UNIVERSITATEA POLITEHNICA TIMIȘOARA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Anul universitar 2019 – 2020

SISTEME ÎNCORPORATE

Labirint

Proiect realizat de: Moga Cristina
Prapugicu Alex
Profesor coordonator: Sergiu Nimara

1.Enuntul proiectului

Implementarea unui joc care presupune ieşirea dintr-un labirint, folosind un microcontroler şi o matrice cu led-uri de dimensiune suficient de mare.

Caracteristici:

• Studenţii vor concepe 3-4 hărţi ale unui labirint, care vor fi afişate secvenţial pe matricea cu led-uri.

• Personajul care se deplasează prin labirint va fi reprezentat la fiecare moment prin aprinderea unui led de altă culoare decât cele care reprezintă limitele labirintului.

• Se vor utiliza 4 butoane care vor corespunde deplasării personajului în cele 4 sensuri posibile (o apăsare a unui buton corespunde deplasării cu o poziţie).

• Se va genera aleator apariţia unui obstacol care se poate deplasa, la rândul său, prin labirint. Obstacolul va fi reprezentat printr-un led de altă culoare.

• Jocul se termină cu succes dacă utilizatorul reuşeşte să ghideze personajul prin labirint până la ieşire, evitând obstacolul.

2. Descrierea plăcii de dezvoltare utilizate

Placa pe care o vom folosi este Arduino Mega 2560, bazată pe microcontrollerul Atmel AVR ATMega2560, pe 8 biti.

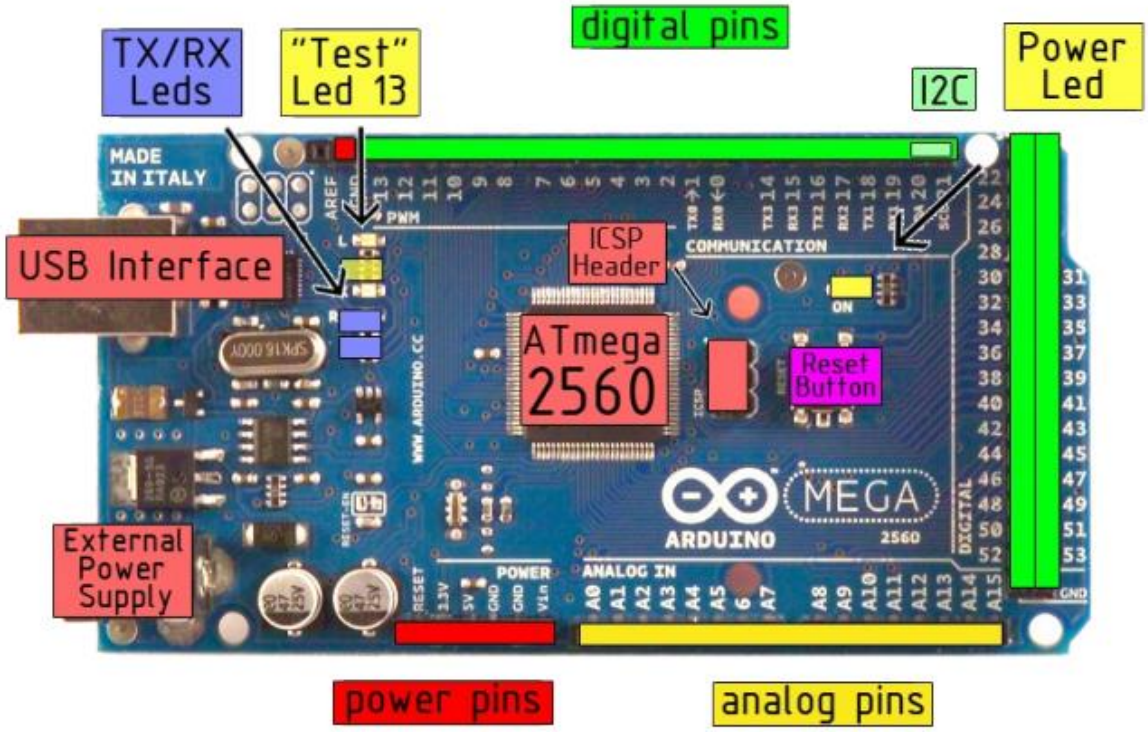
Are:

- 54 de pini de intrare / ieșire digitali (dintre care 15 pot fi folosiți că ieșiri PWM)
- 16 intrări analogice, 4 UART-uri (porturi seriale hardware)
- un oscilator de cristal de 16 MHz
- conexiune USB
- priză de alimentare, un antet ICSP
- un buton de resetare.

2.1.Specificatii:

- Microcontroller: ATmega2560
- Tensiune de alimentare: 5V
- Tensiune de intrare (recomandat): 7-12V

- Pini Digital I/O: 54 (din care 14 oferă ieșire PWM)
- Pini de intrare analogici: 16
- Curent DC pe I/O: 40 mA
- Curent DC pentru Pinul 3.3V: 50 mA
- Flash Memory: 256 KB
- Frecventa: 16MHz



2.2.Structura placii:

• Microcontroler

• Buton reset

• LED indicator funcționare

• LED-uri Tx (Indicator transmisie date), Rx (Indicator recepție date)

• LED Load. Se poate controla de la Pin 13

• USB (Sursă de alimentare și port serial pentru încărcarea programului)

• DC Power Jack (Mufă pentru alimentare placă)

• Pini pentru alimentare dispozitive externe (3.3V, 5V, GROUND)

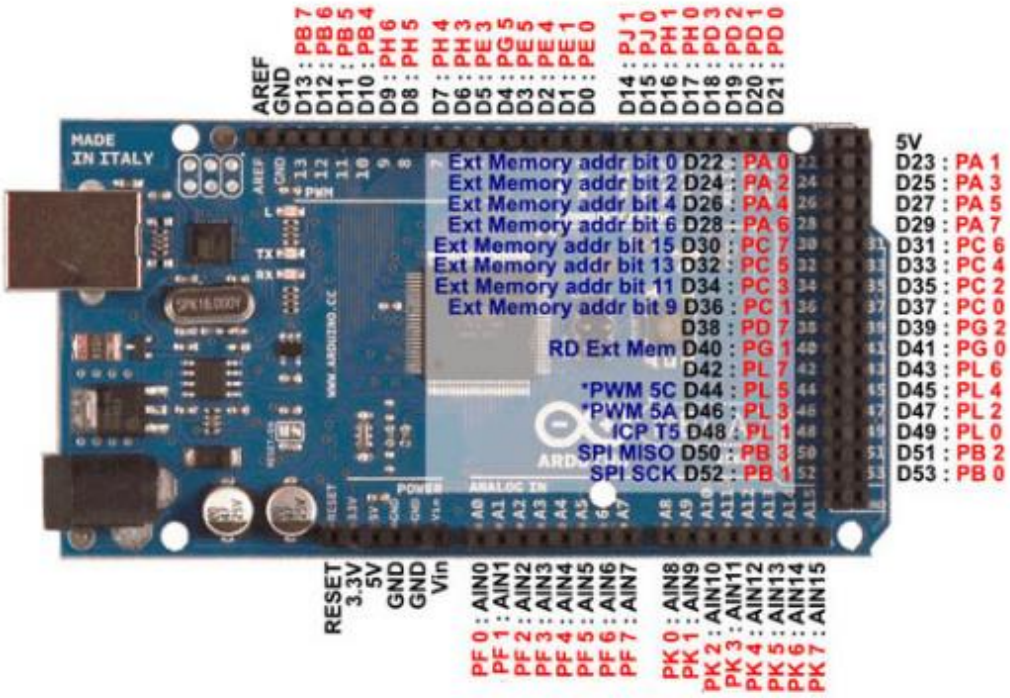
• Pini pentru alimentare placă de la baterie de 9V (Vin, GROUND)

• A0-A15. 16 pini analogici. ADC (Analog to Digital Converter). Tensiunile între 0-5V aplicate la intrarea oricăruia dintre acești pini sunt citite ca valori între 0-1024(10 biti rezolutie)

• 14 pini digitali IN/OUT. Pinii notați cu (~) sunt capabili să genereze semnal PWM (Pulsewidth modulation)

• AREF. Folosit pentru setarea unei limite superioare de referință pentru pinii analogici

2.3.Configurarea pinilor:



- Comunicarea cu calculatorul, si placa arduino se realiză prin portul **USB.**

2.4.Microcontroller ATmega328

ATmega2560 este un microcontroller cu un singur chip creat de Atmel în familia megaAVR (ulterior Microchip Technology a achiziționat Atmel în 2016).

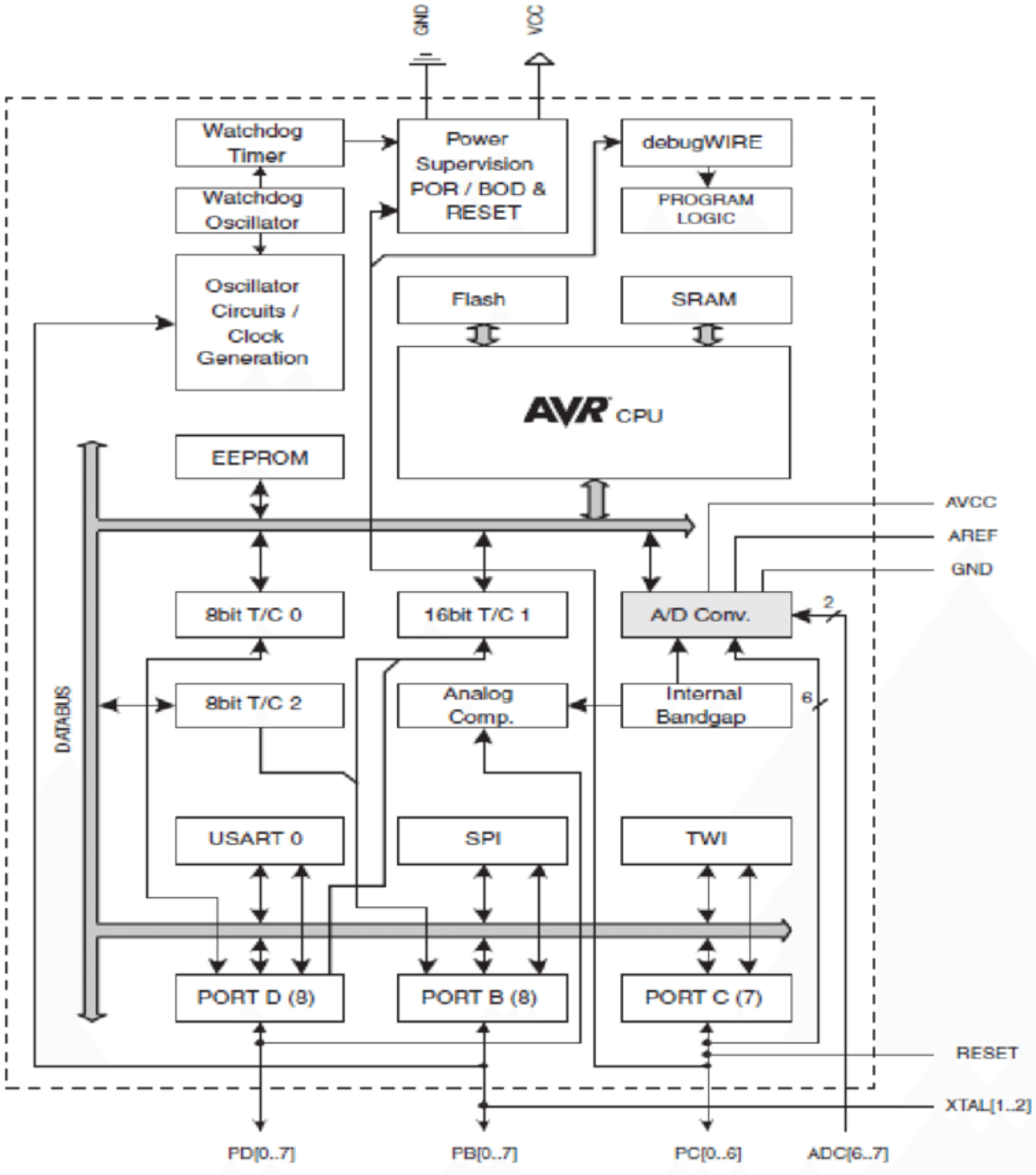
- Are un nucleu modificat al procesorului RISC pe 8 biți a arhitecturii Harvard.
- ATMega2560 este un controler de înaltă performanță, cu putere redusă.
- Este cel mai popular dintre toate controlerele AVR, deoarece este utilizat în plăci ARDUINO.

Microcontrolerul AVR RISC pe 8 biți combină:

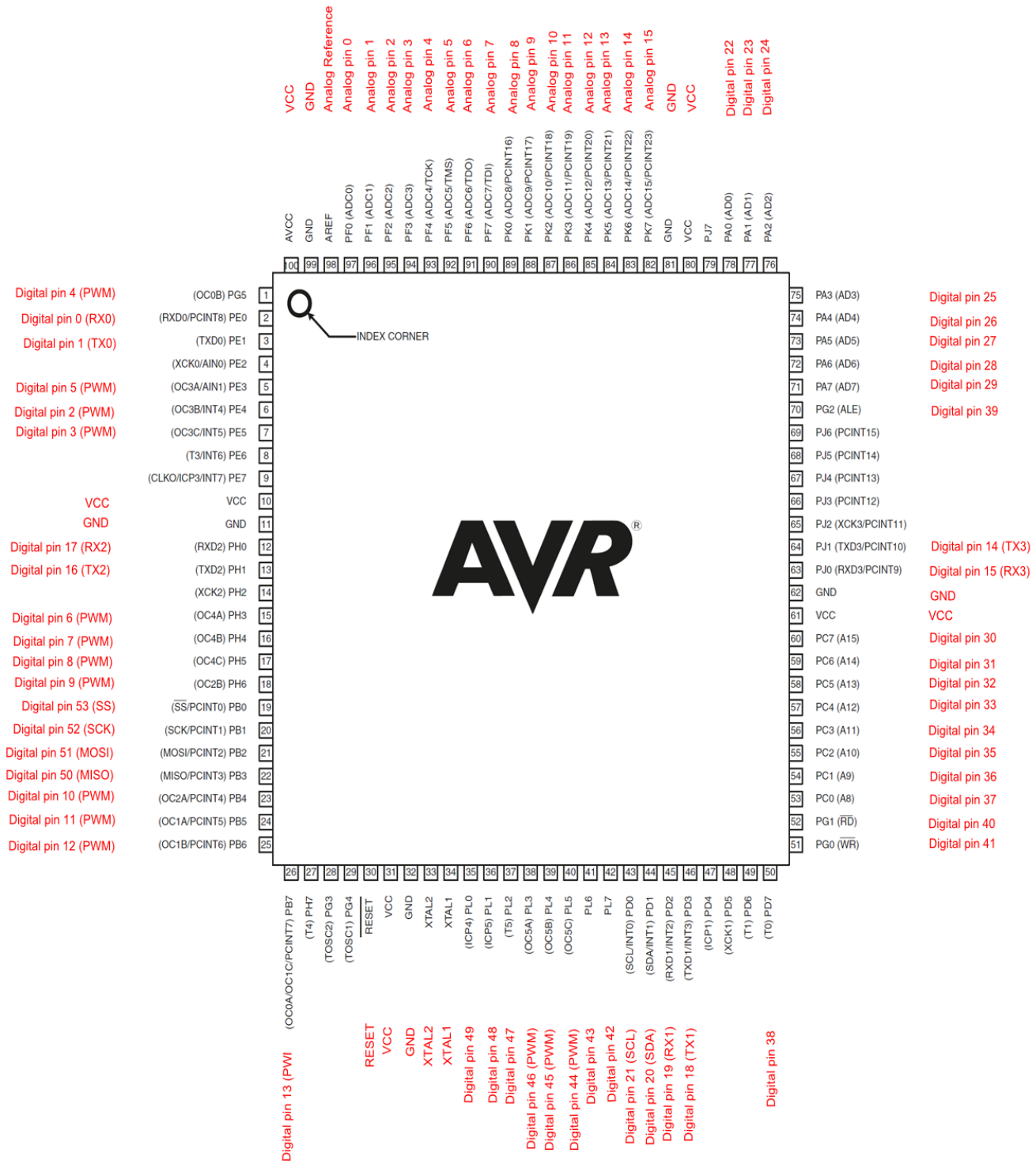
- memoria flash ISP de 256 KB cu capacități de citire-scriere dintre care 8kb bootloader
- 4 KB EEPROM
- 8 KB SRAM
- 54 linii I / O de uz general
- 32 de registre de lucru general
- trei cronometre flexibile / contoare cu moduri de comparare
- întreruperi interne și externe
- USART programabil serial
- o interfață serială cu 2 fire orientată pe octeți
- port serial SPI
- convertor A / D pe 6 canale pe 10 biți (8 canale în pachetele TQFP și QFN / MLF)
- cronometru programabil de veghe cu oscilator intern
- cinci moduri de economisire a energiei selectabile software

Aparatul funcționează între 1,8-5,5 volți. Dispozitivul realizează un randament care se apropie de 1 MIPS pe MHz.

2.4.1.Schema bloc a microcontroller-ului



2.4.2.Maparea pinilor



Pin Number	Pin Name	Mapped Pin Name
1	PG5 (OC0B)	Digital pin 4 (PWM)
2	PE0 (RXD0/PCINT8)	Digital pin 0 (RX0)
3	PE1 (TXD0)	Digital pin 1 (TX0)
4	PE2 (XCK0/AIN0)	
5	PE3 (OC3A/AIN1)	Digital pin 5 (PWM)
6	PE4 (OC3B/INT4)	Digital pin 2 (PWM)
7	PE5 (OC3C/INT5)	Digital pin 3 (PWM)
8	PE6 (T3/INT6)	
9	PE7 (CLKO/ICP3/INT7)	
10	VCC	VCC
11	GND	GND
12	PH0 (RXD2)	Digital pin 17 (RX2)
13	PH1 (TXD2)	Digital pin 16 (TX2)
14	PH2 (XCK2)	
15	PH3 (OC4A)	Digital pin 6 (PWM)
16	PH4 (OC4B)	Digital pin 7 (PWM)
17	PH5 (OC4C)	Digital pin 8 (PWM)
18	PH6 (OC2B)	Digital pin 9 (PWM)
19	PB0 (SS/PCINT0)	Digital pin 53 (SS)
20	PB1 (SCK/PCINT1)	Digital pin 52 (SCK)
21	PB2 (MOSI/PCINT2)	Digital pin 51 (MOSI)
22	PB3 (MISO/PCINT3)	Digital pin 50 (MISO)
23	PB4 (OC2A/PCINT4)	Digital pin 10 (PWM)
24	PB5 (OC1A/PCINT5)	Digital pin 11 (PWM)
25	PB6 (OC1B/PCINT6)	Digital pin 12 (PWM)
26	PB7 (OC0A/OC1C/PCINT7)	Digital pin 13 (PWM)
27	PH7 (T4)	
28	PG3 (TOSC2)	
29	PG4 (TOSC1)	
30	RESET	RESET
31	VCC	VCC

Pin Number	Pin Name	Mapped Pin Name
32	GND	GND
33	XTAL2	XTAL2
34	XTAL1	XTAL1
35	PL0 (ICP4)	Digital pin 49
36	PL1 (ICP5)	Digital pin 48
37	PL2 (T5)	Digital pin 47
38	PL3 (OC5A)	Digital pin 46 (PWM)
39	PL4 (OC5B)	Digital pin 45 (PWM)
40	PL5 (OC5C)	Digital pin 44 (PWM)
41	PL6	Digital pin 43
42	PL7	Digital pin 42
43	PD0 (SCL/INT0)	Digital pin 21 (SCL)
44	PD1 (SDA/INT1)	Digital pin 20 (SDA)
45	PD2 (RXDI/INT2)	Digital pin 19 (RX1)
46	PD3 (TXD1/INT3)	Digital pin 18 (TX1)
47	PD4 (ICP1)	
48	PD5 (XCK1)	
49	PD6 (T1)	
50	PD7 (T0)	Digital pin 38
51	PG0 (WR)	Digital pin 41
52	PG1 (RD)	Digital pin 40
53	PC0 (A8)	Digital pin 37
54	PC1 (A9)	Digital pin 36
55	PC2 (A10)	Digital pin 35
56	PC3 (A11)	Digital pin 34
57	PC4 (A12)	Digital pin 33
58	PC5 (A13)	Digital pin 32
59	PC6 (A14)	Digital pin 31
60	PC7 (A15)	Digital pin 30
61	VCC	VCC
62	GND	GND

Pin Number	Pin Name	Mapped Pin Name
63	PJ0 (RXD3/PCINT9)	Digital pin 15 (RX3)
64	PJ1 (TXD3/PCINT10)	Digital pin 14 (TX3)
65	PJ2 (XCK3/PCINT11)	
66	PJ3 (PCINT12)	
67	PJ4 (PCINT13)	
68	PJ5 (PCINT14)	
69	PJ6 (PCINT 15)	
70	PG2 (ALE)	Digital pin 39
71	PA7 (AD7)	Digital pin 29
72	PA6 (AD6)	Digital pin 28
73	PA5 (AD5)	Digital pin 27
74	PA4 (AD4)	Digital pin 26
75	PA3 (AD3)	Digital pin 25
76	PA2 (AD2)	Digital pin 24
77	PA1 (AD1)	Digital pin 23
78	PA0 (AD0)	Digital pin 22
79	PJ7	
80	VCC	VCC
81	GND	GND
82	PK7 (ADC15/PCINT23)	Analog pin 15
83	PK6 (ADC14/PCINT22)	Analog pin 14
84	PK5 (ADC13/PCINT21)	Analog pin 13
85	PK4 (ADC12/PCINT20)	Analog pin 12
86	PK3 (ADC11/PCINT19)	Analog pin 11
87	PK2 (ADC10/PCINT18)	Analog pin 10
88	PK1 (ADC9/PCINT17)	Analog pin 9
89	PK0 (ADC8/PCINT16)	Analog pin 8
90	PF7 (ADC7)	Analog pin 7
91	PF6 (ADC6)	Analog pin 6
92	PF5 (ADC5/TMS)	Analog pin 5
93	PF4 (ADC4/TMK)	Analog pin 4

Pin Number	Pin Name	Mapped Pin Name
94	PF3 (ADC3)	Analog pin 3
95	PF2 (ADC2)	Analog pin 2
96	PF1 (ADC1)	Analog pin 1
97	PF0 (ADC0)	Analog pin 0
98	AREF	Analog Reference
99	GND	GND
100	AVCC	VCC

3.Modulele microcontrolerului implicate în realizarea proiectului

Pulse Width Modulation sau PWM este o tehnică de a obține rezultate analogice din inputuri digitale.

Controlul digital este folosit pentru a obține un semnal rectangular,un semnal modificat între pornit și oprit.

Acest șablon de pornit-oprit poate simula voltaje până la 5 Volți pentru pornit și 0 Volți pentru oprit, schimbând porțiunea de timp în care semnalul este pornit și cea în care semnalul este oprit.

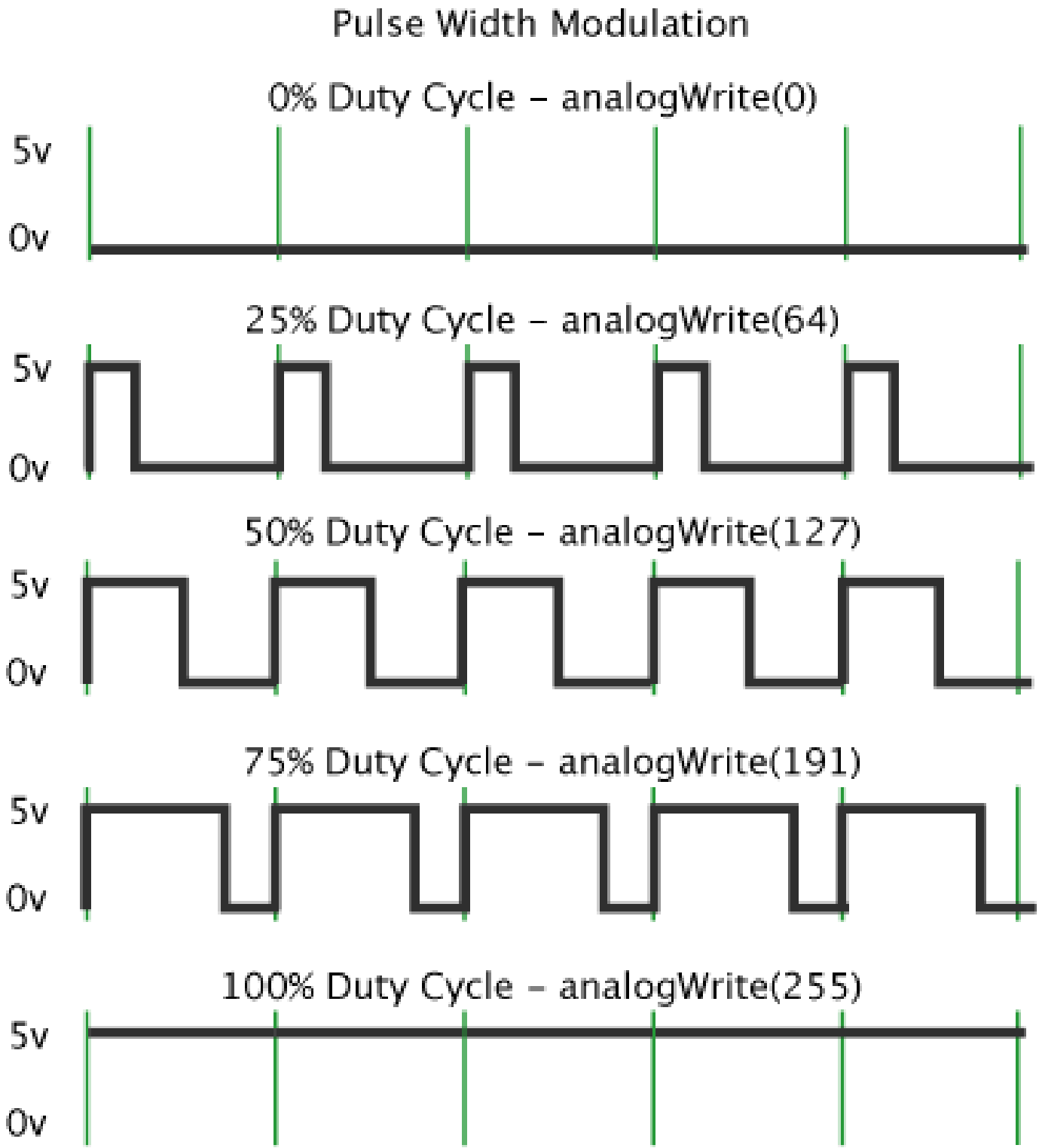
Durata timpului este numită **lățimea pulsului**(pulse width).

Pentru a obține valori analogice variabile se schimbă sau modulează această lățime a pulsului.

Dacă se repetă acest șablon de pornit-oprit destul de rapid cu un LED, rezultatul va fi unul asemănător situației în care voltajul ar fi stabil între 0 și 5 Volți, controlând luminozitatea LED-ului.

În graficul de dedesubt, liniile verzi reprezintă o perioada de timp regulate.
Durata acestei perioade este inversul frecvenței PWM.

În alte cuvinte , cu frecvența PWM a plăcuței Arduino la aproximativ 500Hz, liniile verzi vor măsură 2 milisecunde fiecare. Un apel la funcția analogWrite() pe o scară de la 0 la 255 , că de exemplu analogWrite(255) va solicită 100% din load(duty cycle) sau analogWrite(127) va solicită 50% duty cycle.


Pulse Width Modulation
0% Duty Cycle – analogWrite(0)
5v
0v
25% Duty Cycle – analogWrite(64)
5v
0v
50% Duty Cycle – analogWrite(127)
5v
0v
75% Duty Cycle – analogWrite(191)
5v
0v
100% Duty Cycle – analogWrite(255)
5v
0v

4.Sistemului de afişaj: Matrice de LED-uri

Sistemul de afișaj ales este matricea de LED-uri.

Matricea este de 8x15, avem 8 coloane si 15 randuri,adica 15 benzi cu cate 8 LED-uri.

Bandă de LED-uri este construită cu LED-uri WS2812B legate în serie.

Aceste LED-uri au un circuit integrat construit chiar în LED.

Acesta permite o comunicare printr-o interfață printr-un singur fir.

Această înseamnă că putem controla o multitudine de LED-uri cu doar un pin digital de pe Arduino.

LED-urile au următoarele specificații:

- Tensiune de alimentare: 5V
- Consum individual: 0.3W
- Lățime bandă: 10mm
- LED: 5x5mm

Fiecare circuit este compus din patru bucăți de siliciu: trei leduri de culori diferite (RGB) și un circuit de control și limitare a curentului.

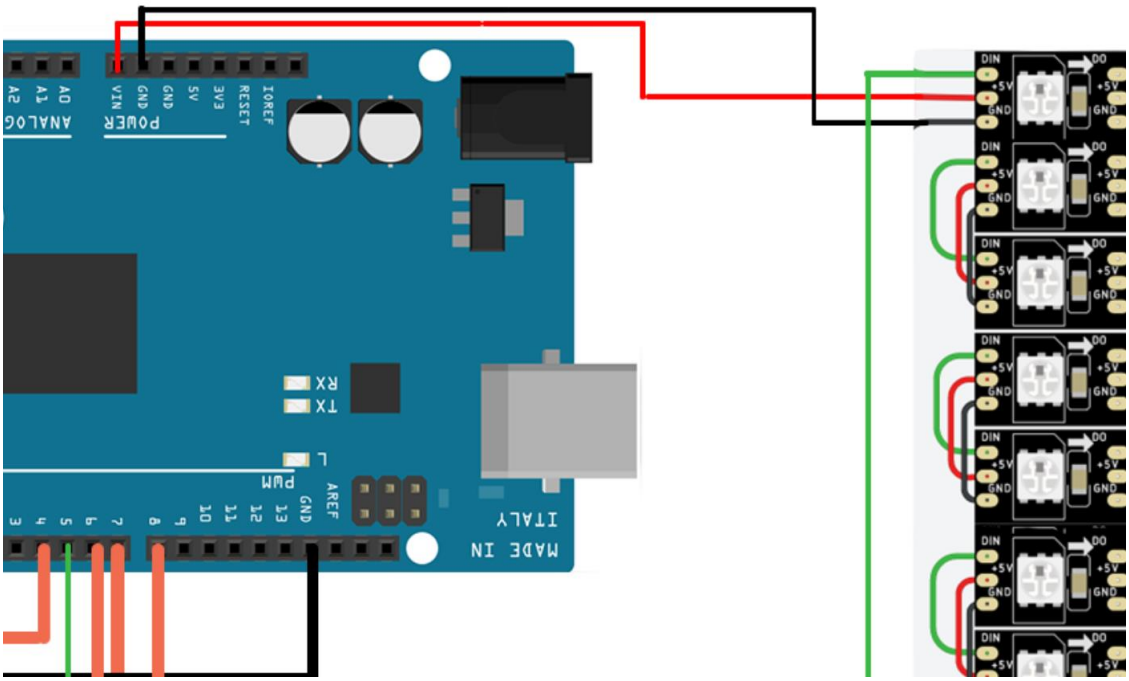
Se poate programa atât intensitatea luminii emanate de Led-uri cât și culoarea acesteia.

Fiecare led are un condensator legat în paralel, cu rolul de a proteja ledul.

4.1.Conectarea Led-urilor la Arduino:

Led-urile sunt conectate la placa Arduino astfel:

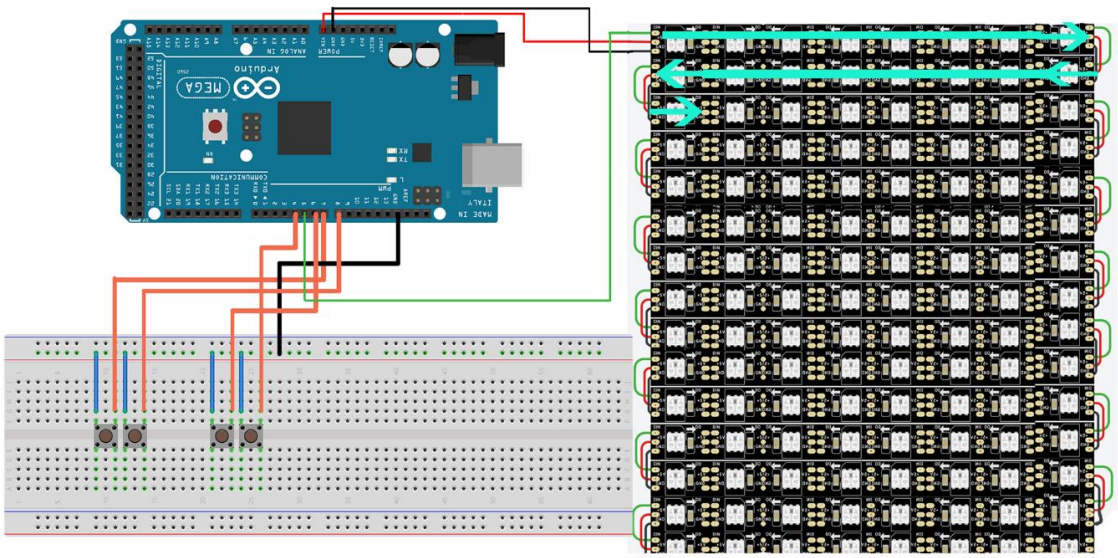
- Alimentarea la Vin (5V)
- GND la GND
- Din la Pin-ul ~5(PWM)



Fiecare banda cu 8 Led-uri este legata de banda imediat urmatoare cu fire de legatura:

- DIN/D0 la DIN/D0
- GND la GND
- +5V la +5V

➢ Sensul în care se parcurg benzile este următorul: (Practic se parcurg în serpentină)



Asadar Led-urile vor fi in ordine:

0 1 2 3 4 5 6 7

15 14 13 12 11 10 9 8

16 17 18 19 20 21 22 23 etc.

5.Alte componente folosite: Butoane

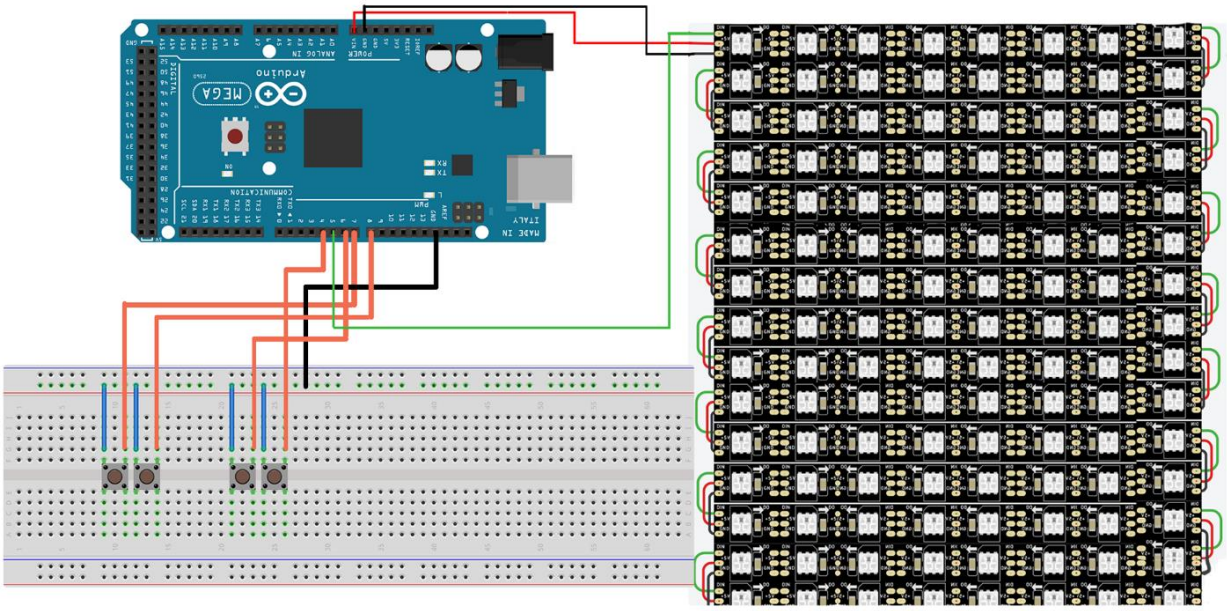
Un comutator cu buton apăsat este un mecanism mic, sigilat, care completează un circuit electric atunci când este apăsat.

Când este activ, un mic arc metalic din interior face contact cu două fire, permițând trecerea curentului electric. Când este inactiv, arcul se retrage, contactul este întrerupt și curentul nu va mai trece.

Acesta se conectează la un pin digital și la GND.

Butonul conectat la GND va fi active pe LOW.

6.Conectarea Hardware



Led-urile sunt conectate la placă Arduino astfel:

- Alimentarea la Vin (5V)
- GND la GND
- Din la Pin-ul ~5(PWM)

Benzile de led-uri sunt conectate între ele prin fire de legătură:

- GND la GND
- V5 la V5
- Din la Din

Butoanele sunt conectate astfel:

Avem patru butoane unul pentru fiecare direcție de mișcare:

- SUS - conectat la pin-ul 8 PWM
- JOS – conectat la pin-ul 7 PWM
- STÂNGA – conectat la pin-ul 6 PWM
- DREAPTA – conectat la pin-ul 4 PWM

Și câte un piciorus la GND.

Placă este conectată la laptop prin intermediul portului USB.

7.Programul aferent jocului

Am implementat un joc de labirint care funcționează după cum urmează :

La alimentarea plăcuței , pe aceasta va apărea un joc de lumini realizat cu ajutorul librăriei FASTLED.

Dacă se dorește începerea jocului se apasă butonul albastru.

În acest moment se poate face selecția hărții labirintului cu ajutorul aceluiași buton. Avem la dispoziție 3 hărți de joc.
Dacă s-a decis care harta se dorește, se apasă butonul negru.

În acest moment player-ul este poziționat la începutul labirintului.

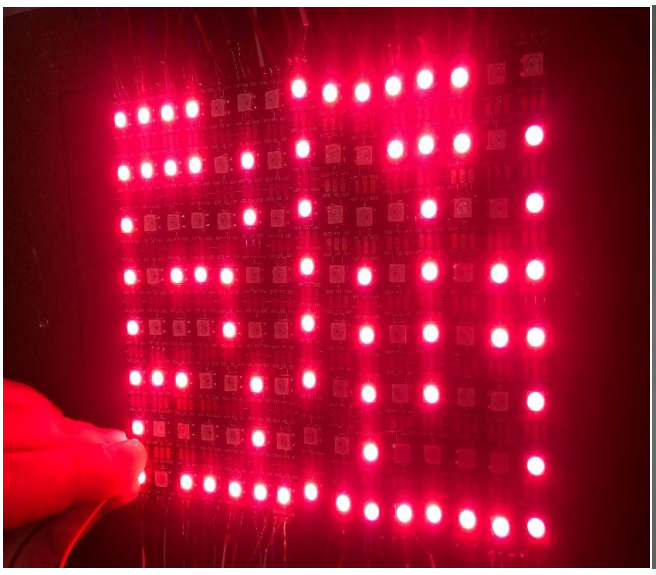
În același timp este generat și un inamic la o poziție din labirint și își începe mișcarea.

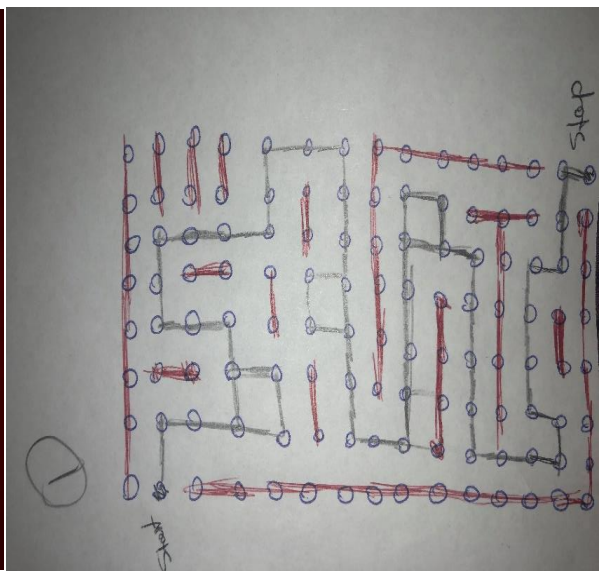
Se pot folosi cele 4 butoane de up/down/left/right pentru a se mișca player-ul prin labirint.

În cazul în care inamicul atinge player-ul, player-ul va fi repoziționat la începutul labirintului.

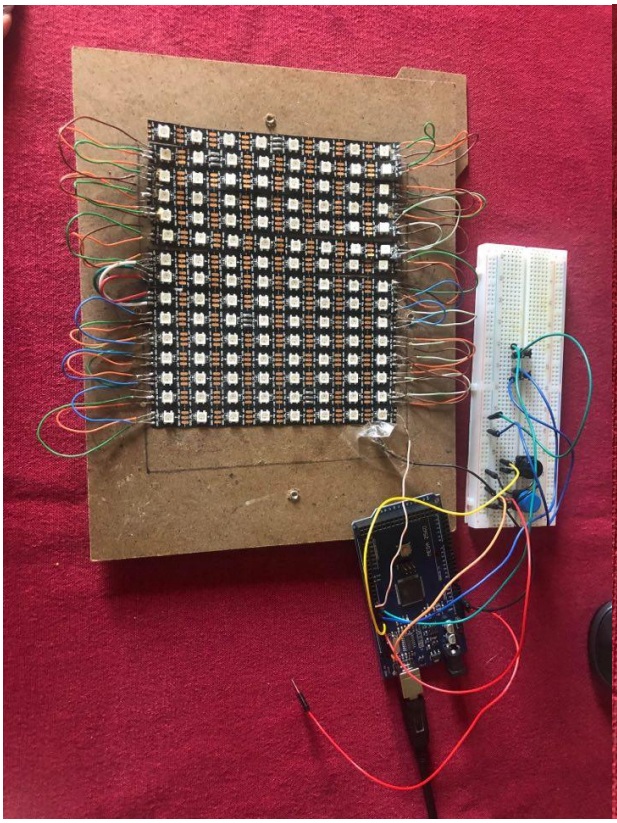
În cazul ajungerii la sfârșitul labirintului, va avea loc o animație de dimming, iar apoi după câteva secunde va reîncepe animația de început astfel putând alege din nou mapa pe care se dorește juca.

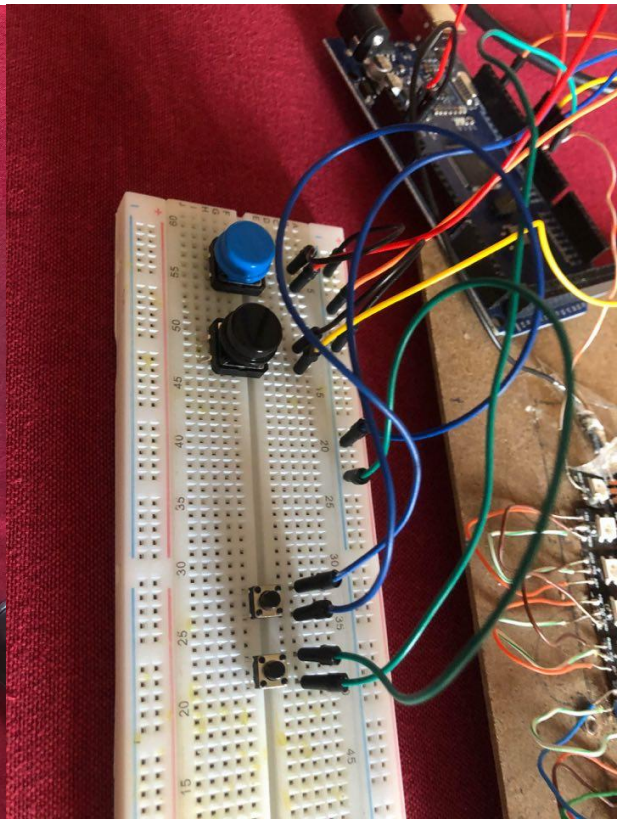
Exemplu de mape:

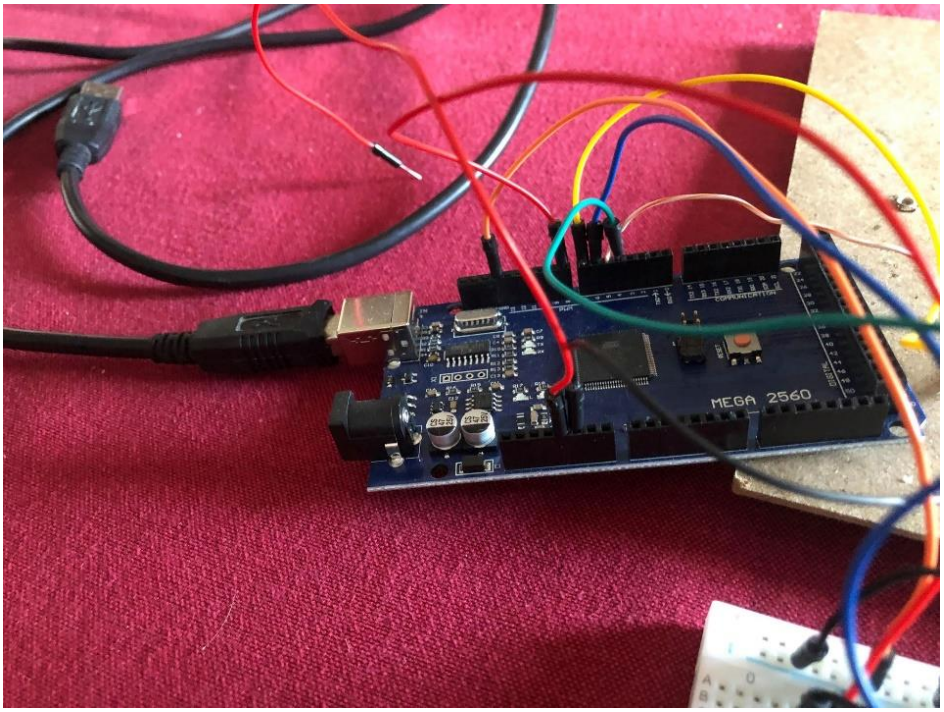




❖ **Conectarea firelor și a componentelor:**

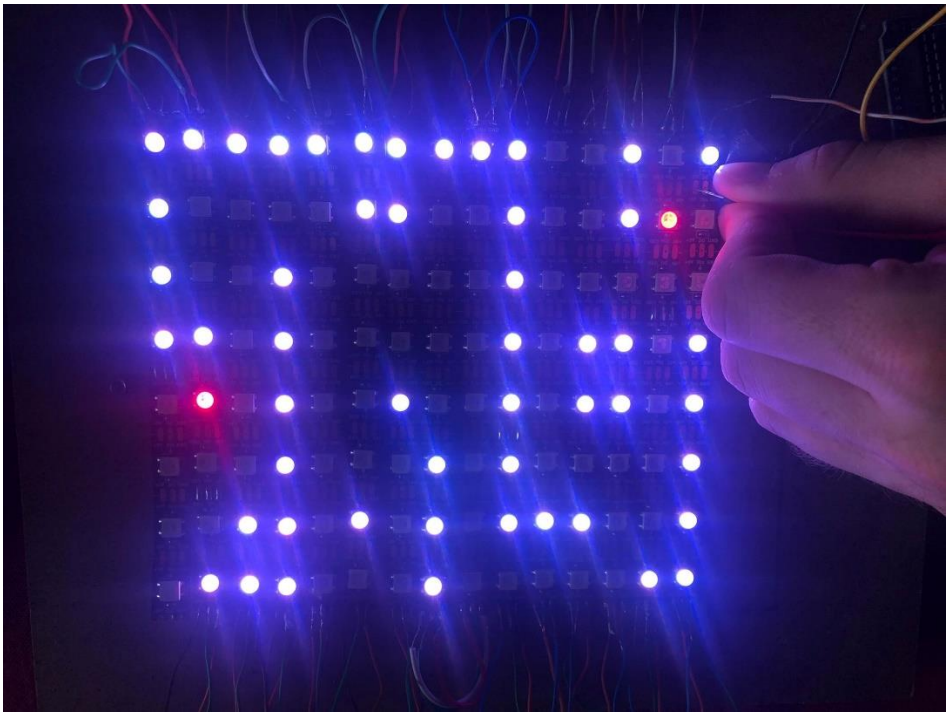




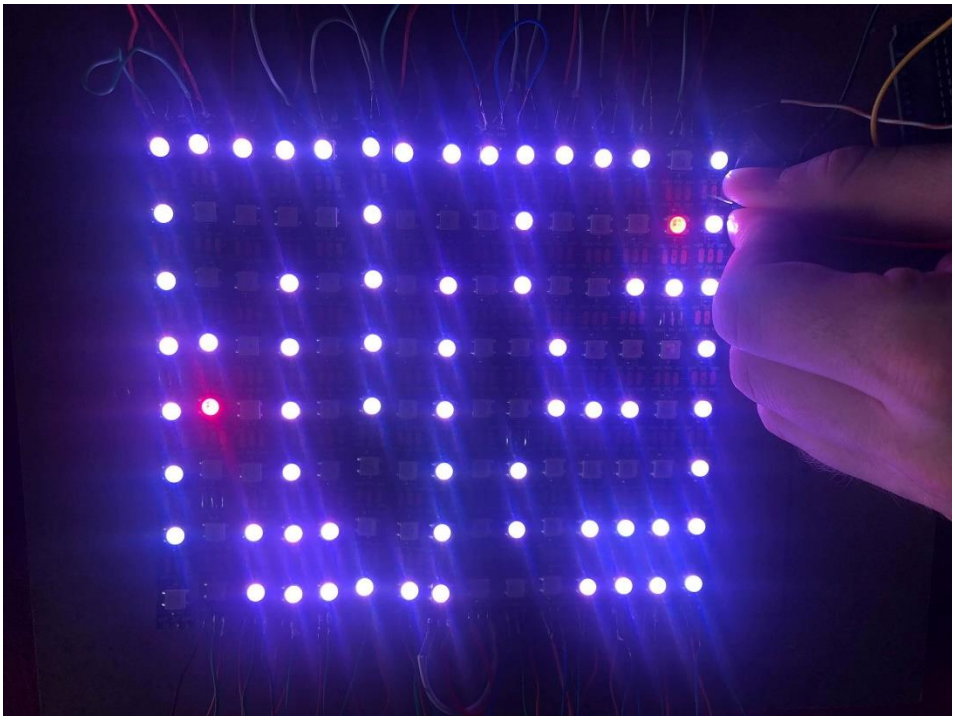


❖ **Cele 3 hărți implementate transpuse pe LED-uri:**

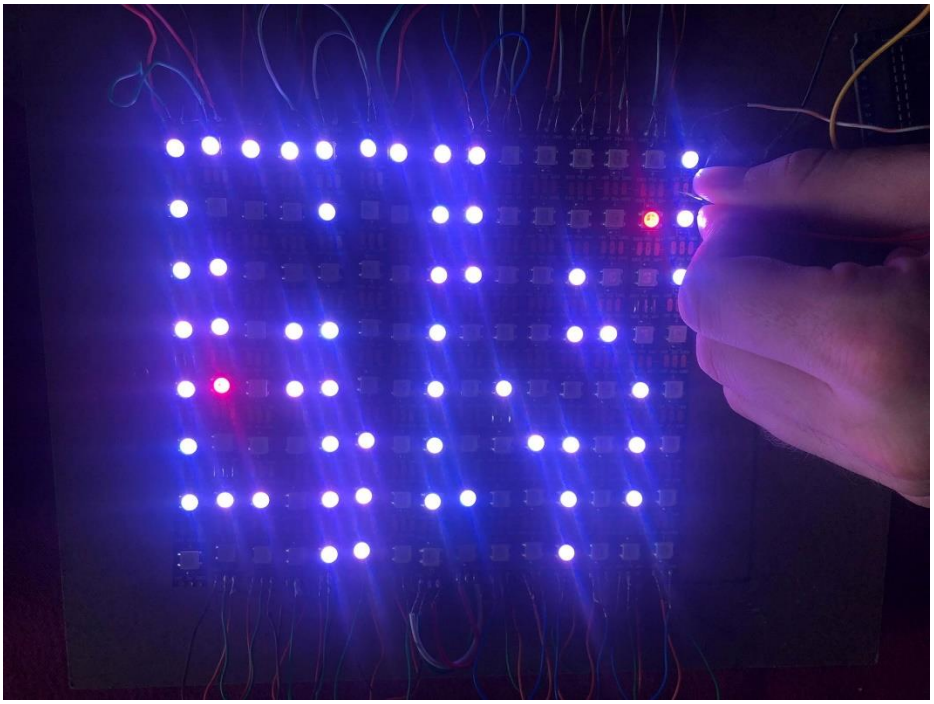
Harta 1:



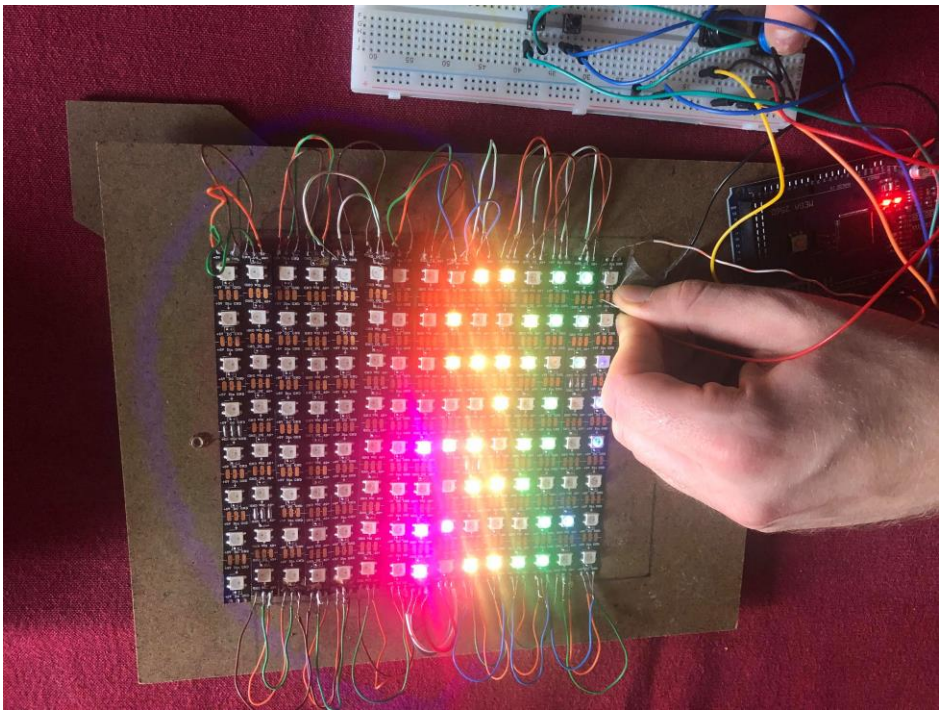
Harta 2:



Harta 3:



Tranziția de început:



❖ Codul:

```
#include <FastLED.h>

/*

  DEFINIT CHIPSET PENTRU A-L FOLOSI CU BIBLIOTECA FASTLED

  PIN 5 PWM PENTRU CONECTARE DATA IN

  DEFINITE MAX SI MIN BRIGHTNESS PENTRU FUCNTII DIN FASTLED

  CARE PORNESC LUMINOZITATEA SI AFISEAZA LEDURILE


  BRIGHTNESS PIN SE FOLOSESTE DOAR DACA VREI SA CONTROLEZI LUMINOZITATEA

  DINTR-UN POTENTIOMETRU


  PENTRU JOC ESTE LASAT STANDARD SI NU SE MODIFICA PRIN POTENTIOMETRU

*/

#define FAST_LED_CHIPSET WS2812B

#define FAST_LED_DATA_PIN 5

#define MAX_BRIGHTNESS 240  // 255 ESTE MAXIM DAR AM LIMITAT LA 240 LUMINOZITATEA

#define MIN_BRIGHTNESS 37

const int brightnessPIN = A0;


#define BLUEBTN 8  // BUTON INAINTE, PINUL 8

#define REDBTN 7   // BUTON INAPOI, PINUL 7
```

```
#define LFTBTN 6   // BUTON STANGA, PINUL 9
#define RGTBTN 4  // BUTON DREAPTA< PINUL 10



#define LINII 15
#define COLOANE 8
#define MAX_MAPS 3  // numar maxim de mape
/*
  Define for colors in hexa
*/
#define  PINK  0x1DEC12
#define  RED    0xFF0000
#define  BLUE   0x0000FF
#define  YELLOW 0xFFFF00
#define  CHOCOLATE  0xD2691E
#define  PURPLE 0x9966CC
#define  WHITE  0XFFFFFF
#define  AQUA   0x00FFFF
#define  HOTPINK 0xFF1493
#define  DARKORANGE 0xFF8C00
#define  BLACK 0x000000

//structura pentru mapa, care transpune matricile pe matricea de LED-uri
```

```
typedef struct MAP {
  uint8_t pixels[LINII][COLOANE];
  unsigned int long color[LINII][COLOANE];
} MAP;
MAP field;
MAP activeMap;



//structura pentru a definii labirinturile
typedef struct randomMap {
  byte pixels[LINII][COLOANE];
} RandomMap;

//structura pentru jucator
typedef struct Player {
  //pozitii player in harta
  byte xpos;//pe a cata linie
  byte ypos;//a cata coloana
  unsigned int color;//culoare led jucator
} Player;
Player myPlayer;
Player enemyPlayer;
```

```
//un jucator nou este pozitionat la o anumita pozitie in labirint, am ales pozitia prestabilita 1,1
void newPlayer(byte x, byte y) {
  myPlayer.xpos = x;
  myPlayer.ypos = y;
  myPlayer.color = DARKORANGE;
}

//define-uri pentru butoanele folosite
#define BTNUP 1
#define BTNDWN 2
#define BTNLFT 3
#define BTNRGT 4

byte currentControl = 0; // pentru a determina ce buton e apasat, initializat pe niciun buton apasat

//citim starile butoanelor, determinam ce buton s-a apasat
//butoane active pe low
void readInput()
{
  byte upST = digitalRead(BLUEBTN);
  byte downST = digitalRead(REDBTN);
  byte leftST = digitalRead(LFTBTN);
  byte rightST = digitalRead(RGTBTN);
```

```
  if (upST == LOW) {
    currentControl = BTNUP;
    delay(300);
  }
  if (downST == LOW) {
    currentControl = BTNDWN;
    delay(300);
  }
  if (leftST == LOW) {
    currentControl = BTNLFT;
    delay(300);
  }
  if (rightST == LOW) {
    currentControl = BTNRGT;
    delay(300);
  }}
// Functie folosita pentru testarea pozitiilor curente ale player-ului in labirint | Am folosit-o doar pentru testare.
void checkPos() {
  Serial.print(myPlayer.xpos);
  Serial.print(myPlayer.ypos);
}
```

```
//in functie de ce buton se apasa, se apeleaza functia corespunzatoare care determina miscarea player-ului

void controlPlayer()

{

  switch (currentControl)

  {

    case BTNUP: moveUp(); checkPos(); break;

    case BTNDWN: moveDown(); checkPos(); break;

    case BTNLFT: moveLeft(); checkPos(); break;

    case BTNRGT: moveRight(); checkPos(); break;

  }

}


//Urmeaza functiile de miscare a player-ului si testarea de coliziuni

void moveUp() {

  // AICI AM IMPLEMENTAT COLIZIUNILE

  // PROBLEMA E CA NU LE PUTEM INCARCA INCA IN PROIECTUL FINAL

  // CA A DEPASIT SPATIUL TOTAL DE MEMORIE DINAMICA

  // URMEAZA SA INCERCAM SA MUTAM PE O MEMORIE EXTERNA SAU EEPROM hartile

  myPlayer.ypos++;

  if (checkWallCollision(myPlayer.xpos, myPlayer.ypos)) {

    myPlayer.ypos--;

  }
```

```
  if (checkOutOfMap(myPlayer.xpos, myPlayer.ypos)) {
    myPlayer.ypos--;
  }
}

void moveDown() {

  myPlayer.ypos--;
  if (checkWallCollision(myPlayer.xpos, myPlayer.ypos)) {
    myPlayer.ypos++;
  }

  if (checkOutOfMap(myPlayer.xpos, myPlayer.ypos)) {
    myPlayer.ypos++;
  }
}

void moveLeft() {
  myPlayer.xpos--;
  if (checkWallCollision(myPlayer.xpos, myPlayer.ypos)) {
    myPlayer.xpos++;
  }
```

```
  if (checkOutOfMap(myPlayer.xpos, myPlayer.ypos)) {
    myPlayer.xpos++;
  }
}



void moveRight() {
  myPlayer.xpos++;
  if (checkWallCollision(myPlayer.xpos, myPlayer.ypos)) {
    myPlayer.xpos--;
  }
  if (checkOutOfMap(myPlayer.xpos, myPlayer.ypos)) {
    myPlayer.xpos--;
  }
}

bool checkWallCollision(byte x, byte y) {
  //daca coordonatele curente reprezinta zid, mutarea nu e permisa
  if (activeMap.pixels[x][y] == 1) {
    return true;
  }
  else {
    return false;
```

```
  }
}
bool checkPlayerCollision(byte x, byte y) {
   //daca coordonatele curente sunt egale cu cele ale enemy-ului atunci mutarea nu este permisa
  if (x == myPlayer.xpos && y == myPlayer.ypos) {
    return true;
  }
  else {
    return false;
  }
}

bool checkOutOfMap(byte x, byte y) {
  //daca coordonatele curente sunt in afara hartii de joc, mutarea nu e permisa
  if (x < 0 || x >= LINII || y < 0 || y >= COLOANE) {
    return true;
  }
  else {
    return false;
  }
}

//definirea a trei mape de labirint
```

```
/*
   '1': Wall
   '0': Wove Zone
*/
bool visited[LINII][COLOANE];


RandomMap mapLib[3] = {
  {
    { {1, 1, 1, 1, 1, 1, 1, 1},
      {0, 0, 1, 0, 0, 0, 1, 1},
      {1, 0, 1, 0, 1, 0, 1, 1},
      {1, 0, 0, 0, 1, 0, 1, 1},
      {1, 0, 0, 1, 1, 0, 0, 0},
      {1, 1, 1, 0, 0, 1, 1, 0},
      {1, 0, 0, 0, 0, 0, 0, 0},
      {1, 0, 1, 1, 1, 1, 1, 1},
      {1, 0, 0, 0, 0, 0, 0, 1},
      {1, 1, 1, 1, 1, 0, 0, 1},
      {1, 0, 0, 0, 0, 0, 1, 1},
      {1, 0, 1, 1, 1, 1, 1, 1},
      {1, 0, 0, 0, 0, 0, 1, 1},
      {1, 0, 0, 1, 1, 0, 0, 0},
      {1, 1, 1, 1, 1, 1, 1, 0}
```

```
  }
},
{
  { {1, 1, 1, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 1, 1, 1, 0},
    {0, 0, 0, 1, 0, 0, 0, 0},
    {0, 0, 1, 1, 0, 1, 1, 1},
    {0, 0, 0, 0, 0, 1, 0, 0},
    {0, 0, 0, 0, 1, 0, 0, 0},
    {1, 1, 1, 0, 0, 0, 1, 0},
    {1, 1, 1, 1, 1, 1, 1, 0},
    {1, 0, 0, 0, 0, 0, 0, 0},
    {1, 0, 0, 0, 0, 1, 1, 1},
    {1, 1, 0, 1, 1, 1, 1, 1},
    {1, 0, 0, 1, 1, 0, 0, 0},
    {1, 0, 0, 0, 0, 0, 1, 0},
    {1, 0, 1, 1, 1, 0, 1, 0},
    {1, 1, 1, 1, 1, 1, 1, 0}
  }
},
{
  { {1, 0, 0, 1, 1, 1, 1, 1},
    {0, 0, 0, 0, 0, 0, 0, 1},
```

```
      {1, 1, 0, 1, 1, 0, 0, 0},
      {0, 0, 0, 1, 1, 0, 1, 0},
      {0, 0, 0, 0, 0, 0, 1, 0},
      {1, 1, 1, 1, 1, 1, 1, 0},
      {1, 0, 0, 0, 0, 0, 0, 0},
      {1, 0, 0, 0, 0, 1, 1, 1},
      {1, 1, 0, 0, 1, 0, 0, 0},
      {1, 1, 0, 0, 0, 0, 1, 0},
      {1, 0, 0, 0, 0, 0, 0, 0},
      {1, 0, 1, 1, 1, 1, 1, 1},
      {1, 0, 0, 0, 0, 0, 1, 1},
      {1, 0, 0, 1, 0, 0, 0, 1},
      {1, 1, 1, 1, 0, 0, 0, 0}
    }
  }
};

// variabila folosita pentru determinarea mapei selectate curent
byte selectedMap = 0;

//punem mapa pe led-uri
void newActiveMap()
{
```

```
  uint8_t x, y;
  for (y = 0; y < COLOANE; y++)
  {
    for (x = 0; x < LINII; x++)
    {
      activeMap.pixels[x][y] = (mapLib[selectedMap]).pixels[x][y];
    }
  }
}

//stinge toate led-urile
void clearField()
{
  uint8_t x, y;
  for (y = 0; y < COLOANE; y++)
  {
    for (x = 0; x < LINII; x++)
    {
      field.pixels[y][x] = 0;
      field.color[y][x] = BLACK;
    }
  }
}
```

```
#define PIXELS LINII*COLOANE //numarul total de led-uri

CRGB leds[PIXELS];  //led-urile

// initializeaza ledurile cu ajutorul FASTLED library
void initPixels()
{
  FastLED.addLeds<FAST_LED_CHIPSET, FAST_LED_DATA_PIN, GRB>(leds, PIXELS);
}


// Initializeaza jocul - curata mapa, creeaza una noua si creeaza player.
void initGame() {
  clearField();
  newActiveMap();
  newPlayer(1,1);
  initEnemy(13, 4);
}

// Aprinde ledurile cu o valoarea intre minimul si maximul pentru luminozitate
void showPixels()
{
  int value = map(analogRead(brightnessPIN), 0, 1023, 0, MAX_BRIGHTNESS);
```

```
  FastLED.setBrightness(constrain(value, MIN_BRIGHTNESS, MAX_BRIGHTNESS));
  FastLED.show();
}



int long getPixel(int pos)
{
  // impacheteaza culorile RGB intr-un long , si ia culoarea de pe ledul curent
  return (leds[pos].red << 16) + (leds[pos].green << 8) + leds[pos].b;
}

//stinge treptat fiecare canal de RGB cu o anumita valoarea dimval
void dim(float dimval)
{
  for (uint8_t i = 0; i < (LINII * COLOANE); i++)
  {
    int currentColor = getPixel(i);
    // aiic se despacheteaza
    int long red = ((currentColor & 0xFF0000) >> 16);
    int green = ((currentColor & 0x00FF00) >> 8);
    int blue = (currentColor & 0x0000FF);

    // pentru fiecare canal de culoare, acea culoare se inmulteste cu coeficientul dat ca parametru functiei, un float intre 0 si 1
```

```
    // rezulta prin inmultire repetata efectul de dimming intrucat se micsoreaza intensitatea de pe acel canal de culoare pana ajunge la R=0 G=0 B=0
    red = red * dimval;
    green = green * dimval;
    blue = blue * dimval;

    currentColor = (red << 16) + (green << 8) + blue;
    setPixel(i, currentColor);
  }
}

//fade de final labirint, se aplica pe fiecare led, functia dim prezentata mai sus
void endFade() {
  byte i;
  for (i = 0; i < LINII * COLOANE; i++) {
    dim(0.3);
    showPixels();
    delay(100);
  }
}

//functia care verifica daca player-ul a ajuns la coordonatele de finish
bool endGame() {
  byte x, y;
```

```
  if (myPlayer.xpos == (LINII - 1)  && myPlayer.ypos == (COLOANE - 1)) {
    endFade();
    return true;
  }
  return false;
}

//functia de printare a mapei si a player-ului
void Field()
{
  byte x, y;
  for (y = 0; y < COLOANE; y++)
  {
    for (x = 0; x < LINII; x++)
    {

      if (activeMap.pixels[x][y] == 0) {
        setFieldPixel(x, y, BLACK);//move zone=led stins

      }
      else if (activeMap.pixels[x][y] == 1) {
        setFieldPixel(x, y, PURPLE);//perete=led aprins
      }
```

```
    }

  }

  setFieldPixel(myPlayer.xpos, myPlayer.ypos, myPlayer.color);//printeaza player-ul la pozitia curenta in harta

  setFieldPixel(enemyPlayer.xpos, enemyPlayer.ypos, enemyPlayer.color);//printeaza enemy player-ul la pozitia curenta in harta

  showPixels();

}



#define NUM_LEDS (COLOANE * LINII)

#define LAST_VISIBLE_LED 119



//functia care ne returneaza numarul led-ului (exemplu:XY(5,4) led-ul 37 de pe banda de led-uri)

byte XY (byte x, byte y) {

  // Daca se depaseste pozitia maxima de leduri posibile se returneaza ultimul led

  if ( (x >= COLOANE) || (y >= LINII) ) {

    return (LAST_VISIBLE_LED + 1);

  }

  // tabela cu numerele ledurilor de la 0-119

  const uint8_t XYTable[] = {

    0,   1,   2,   3,   4,   5,   6,   7,

    15,  14,  13,  12,  11,  10,   9,   8,

    16,  17,  18,  19,  20,  21,  22,  23,

    31,  30,  29,  28,  27,  26,  25,  24,
```

```
   32,  33,  34,  35,  36,  37,  38,  39,
   47,  46,  45,  44,  43,  42,  41,  40,
   48,  49,  50,  51,  52,  53,  54,  55,
   63,  62,  61,  60,  59,  58,  57,  56,
   64,  65,  66,  67,  68,  69,  70,  71,
   79,  78,  77,  76,  75,  74,  73,  72,
   80,  81,  82,  83,  84,  85,  86,  87,
   95,  94,  93,  92,  91,  90,  89,  88,
   96,  97,  98,  99, 100, 101, 102, 103,
   111, 110, 109, 108, 107, 106, 105, 104,
   112, 113, 114, 115, 116, 117, 118, 119
  };


  byte i = (y * COLOANE) + x;
  byte j = XYTable[i];
  return j;
}


//seteaza un led de la o anumita pozitie, cu o anumita culoare cu ajutorul tabelului de mai sus pentru pozitie si a functiei setPixel de mai jos

void setFieldPixel(int x, int y, int long color)
{
  byte  pos = XY(y, x);
  setPixel(pos, color);
```

```
}


void setPixel(int pos, int long color)
{
  leds[pos] = CRGB(color);
}


//functia care citeste butoanele si selecteaza mapa de joc
void mapSelect() {
  clearField();
  bool mapNotSelected = true;
  while (mapNotSelected) {//cat timp mapa nu e selectata se poate alege in continuare mapa pe care se vrea a se juca
    delay(200);
    Field();
    byte blueBtnState = digitalRead(BLUEBTN);
    if (blueBtnState == LOW) {
      selectedMap++;
      if (selectedMap > (MAX_MAPS - 1)) {//mapele merg circular, daca se ajunge la ultima mapa se revine la prima
        selectedMap = 0;
      }
      newActiveMap();
      delay(300);
```

```
    }

    byte redBtnState = digitalRead(REDBTN);//la apasarea butonului rosu s-a selectat mapa de joc

    if (redBtnState == LOW) {

      mapNotSelected = false;

    }

  }

  if (selectedMap == 0) {

    initEnemy(6, 2);

  }

  else if (selectedMap == 1) {

    initEnemy(8, 2);

  }

  else {

    initEnemy(10, 2);

  }



}



bool mazeRunning = false;

// variabile folosite pentru determinarea mutarii curente a enemy-ului

byte UP = 0, RIGHT = 1, DOWN = 2, LEFT = 3;

// in last command se retine ultima miscare a enemy-ului pentru a evita un loop intre "move up" si "move down" pe o singura pozitie
```

```
byte lastCommand = 0;
void runMaze() {
  initGame();//initializare joc
  mazeRunning = true;
  mapSelect();
  /*
     cat timp jocul ruleaza se tot citeste inputul pentru control player
     si se reprinteaza mapa si player-ul la noile pozitii
  */
  while (mazeRunning) {
    currentControl = 0;
    readInput();
    controlPlayer();
    // O data la 600 de milisecunde se apeleaza functia walkEnemy care muta pozitia enemy-ului
    EVERY_N_MILLISECONDS(600) {
      walkEnemy();
    }
    Field();
    if (endGame()) {
      mazeRunning = false;
    }
  }
}
```

```
/*
  Se verifica ultima commanda apoi se trece pe branch-ul potrivit, daca cumva nu se poate face mutarea - fie din cauza coliziunii cu zidul sau iesirea din mapa
  se va face un salt la celalalt if pentru a testa o alta posibila mutare
*/
void walkEnemy() {
  if (lastCommand == UP) {
WALKUP: if (!canWalkUp()) {
      goto WALKDOWN;
    }
  }

  if (lastCommand == DOWN) {
WALKDOWN: if (!canWalkDown()) {
      goto WALKUP;
    }
  }
}

/*
  Se face mutarea dupa un delay convenabil apoi se testeaza :
  Daca enemy atinge player-ul, player-ul revine la pozitia initiala si returneaza false
```

Daca enemy atinge zid-ul acesta revine in pozitia de dinainte si returneaza false

Daca enemy atinge o pozitie dinafara mapei, revine la pozitia precedenta si returneaza false

In orice alt caz returneaza true si seteaza ultima comanda ca fiind cea executata

*/

```
bool canWalkDown() {
  delay(200);
  enemyPlayer.ypos--;
  if (checkPlayerCollision(enemyPlayer.xpos, enemyPlayer.ypos))
  {
    newPlayer(1,1);
  }
  if (checkWallCollision(enemyPlayer.xpos, enemyPlayer.ypos))
  {
    enemyPlayer.ypos++;
    return false;
  }
  if (checkOutOfMap(enemyPlayer.xpos, enemyPlayer.ypos))
  {
    enemyPlayer.ypos++;
    return false;
  }
```

```
  lastCommand = DOWN;
  return true;
}

// Asemenea functie canWalkUp() doar ca miscarea se face in directia opusa
bool canWalkUp() {
  delay(200);
  enemyPlayer.ypos++;
  if (checkPlayerCollision(enemyPlayer.xpos, enemyPlayer.ypos))
  {
    newPlayer(1,1);
  }
  if (checkWallCollision(enemyPlayer.xpos, enemyPlayer.ypos))
  {
    enemyPlayer.ypos--;
    return false;
  }
  if (checkOutOfMap(enemyPlayer.xpos, enemyPlayer.ypos))
  {
    enemyPlayer.ypos--;
    return false;
  }
  lastCommand = UP;
```

```
  return true;
}



// Initializeaza pozitia enemy-ului la xposition si yposition
// De asemenea seteaza si culoarea acestuia
void initEnemy(byte xposition, byte yposition) {
  enemyPlayer.xpos = xposition;
  enemyPlayer.ypos = yposition;
  enemyPlayer.color = PINK;
}
/*
   Animatie din fastled
   Se aprind ledurile unul dupa altul si lasa in urma
   un trace care se dim-uie
   gHue este saturatia care se modifica in Loop o data la 20 de milisecunde
*/
byte gHue = 0;
void RunningLeds()
{
  fadeToBlackBy( leds, 120, 20);
  int pos = beatsin16( 13, 0, 120 - 1 );
  leds[pos] += CHSV( gHue, 255, 192);
```

```
}

/*
   Setarea pinilor pentru butoane si initializarea benzii led
*/
void setup() {
  Serial.begin(9600);
  pinMode(BLUEBTN, INPUT_PULLUP);
  pinMode(REDBTN, INPUT_PULLUP);
  pinMode(LFTBTN, INPUT_PULLUP);
  pinMode(RGTBTN, INPUT_PULLUP);
  initPixels();
  showPixels();
}

void loop() {
  // aici se aprinde animatia led pana cand se apasa butonul albastru
  FastLED.show();
  FastLED.delay(1000 / 60);
  EVERY_N_MILLISECONDS( 20 ) {
    gHue = gHue + 5;
  };
  RunningLeds();
```

```
  byte blueBtnState = digitalRead(BLUEBTN);
  if (blueBtnState == LOW) {
    clearField();
    runMaze();
  }
}
```

8.Bibliografie:

http://users.utcluj.ro/~rdanescu/pmp-lab01.pdf

https://www.arduino.cc/en/Hacking/PinMapping2560

https://www.researchgate.net/figure/Block-Diagram-of-Microcontroller-Arduino-Mega-2560_fig2_320034070

https://docs.rs-online.com/9ab3/0900766b80e8ba22.pdf

https://www.robotshop.com/media/files/pdf/arduinomega2560datasheet.pdf

https://www.theengineeringprojects.com/2018/06/introduction-to-arduino-mega-2560.html

https://d2drzakx2pq6fl.cloudfront.net/production/products/resources/10/ATmega2560-Arduino-Pin-Mapping.pdf?1431001426