



# **UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA**

---

## **Snake Game cu Display OLED si Joystick**

---

**Autor:** Campean Bogdan

**Grupa:** 30237

FACULTATEA DE AUTOMATICA SI CALCULATOARE

6 ianuarie 2026

# Cuprins

<b>1</b>	<b>Introducere</b>	<b>2</b>
1.1	Descrierea Proiectului	2
1.2	Obiective	2
<b>2</b>	<b>Hardware Components</b>	<b>2</b>
2.1	Lista de Componente	2
2.2	Descrierea Conexiunilor	2
2.2.1	Display-ul OLED (Protocol I2C)	2
2.2.2	Joystick-ul Analogic	2
<b>3</b>	<b>Software</b>	<b>3</b>
3.1	Control si Input (Joystick Logic)	3
3.2	Miscare si Prevenirea Erorilor	3
3.3	Generarea Mancarii	4
3.4	Detectia Coliziunilor (Game Over)	4
<b>4</b>	<b>Testare</b>	<b>5</b>
4.1	Testare Hardware	5
4.2	Testare Funcțională	5

# 1 Introducere

## 1.1 Descrierea Proiectului

Acest proiect implementeaza jocul clasic *Snake* pe un sistem embedded bazat pe Arduino. Jocul este afisat pe un ecran OLED monocrom si controlat prin intermediul unui joystick analogic. Scopul proiectului este de a demonstra interactiunea in timp real dintre intrari (senzori), logica de procesare (microcontroller) si iesiri (display), intr-un format interactiv si distractiv.

## 1.2 Obiective

- Interfatarea unui display OLED SSD1306 prin protocolul I2C.
- Citirea si interpretarea datelor analogice de la un joystick pe doua axe.
- Implementarea logicii jocului (misiune, coliziuni, scor) in C++.
- Crearea unui sistem de meniu si restart fara resetarea hardware a placii.

# 2 Hardware Components

## 2.1 Lista de Componente

Sistemul este compus din urmatoarele elemente hardware:

1. **Arduino Uno** (sau compatibil): Microcontrollerul principal (ATmega328P).
2. **Display OLED 0.96 inch**: Rezolutie 128x64 pixeli, driver SSD1306.
3. **Modul Joystick**: Include doua potentiometre pentru axele X/Y si un buton digital.
4. **Breadboard si fire**: Pentru conexiuni.

## 2.2 Descrierea Conexiunilor

### 2.2.1 Display-ul OLED (Protocol I2C)

Ecranul comunica prin magistrala I2C, folosind doar doua fire pentru date:

- **SDA (Data)** → Conectat la pinul **A4** al Arduino.
- **SCL (Clock)** → Conectat la pinul **A5** al Arduino.
- Alimentare: 5V si GND.

### 2.2.2 Joystick-ul Analogic

Joystick-ul functioneaza ca un divizor de tensiune bazat pe doua potentiometre interne.

- **Valori**: Arduino citeste tensiunea de pe axe X si Y si o converteste digital intr-o valoare intre **0 si 1023**.
- **Repaus**: Cand joystick-ul este in pozitia centrala (liber), valoarea citita este aproximativ **512** (jumatea intervalului, corespunzator la 2.5V).

- **Extreme:** Miscarea maxima intr-o directie duce valoarea spre 1023, iar in directia opusa spre 0.

Conexiunile sunt:

- **VRx (Axa X)** → Pin Analogic **A0**.
- **VRy (Axa Y)** → Pin Analogic **A1**.
- **SW (Switch)** → Pin Digital **2** (INPUT\_PULLUP).

## 3 Software

Codul este scris in C++ folosind mediul Arduino IDE. Structura programului este impartita in functii modulare pentru a gestiona separat input-ul, logica si desenarea.

### 3.1 Control si Input (Joystick Logic)

Aceasta este partea care face legatura intre hardware si software. Joystick-ul returneaza valori intre 0 si 1023.

Pentru a interpreta corect directia, am implementat o **Dead Zone** intre valorile 300 si 700. Aceasta este necesara deoarece joystick-ul in pozitie de repaus nu sta perfect la 512, ci fluctueaza usor. Orice valoare intre 300 si 700 este ignorata, fiind considerata "centru".

Logica de decizie este urmatoarea:

- **DREAPTA:** Valoarea pe X depaseste 700 (tensiune mare).
- **STANGA:** Valoarea pe X scade sub 300 (tensiune mica).
- **JOS:** Valoarea pe Y depaseste 700.
- **SUS:** Valoarea pe Y scade sub 300.

```

1 xVal = analogRead(xPin);
2 yVal = analogRead(yPin);
3
4 if (xVal > 700) {
5     next_dir = right;
6 } else if (xVal < 300) {
7     next_dir = left;
8 } else if (yVal > 700) {
9     next_dir = down;
10 } else if (yVal < 300) {
11     next_dir = up;
12 }
```

Listing 1: Citirea si interpretarea joystick-ului

### 3.2 Miscare si Prevenirea Erorilor

Dupa stabilirea directiei dorite (**next\_dir**), codul verifica daca miscarea este valida. Este implementata o protectie care impiedica sarpele sa se intoarca la 180 de grade (ex: sa meargă

stanga cand el se deplaseaza spre dreapta), deoarece acest lucru ar duce la o coliziune imediata cu propriul gat.

Algoritmul de miscare functioneaza pe principiul "coada urmeaza capul". Parcurgem vectorul sarpele de la coada spre cap, mutand fiecare segment in pozitia celui din fata lui.

```

1 void move_right() {
2     // Mutam corpul: segmentul i ia locul segmentului i-1
3     for (int i = snake_length - 1; i > 0; i--) {
4         snakeX[i] = snakeX[i - 1];
5         snakeY[i] = snakeY[i - 1];
6     }
7     // Mutam capul in noua directie
8     snakeX[0] += block_length;
9 }
```

Listing 2: Algoritmul de miscare a corpului

### 3.3 Generarea Mancarii

Mancarea trebuie sa apara aleatoriu pe ecran, dar aliniata perfect la grila de joc. Deoarece sarpele are dimensiunea de 4x4 pixeli (block\_length), coordonatele mancarii trebuie sa fie multipli de 4.

Folosim functia `random()` pentru a genera o pozitie valida, evitand marginile ecranului.

```

1 void spawn_food() {
2     if (!foodExists) {
3         // Generam o pozitie aleatorie in interiorul grilei
4         foodX = (random(1, (SCREEN_WIDTH / block_length) - 2)) * block_length;
5         foodY = (random(1, (SCREEN_HEIGHT / block_width) - 2)) * block_width;
6         foodExists = true;
7     }
8     // Desenam mancarea pe ecran
9     display.fillRect(foodX, foodY, block_length, block_width, SSD1306_WHITE);
10 }
```

Listing 3: Generarea coordonatelor pentru mancare

### 3.4 Detectia Coliziunilor (Game Over)

Exista doua tipuri de evenimente care declanseaza sfarsitul jocului. Functia `detect_col()` verifică aceste conditii la fiecare frame:

1. **Coliziunea cu Peretii:** Se verifica daca coordonatele capului (`snakeX[0]`, `snakeY[0]`) au iesit din dimensiunile ecranului (0-128 pe X, 0-64 pe Y).
2. **Coliziunea cu Sine Insusi:** Se parurge vectorul corpului incepand de la al doilea segment. Daca coordonatele capului coincid cu coordonatele oricarui alt segment, inseamna ca sarpele s-a muscat singur.

```

1 if (snake_length > 3) {
2     for (int i = 1; i < snake_length; i++) {
3         // Daca capul este in aceeasi pozitie cu un segment din corp
4         if (snakeX[0] == snakeX[i] && snakeY[0] == snakeY[i]) {
```

```
5     game_over();
6     return;
7 }
8 }
9 }
```

Listing 4: Verificarea coliziunii cu propriul corp

## 4 Testare

### 4.1 Testare Hardware

S-a verificat alimentarea corecta a componentelor si adresa I2C a ecranului (0x3C). Valorile joystick-ului au fost monitorizate prin Serial Plotter pentru a confirma ca revin la valoarea 512(aprox) cand maneta este eliberata.

### 4.2 Testare Functionala

1. **Miscare:** Sarpele raspunde prompt la comenzi, fara intarzieri vizibile.
2. **Mancare:** Generarea functioneaza corect, iar sarpele creste in dimensiune la contact.
3. **Coliziuni:** Limitele ecranului sunt respectate strict. Jocul se opreste corect la impactul cu peretele sau cu propria coada.
4. **Restart:** Butonul joystick-ului (pin 2) reseteaza cu succes jocul dupa ecranul de final.