UNIVERSITATEA POLITEHNICA TIMIŞOARA

FACULTATEA DE AUTOMATICA ŞI CALCULATOARE

SISTEME INCORPORATE

IMPLEMENTAREA JOCULUI BATTLESHIP

Daniel Vincze

Andreea-Oana Zăvadă

An III, CTI, subgrupa 6.2

An universitar 2017-2018

**1.Enunţul proiectului:**

Implementarea jocului Battleship, folosind un microcontroler şi o matrice cu LED-uri (2 studenţi)

Caracteristici:

• Se va împărţi afişajul în două regiuni egale, una pentru fiecare jucător;

• Se vor genera cel puţin două forme diferite pentru fiecare jucător, plasate aleator pe spaţiul de joc;

• Poziţia de lovire va fi selectată de un jucător folosind 2 butoane pentru direcţiile dreapta, jos. Confirmarea poziţiei se va realiza folosind un alt buton.

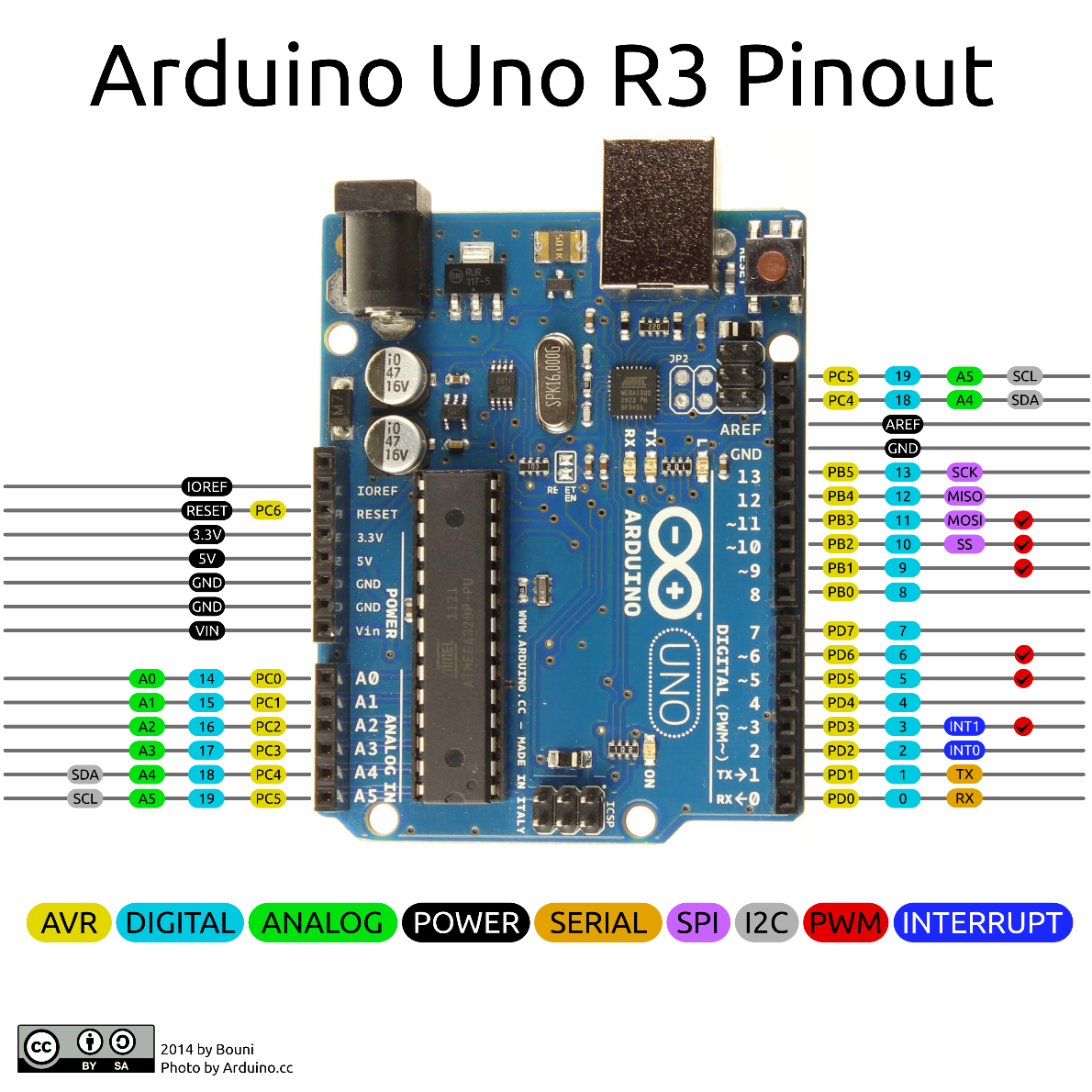
**2.Descrierea plăcii de dezvoltare utilizate:**

**Arduino UNO** este o placa de dezvoltare bazata pe microcontrolerul ATmega328. Consta într-o platforma de mici dimensiuni (6.8 cm / 5.3 cm – în cea mai des întâlnită variantă) construită în jurul unui procesor de semnal şi este capabilă de a prelua date din mediul înconjurător printr-o serie de senzori şi de a efectua acţiuni asupra mediului prin intermediul luminilor, motoarelor, servomotoarelor, şi alte tipuri de dispozitive mecanice. Procesorul este capabil să ruleze cod scris într-un limbaj de programare care este foarte similar cu limbajul C++.

Placa Arduino UNO se conectează la portul USB al calculatorului folosind un cablu de tip USB A-B, disponibil în varianta de 1.5 metri. Poate fi alimentată extern (din priză) folosind un alimentator extern. Alimentarea externă este necesara în situaţia în care consumatorii conectaţi la placă necesită un curent mai mare de câteva sute de miliamperi. În caz contrar, placa se poate alimenta direct din PC, prin cablul USB.

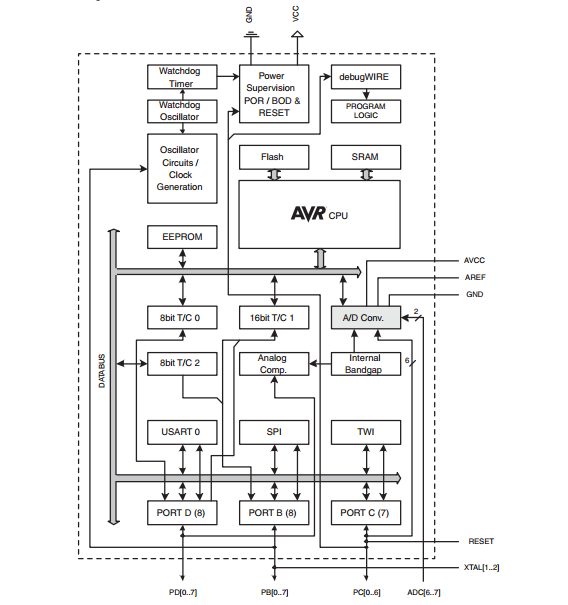
**Caracteristici:**

* 14 pini digitali de input/output (din care 6 pot fi folosiţi ca output-uri de PWM)
* 6 pini analogici de input
* un cristal de cuarţ de 16MHz
* o conexiune USB
* un jack de putere
* un header ICSP
* un buton de reset.
* Configuraţie Pinout :



**Microcontroler: Atmel ATMega328P**

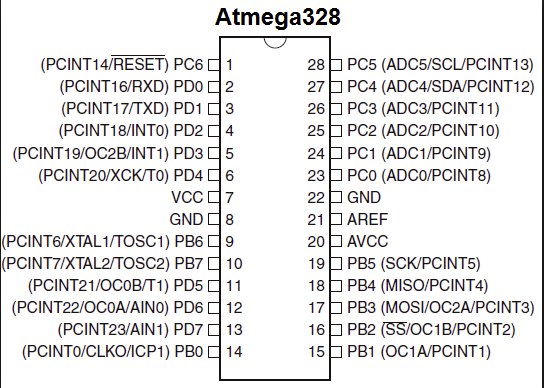
Schema bloc:



**Specificaţii :**

* Microcontroler: ATmega328
* Tensiune de lucru: 5V
* Tensiune de intrare (recomandat): 7-12V
* Tensiune de intrare (limită):  6-20V
* Pini digitali: 14 (6 PWM output)
* Pini analogici: 6
* Curent per pin I/O: 40 mA
* Curent 3.3V: 50 mA
* Memorie Flash: 32 KB (ATmega328)  0.5 KB pentru bootloader
* SRAM: 2 KB (ATmega328)
* EEPROM: 1 KB (ATmega328)
* Clock Speed: 16 MHz

**Configuraţia pinilor:**



**3.Descrierea detaliată a modulelor microcontrolerului care au fost implicate în realizarea proiectului de faţă:**

**1.Modulul de întreruperi:**

Procesorul de pe placa Arduino are 2 tipuri diferite de întreruperi : externe şi “pin change”.

Pentru întreruperile externe există doar 2 pini configuraţi pe microcontrolerul ATmega 328P , adică în cazul Arduino Uno , aceştia fiind mapaţi la pinii 2 şi 3. Aceste întreruperi pot fi setate sa se activeze atât la fronturile crescătoare cat şi descrescătoare ale unui semnal sau la nivelul inferior al semnalului. Activările sunt interpretate de hardware iar întreruperea este foarte rapida.

În celălalt caz , al întreruperilor bazate pe “pin change”, acestea pot fi folosite cu toţi ceilalţi pini ai plăcii de dezvoltare. Acest tip de întreruperi pot fi activate doar pe fronturile crescătoare şi descrescătoare ale semnalelor.

**2.Mini Push Button Switch**

Acest tip de butoane reprezintă switch-uri Single Pull Single Throw în miniatură. Acestea sunt de calitate ridicata Omron de tip B3F folosite momentan pe switchuri. Se folosesc pentru resetări switch tactile. Se montează direct într-un bread board standard. Funcţionează pana la 50 mA.

**3.Bread Board**

Un Bread Board, numit şi protoboard, reprezintă o bază de construcţie pentru realizarea de prototipuri în electronică.

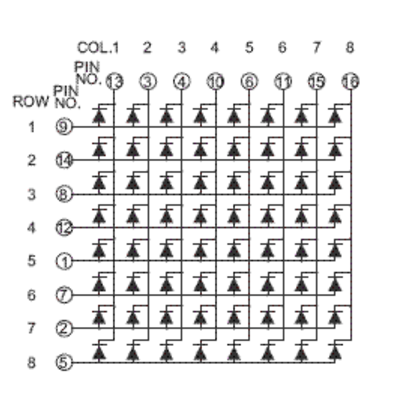
Termenul de “breadboard” este de obicei utilizat pentru a face referire la faptul că acest gen de placă nu necesită lipituri pentru a realiza conexiunea elementelor (fiind de genul plugboard). Datorită faptului că elementele nu trebuie lipite pe placă, aceasta poate fi refolosită. Acest lucru facilitează crearea de noi prototipuri temporare şi de diferite design-uri de circuite. Variante mai vechi de breadboard nu dispuneau de aceasta facilitate.

**4.Explicarea funcţionării sistemului de afişaj ales (schema + descriere):**

MOD-LED 8x8 este o matrice (de dimensiune 50 x 50 mm) de led-uri înlănţuibilă care iţi permite sa obţii afişaje cu led-uri de orice dimensiune pur şi simplu conectând doua sau mai multe matrici împreună.

Proiectul va folosi 2 matrice led de mărime 8\*8 , fiecare conectată la câte un cip MAX7219.

Schema de circuit a unei matrice:



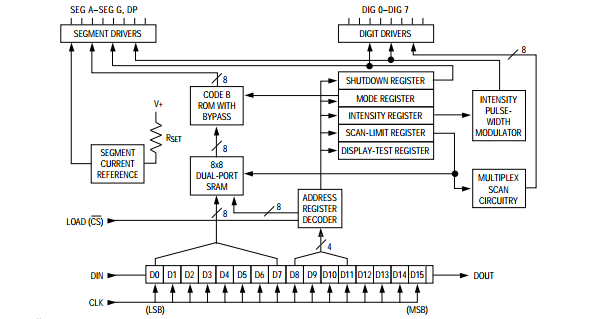
**5.Scurtă descriere a circuitului dedicat utilizat pentru realizarea proiectului:**

Proiectul foloseşte 2 circuite MAX7219 pentru a realiza comanda celor 2 matrice de LED-uri.

Acestea pot comanda un display cu led-uri cu catod comun de maxim 64 de LED-uri (o matrice de 8\*8) . În cip sunt incluse un decodificator BCD , circuite multiplexate de scanare, un static RAM de mărime 8x8 în care se memorează fiecare digit.  
Pentru a seta curentul pentru toate LED-urile este necesar un singur rezistor extern (În acest proiect se va folosi unul de 10KΩ).

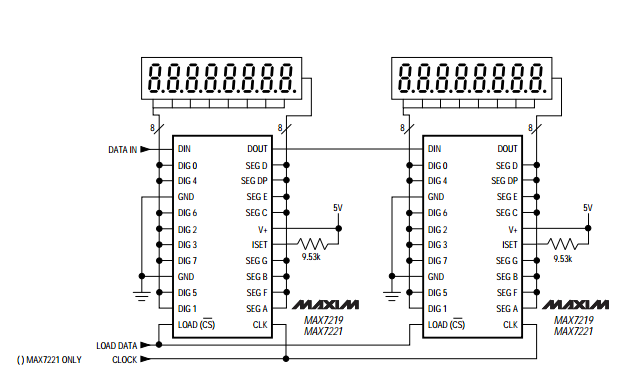
MAX7219 este compatibil cu SPI, QSPI şi Microwire. Proiectul va folosi interfaţa SPI pentru a transmite date circuitului integrat.

Diagrama funcţională:



Conectarea celor 2 matrice de LED-uri se va face prin cascadarea a 2 circuite MAX7219. Astfel se vor folosi doar 3 pini ai plăcii Arduino şi vor putea fi comandate individual toate cele 128 de LED-uri.

Exemplu de cascadare a 2 MAX7219 , folosind display-uri cu 7 segmente, cu 8 digiţi fiecare:



# **Biblioteci folosite:**

Biblioteca folosită de program se numeşte LedControl.

Aceasta permite adresarea individuală a LED-urilor conectate la un număr maxim de 8 circuite MAX72xx.

Acest lucru se face prin crearea în program a unei variabile de tip LedControl.

Ex: LedControl lc=LedControl(12,11,10,1);

Ca parametri la creare se specifică pinul de DATA\_IN , pinul de CLK ,pinul de LOAD şi numărul de MAX72xx conectate in cascadă.

Biblioteca permite setarea Led-urilor in power saving mode (toate led-urile oprite) , setarea intensităţii acestora şi adresarea atât individuală cât şi pe coloane/linii a led-urilor conectate la MAX72xx.

Programul foloseşte doar funcţia de setare individuală a unui LED in funcţia vUpdateBoard(). Se iterează peste toate LED-urile şi se setează in funcţie de valoarea din matricea interna a programului.   
0 in matrice presupune ca LED-ul corespunzător este oprit, orice alt număr presupune ca acesta este pornit.

Funcţia pentru setarea individuala a LED-urilor este :  
setLed(int address,int row , int col, Boolean value);

Address – adresa circuitului integrat (0 sau 1 in cazul actual)

Row/col – rândul şi coloana led-ului.

Value – true pentru pornit , false pentru oprit.

**6.Conectarea şi descrierea hardware :**

**7.Program, comentarii şi descriere:**

#include "LedControlMS.h"

#include <binary.h>

/\*

\* Vom avea nevoie de un obiect LedControl pentru a controla ledurile matricii

\* pin 12 pentru DIN

\* pin 11 pentru CLK

\* pin 10 pentru LOAD

\* Folosim doua matrici de tipul MAX72XX

\*

\*/

#define DEVICES 2

int datainPin = 12;

int clkPin = 11;

int loadPin = 10;

int numDevices = 2;

/\*

\* Butonul ce va misca navele spre dreapta se va plasa pe pinul 2

\* Cel pentru a muta navele in jos este plasat pe pinul 3

\* Butonul cu care actionam va fi plasat la pinul 4

\*

\*/

int rightButtonPin = 3; //blue

int downButtonPin = 2; //red

int fireButtonPin = 5; //yellow

//Obiect creat pentru controlarea matricii de leduri

LedControl lc = LedControl(datainPin, clkPin, loadPin, DEVICES);

//matrici backend, pentru user si cpu

/\*

\* valori posibile:

\* 0 - apa

\* 1 - nava

\* 2 - zona lovita anterior

\*

\*/

int playerMatrix[8][8];

int cpuMatrix[8][8];

unsigned int smileyFace[8] = { // sir de randuri de biti pentru afisarea unei fete zambarete

B00111100,

B01000010,

B10100101,

B10000001,

B10100101,

B10011001,

B01000010,

B00111100

};

unsigned int straightFace[8] = { // sir de randuri de biti pentru afisarea unei fete lungi

B00111100,

B01000010,

B10100101,

B10000001,

B10111101,

B10000001,

B01000010,

B00111100

};

int pRow, pColumn; //randul si coloana curenta a user-ului

/\*

\* lastHit va retine starea in care se afla CPU la un moment dat in timpul jocului

\* starea 0 - CPU inca nu a lovit nicio nava, se alege o pozitie aleatoare la urmatoarea tura

\* starea 1 - CPU a lovit o pozitie a navei, se alege o pozitie adiacenta la urmatoarea tura

\* starea 2 - CPU a lovit inca o pozitie a navei din starea 1, la acest moment se cunoaste daca e nava orizontala sau verticala, si se alege

\* urmatoarea miscare in functie de pozitia navei. Daca nava e orizontala, se distruge complet in aceasta stare si se revine la starea 0

\* starea 3 - CPU mai are de nimerit o pozitie a navei verticale

\*/

int lastHit = 0;

// aici se salveaza randul si coloana aleasa de CPU in tura anterioara. Se alege miscarea urmatoare in functie de asta

int cpuPrev[2] = {

0, // prev row

0 // prev column

};

int cpuNextPos[8] = { // pozitiile urmatoare posibile pentru un atac mai inteligent, valaore initiala: -1

// -1 poate insemna si pozitie indisponibila (afara din tabla de joc / pozitie deja lovita)

-1, -1 // upper

-1, -1 // to the right

-1, -1 // lower

-1, -1 // to the left

};

// valori boolene pentru a identifica daca nava lovita e orizontala sau verticala

int cpuHorizontal = 0;

int cpuVertical = 0;

//initializeaza valorile random ale unei matrici

/\*

\* se genereaza valori la intamplare pentru pozitia navelor unei matrici backend

\* matrix - matricea de initilizat

\*

\*/

void initializareShips(int matrix[8][8]){

//initializare ship orizontal de marime 4

int row = random(0,8); //se genereaza orice rand

int column = random(0,5); //se genereaza pana la coloana 4 deoarece nava are o marime de 4 leduri

for(int i = column; i < column + 4; i++){

//Serial.println(i);

matrix[row][i] = 1; //valorile alese se seteaza pe 1, acestea se folosesc pentru a aprinde ledurile

}

//initializare ship vertical de marime 3

int column1 = random(0,8); //initial, se genereaza pe orice coloana

int row1 = random(0,6); // se generaza pana la randul 5 pentru ca nava are o marime de 3 leduri

while(column1 < column + 4 && column1 > column - 1){ //cat timp navele se suprapun, se genereaza nava in alt loc

if(row1 > row || row1 < row - 2){

break;

}

else{

column1 = random(0,8);

row1 = random(0,6);

}

}

//mai intai se genereaza capul navei, dupa care se genereaza restul corpului

for(int i = row1; i < row1 + 3; i++){

//Serial.println(i);

matrix[i][column1] = 1; //valorile alese se seteaza pe 1

}

}

//afisarea efectiva pe matricea de leduri, folosind matricile backend

// matrix - matricea de afisat

void afisareMatrix(int matrix[8][8]){

for(int i = 0; i < 8; i++){ //pentru fiecare rand din matrix

unsigned int binary = 0; //se seteaza o valoare binary ce va salva valoarea efectiva a unui rand in functie de combinatia binara a randului

for(int j = 0; j < 8; j++){ //in aceasta bucla, pentru fiecare coloana a randului i, se aduna valoarea binara la binary

//Serial.print(matrix[i][j]);

//Serial.print("Row: "); Serial.print(i); Serial.print(": "); Serial.print(binary); Serial.println(); Serial.println();

if(matrix[i][j] == 1){

binary += (matrix[i][j] << (7-j)); //formula de translatie binar -> zecimal (valoarea shiftata la stanga cu 7 - <nr. coloana> pozitii)

}

}

//Serial.println();

//Serial.print("Row: "); Serial.print(i); Serial.print(": "); Serial.print(binary); Serial.println(); Serial.println();

//Serial.print("row: "); Serial.print(i); Serial.print(";"); Serial.print(binary); Serial.println();

lc.setRow(0, i , binary); //afisarea randului calculat pe matricea de leduri

}

}

/\*

void debugCPU(){

Serial.println("Matrice oponent:");

for(int i = 0; i < 8; i++){

for(int j = 0; j < 8; j++){

Serial.print(cpuMatrix[i][j]);

}

Serial.println();

}

}

\*/

//initializarea jocului

void setup() {

Serial.begin(9600);

randomSeed(analogRead(0)); //seed pentru functia random()

for(int i = 0; i < numDevices; i++){

lc.shutdown(i,false); //dezactivare power save pentru matricea i

lc.setIntensity(i,8); // setare intensitate leduri pentru matricea i (max: 8)

lc.clearDisplay(i); //opreste toate ledurile

}

//initializare matricilor backend (0 inseamna led off)

for(int i = 0; i < 8; i++){

for(int j = 0; j < 8; j++){

playerMatrix[i][j] = 0;

cpuMatrix[i][j] = 0;

}

}

initializareShips(playerMatrix); //generam matricea user-ului

delay(200);

initializareShips(cpuMatrix); //generam matricea oponentului

delay(200);

afisareMatrix(playerMatrix); //afisarea efectiva a ledurilor

Serial.println();

//debugCPU();

//setarea butoanelor ca pini de input

pinMode(rightButtonPin,INPUT);

pinMode(downButtonPin,INPUT);

pinMode(fireButtonPin,INPUT);

}

//efect de blink pentru alegerea shot-ului. Se va stinge si se va aprinde de doua ori led-ul ales

/\*

\* dev - indexul matricii de leduri

\* row - randul ledului ales

\* column - coloana ledului ales

\*

\*/

void playerBlink(int dev, int row, int column){

delay(150);

for(int i = 0; i < 2; i++){

lc.setLed(dev, row, column, false);

delay(500);

lc.setLed(dev, row, column, true);

delay(500);

}

}

// alegerea shot-ului de catre user

/\*

\* row - randul ledului ales

\* column - coloana ledului ales

\*

\*/

void playerShot(int row, int column){

playerBlink(1, row, column); //efect de blink pentru shot-ul ales

if(cpuMatrix[row][column] == 0){ //daca am nimerit apa

lc.setLed(1, row, column, false); //nava netintita, deci se stinge led-ul

}

//daca s-a lovit nava, atunci ledul ramane aprins din functia anterioara playerBlink()

cpuMatrix[row][column] = 2; //se seteaza pe 2 pentru a stii faptul ca aici s-a mai lovit si a sari peste aceasta zona

}

// functie ce elimina valorile imposibile din cpuNextPos. Valori imposibile inseamna coordonate deja lovite sau ce depasesc placa de joc

void eliminate(){

for(int i = 0; i < 8; i += 2){ // daca oricare dintre pozitiile anterior setate sunt imposibile, se vor reseta pe -1

if(cpuNextPos[i] < 0 || cpuNextPos[i] > 7 ||

cpuNextPos[i + 1] < 0 || cpuNextPos[i + 1] > 7 ||

playerMatrix[cpuNextPos[i]][cpuNextPos[i + 1]] == 2){

cpuNextPos[i] = -1;

cpuNextPos[i + 1] = -1;

}

}

}

/\*

void displayCPU(){

Serial.println();

Serial.println("cpuPrev: ");

for(int i = 0; i < 2; i++){

Serial.print(cpuPrev[i]);

Serial.print(" ");

}

Serial.println();

Serial.println("cpuNextPos : ");

for (int i = 0; i < 8; i++){

Serial.println(cpuNextPos[i]);

}

}

\*/

// alegerea oponentului

/\*

\* alegerea se face generand randul si coloana ledului aleator prin functia random()

\*/

void CPUShot(){

delay(200);

/\*

\* row - randul curent ales

\* column - coloana aleasa curent

\* randPos - o pozitie aleatoare generata cu random()

\*

\*/

int row, column, randPos;

switch(lastHit){

case 0: //starea 0, nicio portiune de nava nu a fost lovita

do{ //se genereaza o pozitie aleatoare nelovita anterior

row = random(0,8);

delay(200);

column = random(0,8);

}while(playerMatrix[row][column] == 2); //daca s-a ales deja un loc in care s-a impuscat deja, se genereaza altul

if(playerMatrix[row][column] == 1){ // daca s-a lovit o nava

lastHit = 1; // se trece in starea 1

// se salveaza row si column in cpuPrev

cpuPrev[0] = row;

cpuPrev[1] = column;

// daca nava verticala nu a fost inca distrusa, cpuNextPos ia valorile de mai sus si de mai jos de ledul curent lovit

if(cpuVertical != -1){

cpuNextPos[0] = row - 1; cpuNextPos[1] = column; //upper

cpuNextPos[4] = row + 1; cpuNextPos[5] = column; //lower

}

//daca nava orizontala nu a fost inca distrusa, cpuNextPos ia valorile din stanga si din dreapta ledului curent lovit

if(cpuHorizontal != -1){

cpuNextPos[2] = row; cpuNextPos[3] = column + 1; //to the right

cpuNextPos[6] = row; cpuNextPos[7] = column - 1; //to the left

}

}

break;

case 1: // starea 1

do{ //se genereaza o pozitie aleatoare din cele 4 pozitii din cpuNextPos

randPos = random(0,4);

randPos \*= 2; //vom lua doar valorile lui x (multiplu de 2 in matricea cpuNextPos)

}while(cpuNextPos[randPos] == -1); // cat timp aceasta este posibila

// se seteaza row si column in functie de pozitia generata

row = cpuNextPos[randPos];

column = cpuNextPos[randPos + 1];

// daca pozitia aleasa este nava

if(playerMatrix[row][column] == 1){

lastHit = 2; // se trece in starea 2

if(row - cpuPrev[0] == -1){ // daca a doua tintire e mai sus de cea anterioara

cpuVertical = 1; // nava e verticala

// pentru cpuNextPos, se aleg doar pozitiile verticale posibile, cele orizontale se seteaza pe -1

cpuNextPos[0] = row - 1; cpuNextPos[1] = column;

cpuNextPos[2] = -1; cpuNextPos[3] = -1;

cpuNextPos[4] = row + 2; cpuNextPos[5] = column;

cpuNextPos[6] = -1; cpuNextPos[7] = -1;

}

else if(row - cpuPrev[0] == 1){ //a doua tintire e mai jos de cea anterioara

cpuVertical = 1; // la fel ca mai sus, doar ca se pun alte valori in cpuNextPos

cpuNextPos[0] = row - 2; cpuNextPos[1] = column;

cpuNextPos[2] = -1; cpuNextPos[3] = -1;

cpuNextPos[4] = row + 1; cpuNextPos[5] = column;

cpuNextPos[6] = -1; cpuNextPos[7] = -1;

}

else if(column - cpuPrev[1] == -1){ // daca a doua tintire e la stanga celei anterioare

cpuHorizontal = 1; // nava e orizontala

// se aleg doar pozitiile orizontale, restul se seteaza pe -1

cpuNextPos[0] = -1; cpuNextPos[1] = -1;

cpuNextPos[2] = row; cpuNextPos[3] = column + 2;

cpuNextPos[4] = -1; cpuNextPos[5] = -1;

cpuNextPos[6] = row; cpuNextPos[7] = column - 1;

}

else if(column - cpuPrev[1] == 1){ // daca a doua tintire e la dreapta celei anterioare

cpuHorizontal = 1; // analog

cpuNextPos[0] = -1; cpuNextPos[1] = -1;

cpuNextPos[2] = row; cpuNextPos[3] = column + 1;

cpuNextPos[4] = -1; cpuNextPos[5] = -1;

cpuNextPos[6] = row; cpuNextPos[7] = column - 2;

}

// in final, se seteaza cpuPrev cu valorile curente

cpuPrev[0] = row;

cpuPrev[1] = column;

}

break;

case 2: //starea 2

do{

randPos = random(0,4);

randPos \*= 2; //vom lua doar valorile lui x (multiplu de 2 in matricea cpuNextPos)

}while(cpuNextPos[randPos] == -1);

row = cpuNextPos[randPos];

column = cpuNextPos[randPos + 1];

if(playerMatrix[row][column] == 1){ //daca valoarea generata e nava

if(cpuVertical){ // daca a fost nava verticala, aceasta este distrusa complet, deci se revine la starea 0

lastHit = 0;

cpuVertical = -1; // se seteaza nava verticala pe imposibil pentru a nu mai genera valori in cpuNextPos pentru aceasta

}

else if(cpuHorizontal){ // daca e nava orizontala

lastHit = 3; // se trece in starea 3

// se aleg valori pentru cpuNextPos in functie de coloana curenta si cea anterioara

if(column - cpuPrev[1] == -1){

cpuNextPos[0] = -1; cpuNextPos[1] = -1;

cpuNextPos[2] = row; cpuNextPos[3] = column + 3;

cpuNextPos[4] = -1; cpuNextPos[5] = -1;

cpuNextPos[6] = row; cpuNextPos[7] = column - 1;

}

else if(column - cpuPrev[1] == 1){

cpuHorizontal = 1;

cpuNextPos[0] = -1; cpuNextPos[1] = -1;

cpuNextPos[2] = row; cpuNextPos[3] = column + 1;

cpuNextPos[4] = -1; cpuNextPos[5] = -1;

cpuNextPos[6] = row; cpuNextPos[7] = column - 3;

}

}

}

break;

case 3: // starea 3

// se genereaza din ultimele doua pozitii posibile

do{

randPos = random(0,4);

randPos \*= 2; //vom lua doar valorile lui x (multiplu de 2 in matricea cpuNextPos)

}while(cpuNextPos[randPos] == -1);

row = cpuNextPos[randPos];

column = cpuNextPos[randPos + 1];

// daca s-a nimerit, starea se reseteaza pe 0, si nava orizontala se seteaza pe imposibil

if(playerMatrix[row][column] == 1){

lastHit = 0;

cpuHorizontal = -1;

}

break;

}

playerBlink(0, row, column); //efect de blink pentru alegere

playerMatrix[row][column] = 2; // se seteaza 2 pentru a nu impusca din nou in acelasi loc

eliminate(); // se elimina valorile imposibile din cpuNextPos

lc.setLed(0, row, column, false); //se stinge ledul ales, fie ca a nimerit, fie ca nu

}

//verifica daca tot randul a fost deja tintit pentru a-l evita

/\*

\* daca un rand a fost deja ocupat cu incercari de tintire, acel rand trebuie evitat de ledul de selectie a urmatoarei miscari

\* row - randul de verificat

\* matrix - matricea backend in care se afla randul

\*

\*/

boolean fullRow(int row, int matrix[8][8]){

for(int i = 0; i < 8; i++){

if(matrix[row][i] != 2){

return false;

}

}

return true;

}

//verifica daca intreaga coloana a fost deja tintita

/\*

\* asemenea fullRow, dar se verifica coloana

\* column - coloana de verificat

\* matrix - matricea in care se afla column

\*

\*/

boolean fullColumn(int column, int matrix[8][8]){

for(int i = 0; i < 8; i++){

if(matrix[i][column] != 2){

return false;

}

}

return true;

}

//functie de mutare a ledului spre dreapta

/\*

\* se verifica mai intai daca poozitiile din dreapta ledului aprins au fost deja incercate, iar daca da, se sare peste aceste pozitii

\* row - randul curent

\* column - coloana curenta

\*

\*/

void moveRight(){

lc.setLed(1, pRow, pColumn, false);

do { //cat timp zona curenta a fost deja lovita, se trece la urmatoarea

pColumn++;

pColumn %= 8;

while(fullRow(pRow, cpuMatrix)){ //cat timp randurile curente sunt deja lovite, se trec la urmatoarele

pRow++;

pRow %= 8;

}

}while(cpuMatrix[pRow][pColumn] == 2);

lc.setLed(1, pRow, pColumn, true); // mutarea efectiva a ledului (in frontend)

}

//functie de mutare a ledului in jos, similar moveRight, dar se interschimba row cu column

/\*

\* row - randul curent

\* column - coloana curenta

\*

\*/

void moveDown(){

lc.setLed(1, pRow, pColumn, false);

do{

pRow++;

pRow %= 8;

while(fullColumn(pColumn, cpuMatrix)){

pColumn++;

pColumn %= 8;

}

} while(cpuMatrix[pRow][pColumn] == 2);

lc.setLed(1, pRow, pColumn, true); //afisare in frontend

}

//afisare fete de final la infinit pana la apasarea butonului reset (la sfarsitul jocului)

/\*

\* face1 - fata de final a jucatorului

\* face2 - fata de final cpu-ului

\* fetele pot fi smileyFace say straightFace

\*

\*/

void displayFaces(unsigned int face1[], unsigned int face2[]){

delay(200);

for(int i = 0; i < 8; i++){

lc.setRow(0, i, face1[i]);

lc.setRow(1, i, face2[i]);

}

while(true){

;

}

}

/\*

\* functie ce verifica daca jucatorul sau cpu-ul a pierdut, pentru a termina jocul. Se verifica in functie de matricile backend a acestora.

\* daca o matrice nu mai contine nicio valoare de 1 ( nu mai exista nicio nava nedistrusa )

\* matrix - matricea de verificat

\*

\*/

int lostCheck(int matrix[8][8]){

for(int i = 0; i < 8; i++){ //se parcurge matricea, iar la prima valoare de 1 ( exista nava nedistrusa ) functia returneaza 0

for(int j = 0; j < 8; j++){

if(matrix[i][j] == 1){

return 0;

}

}

}

// altfel se returneaza 1, iar jocul se termina

return 1;

}

void loop() {

//miscarea user-ului

pRow = 0;

pColumn = -1;

moveRight();

int rightButtonState, downButtonState, fireButtonState;

while(true){

delay(150);

//citesc starile butoanelor

rightButtonState = digitalRead(rightButtonPin);

downButtonState = digitalRead(downButtonPin);

fireButtonState = digitalRead(fireButtonPin);

//daca se apasa butonul dreapta

if(rightButtonState == HIGH){

moveRight();

//Serial.println("right pressed");

}

if(downButtonState == HIGH){ //daca se apasa butonul down

moveDown();

//Serial.println("down pressed");

}

if(fireButtonState == HIGH){ //daca se apasa butonul fire

//Serial.println("fire pressed");

playerShot(pRow, pColumn); //se incearca shot-ul prin functia playerShot

break;

}

}

if(lostCheck(cpuMatrix)){ //se verifica daca CPU a pierdut

displayFaces(smileyFace, straightFace); //se afiseaza fetele finale

}

else{

CPUShot(); // altfel, e randul oponentului

if(lostCheck(playerMatrix)) //se verifica daca jucatorul a pierdut

displayFaces(straightFace, smileyFace); //se afiseaza fetele finale

}

}

**8.Bibliografie:**

<http://www.arduino.cc/en/Tutorial/Pushbutton>

<http://arduino.cc/en/reference/random>

<http://fritzing.org/home/>

<http://en.wikipedia.org/wiki/Arduino>

<http://arduino.cc/en/Main/arduinoBoardUno>

<http://en.wikipedia.org/wiki/Breadboard>

<https://www.robofun.ro/arduino_uno_v3>

<https://github.com/Dany9966/Battleship>