COMP 1045: INTERNET OF THINGS USING ARDUINO

Final Project – Memory Game

Student name: Davi Soares Mendes

Student name: Erandi Canafistula Araujo

1. Introduction

In this project, we built a memory game. This an interactive circuit designed to test and improve memory and attention skills. In this game, the Arduino controls LEDs and a buzzer to display a sequence of lights and sounds, which the player must then replicate by pressing corresponding keys in the keyboard (using the Serial Monitor). This project demonstrates Arduino programming, LCD integration, and user input handling.

2. Objectives

- Implement a Memory Game: design a game where the Arduino generates a random sequence of LED lights and sounds that the player must repeat;
- Use LCD for Display: utilize an LCD to provide instructions and feedback to the player;
- Integrate LEDs and Buzzer: employ LEDs to visually indicate the sequence and a buzzer to produce corresponding tones;
- Develop a Robust Game Logic: ensure that the game handles user inputs correctly, checks for correctness, and provides appropriate feedback.

3. Methodology

The project was developed and tested using Tinkercad, a powerful online circuit simulation platform that facilitated both virtual construction and debugging. The process began with designing the circuit in Tinkercad, where virtual components including LEDs, a buzzer, and an LCD display were configured and connected. The simulator's interface allowed for precise placement of components and wiring, ensuring that all connections were correctly established.

With the circuit design complete, the Arduino code was written and uploaded directly into Tinkercad's integrated code editor. This code managed the core game functions, such as generating and displaying sequences, controlling the LEDs and buzzer, and processing user inputs. The ability to run simulations within Tinkercad enabled real-time observation of the game's behavior. This was crucial for verifying that the LEDs lit up in the correct sequence, the buzzer emitted the appropriate tones, and the LCD displayed the intended messages.

Testing and debugging were integral parts of the simulation process. Any discrepancies or issues identified during simulation—such as incorrect LED responses or LCD display errors—were addressed by adjusting the circuit connections and refining the code. Tinkercad’s debugging tools played a vital role in tracing and resolving these issues efficiently.

The interaction between the user and the game was simulated using Tinkercad’s virtual Serial Monitor, which allowed for the testing of user inputs. This simulation ensured that the game correctly captured user inputs and provided accurate feedback on the LCD.

Upon completing the simulations and confirming that all components and interactions functioned as intended, the project was validated. The successful simulation of the memory game demonstrated that it operated correctly, with sequences displayed as expected, accurate user input handling, and effective feedback mechanisms. Using Tinkercad streamlined the development process, allowing for comprehensive testing and refinement in a virtual environment before any physical hardware implementation.

4. Components and sensors used

- Arduino Uno: the microcontroller board used to control and program the game logic;
- Arduino Breadboard Small and Breadboard Mini: used to prototype the circuit by providing a platform to connect and organize the components;
- 16x2 LCD Display: used to display game instructions and status messages to the player.
- 4 LEDs (green, yellow, blue, and red): represent the game sequence; the LEDs light up in a specific order that the player needs to replicate;
- Buzzer: produces sound signals that correspond to the LED lights, providing audio feedback to the player.

- 5 Resistors (220 ohms): these resistors are used to limit the current flowing through the LEDs and the LCD display;
- Wires: used to make electrical connections between the components and the Arduino board.

5. Code explanation

The code for the memory game begins by including the LiquidCrystal library, which is essential for controlling the LCD display. The LCD is connected to the Arduino via specific pins for enabling, registering, and data transfer, and is initialized with the statement LiquidCrystal lcd(12, 11, 5, 4, 3, 2);. The pin configuration for the LEDs and the buzzer is also defined, with each LED assigned a unique pin and associated with a character used for player inputs.

The setup() function is responsible for initializing the game. It starts by setting up the LCD display, where a welcome message prompts the user to press any key to start the game. This function also configures the pins connected to the LEDs and the buzzer as outputs, and seeds the random number generator with a value derived from an analog read to ensure a new sequence is generated each time the game starts.

In the loop() function, the game's main logic is executed. Initially, it checks whether the game has started. If not, it waits for serial input from the user. When any input is detected, the startGame() function is invoked. This function resets the game level to zero, generates a random sequence of LED indices, and updates the LCD to instruct the player to follow the sequence.

Once the game has started, the loop() function determines whether it should display the sequence or wait for player input. If it's time to show the sequence, the showSequence() function is called. This function iterates through the sequence, turning on each LED in turn and playing the corresponding note using the buzzer. It also introduces delays to ensure the sequence is presented clearly. After displaying the sequence, the LCD prompts the player to input their response.

When it's time for the player to input their response, the playerInput() function is used to capture and evaluate the input. The function reads characters from the serial input, maps them to corresponding LEDs, and checks whether they match the sequence. If the player’s sequence matches the generated sequence, the game advances to the next level. If there’s an error, the game ends, and an appropriate message is displayed on the LCD. The function also incorporates a short delay before resetting the state to wait for the next sequence display.

Overall, the code effectively manages the game’s flow by handling initialization, sequence generation, player input, and game progression. Each function is designed to control a specific aspect of the game, ensuring that the gameplay experience is smooth and interactive.

6. Troubleshooting steps

When troubleshooting issues with the memory game circuit, the first step is to address any problems with the LCD display. If the LCD is not showing any output, start by verifying the connections to the LCD. Ensure that the pins used for enabling, registering, and data transfer are correctly connected to the Arduino and match the pin configuration specified in the code. Additionally, confirm that the LCD is properly powered. If the connections are correct and the display still doesn’t work, double-check the initialization settings in the code and ensure that the LCD library is correctly included and used.

If the LEDs are not lighting up as expected, inspect the wiring between the LEDs and the Arduino. Each LED should be connected to the correct pin, as defined in the ledPins array in the code. Make sure that the LED pins are configured as outputs in the setup() function. If the LEDs are correctly connected but not functioning, test them individually with a simple blink code to confirm that they are operational and that the circuit is correctly wired.

When the buzzer fails to produce sound, first verify the buzzer's connection to the correct pin as specified in the code. Ensure that the pin is set as an output in the setup() function. To diagnose the issue, upload a basic tone-generating sketch to test the buzzer. If the buzzer is still silent, check the connections and consider testing with a different buzzer to rule out hardware faults.

For issues with serial input, where the game does not respond to user inputs, ensure that the Serial Monitor settings match the baud rate specified in the Serial.begin(9600) command. Verify that the serial communication is correctly initialized and that inputs are correctly read and processed by the playerInput() function. If the input is not being recognized, confirm that the characters entered in the Serial Monitor correspond to the expected input values and that the Serial Monitor is configured to send newline characters if required by the code.

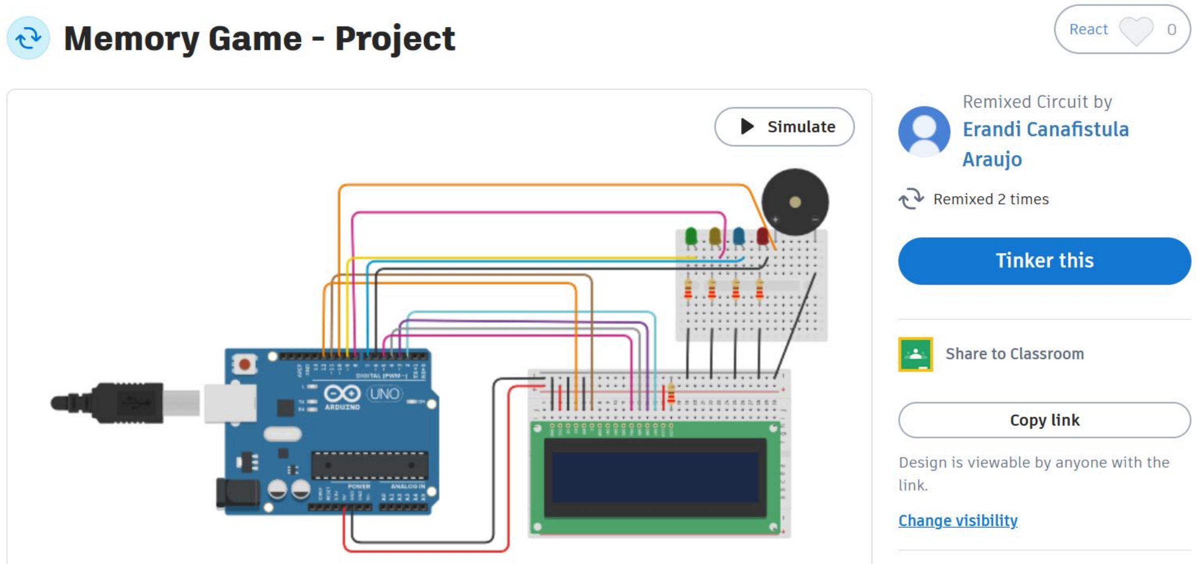
Lastly, if the game's logic does not work as intended, such as incorrect sequence handling or unexpected game behavior, review the game logic implemented in functions like startGame(), showSequence(), and playerInput(). Insert debug statements to output intermediate values to the Serial Monitor, which can help trace where the logic might be failing. Ensure that the sequence generation, display, and comparison mechanisms are functioning as intended and that the game flow transitions correctly between different states.

7. Conclusion and future scope

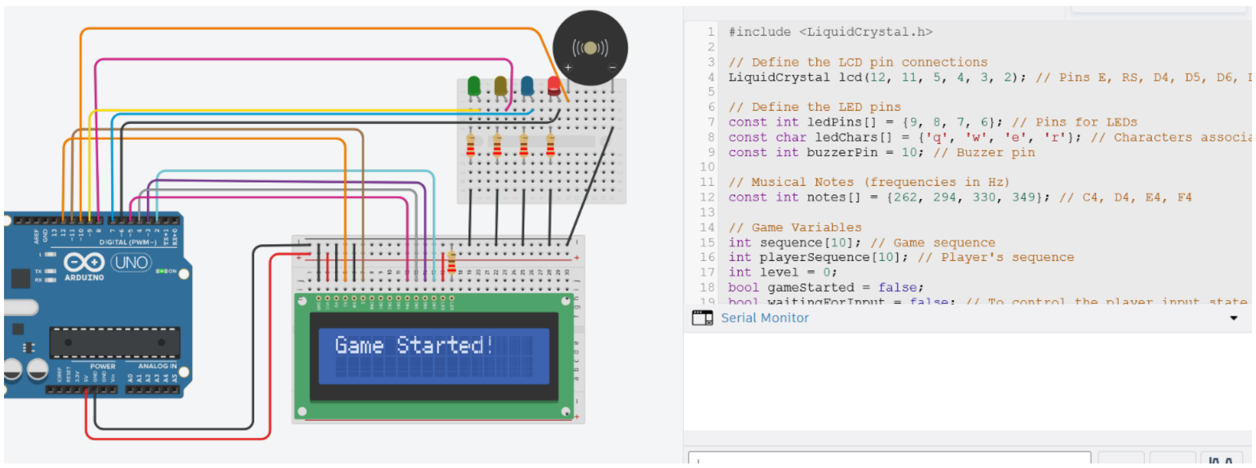
The memory game project successfully demonstrates the integration of various components with an Arduino to create an engaging and interactive game. By utilizing LEDs, a buzzer, and an LCD display, the project provides a functional and educational example of how to implement game logic and user interaction in an embedded system. The use of Tinkercad for simulation ensured that the game was thoroughly tested and refined in a virtual environment before any physical hardware was assembled. The successful implementation of the game highlights the versatility of the Arduino platform in developing electronic projects that involve both visual and auditory feedback.

There are several opportunities to expand and enhance the memory game in future iterations. Increasing the complexity of the game by extending the sequence length and adding multiple difficulty levels could provide a greater challenge and make the game more engaging. Implementing physical buttons or a touchscreen for user input would enhance the interactivity and usability of the game. Additionally, integrating a scoring system to track player performance over multiple rounds could add a competitive element and increase replayability. Exploring advanced features such as multiplayer modes or timed challenges could further enhance the game's appeal and functionality. These improvements would not only refine the game but also provide valuable learning experiences in both hardware and software development.

8. All screen shots

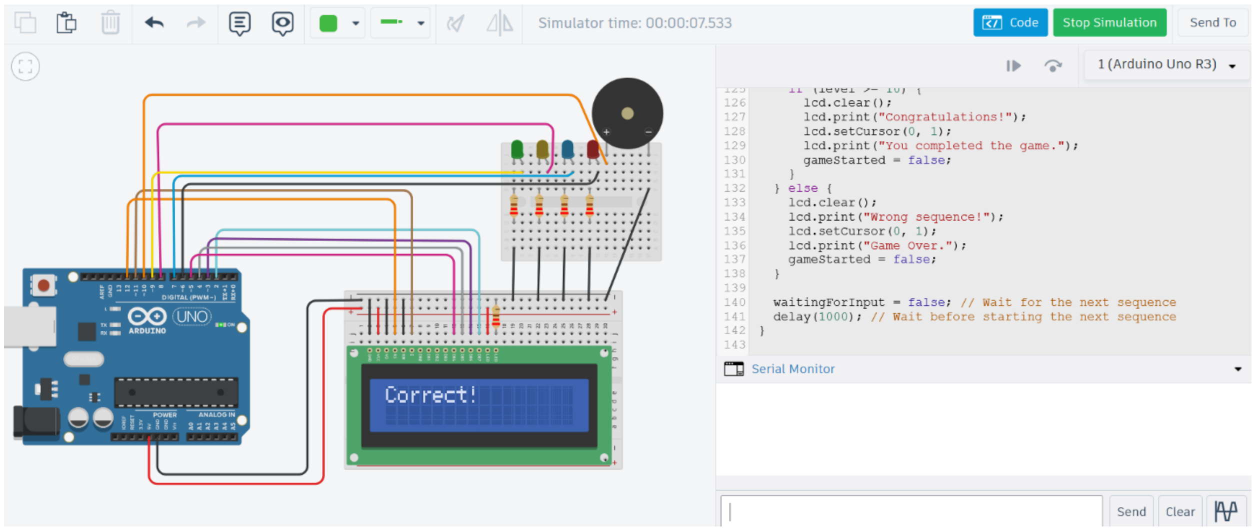


The circuit at the start of the game:



```
#include <LiquidCrystal.h>

// Define the LCD pin connections
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Pins E, RS, D4, D5, D6, 

// Define the LED pins
const int ledPins[] = {9, 8, 7, 6}; // Pins for LEDs
const char ledChars[] = {'q', 'w', 'e', 'r'}; // Characters associa
const int buzzerPin = 10; // Buzzer pin

// Musical Notes (frequencies in Hz)
const int notes[] = {262, 294, 330, 349}; // C4, D4, E4, F4

// Game Variables
int sequence[10]; // Game sequence
int playerSequence[10]; // Player's sequence
int level = 0;
bool gameStarted = false;
bool waitingForInput = false; // To control the player input state
```

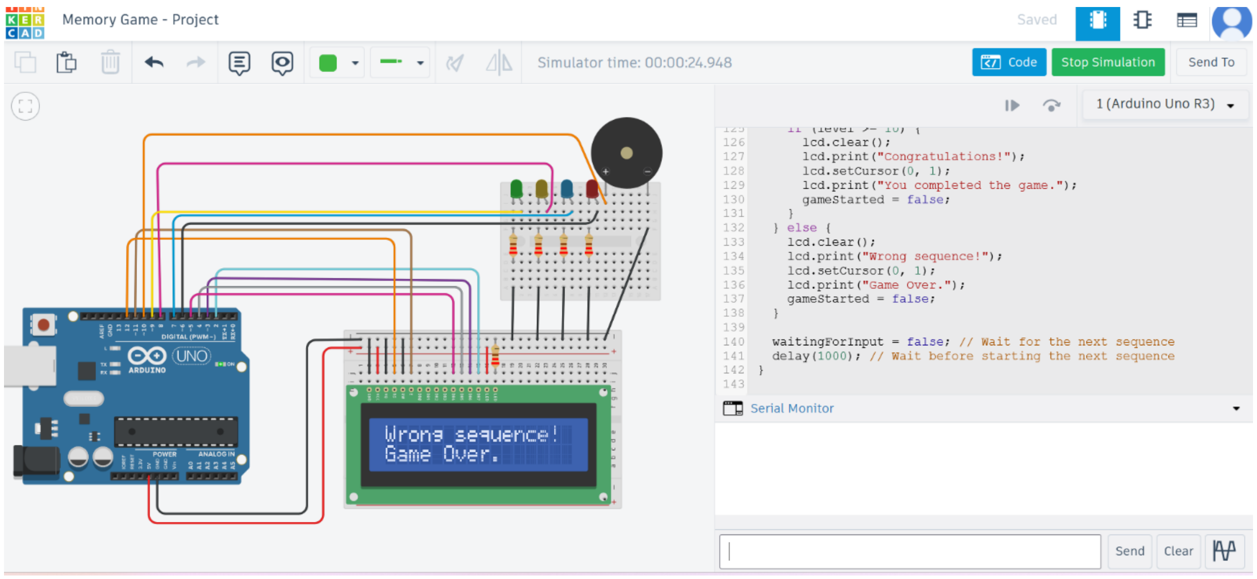
The circuit when the player gets the sequence right:



```
Correct!
```

```
    lcd.clear();
    lcd.print("Congratulations!");
    lcd.setCursor(0, 1);
    lcd.print("You completed the game.");
    gameStarted = false;
  }
} else {
  lcd.clear();
  lcd.print("Wrong sequence!");
  lcd.setCursor(0, 1);
  lcd.print("Game Over.");
  gameStarted = false;
}

waitingForInput = false; // Wait for the next sequence
delay(1000); // Wait before starting the next sequence
}
```

The circuit when the player fails to follow the sequence:



```
Wrong sequence!
Game Over.
```

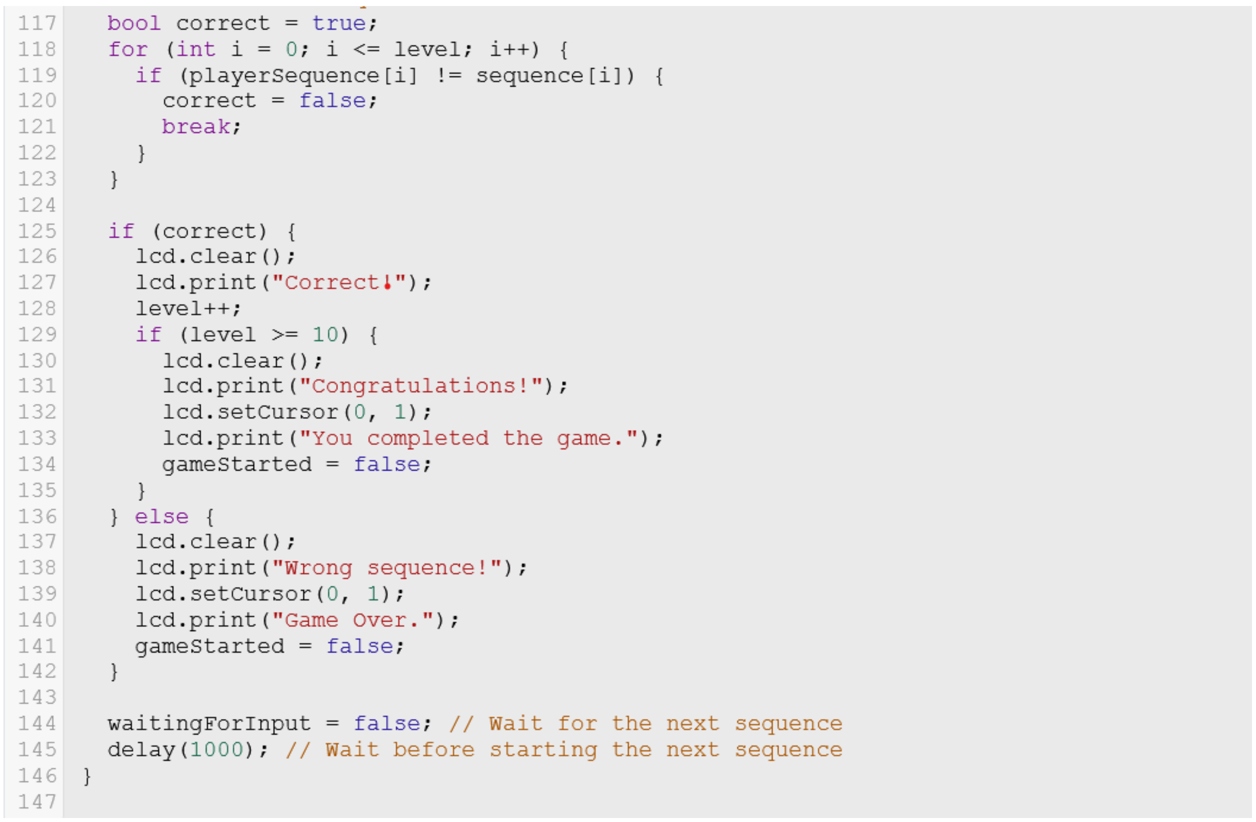
```
    lcd.clear();
    lcd.print("Congratulations!");
    lcd.setCursor(0, 1);
    lcd.print("You completed the game.");
    gameStarted = false;
  }
} else {
  lcd.clear();
  lcd.print("Wrong sequence!");
  lcd.setCursor(0, 1);
  lcd.print("Game Over.");
  gameStarted = false;
}

waitingForInput = false; // Wait for the next sequence
delay(1000); // Wait before starting the next sequence
}
```

The circuit code:

```
#include <LiquidCrystal.h>

// Student name: Davi Soares Mendes - Student number: 200613140
// Student name: Erandi Canafistula Araujo - Student number: 200597545


// Define the LCD pin connections
LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Pins E, RS, D4, D5, D6, D7

// Define the LED pins
const int ledPins[] = {9, 8, 7, 6}; // Pins for LEDs
const char ledChars[] = {'q', 'w', 'e', 'r'}; // Characters associated with LEDs
const int buzzerPin = 10; // Buzzer pin

// Musical Notes (frequencies in Hz)
const int notes[] = {262, 294, 330, 349}; // C4, D4, E4, F4

// Game Variables
int sequence[10]; // Game sequence
int playerSequence[10]; // Player's sequence
int level = 0;
bool gameStarted = false;
bool waitingForInput = false; // To control the player input state

void setup() {
  // Initialize LCD
  lcd.begin(16, 2); // 16 columns and 2 rows
  lcd.print("Press any key");
  lcd.setCursor(0, 1);
```

```
  lcd.print("to start game");

  // Set up LEDs as outputs
  for (int i = 0; i < 4; i++) {
    pinMode(ledPins[i], OUTPUT);
  }

  pinMode(buzzerPin, OUTPUT); // Buzzer as output

  Serial.begin(9600);
  randomSeed(analogRead(A0)); // Initialize random seed
}

void loop() {
  if (!gameStarted) {
    if (Serial.available()) {
      Serial.read(); // Clear the buffer
      startGame();
      gameStarted = true;
      lcd.clear();
      lcd.print("Game Started!");
      delay(1000); // Short delay to show message
    }
    return;
  }

  if (!waitingForInput) {
    showSequence();
    waitingForInput = true;
  } else {
    playerInput();
  }
}

void startGame() {
  level = 0;
  for (int i = 0; i < 10; i++) {
    sequence[i] = random(4); // Generate a random sequence of LEDs
  }
  lcd.clear();
  lcd.print("Follow the sequence");
}

void showSequence() {
  for (int i = 0; i <= level; i++) {
    int ledIndex = sequence[i];
    digitalWrite(ledPins[ledIndex], HIGH);
    tone(buzzerPin, notes[ledIndex], 500); // Play corresponding note
    delay(500);
    digitalWrite(ledPins[ledIndex], LOW);
    noTone(buzzerPin);
    delay(200);
  }
  lcd.clear();
  lcd.print("Your turn! Enter:");
  lcd.setCursor(0, 1);
  lcd.print("q, w, e, r");
}
```

```

void playerInput() {
  int inputCount = 0;

  while (inputCount <= level) {
    if (Serial.available()) {
      char input = Serial.read(); // Read the entered character

      // Check if the character is valid and corresponds to an LED
      for (int j = 0; j < 4; j++) {
        if (input == ledChars[j]) {
          playerSequence[inputCount] = j;
          tone(buzzerPin, notes[j], 200); // Play corresponding note
          delay(200);
          noTone(buzzerPin);
          inputCount++;
          break;
        }
      }

      if (input == '\n' || input == '\r') { // Ignore new line or return characters
        continue;
      }

      if (inputCount > level) break; // If we have enough inputs
    }
  }

  // Check if the sequence is correct
```



```
  bool correct = true;
  for (int i = 0; i <= level; i++) {
    if (playerSequence[i] != sequence[i]) {
      correct = false;
      break;
    }
  }

  if (correct) {
    lcd.clear();
    lcd.print("Correct!");
    level++;
    if (level >= 10) {
      lcd.clear();
      lcd.print("Congratulations!");
      lcd.setCursor(0, 1);
      lcd.print("You completed the game.");
      gameStarted = false;
    }
  } else {
    lcd.clear();
    lcd.print("Wrong sequence!");
    lcd.setCursor(0, 1);
    lcd.print("Game Over.");
    gameStarted = false;
  }

  waitingForInput = false; // Wait for the next sequence
  delay(1000); // Wait before starting the next sequence
}
```