# Documentație Tehnică

Tetris pe Arduino Mega

Arduino Mega + ESP32 + Python Bridge

Student: Tarța Xenia-Maria
Facultatea de Automatică și Calculatoare

11 ianuarie 2026

## Cuprins

# 1 Introducere

Jocul Tetris este un exemplu clasic de algoritm bazat pe matrici și gestionare a stărilor. În acest proiect, am extins conceptul tradițional prin adăugarea componentelor IoT (Internet of Things), permițând vizualizarea jocului în timp real pe un dispozitiv mobil, fără o conexiune fizică directă între controler și ecranul secundar.

## 1.1 Obiectivele Proiectului

- Implementarea motorului de joc (Game Engine) pe un microcontroler (Arduino Mega).
- Salvarea permanentă a scorului maxim (High Score) folosind memoria non-volatilă EEPROM.
- Crearea unui "Bridge" software în Python pentru a interconecta două porturi seriale distincte și a afișa interfața grafică.
- Implementarea unui server web asincron pe ESP32 pentru streaming-ul matricii de joc către un browser mobil (Smartphone).

# 2 Arhitectura Sistemului

Sistemul utilizează o topologie de tip **Hub-and-Spoke**, unde Laptopul funcționează ca nod central de procesare și rutare.

## 2.1 Fluxul Datelor

1. **Arduino Mega:** Citește Joystick-ul → Calculează Matricea → Trimite prin Serial ('FRAME').
2. **Python (PC):** Primește datele → Desenează pe Ecran → Extrage Scorul și Grila → Trimite prin al doilea port Serial către ESP32.
3. **ESP32:** Primește datele → Actualizează variabila globală → Servește pagina HTML → Trimite JSON către telefon prin Wi-Fi.

# 3 Detalii de Implementare Software

## 3.1 1. Algoritmul de Joc (Game Engine Logic)

Implementarea logicii Tetris pe Arduino Mega se bazează pe standardul *SRS (Super Rotation System)*. Spațiul de joc este reprezentat printr-o matrice bidimensională de numere întregi 'int grid[20][10]'.

Esența algoritmului este funcția de predicție a coliziunilor ('isValid'). Înainte ca o piesă să fie mutată sau rotită, algoritmul calculează coordonatele viitoare ale celor 4 blocuri componente. Mutarea este acceptată doar dacă:

1. Toate blocurile se află în interiorul limitelor matricii ($x \in [0, 9], y \in [0, 19]$).

2. Pozițiile viitoare nu sunt ocupate deja de alte valori nenule în matrice.

```
bool isValid(int x, int y, int rot) {
  for (int i = 0; i < 4; i++) {
    int px = x + shapes[currentType][rot][i][0];
    int py = y + shapes[currentType][rot][i][1];

    if (px < 0 || px >= WIDTH || py >= HEIGHT)
      return false;

    if (py >= 0 && grid[py][px] != 0)
      return false;
  }
  return true;
}
```

Listing 1: Algoritmul de detectare a coliziunilor (Arduino)

## 3.2 2. Arduino Mega - Bucla Principală și Control

Funcția 'loop()' este inima sistemului și rulează în mod repetat. Aceasta implementează o Mașină de Stări Finite (FSM) cu trei stări principale: **Meniu**, **Game Over** și **Joc Activ**.

Pentru a asigura un gameplay fluid, nu am utilizat funcția 'delay()' (care ar bloca procesorul), ci am folosit cronometrul intern 'millis()' pentru a gestiona input-ul și gravitația în paralel (Multitasking cooperativ).

**Explicația detaliată a codului:**

- **Starea de Meniu:** Cât timp 'gameStarted' este fals, microcontrolerul trimite mesajul "START" către PC și așteaptă apăsarea butonului de Joystick (pin digital LOW) pentru a inițializa variabilele și a începe jocul.
- **Control Joystick (Input):** Citim valorile analogice (0-1023).
  - Axa X: Valori $< 200$ înseamnă Dreapta, valori $> 800$ înseamnă Stânga.
  - Axa Y: Valori $< 200$ declanșează rotația (incrementare modulo 4), valori $>$ 800 activează "Fast Drop" (scade intervalul de cădere la 50ms).
- **Gravitația:** La fiecare interval de timp ('dropInterval'), piesa coboară automat un rând. Dacă mișcarea nu este validă (piesa a atins baza), se apelează 'lockPiece()'.

```
void loop() {
  if (!gameStarted) {
    Serial.println("START");

    if (digitalRead(pinButon) == LOW) {
      gameStarted = true;
      gameOver = false;
      score = 0;
      memset(grid, 0, sizeof(grid));
      spawnPiece();
      delay(500);
    }
    delay(100);
```

```
    return;
  }

  if (gameOver) {
    if (digitalRead(pinButon) == LOW) {
      gameStarted = false;
      delay(1000);
    }
    return;
  }
  unsigned long now = millis();

  if (now - lastMoveTime > moveInterval) {
    int xVal = analogRead(pinX);
    int yVal = analogRead(pinY);

    if (xVal > 800) {
      if (isValid(currentX - 1, currentY, currentRotation))
        currentX--;
      lastMoveTime = now;
    }
    else if (xVal < 200) {
      if (isValid(currentX + 1, currentY, currentRotation))
        currentX++;
      lastMoveTime = now;
    }

    if (yVal < 200) {
      int newRot = (currentRotation + 1) % 4;
      if (isValid(currentX, currentY, newRot))
        currentRotation = newRot;
      delay(150); multipla
    }

    if (yVal > 800)
      dropInterval = 50;
    else
      dropInterval = 500;
  }

  if (now - lastDropTime > dropInterval) {
    if (isValid(currentX, currentY + 1, currentRotation))
      currentY++; // La fiecare interval piesa coboara un rand
    else
      lockPiece(); // Daca nu mai poate cobori, se blocheaza
    lastDropTime = now;
  }

  sendDisplayData();
  delay(30);
}
```

Listing 2: Implementarea completă a buclei de joc (Arduino Mega)

## 3.3 3. Python Bridge - Streaming și Rate Limiting

Una dintre cele mai mari provocări a fost instabilitatea modulului ESP32 la primirea unui volum mare de date. Am implementat în Python un algoritm de "Rate Limiting" care trimite datele către ESP32 doar dacă au trecut 0.2 secunde de la ultima transmisie și doar dacă matricea s-a modificat.

Listing 3: Logica de Streaming din Python

```
last_esp_update = 0
last_sent_grid = ""

if ser_mega.in_waiting > 0:
    line = ser_mega.readline().decode('utf-8').strip()

    if line.startswith("FRAME:"):
        parts = line.split(":")
        current_grid = parts[1]

        draw_game(current_grid, current_score, current_high)

        now = time.time()
        if ser_esp and (now - last_esp_update > 0.2):
            if current_grid != last_sent_grid:
                msg = f"G:{current_grid},{current_score}\n"
                ser_esp.write(msg.encode('utf-8'))

                last_sent_grid = current_grid
                last_esp_update = now
```

## 3.4 4. ESP32 - Optimizare Hardware și Web Server

Pentru a preveni erorile de tip "Brownout" (cădere de tensiune) cauzate de consumul Wi-Fi simultan cu procesarea Serială, am redus frecvența procesorului și puterea de transmisie.

```
#include "esp32-hal-cpu.h"

void setup() {
  setCpuFrequencyMhz(80);

  Serial.begin(115200);
  Serial.setTimeout(5);

  WiFi.mode(WIFI_AP);
  WiFi.softAP("Tetris_Arena", "parola_tetris");

  WiFi.setTxPower(WIFI_POWER_11dBm);

  server.on("/", handleRoot);
  server.on("/data", handleData);
  server.begin();
}
```

Listing 4: Configurarea Low-Power pe ESP32

# 4 Bibliografie și Resurse Utilizate

Implementarea acestui proiect s-a bazat pe documentația oficială a platformelor utilizate și pe algoritmi standard de programare a jocurilor.

# Bibliografie


[1] **Tetris Guideline  SRS (Super Rotation System)**.
Descrierea algoritmului matematic standard pentru rotația pieselor și definirea formelor (I, J, L, O, S, T, Z) într-un sistem de coordonate cartezian.
Sursă: `https://tetris.wiki/Super_Rotation_System`

[2] **Arduino Reference Documentation**.
Documentația pentru funcțiile `Serial`, `EEPROM` și manipularea pinilor digitali/analogici.
Disponibil la: `https://www.arduino.cc/reference/en/`

[3] **Espressif ESP32 WebServer Library**.
Resurse pentru crearea punctului de acces (SoftAP) și gestionarea cererilor HTTP asincrone.
Sursă: `https://github.com/espressif/arduino-esp32/tree/master/libraries/WebServer`

[4] **Pygame Documentation**.
Utilizat pentru desenarea primitivelor grafice (‘pygame.draw.rect‘) și gestionarea ferestrei de afișare pe laptop.
Disponibil la: `https://www.pygame.org/docs/`

[5] **pySerial Documentation**.
Biblioteca folosită pentru comunicarea UART (Universal Asynchronous Receiver-Transmitter) între Python și microcontrolere.
Disponibil la: `https://pyserial.readthedocs.io/`