Battleship LED

Introducere

Battleship este un boardgame in care 2 jucatori plaseaza unitati sub forma de **nave de razboi** pe un grid unde fiecare celula a grilei este identificata printr-un set de coordonate.

Scopul jocului este de a distruge flota adversarului prin ghicirea pozitiilor navelor acestuia pe grid.

Battleship LED are obiectivul de a fi un hybrid intre experienta fizica si cea digitala, prin adaugarea elementelor digitale unui joc traditional de strategie.

Descriere generală

Jocul are 2 faze principale: **Battleship Placement** si **Target Selection**. Fiecare jucator are propria lor matrice LED si propriul lor panou de control, separate print-un perete de cele ale adversarului.

Ecranul LCD 1602 este plasat pe margine unde poate fi vazut de ambii jucatori. El este folosit pentru a oferi informatii cruciale, cum ar fi cate nave mai are un jucator, al carui jucator este tura etc .

Buzzerele sunt folosite pentru a reda sunete specifice pentru urmatoarele situatii: **start game**, **hit**, **miss**, **ship sunken**, **end of game**.

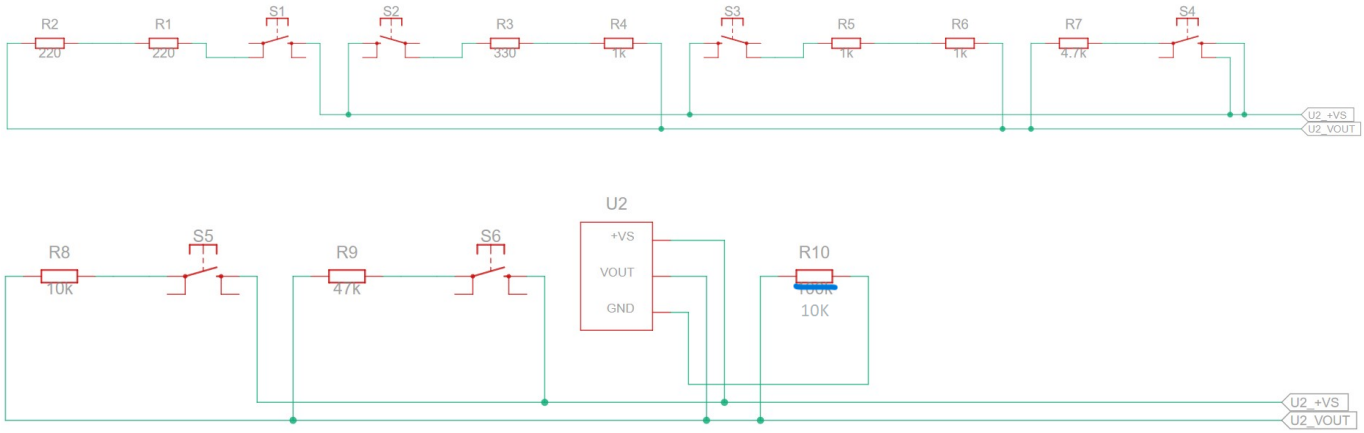
Panoul de control este alcatuit din 6 pushbuttons multiplexate prin intermediul unui sistem de divizoare de tensiune. Cele 4 butoane directionale deplaseaza un 'cursor' pe ecranul LED reprezentat de un flicker controlat printr-un timer intern. Butonul stanga sus are functia de **Rotate** in faza de **Battleship Placement** pentru a roti navele si **Change Screen** in faza de **Target Selection** pentru schimba intre boardu tau si cel inamic. **Confirm** pentru a confirma selecia de locatie pentru nave / tinta.

}

Hardware Design

Full Circuit

Control Panel



Distributie pinuri

- **D13(SCK), D11(MOSI), D10(SS)** sunt folositi pentru a transmite date catre matricele LED prin **SPI**. Modulele MAX7219 fac posibila interfatarea celor doua matrice LED folosind acelasi latch pin (**D10**)
- **A0** si **A1** citesc valori de la cele doua panouri de control care sunt procesate de convertorul **ADC** din Arduino
- **A4(SDA), A5(SCL)** sunt conectati la ecranul LCD printr-un adaptor **I2C** care faciliteaza protocolul respectiv

Lista piese

- Arduino Nano (ATmega328p și CH340)
- LCD1602 + I2C module
- 2xMAX7219 conectate la 2 matrice LED 8×8
- 2*6 pushbuttons panouri de control
- Pushbutton reset/new game
- 7*2 rezistente sisteme divizor de tensiune
- 2 Buzzer
- 9V battery
- Switch button
- Solder boards
- Fire conductoare

Software Design

- Am folosit PlatformIO cu VScode si fisiere structurate cu un header comun
- Platforma finala va fi PlatformIO cu VScode
- Implementarile la Timer,ADC,I2C si intreruperi sunt luate din laboratoare

Implementare

Afisare

Intern, sunt folosite 2 matrici 2×8 ungsigned char pentru a transmite stadiul actual al jocului catre ecranele LED:

- LEDS - tine cont de locatiile atacate de inamic
- Ships - tine cont de locatiile unde sunt navele

Ecranul unui jucator poate sa fie in 3 moduri de functionare:

1. Se uita la propriul board, unde locatiile navelor sunt vizibile
2. Se uita la board-ul inamic, unde doar partile lovite din nave sunt vizibile
3. Se uita la navele plasate in matricea Ships

View_Self	no hit	hit
no ship	0	1
ship	f1	1

View_Enemy	no hit	hit
no ship	0	1
ship	0	f1

View_Placement	no ship	ship
no ship	0	1
ship	1	f1

```
Unde:
  * 1 - LED ON
  * 0 - LED OFF
  * f1 - flicker 1 folosit pentru nave, prin timer 2 alterneaza intre 1 si 0 la 1000 ms
  * f2 - flicker 2 folosit pentru cursor, prin timer 1 alterneaza intre 1 si 0 la 50 ms

#define View_Enemy(x,y) ((x&~y)|(x&f1))
#define View_Self(x,y) ((y&f1)|x)
#define View_Placement(x,y) ((~x&y)|(x&~y)|(y&f1))
```

Control Inputs

Prin intermediul ADC conectat la A0 pentru Player1 si A1 pentru Player2 citim valori si verificam pentru urmatoarele valori:

```
#define BTN1 980 //Rotate / Change Screen
#define BTN2 903 //UP
#define BTN3 853 //Confirm
#define BTN4 696 //LEFT
#define BTN5 512 //DOWN
#define BTN6 179 //RIGHT

#define VALUE_IN_RANGE(read_value, threshold_value) \
  ((read_value - threshold_value) <= 15 || ((threshold_value - read_value) <= 15))
//Pentru perturbari fizice
```

In PLACE_PHASE un jucator poate sa-si vada doar propriul ecran si sa-si miste cursorul pentru a plasa nave. Odata ce a plasat 5 nave cursorul devine inactiv si nu se mai poate misca.

In BATTLE_PHASE cursorul jucatorului curent este plasat pe matricea LEDS[enemy] unde are control. Cat timp este tura lui, jucatorul nu poate sa schimbe ecranul, iar jucatorul inamic are cursorul deactivat si poate doar sa schimbe ecranul.

Buzzer

Functiile tone() si notone() din Arduino.h au prezentat doua probleme in acest proiec:

1. doar un singur pin analog poate sa creeze iesiri cu tone() la un moment dat (folosesc 2 buzzer-e pentru fiecare jucator, PD5 pt. Player1 si PD6 pt. Player2)
2. PWM output la 3 and 11 sunt afectate pe placa utilizata

Asa ca am scris o alternativa:

```
void playTone(uint8_t buzzerpin, int frequency, int duration)
{
  long period = 1000000L / frequency;
  long cycles = (long)duration * 1000L / period;
  for (long i = 0; i < cycles; i++)
  {
      SET(PORTD, buzzerpin);
      delayMicroseconds(period / 2);
      CLR(PORTD, buzzerpin);
      delayMicroseconds(period / 2);
  }
}
```

LCD Screen

Prin folosirea librariilor din lab-ul de I2C am recreat cateva functionalitatile din LiquidCrystal_I2C

```
void lcd_send_command(uint8_t command);
void lcd_send_data(uint8_t data);
void lcd_init(void);
void lcd_set_cursor(uint8_t row, uint8_t col);
void lcd_print(char *str);

void lcd_send(uint8_t value, uint8_t mode)
{
  uint8_t highNib = value & 0xF0;
  uint8_t lowNib = (value << 4) & 0xF0;
```

```
  // Send high nibble
  twi_start();
  twi_write(LCD_ADDRESS << 1);
  twi_write(mode | highNib | 0x0C); // En=1, Rs/Rw as per mode
  twi_write(mode | highNib | 0x08); // En=0, Rs/Rw as per mode
  twi_stop();
```

```
  // Send low nibble
```

```
  twi_start();
  twi_write(LCD_ADDRESS << 1);
  twi_write(mode | lowNib | 0x0C); // En=1, Rs/Rw as per mode
  twi_write(mode | lowNib | 0x08); // En=0, Rs/Rw as per mode
  twi_stop();
}
```

Game Logic

In PLACE_PHASE initial ambi jucatori incep jocul cu 0 nave. La pozitia cursorului este 'plasata' o nava de lungime {2,3,3,3,4} care se 'misca' cu acesta. Ea poate sa treaca peste alte nave, nar nu prin margini.

```
if (P_cursor[player].r <= row && P_cursor[player].r + length - 1 >= row)
    displayData[player] = View_Placement(Ships[player][row], (1 <<
P_cursor[player].c));
//Daca nava este verticala

if (row == P_cursor[player].r)
          {
              unsigned char temp = 0x00;
              for (uint8_t j = 0; j < length; j++)
                  temp |= (1 << (P_cursor[player].c + j));
              displayData[player] = View_Placement(Ships[player][row],
temp);
          }
//Daca nava este orizontala
```

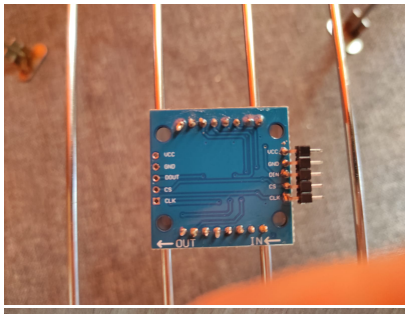
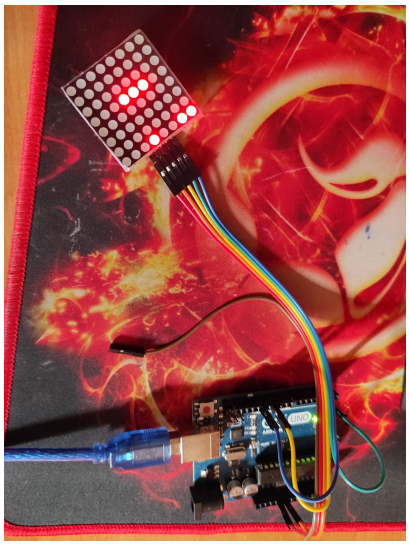
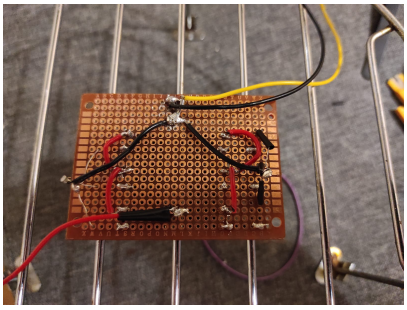
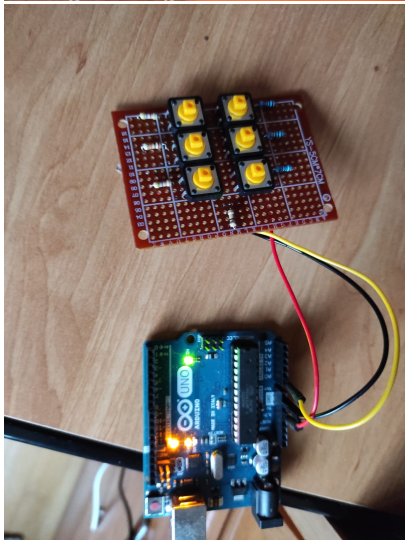
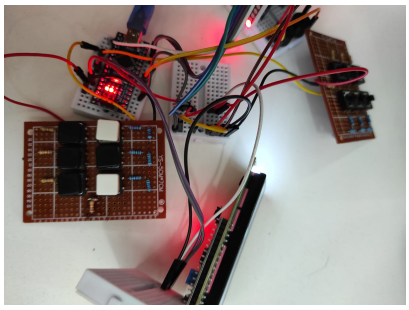
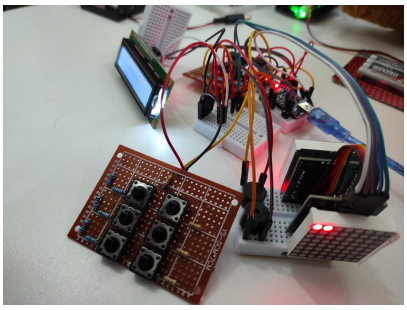
O nava poate fi introdusa in joc (adaugata la unsigned char Ships si la struct) daca toate segmentele sale nu sunt deja ocupate (idem pentru selectarea unui atac); Odata ce ambii jucatori au plasat cele 5 nave incepe BATTLE_PHASE. Cand se selecteaza o pozitie pentru atac, jocul prima data verifica daca a lovita inainte. Daca e un 'Hit' jucatorul poate sa continue sa atace pana cand este un 'Miss'.

Cu a 5-ea nava doborata, jucatorul inca in viata este declarat 'winner' si nu se pai primesc inputuri.

```
Librari Folosite

* #include <avr/io.h>
* #include <stdio.h>
* #include <util/delay.h>
* #include <SPI.h>
* #include <avr/interrupt.h>
* #include <util/twi.h>
```

Rezultate Obţinute

Concluzii

Download

```
GitHub Link
```

Battleship-LED

```
Prototipul de aplicatie a fost realizat pe site-ul de simulari
```

Wokwi

Jurnal

Din pacate foarte mult din partea fizica a proiectului a ramas incompleta. Nu am apucat sa realizez scheletul de carton care sa separe cei doi jucatori similar cu boardgame-ul real. Pe partea unui jucator urma sa fie 1 panou de control, 1 ecran LED si un buzzer, iar ecranul LCD urma sa fie vizibil la o parte de ambii jucatori.

Nu am mai avut timp sa fac conecturi complete pentru prezentare, asa ca am mers cu varianta auxiliara de a conecta totul la breadboard-uri, ceea ce a lasat bateria si butonul de reset pentru prezentare neatasate si jocul greu de manevrat.

Initial, am vrut sa am tot proiectul in C, dar am aftat foarte tarziu ca SPI.h lucreaza cu clase, si am fost nevoit sa schimb la C++. Daca as fi lucrat cu C++ de la inceput m-as fi folosit de clase pentru implementarea proiectului in loc de struct.

Chiar si asa, jocul a functionat la testare doar cu cateva erori la inputuri cauzate de suduri slabe si panou nefixat pe loc.

Bibliografie/Resurse

- https://en.wikipedia.org/wiki/Battleship (game)
- https://ocw.cs.pub.ro/courses/pm/lab/lab3-2023-2024 (schelet Timer si Intreruperi)
- https://ocw.cs.pub.ro/courses/pm/lab/lab4-2023-2024 (schelet ADC)
- https://ocw.cs.pub.ro/courses/pm/lab/lab6-2023-2024 (schelet I2C)
- https://www.arduino.cc/reference/en/language/functions/communication/spi/ (SPI.h)
- https://lastminuteengineers.com/max7219-dot-matrix-arduino-tutorial/ (MAX7219 tutorial)