : flagButton = 0;
    delay(100);
    for(int i=0; i<9; i++){
      int flagRed = 0; // pt mov
      for(int j=0; j<9; j++){

         if(player1_board[bomb_x][bomb_y] == redPositionPlayer1[j]){
            flagRed = 1;
         }}

      if(player1_board[bomb_x][bomb_y] == shipsPlayer1[i] && flagRed == 0){
        flagOver = 1;
        break;
      }else{
        flagOver = 0;
      }
    }
    if ((xValue == 0) && (yValue >= 400 && yValue <= 600)){// Deplasare la stânga
      if((bomb_x - 1) >= 0){
        int flagBlue = 1;
        if(flagOver == 1){
          changeColorPlayer1(bomb_x, bomb_y, color_yellow);
          changeColorPlayer1(bomb_x-1, bomb_y, color_black);
        }else{
          for(int i=0; i<=indexRedPlayer1; i++){
            if(player1_board[bomb_x][bomb_y] == redPositionPlayer1[i]){
              changeColorPlayer1(bomb_x, bomb_y, color_red);
              changeColorPlayer1(bomb_x-1, bomb_y, color_black);
```

```
        flagBlue=0;
        break;
       }
      }
      for(int i=0; i<=indexGreenPlayer1; i++){
       if(player1_board[bomb_x][bomb_y] == greenPositionPlayer1[i]){
        changeColorPlayer1(bomb_x, bomb_y, color_green);
        Serial.println("fac verde la stanga");
        changeColorPlayer1(bomb_x-1, bomb_y, color_black);
        flagBlue=0;
        break;
       }
      }
      if(flagBlue){
       changeColorPlayer1(bomb_x, bomb_y, color_blue);
       changeColorPlayer1(bomb_x-1, bomb_y, color_black);
      }

  }
  delay(100);
  bomb_x = bomb_x - 1;
  }
}

if ((xValue >= 1000 && xValue <= 1023) && (yValue >= 400 && yValue <= 600)) { // Deplasare la dreapta
  if((bomb_x + 1) < 15){
     int flagBlue=1;
   if(flagOver == 1){
     changeColorPlayer1(bomb_x, bomb_y, color_yellow);
     changeColorPlayer1(bomb_x+1, bomb_y, color_black);
   }else{
     for(int i=0; i<indexRedPlayer1; i++){
       if(player1_board[bomb_x][bomb_y] == redPositionPlayer1[i]){
         changeColorPlayer1(bomb_x, bomb_y, color_red);
         changeColorPlayer1(bomb_x+1, bomb_y, color_black);
         flagBlue=0;
         break;
       }
     }
     for(int i=0; i<indexGreenPlayer1; i++){
       if(player1_board[bomb_x][bomb_y] == greenPositionPlayer1[i]){
         changeColorPlayer1(bomb_x, bomb_y, color_green);
         changeColorPlayer1(bomb_x+1, bomb_y, color_black);
         flagBlue=0;
         break;
       }
     }
     if(flagBlue){
       if(shipsPlayer1[0] != 400){
         changeColorPlayer1(0, 0, color_yellow);
       }
       changeColorPlayer1(bomb_x, bomb_y, color_blue);
       changeColorPlayer1(bomb_x+1, bomb_y, color_black);
     }
   }
   delay(100);
   bomb_x = bomb_x + 1;
  }
}
```

```
if ((yValue == 0) && (xValue >= 400 && xValue <= 600)) {// Deplasare în sus
  if((bomb_y + 1) <= 7){
    int flagBlue=1;
    if(flagOver == 1){
      changeColorPlayer1(bomb_x, bomb_y, color_yellow);
      changeColorPlayer1(bomb_x, bomb_y+1, color_black);
    }else{
      for(int i=0; i<indexRedPlayer1; i++){
        if(player1_board[bomb_x][bomb_y] == redPositionPlayer1[i]){
          changeColorPlayer1(bomb_x, bomb_y, color_red);
          changeColorPlayer1(bomb_x, bomb_y+1, color_black);
          flagBlue=0;
          break;
        }
      }
      for(int i=0; i<indexGreenPlayer1; i++){
        if(player1_board[bomb_x][bomb_y] == greenPositionPlayer1[i]){
          changeColorPlayer1(bomb_x, bomb_y, color_green);
          changeColorPlayer1(bomb_x, bomb_y+1, color_black);
          flagBlue=0;
          break;
        }
      }
      if(flagBlue){
        if(shipsPlayer1[0] != 400){
          changeColorPlayer1(0, 0, color_yellow);
        }
        changeColorPlayer1(bomb_x, bomb_y, color_blue);
        changeColorPlayer1(bomb_x, bomb_y+1, color_black);
      }
    }
    delay(100);
    bomb_y = bomb_y + 1;
  }
}
if ((yValue >= 1000 && yValue <= 1023) && (xValue >= 400 && xValue <= 600)) {// Deplasare în jos
  if((bomb_y - 1) >= 0){
    int flagBlue=1;
    if(flagOver == 1){
      changeColorPlayer1(bomb_x, bomb_y, color_yellow);
      changeColorPlayer1(bomb_x, bomb_y-1, color_black);
    }else{
      for(int i=0; i<=indexRedPlayer1; i++){
        if(player1_board[bomb_x][bomb_y] == redPositionPlayer1[i]){
          changeColorPlayer1(bomb_x, bomb_y, color_red);
          changeColorPlayer1(bomb_x, bomb_y-1, color_black);
          flagBlue=0;
          break;
        }
      }
      for(int i=0; i<=indexGreenPlayer1; i++){
        if(player1_board[bomb_x][bomb_y] == greenPositionPlayer1[i]){
          changeColorPlayer1(bomb_x, bomb_y, color_green);
          changeColorPlayer1(bomb_x, bomb_y-1, color_black);
          flagBlue=0;
          break;
        }
      }
      if(flagBlue){
        changeColorPlayer1(bomb_x, bomb_y, color_blue);
```

```
        changeColorPlayer1(bomb_x, bomb_y-1, color_black);
       }

     }
     delay(100);
     bomb_y = bomb_y - 1;
    }
   }
   buttonState = digitalRead(JOY_BTN);
   buttonState ? flagButton = 1 : flagButton = 0;
   delay(100);

}

int position_bomb = 20;
if(flagButton == 0){
    int flagExitPressed = 0;
    for(int i=0; i<indexGreenPlayer1; i++){ //green
      if(player1_board[bomb_x][bomb_y] == greenPositionPlayer1[i]){
        changeColorPlayer1(bomb_x, bomb_y, color_green);
        Serial.println("fac verde la testare vector verzi");
        flagExitPressed = 1;
      }
    }
    for(int i=0; i<indexRedPlayer1; i++){ //red
      if(player1_board[bomb_x][bomb_y] == redPositionPlayer1[i]){
        changeColorPlayer1(bomb_x, bomb_y, color_red);
        flagExitPressed = 1;
      }
    }
    if(flagExitPressed == 0){
      for(int i=0;i<9;i++){
        if(player1_board[bomb_x][bomb_y] == shipsPlayer1[i]){
          position_bomb = i;
          remain_player1_ships --;
          break;
        }
      }
    if(position_bomb > 9){
      changeColorPlayer1(bomb_x, bomb_y, color_green);
      greenPositionPlayer1[indexGreenPlayer1] = player1_board[bomb_x][bomb_y];
      indexGreenPlayer1++;
    }
    if(position_bomb == 0 || position_bomb == 1){    //daca atacam nava 1
      for (int i = 0; i < 16; i++) {
        for(int j=0; j<8;j++){
         if(player1_board[i][j] == shipsPlayer1[0] || player1_board[i][j] == shipsPlayer1[1]){
           changeColorPlayer1(i,j,color_red);
           redPositionPlayer1[indexRedPlayer1] = player1_board[i][j];
           indexRedPlayer1++;
         }
        }

      }
    }

    if(position_bomb == 2 || position_bomb == 3|| position_bomb == 4){//daca atacam nava 2
      for (int i = 0; i < 16; i++) {
        for(int j=0; j<8;j++){
```

```
        if(player1_board[i][j] == shipsPlayer1[2] || player1_board[i][j] == shipsPlayer1[3]|| player1_board[i][j] == shipsPlayer1[4]){
          changeColorPlayer1(i,j,color_red);
          redPositionPlayer1[indexRedPlayer1] = player1_board[i][j];
          indexRedPlayer1++;
        }
       }

      }
     }

    if(position_bomb == 5 || position_bomb == 6 || position_bomb == 7|| position_bomb == 8){//daca atacam nava 3
      for (int i = 0; i < 16; i++) {
       for(int j=0; j<8;j++){
        if(player1_board[i][j] == shipsPlayer1[5] || player1_board[i][j] == shipsPlayer1[6]|| player1_board[i][j] == shipsPlayer1[7]||player1_board[i][j] == shipsPlayer1[8]){
          changeColorPlayer1(i,j,color_red);
          redPositionPlayer1[indexRedPlayer1] = player1_board[i][j];
          indexRedPlayer1++;
        }
       }

      }
     }
     }
    delay(200);
   }
   return;
}

void moveShip_player1(int copie_x, int copie_y, int size, int numberShips) { //functie de miscare a unei nave pentru jucator 1
  int poz_init_x = copie_x;
  int poz_init_y = copie_y;
  size = size +1;
  int xValue, yValue, buttonState;
  buttonState = digitalRead(JOY_BTN);
  int flagButton = 0;
  Serial.print("starea initiala buton: ");
  Serial.println(buttonState);
  buttonState ? flagButton = 1 : flagButton = 0;
  while(flagButton == 1){
   // Actualizarea valorilor citite ale joystickului
   xValue = analogRead(JOY_X);
   yValue = analogRead(JOY_Y);
   buttonState = digitalRead(JOY_BTN);
   buttonState ? flagButton = 1 : flagButton = 0;
   delay(100);

   if ((xValue == 0) && (yValue >= 400 && yValue <= 600)){// Deplasare la stânga
    if((copie_x - size) >= 0){
     placeShip_player1(copie_x, copie_y, size-1,color_blue);
     placeShip_player1(copie_x-size, copie_y, size-1, color_yellow);
     reloadFieldPlayer1();
     strip.show();
     delay(100);
     copie_x = copie_x - size;
     //Serial.print("copie_x_stanga:");
     //Serial.println(copie_x);
```

```
  }
 }

 if ((xValue >= 1000 && xValue <= 1023) && (yValue >= 400 && yValue <= 600)) { // Deplasare la dreapta
  if(numberShips == 1){
     if((copie_x + size + 3) < 15){
     placeShip_player1(copie_x, copie_y, size-1, color_blue);
     placeShip_player1(copie_x+size, copie_y, size-1, color_yellow);
     reloadFieldPlayer1();
     strip.show();
     delay(100);
     copie_x = copie_x + size;
     //Serial.print("copie_x_dreapta:");
     //Serial.println(copie_x);
    }
  }else{
     if((copie_x + size) < 15){
     placeShip_player1(copie_x, copie_y, size-1, color_blue);
     placeShip_player1(copie_x+size, copie_y, size-1, color_yellow);
     reloadFieldPlayer1();
     strip.show();
     delay(100);
     copie_x = copie_x + size;
     //Serial.print("copie_x_dreapta:");
     //Serial.println(copie_x);
    }
  }
 }



 if ((yValue == 0) && (xValue >= 400 && xValue <= 600)) {// Deplasare în sus
  if((copie_y + 1) <= 7){
    placeShip_player1(copie_x, copie_y, size-1, color_blue);
    placeShip_player1(copie_x, copie_y+1, size-1, color_yellow);
    reloadFieldPlayer1();
    strip.show();
    delay(100);
    copie_y = copie_y + 1;
    //Serial.print("copie_y_sus:");
    //Serial.println(copie_y);
  }
 }

 if ((yValue >= 1000 && yValue <= 1023) && (xValue >= 400 && xValue <= 600)) {// Deplasare în jos
  if((copie_y - 1) >= 0){
    placeShip_player1(copie_x, copie_y, size-1, color_blue);
    placeShip_player1(copie_x, copie_y-1, size-1, color_yellow);
    reloadFieldPlayer1();
    strip.show();
    delay(100);
    copie_y = copie_y - 1;
    //Serial.print("copie_y_jos:");
    //Serial.println(copie_y);
  }
 }
 buttonState = digitalRead(JOY_BTN);
 buttonState ? flagButton = 1 : flagButton = 0;
 delay(100);
}
```

```
  buttonState ? flagButton = 1 : flagButton = 0;
   delay(100);
   if(flagButton == 0){

       if(numberShips == 2){ // daca suntem cu nava 3
         for(int i=0; i<9; i++){
           Serial.println("intru in if numberShips == 2");
           if((player1_board[copie_x][copie_y] == shipsPlayer1[i])||(player1_board[copie_x+1][copie_y] ==
shipsPlayer1[i])||(player1_board[copie_x+2][copie_y] == shipsPlayer1[i])){
             current_ship_position_x=0;
             current_ship_position_y=0;
             player1_ships++;
             current_ship_size--;
             reloadFieldBluePlayer1();

             return;
           }
         }
       }

       if(numberShips == 1){ // daca suntem la prima nava
         for(int i=0; i<9; i++){
           if((player1_board[copie_x][copie_y] == shipsPlayer1[i])||(player1_board[copie_x+1][copie_y] ==
shipsPlayer1[i])||(player1_board[copie_x+2][copie_y] == shipsPlayer1[i])||(player1_board[copie_x+3][copie_y]
== shipsPlayer1[i])){
             current_ship_position_x=1;
             current_ship_position_y=1;
             player1_ships++;
             current_ship_size--;
             reloadFieldBluePlayer1();
             return;
           }
         }
       }


     for(int i=0;i<size;i++){
       shipsPlayer1[indexPlayer1] = player1_board[copie_x+i][copie_y];
       indexPlayer1++;
     }
   }
  return;
}

void placeShip_player1(int x, int y, int size, uint32_t color) {   //functie pentru plasarea unei nave pentru jucator
1
  for(int i=0; i<=size; i++){
    strip.setPixelColor(player1_board[x+i][y], color);//setam de la pozitia x pana la x+i led-urile in culoarea
oferita ca parametru
  }
}

void reloadFieldPlayer1(){                              // functie de resetare a pozitiilor navelor jucator 1
  for(int i=0; i<9; i++){
    if(shipsPlayer1[i] != 400){
      for (int j = 0; j < 16; j++) {
        for(int k=0; k<8;k++){
          if(player1_board[j][k] == shipsPlayer1[i]){
            changeColorPlayer1(j, k, color_yellow);
```

```
        }
       }
     }
    }
   }
}

void reloadFieldBluePlayer1(){                          // functie de resetare a plansei jucator 1

  for (int j = 0; j < 16; j++) {
     for(int k=0; k<8;k++){
           changeColorPlayer1(j, k, color_blue);
      }
     }
  reloadFieldPlayer1();
}

void reloadFieldBluePlayer2(){                          // functie de resetare a plansei jucator 2

  for (int j = 0; j < 16; j++) {
     for(int k=0; k<8;k++){
           changeColorPlayer2(j, k, color_blue);
      }
     }
  reloadFieldPlayer2();
}

void reloadFieldPlayer2(){                              // functie de resetare a pozitiilor navelor jucator 1
  for(int i=0; i<9; i++){
    if(shipsPlayer2[i] != 0){
      for (int j = 0; j < 16; j++) {
         for(int k=0; k<8;k++){
           if(player2_board[j][k] == shipsPlayer2[i]){
            changeColorPlayer2(j, k, color_purple);
           }
         }
      }
    }
   }
}

void moveShip_player2(int copie_x, int copie_y, int size, int numberShips) {//functie de miscare a unei nave
pentru jucator 2
  size = size +1;
  int xValue, yValue, buttonState;
  buttonState = digitalRead(JOY_BTN);
  int flagButton = 0;
  Serial.print("starea initiala buton: ");
  Serial.println(buttonState);
  buttonState ? flagButton = 1 : flagButton = 0;
  while(flagButton == 1){
    // Actualizarea valorilor citite ale joystickului
    xValue = analogRead(JOY_X);
    yValue = analogRead(JOY_Y);
    buttonState = digitalRead(JOY_BTN);
    buttonState ? flagButton = 1 : flagButton = 0;
    delay(100);

    if ((xValue == 0) && (yValue >= 400 && yValue <= 600)){// Deplasare la stânga
      if((copie_x - size) >= 0){
```

```
    placeShip_player2(copie_x, copie_y, size-1,color_blue);
    placeShip_player2(copie_x-size, copie_y, size-1, color_purple);
    reloadFieldPlayer2();
    strip.show();
    delay(100);
    copie_x = copie_x - size;
    //Serial.print("copie_x_stanga:");
    //Serial.println(copie_x);
   }
}

if ((xValue >= 1000 && xValue <= 1023) && (yValue >= 400 && yValue <= 600)) { // Deplasare la dreapta
  if(numberShips == 1){
     if((copie_x + size + 3) < 15){
       placeShip_player2(copie_x, copie_y, size-1, color_blue);
       placeShip_player2(copie_x+size, copie_y, size-1, color_purple);
       reloadFieldPlayer2();
       strip.show();
       delay(100);
       copie_x = copie_x + size;
       //Serial.print("copie_x_dreapta:");
       //Serial.println(copie_x);
    }
  }else{
    if((copie_x + size) < 15){
      placeShip_player2(copie_x, copie_y, size-1, color_blue);
      placeShip_player2(copie_x+size, copie_y, size-1, color_purple);
      reloadFieldPlayer2();
      strip.show();
      delay(100);
      copie_x = copie_x + size;
      //Serial.print("copie_x_dreapta:");
      //Serial.println(copie_x);
    }
  }
}

if ((yValue == 0) && (xValue >= 400 && xValue <= 600)) {// Deplasare în sus
  if((copie_y + 1) <= 7){
    placeShip_player2(copie_x, copie_y, size-1, color_blue);
    placeShip_player2(copie_x, copie_y+1, size-1, color_purple);
    reloadFieldPlayer2();
    strip.show();
    delay(100);
    copie_y = copie_y + 1;
   // Serial.print("copie_y_sus:");
    //Serial.println(copie_y);
  }
}

if ((yValue >= 1000 && yValue <= 1023) && (xValue >= 400 && xValue <= 600)) {// Deplasare în jos
  if((copie_y - 1) >= 0){
    placeShip_player2(copie_x, copie_y, size-1, color_blue);
    placeShip_player2(copie_x, copie_y-1, size-1, color_purple);
    reloadFieldPlayer2();
    strip.show();
    delay(100);
    copie_y = copie_y - 1;
    //Serial.print("copie_y_jos:");
    //Serial.println(copie_y);
```

```
    }
   }
   buttonState = digitalRead(JOY_BTN);
   buttonState ? flagButton = 1 : flagButton = 0;
   delay(100);
  }

  buttonState ? flagButton = 1 : flagButton = 0;
   delay(100);
   if(flagButton == 0){
    if(numberShips == 2){ // daca suntem cu nava 3
        for(int i=0; i<9; i++){
         if((player2_board[copie_x][copie_y] == shipsPlayer2[i])||(player2_board[copie_x+1][copie_y] ==
shipsPlayer2[i])||(player2_board[copie_x+2][copie_y] == shipsPlayer2[i])){
           current_ship_position_x_2=0;
           current_ship_position_y_2=0;
           player2_ships++;
           current_ship_size_2--;
           reloadFieldBluePlayer2();
           return;
         }
        }
      }

      if(numberShips == 1){ // daca suntem la prima nava
        for(int i=0; i<9; i++){
         if((player2_board[copie_x][copie_y] == shipsPlayer2[i])||(player2_board[copie_x+1][copie_y] ==
shipsPlayer2[i])||(player2_board[copie_x+2][copie_y] == shipsPlayer2[i])||(player2_board[copie_x+3][copie_y]
== shipsPlayer2[i])){
           current_ship_position_x_2=1;
           current_ship_position_y_2=1;
           player2_ships++;
           current_ship_size_2--;
           // lcd.setCursor(3, 0);
           //  lcd.print("Nave su");
           //  delay(100);
           reloadFieldBluePlayer2();
           return;
         }
        }
      }


    for(int i=0;i<size;i++){
     shipsPlayer2[indexPlayer2] = player2_board[copie_x+i][copie_y];
     indexPlayer2++;
    }
    Serial.println("BUTONUL A FOST APASAT");
   }
  return;
}

// void afisareVector(int *vector, int size){                  //functie pentru debugging
//    for(int i=0; i<size; i++){
//      Serial.print("shipsPlayer[");
//      Serial.print(i);
//      Serial.print("]=");
//      Serial.println(vector[i]);
//    }
// }
```

```
void setup(){


  Serial.begin(9600);
  strip.begin();
  fillBackground(player1_board, color_blue);
  fillBackground(player2_board, color_blue);

  pinMode(JOY_BTN, INPUT_PULLUP); // Activarea rezistenței de pull-up

  Wire.begin();
  lcd.init();
  lcd.backlight();
  lcd.setCursor(3, 0);
  lcd.print("Start joc!");
  delay(100);
}

void loop() {
  int copie_x, copie_y;
  delay(100);
  lcd.setCursor(7, 1);
    lcd.print(player1_ships);
    lcd.print("-");
    lcd.print(player2_ships);
    delay(100);
  while(player1_ships != 0){
    int flagBoat3=0;
    Serial.print("Cate barci mai am de pus? ");
    Serial.println(player1_ships);
    if(player1_ships <= 2){
      lcd.setCursor(3, 0);
      lcd.print("Jucatorul 1");
      delay(100);
      if(player1_board[current_ship_position_x][current_ship_position_y] == shipsPlayer1[indexPlayer1-1]){
        flagBoat3=1;
        current_ship_position_x = current_ship_position_x + current_ship_size;
      }else{
        if(player1_board[current_ship_position_x+1][current_ship_position_y]== shipsPlayer1[indexPlayer1-1]){
          flagBoat3=1;
          current_ship_position_x = current_ship_position_x + current_ship_size + 1;
        }
        else{
          if(player1_board[current_ship_position_x+2][current_ship_position_y]== shipsPlayer1[indexPlayer1-1]){
            flagBoat3=1;
          current_ship_position_x = current_ship_position_x + current_ship_size + 2;
          }else{
            if(player1_board[current_ship_position_x+3][current_ship_position_y]== shipsPlayer1[indexPlayer1-1]){
              flagBoat3=1;
              current_ship_position_x = current_ship_position_x + current_ship_size + 3;
            }else{
              if(player1_ships == 1 && flagBoat3 == 0){
                if(player1_board[current_ship_position_x+4][current_ship_position_y]== shipsPlayer1[indexPlayer1-
1]){

                  current_ship_position_x = current_ship_position_x + current_ship_size + 3;
                }
                }

              }
```

```
        }
       }
      }
     }

   placeShip_player1(current_ship_position_x, current_ship_position_y, current_ship_size,color_yellow);
   copie_x = current_ship_position_x;
   copie_y = current_ship_position_y;
   strip.show();
   delay(100);
   moveShip_player1(copie_x, copie_y, current_ship_size, player1_ships);
   strip.show();
   //afisareVector(shipsPlayer1, 9);
   current_ship_position_x++;
   current_ship_position_y++;
   current_ship_size++;
   player1_ships = player1_ships - 1;
  }


  while(player2_ships != 0){
   Serial.print("Cate barci mai am de pus? ");
   Serial.println(player2_ships);
   if(player2_ships <= 2){
    lcd.setCursor(3, 0);
    lcd.print("Jucatorul 2");
    delay(100);
    if(player2_board[current_ship_position_x_2][current_ship_position_y_2] == shipsPlayer2[indexPlayer2-1]){
     current_ship_position_x_2 = current_ship_position_x_2 + current_ship_size_2;
    }else{
     if(player2_board[current_ship_position_x_2+1][current_ship_position_y_2]== shipsPlayer2[indexPlayer2-
1]){
      current_ship_position_x_2 = current_ship_position_x_2 + current_ship_size_2 + 1;
     }
     else{
      if(player2_board[current_ship_position_x_2+2][current_ship_position_y_2]== shipsPlayer2[indexPlayer2-
1]){
      current_ship_position_x_2 = current_ship_position_x_2 + current_ship_size_2 + 2;
      }else{
       if(player2_board[current_ship_position_x_2+3][current_ship_position_y_2]==
shipsPlayer2[indexPlayer2-1]){
         current_ship_position_x_2 = current_ship_position_x_2 + current_ship_size_2 + 3;
        }else{
         if(player2_ships == 1 ){
          if(player2_board[current_ship_position_x_2 + 4][current_ship_position_y_2]==
shipsPlayer2[indexPlayer2-1]){
            current_ship_position_x_2 = current_ship_position_x_2 + current_ship_size_2 + 3;
           }
           }

         }
      }
     }
    }
   }
   placeShip_player2(current_ship_position_x_2, current_ship_position_y_2,
current_ship_size_2,color_purple);
   copie_x = current_ship_position_x_2;
   copie_y = current_ship_position_y_2;
   strip.show();
```

```
   delay(100);
   moveShip_player2(copie_x, copie_y, current_ship_size_2, player2_ships);
   strip.show();
   current_ship_position_x_2++;
   current_ship_position_y_2++;
   current_ship_size_2++;

   player2_ships = player2_ships - 1;
  }
  //afisareVector(shipsPlayer2, 10);
  while(remain_player1_ships != 0 && remain_player2_ships!=0){
   lcd.setCursor(3, 0);
   lcd.print("Jucatorul 1");
   delay(100);
   placeBombPlayer1(0, 0);
   delay(100);
   lcd.setCursor(7, 1);
   lcd.print(remain_player1_ships);
   lcd.print("-");
   lcd.print(remain_player2_ships);
   delay(100);

   if(remain_player2_ships == 0){
    break;
   }
   lcd.setCursor(3, 0);
   lcd.print("Jucatorul 2");
   delay(100);
   placeBombPlayer2(0, 0);
   delay(100);
   lcd.setCursor(7, 1);
   lcd.print(remain_player1_ships);
   lcd.print("-");
   lcd.print(remain_player2_ships);
   delay(100);
  }
  lcd.setCursor(3, 0);
  lcd.print("Stop joc!   ");
  delay(100);
  placeBombPlayer2(0, 0);
  delay(100);
  lcd.setCursor(7, 1);
  lcd.print(remain_player1_ships);
  lcd.print("-");
  lcd.print(remain_player2_ships);
  delay(100);
}
```

Bibliografie:

Arduino Mega Pinout | Arduino Mega 2560 Layout, Specifications (electronicshub.org)

Arduino Mega 2560 Rev3 — Arduino Official Store

Arduino - Wikipedia

Arduino Forum

Matrice LED RGB, Elektroweb, 16x16, WS2812B, NeoPixel, 60mA - eMAG.ro

Feature (adafruit.com)

Joystick analogic cu 3 axe, Cu buton, Negru - eMAG.ro

KY-023-Joy-IT.pdf (datasheetspdf.com)

HS320240A (mouser.com)

Modul interfata I2C pentru LCD1602 | Bitmi.ro✓