SIMON GAME

Arduino Platform Project

Student: GIL Antony

Course: Arduino Platform Project

Professor: Paweł Król

Academic Year: 2025/2026

University: Politechnika Krakowska im. Tadeusza Kościuszki

Politechnika Krakowska
im. Tadeusza Kościuszki

Table des matières

I. PROJECT OVERVIEW

A. Objectives

This project implements a Simon Game on Arduino, a classic electronic memory game that tests and improves the player's ability to remember increasingly complex sequences. The system uses LEDs to display patterns that the player must reproduce by pressing corresponding buttons.

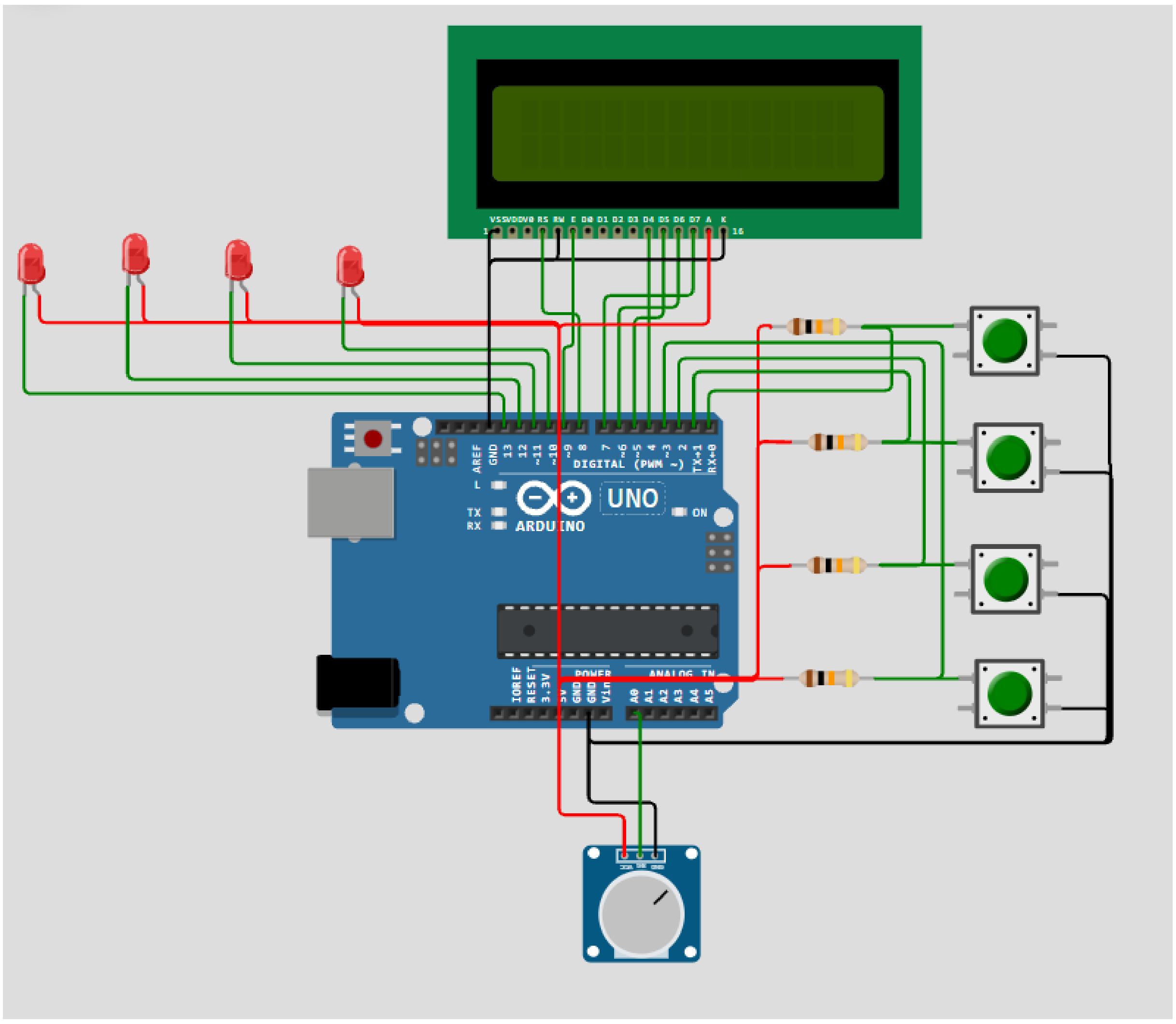
The main objectives are:

- Create an interactive memory game with real-time feedback
- Implement dynamic difficulty adjustment through pattern length progression
- Provide visual feedback through LEDs and textual information via LCD display
- Allow speed customization through potentiometer control
- Demonstrate Arduino programming skills and hardware interfacing

B. Main Features

The Simon Game combines hardware interfacing with game logic to create an interactive memory challenge. Players must reproduce LED patterns of increasing length, with their score and progress displayed on an LCD screen. The potentiometer allows real-time speed adjustment to customize difficulty.

II. SYSTEM DIAGRAM



Hardware Configuration

Component	Variable Name	Pins Used	Logic / Configuration
LED Indicators	leds[]	10, 11, 12, 13	Output (Active LOW)
Push Buttons	buttons[]	0, 1, 2, 3	Input (Active LOW / Pull-up)
LCD Display	lcd	4, 5, 6, 7, 8, 9	4-bit Mode (LiquidCrystal)
Speed Control	sensorValue	A0	Analog Input (0-5V)

Link : https://wokwi.com/projects/359988057917665281

III. CODE IMPLEMENTATION

A. Libraries Used

LiquidCrystal.h

- Standard Arduino library for LCD control
- Provides functions for text display and cursor positioning
- Simplifies 4-bit communication with LCD display

Standard Arduino Functions

- pinMode() - Configure pins as input/output
- digitalWrite() - Control LED states
- digitalRead() - Read button states
- analogRead() - Read potentiometer value
- delay() - Timing control
- millis() - Non-blocking timing and randomization
- random() - Generate random pattern sequences

B. Code Structure

Global Variables Description

Variable	**Type**	**Description / Role**
leds[4]	int Array	Stores the digital output pins connected to the 4 game LEDs.
buttons[4]	int Array	Stores the digital input pins connected to the 4 player buttons.
pattern[20]	int Array	Buffer array used to store the random sequence generated by the game.
patternLength	int	Represents current difficulty. Defines how many steps of the pattern are currently active.
score	int	Accumulates the player's points based on successful rounds.
patternCount	int	Counter for the number of rounds played (used to trigger Game Over at 10).
delayTime	int	The speed of the sequence in milliseconds (dynamically updated by the potentiometer).

Setup Function

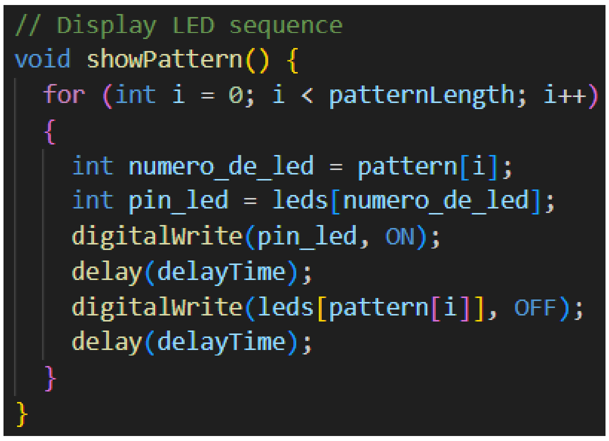
- Initialize all LED pins as OUTPUT and turn them off
- Configure button pins as INPUT
- Read initial potentiometer value for speed setting
- Initialize LCD display (16 columns, 2 rows)
- Display initial game state

Main Loop

1. Read potentiometer for speed adjustment
2. Generate random pattern of current length
3. Display pattern to player using showPattern()
4. Wait for player input with playerTurn()
5. Update score and pattern length based on result
6. Display error flash if player made mistake
7. Update LCD with current game state
8. Check for game over conditions
9. Prepare for next round

C. Key Functions

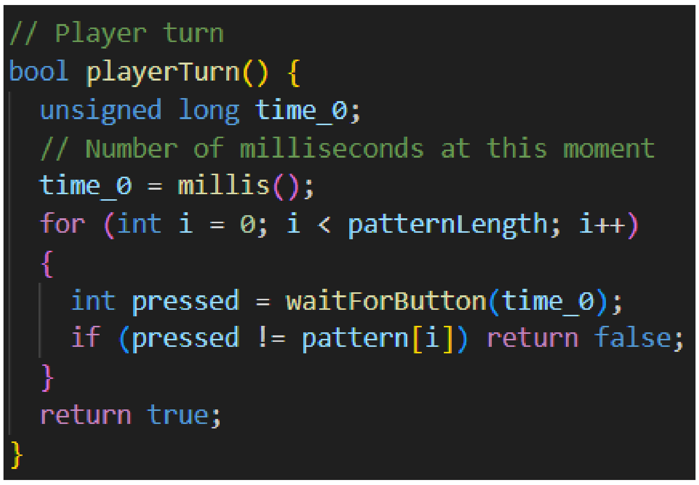
showPattern()



```
// Display LED sequence
void showPattern() {
  for (int i = 0; i < patternLength; i++)
  {
    int numero_de_led = pattern[i];
    int pin_led = leds[numero_de_led];
    digitalWrite(pin_led, ON);
    delay(delayTime);
    digitalWrite(leds[pattern[i]], OFF);
    delay(delayTime);
  }
}
```

- Displays the LED sequence to memorize
- Uses delay time controlled by potentiometer
- Each LED lights up for delayTime milliseconds
- Pauses between LEDs for visual clarity

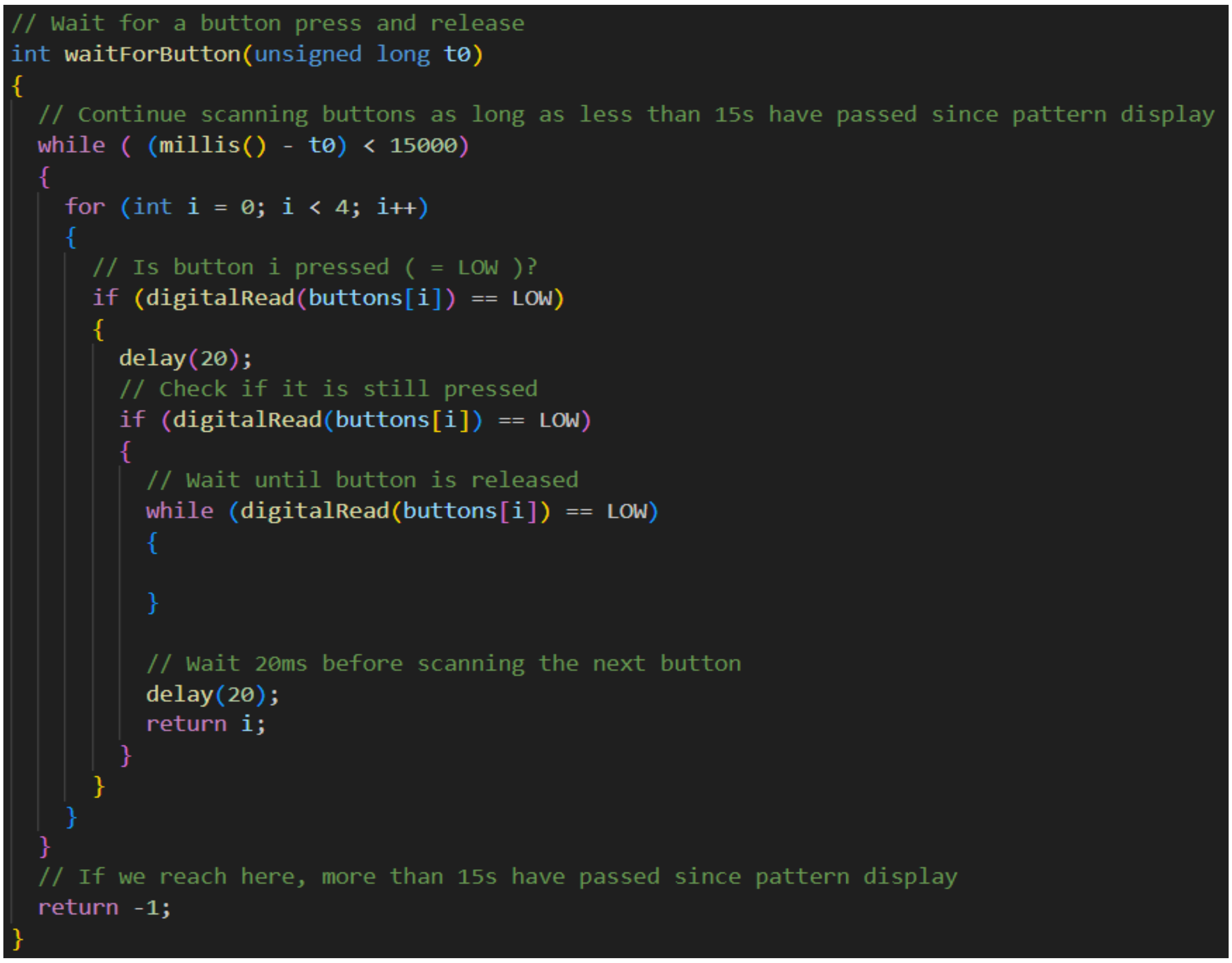
playerTurn()



```
// Player turn
bool playerTurn() {
  unsigned long time_0;
  // Number of milliseconds at this moment
  time_0 = millis();
  for (int i = 0; i < patternLength; i++)
  {
    int pressed = waitForButton(time_0);
    if (pressed != pattern[i]) return false;
  }
  return true;
}
```

- Captures player's button presses
- Compares each press with expected pattern
- Returns true if all presses match
- Returns false on first mismatch

waitForButton()



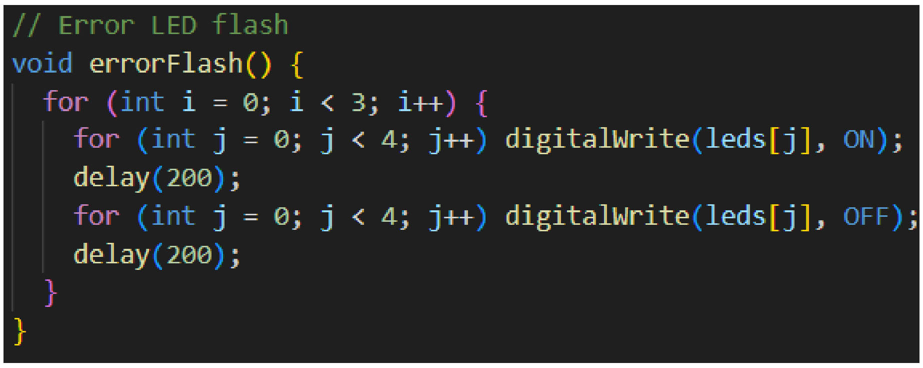
```
// Wait for a button press and release
int waitForButton(unsigned long t0)
{
  // Continue scanning buttons as long as less than 15s have passed since pattern display
  while ( (millis() - t0) < 15000)
  {
    for (int i = 0; i < 4; i++)
    {
      // Is button i pressed ( = LOW )?
      if (digitalRead(buttons[i]) == LOW)
      {
        delay(20);
        // Check if it is still pressed
        if (digitalRead(buttons[i]) == LOW)
        {
          // Wait until button is released
          while (digitalRead(buttons[i]) == LOW)
          {

          }

          // Wait 20ms before scanning the next button
          delay(20);
          return i;
        }
      }
    }
  }
  // If we reach here, more than 15s have passed since pattern display
  return -1;
}
```

- Waits for button press with 15-second timeout
- Implements software debouncing (20ms delay)
- Confirms button press by checking twice
- Waits for button release before returning
- Returns -1 if timeout occurs

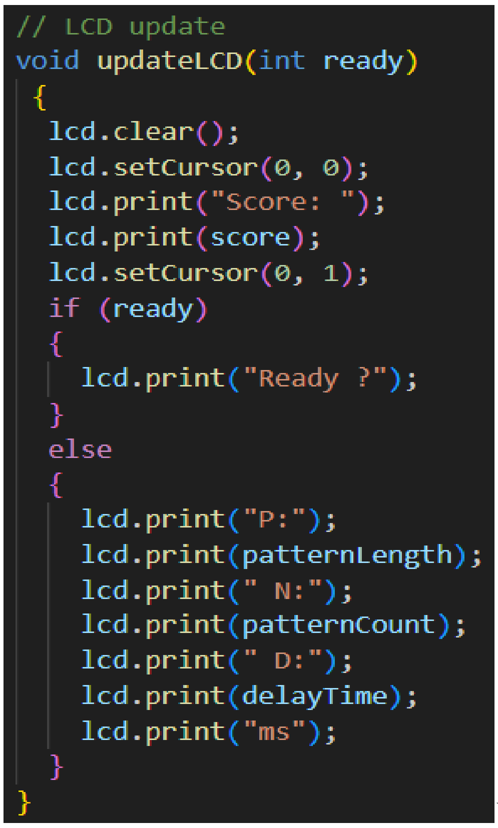
errorFlash()



```
// Error LED flash
void errorFlash() {
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) digitalWrite(leds[j], ON);
    delay(200);
    for (int j = 0; j < 4; j++) digitalWrite(leds[j], OFF);
    delay(200);
  }
}
```

- Provides visual feedback for errors
- Flashes all LEDs simultaneously 3 times
- Clear indication that player made a mistake

updateLCD()

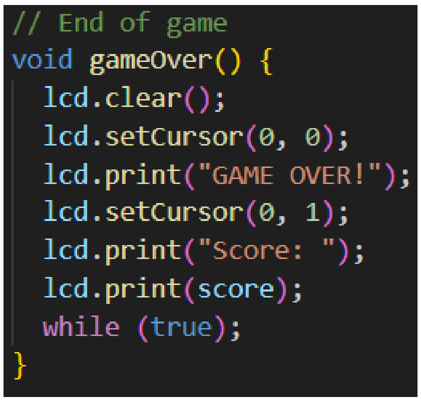


```
// LCD update
void updateLCD(int ready)
 {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Score: ");
  lcd.print(score);
  lcd.setCursor(0, 1);
  if (ready)
  {
    lcd.print("Ready ?");
  }
  else
  {
    lcd.print("P:");
    lcd.print(patternLength);
    lcd.print(" N:");
    lcd.print(patternCount);
    lcd.print(" D:");
    lcd.print(delayTime);
    lcd.print("ms");
  }
}
```

}

- Updates LCD with game information
- Line 1: Current score
- Line 2: Pattern length, round number, delay time
- Shows "Ready?" message between rounds

gameOver()



```
// End of game
void gameOver() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("GAME OVER!");
  lcd.setCursor(0, 1);
  lcd.print("Score: ");
  lcd.print(score);
  while (true);
}
```

- Displays final score
- Halts program execution
- Requires reset to play again

IV. TECHNICAL DESIGN CHOICES

A. Scoring System

Score Calculation:

- Success: score += patternLength
- Failure: score -= patternLength
- Minimum pattern length: 2 (prevents too easy gameplay)

Design Rationale:

- Rewards increase with difficulty
- Penalties proportional to failed challenge
- Negative score possible (adds pressure)
- Pattern length never drops below 2 (maintains minimum challenge)

B. Game Over Conditions

Two End Conditions:

1. 10 rounds completed
2. Score becomes negative

Reasoning:

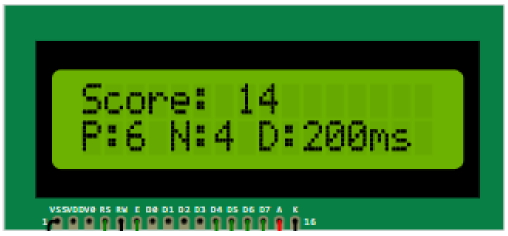
- 10 rounds provides complete game experience
- Negative score creates fail state
- Prevents infinite grinding
- Clear win/lose objectives

C. LCD Information Display

Two Display Modes:

Mode 1 - Active Game:

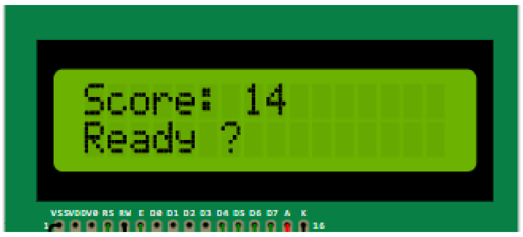
- P: Pattern Length (current difficulty)
- N: Round Number (progress tracker)
- D: Delay Time (current speed)



Mode 2 - Between Rounds

Benefits:

- Compact information display
- Real-time feedback
- Players can track their progress
- Speed adjustment visible immediately

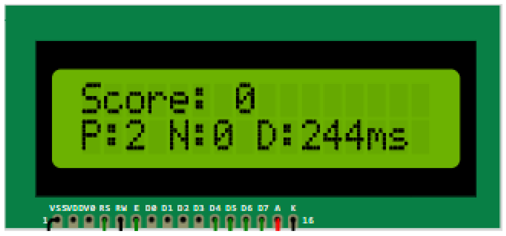


V. SYSTEM FUNCTIONALITY

A. Game Start

When the Arduino powers on:

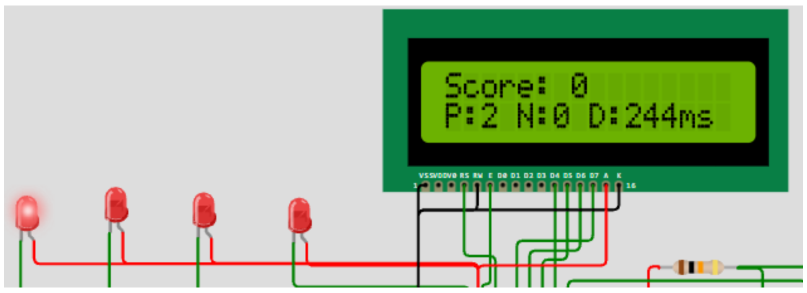
1. All LEDs turn off
2. LCD displays initial score (0)
3. System waits 1.5 seconds
4. First pattern is generated



B. Pattern Display Phase

The system displays the sequence:

- Each LED in the pattern lights up sequentially
- Delay between LEDs controlled by potentiometer
- Pattern length starts at 2, increases with success



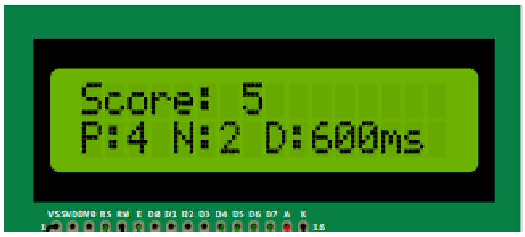
C. Player Input Phase

Player must reproduce the pattern:

- Press buttons in the same order as LEDs lit up
- 15-second timeout for complete sequence
- Visual feedback: LED lights when button pressed
- Timeout or wrong press = error

D. Game Information Display

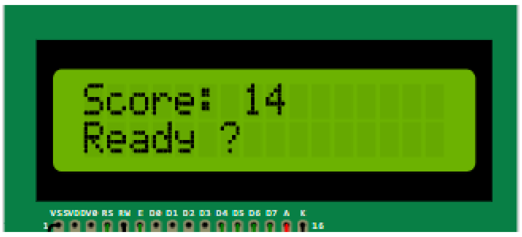
LCD shows real-time information:



During gameplay:

Score: 5

P:4 N:2 D:600ms



Between rounds:

Score: 14

Ready ?

E. Success Feedback

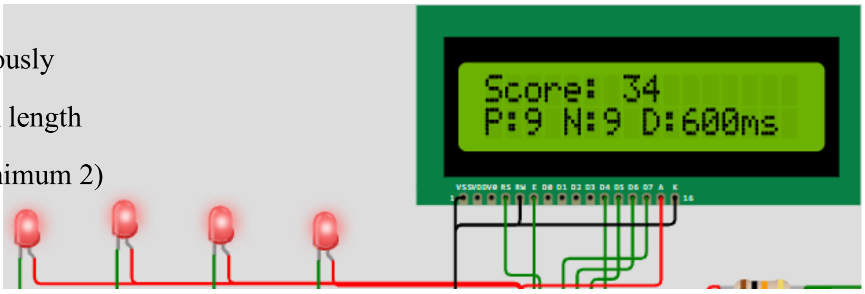
On successful pattern reproduction:

- Score increases by pattern length
- Pattern length increases by 1
- Round counter increments
- "Ready?" message displayed briefly
- Next pattern begins

F. Error Feedback

On incorrect input:

- All LEDs flash 3 times simultaneously
- Score decreases by current pattern length
- Pattern length decreases by 1 (minimum 2)
- Round counter still increments



G. Game Over

Game ends when:

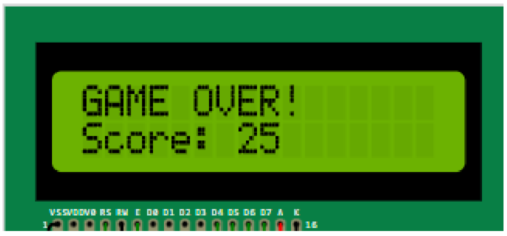
- 10 rounds completed, OR
- Score becomes negative

Final Screen:

GAME OVER!

Score: 25

Program halts, requiring Arduino reset to play again.



H. Speed Adjustment

Players can adjust difficulty:

- Turn potentiometer during gameplay
- Changes take effect in next round
- Faster = more challenging
- Slower = easier to memorize

Speed Range:

- Minimum: 200ms per LED
- Maximum: 1000ms per LED

VI. SOURCE CODE REPOSITORY

The complete source code is available on GitHub at https://github.com/AntonyGil1999/Simon-Game

Installation of the Source Code from GitHub:

1. Create an empty folder on your computer.
2. Open Git Bash and navigate to this folder.
3. Run the following command to clone the repository:

git clone https://github.com/AntonyGil1999/Simon-Game.git

The repository includes:

- src/ folder containing the main application source code.
- include/ and lib/ folders for headers and libraries.
- platformio.ini for project configuration.
- test/ folder for verification scripts.
- Wokwi System Diagram.png showing the hardware connections.
- GIL Antony Projet Arduino Simon Game.pdf, my rapport.

Usage:

- **Open:** Open the project folder in your IDE (VS Code with PlatformIO or Arduino IDE).
- **Setup:** Connect the Arduino Uno via USB.
- **Upload:** Compile and upload the sketch to the board.
- **Execution:** The game starts automatically after the upload is complete.

VII. CONCLUSION

This project implements an interactive Simon Game on Arduino, demonstrating embedded systems programming and hardware interfacing.

Technical Skills Applied:

Programming Concepts:

- Digital I/O for LED control
- Analog input processing for potentiometer
- LCD display integration and control
- Software debouncing implementation
- Non-blocking timing with millis()
- Random number generation and pattern logic
- Modular function design
- Array manipulation and state management
- Timeout handling and input validation

Hardware Integration:

- Multiple component interfacing
- Pin configuration and management
- Electrical connections and circuit design
- Real-time system responsiveness

Learning Outcomes:

Through this project, I gained practical experience with:

- Arduino programming environment and workflow
- Hardware debugging and troubleshooting techniques
- Game logic implementation in embedded systems
- User interface design for constrained displays
- Real-time interaction design and timing management

Possible Enhancements:

Future improvements could include:

- Multiple difficulty levels (easy, medium, hard presets)
- Sound effects using piezo buzzer for audio feedback
- High score saving using EEPROM for persistence
- Multiplayer mode with turn-taking functionality
- Color-coded LEDs for enhanced visual variety
- Progressive speed increase within individual rounds
- Statistics tracking (best streak, average score)

Reflections:

This project demonstrates Arduino's capabilities for creating interactive applications that combine hardware control with engaging user experiences. The modular code structure and clear separation between game logic and hardware control made development and debugging more manageable. The use of software debouncing and timeout handling addressed real-world timing challenges inherent in embedded systems.