Arduino Project
For
Computer Systems Servicing

SIMON GAME DOCUMENTATION

Submitted to:

GLENN S. OLAER

Submitted by:

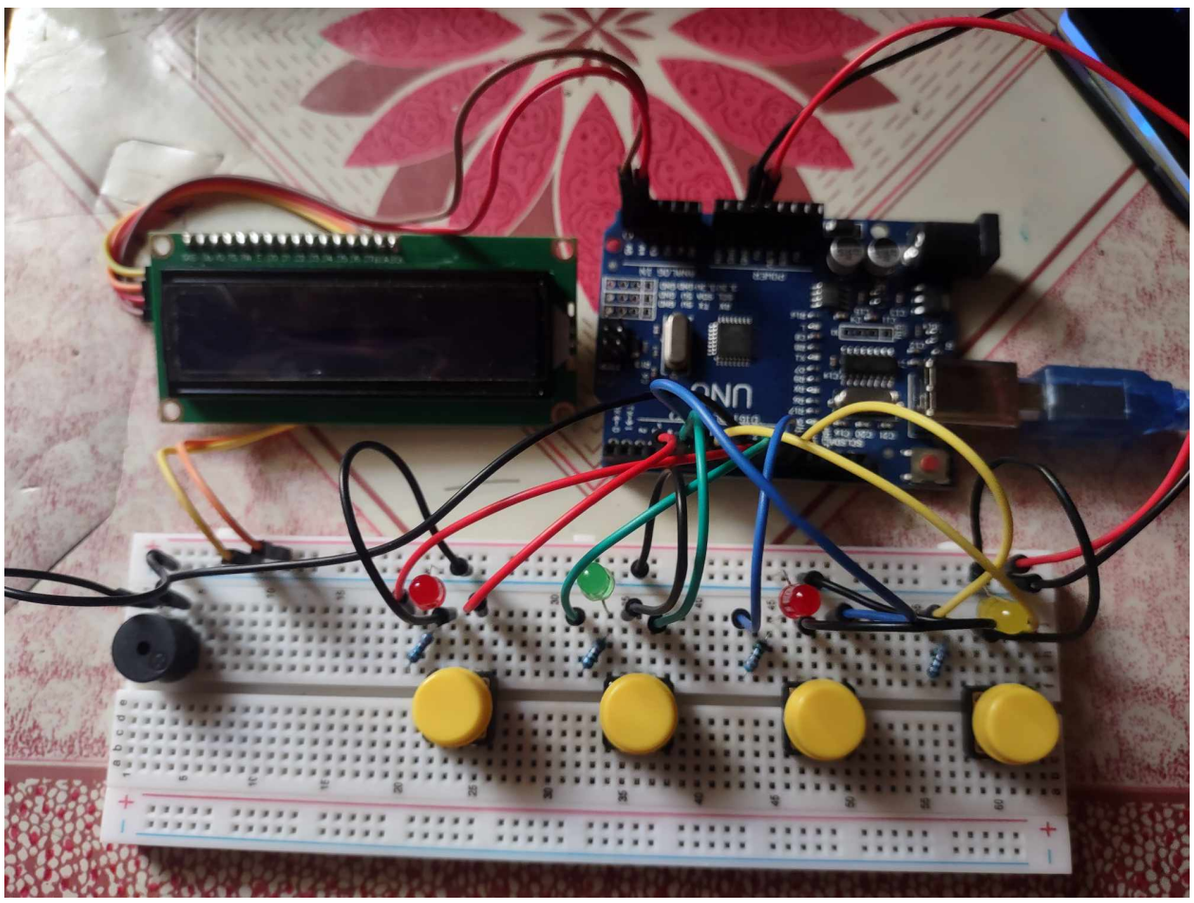
Mark James Felecio
Mary Grace Musico
Kris Ainy Tinaco
April Mae Balaba
John Ivan Baril
Joseph Karl Busano
Angie Manabit
Ivan Galicia
Joseph Gultia
Mark Dino

TABLE OF CONTENTS

Section I. Introduction

The Simon Game is a classic electronic game that tests memory and reflexes. Players must repeat a sequence of lights generated by the game. The game starts with a simple pattern and the sequence progressively increases as the player successfully completes each level. This continues until the player has made a mistake. The game over sound is played and the game will restart.

We created our Arduino version of the Simon Game as our final project for the Computer Systems Servicing (CSS) major. The parts contained in the Arduino kit we purchased were used to make it. An Arduino kit comprises several electronic components and modules that are assembled to enable learning, experimentation, and project development using Arduino microcontroller boards. Typically, it consists of an Arduino board, breadboards, LEDs, resistors, capacitors, transistors, diodes, buttons, sensors, actuators, and jumper wires to connect the various parts.



Simon Game

Section II. Hardware Overview

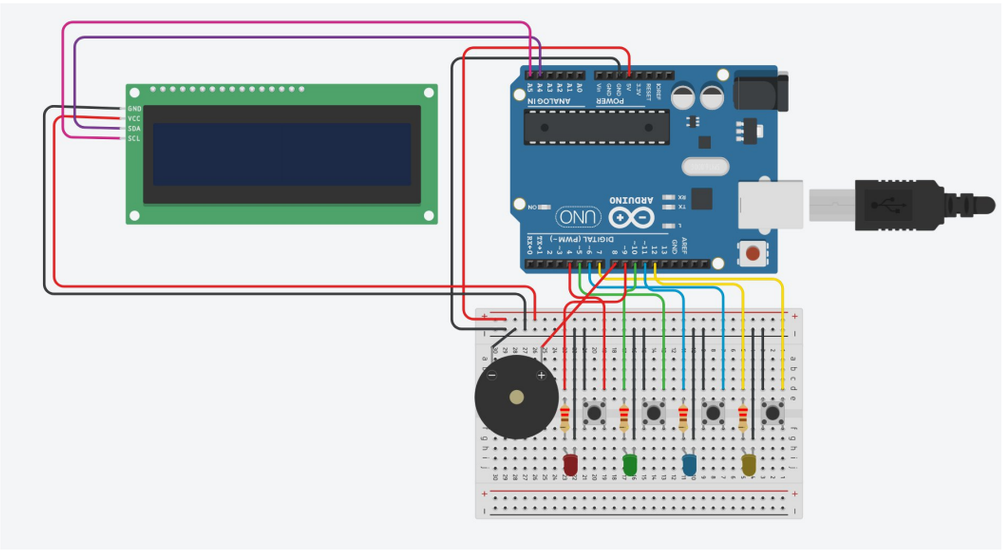
Components Required:

- Arduino Uno
- 4 LED lights (Red, Green, Blue, Yellow)
- 4 pushbuttons (matching the colors of the LEDs)
- LCD 16 x 2 (I2C)
- Buzzer or piezo speaker
- Breadboard
- Jumper wires
- 4 220Ω Resistors

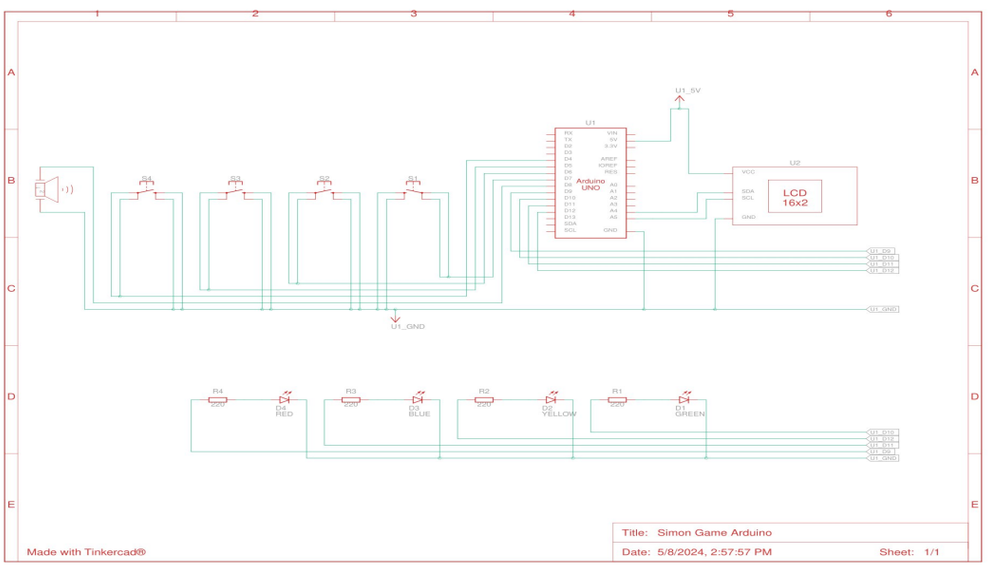
Libraries Required:

- LiquidCrystal_I2C Library

Schematic Diagram:



Wiring Diagram:



Section III. Setting Up

==Hardware==

Step 1: Gather all required components for the project.

Step 2: Place the 4 push buttons in the center of the breadboard.

Step 3: Align the 4 LEDs with their respective buttons. Connect the shorter pin of each LED to the ground row (-) on the breadboard. Connect the longer pin through a resistor and link it to the Arduino.

Step 4: Connect the left pin of each button to the ground row (-) on the breadboard. Connect the right pin of each button to the Arduino.

Step 5: Bridge the ground (-) lane on the breadboard to the GND pin on the Arduino. Connect the positive (+) lane to the 5V pin on the Arduino.

Step 6: Position the buzzer on the breadboard. Connect the positive pin (identified by the (+) symbol) to the Arduino, and link the negative pin to the ground row (-) on the breadboard.

Step 7: Utilize a male-female jumper wire to connect the female connector at the back of the LCD display (I2C). Attach the GND wire to the ground, VCC wire to the power (+), SDA wire to A4 on the Arduino, and SCL wire to A5 on the Arduino.

==Software==

Step 1: Download the Arduino IDE from their website. (https://www.arduino.cc/)

Step 2: Follow the instructions on the website on how to install it based on your operating system.

Step 3: Open the Arduino IDE.

Step 4: Go to this link and copy the code. (https://pastebin.com/EmLfhLic)

Step 5: Paste it to the code editor.

Step 6:. Download the LiquidCrystal i2c library from the website.
https://www.arduino.cc/reference/en/libraries/liquidcrystal-i2c/

Step 7:. Click on the [Sketch] tab at the top. After that, click [Include Library], then [ADD .ZIP Library]. It will open your file manager. Select the LiquidCrystal zip file to add the library to your code.

Step 8: Depending on the LCD (I2C) display you use. You may have to change the parameters in the code. The default address is 0x27 but addresses can be 0x20 or 0x3F. If you are using a bigger LCD, change the values on the parameters.

LiquidCrystal_I2C lcd(0x27, 16, 2);
LiquidCrystal_I2C lcd(address, LCD columns, LCD rows);

Step 9: Change the values on these variables based on the pin number that you have connected each component in your circuit.

```
const byte BUTTON_PIN[] = {4, 5, 6, 7};  // Red, Green, Blue, Yellow
const byte LED_PIN[] = {9, 10, 11, 12};
const byte BUZZER_PIN = 8;
```

You can also change these values:

```
const byte TONES[] = {262, 330, 392, 494};  // Sound each button makes
const byte MAX_ROUNDS = 10;   // Maximum rounds to win
```

Step 10: Connect your Arduino to your pc or laptop using the blue USB cable.

Step 11: Click the Arduino Uno dropdown *(Refer to the image)* and select the port where you have connected your arduino. If it isn't there, connect your Arduino to other ports or use another computer. If the problem persists, browse through the internet to find the solution.

Step 12: Click the the Check ✓ symbol to compile the code. If there are any errors, read it and fix the errors that it gave.

Step 13: If there are no errors click the arrow symbol to upload the code to the Arduino.

Section IV. The Code

```
// Import necessary libraries
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// List of frequencies for musical notes
const int MUSICAL_NOTE_FREQUENCIES[] = {
  31, 33, 35, 37, 39, 41, 44, 46, 49, 52, 55, 58, 62, 65, 69, 73, 78, 82, 87,
93, 98, 104, 110, 117, 123, 131, 139, 147, 156, 165, 175, 185, 196, 208, 220,
233, 247, 262, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494, 523, 554,
587, 622, 659, 698, 740, 784, 831, 880, 932, 988, 1047, 1109, 1175, 1245, 1319,
1397, 1480, 1568, 1661, 1760, 1865, 1976, 2093, 2217, 2349, 2489, 2637, 2794,
2960, 3136, 3322, 3520, 3729, 3951, 4186, 4435, 4699, 4978
};

// Initialize LCD object with address, columns, and rows
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Pins for buttons, LEDs, and buzzer
const byte BUTTON_PIN[] = {4, 5, 6, 7};  // Red, Green, Blue, Yellow
const byte LED_PIN[] = {9, 10, 11, 12};
const byte BUZZER_PIN = 8;
const int TONES[] = {262, 330, 392, 494};  // Sound each button makes
const byte MAX_ROUNDS = 10;   // Maximum rounds to win

// Game variables
int buttonSequence[MAX_ROUNDS]; // An array that stores the generated button
sequence for each round
byte pressedButton = 4;  // Default value for no button pressed
byte roundCounter = 0;
byte score = 0;
byte highScore = 0;
bool gameStarted = false;
bool firstRound = true;

// Initialize pins and libraries
void setup() {
  Wire.begin();   // Begin communication with I2C device
  lcd.init();
  lcd.backlight();    // Initialize LCD and turn on backlight

  // Set button and LED pins as inputs and outputs respectively
  for (byte i = 0; i < 4; i++) {
    pinMode(BUTTON_PIN[i], INPUT_PULLUP);
    pinMode(LED_PIN[i], OUTPUT);
  }
  // Set buzzer pin as output
  pinMode(BUZZER_PIN, OUTPUT);
```

```
  // Seed random number generator with analog pin A0
  // This ensures that the button sequence is unique each time the game runs
  randomSeed(analogRead(A0));

  showMenuScreen();
  delay(1000);
}

// Main game loop
void loop() {
  allLEDOff();

  // If game has not started, start the game
  if (!gameStarted) {
    startGame();
  }

  // Play the generated button sequence
  playButtonSequence();

  // Check if user input matches the generated sequence
  if (checkUserSequence()) {
    handleCorrectInput();
  } else {
    loseSequence();   // Executes losing sequence if a button is wrong
  }

  // Display current score
  displayScore();
}

// FUNCTIONS
void startGame() {
  playGameStartTone();
  displayScore();
  generateButtonSequence();
  roundCounter = 0;
  gameStarted = true;
}

void showMenuScreen() {
  allLEDOn();

  lcd.clear();
  lcd.setCursor(2, 0);
  lcd.print("Simon Game!");
  lcd.setCursor(0, 1);
  lcd.print("<- Press to play");
```

```
  waitForButtonPress();
}

// Display score on LCD
void displayScore() {
  lcd.clear();

  // Updates the highscore
  if (score > highScore) {
    highScore = score;
  }

  // Displays the current score and high score on the LCD
  lcd.setCursor(0, 0);
  lcd.print("High Score: " + String(highScore));
  lcd.setCursor(0, 1);
  lcd.print("Score: " + String(score));
}

// Generate random button sequence for the game
void generateButtonSequence() {
    for (byte i = 0; i < MAX_ROUNDS; i++) {
        buttonSequence[i] = random(0, 4);
    }
}

// Play the generated button sequence with LEDs and tones
void playButtonSequence() {
  for (byte i = 0; i < roundCounter; i++) {
    lightLEDandPlayTone(buttonSequence[i]);
    delay(150); // change this value to control how fast the LED will flash
    allLEDOff();
    delay(150);
  }
}

// Check if user input matches the generated sequence
bool checkUserSequence() {
  for (int i = 0; i < roundCounter; i++) {
    byte expectedButton = buttonSequence[i];
    byte actualButton;

    do {
      actualButton = readButtons(); // waits until the user has pressed a
button
    } while (actualButton == 4);

    lightLEDandPlayTone(actualButton);
```

```
    if (expectedButton != actualButton) {  // compares the pressed button and
the expected button
      return false;  // If any button is incorrect, return false
    }
    delay(300);
    allLEDOff();
  }
  return true;  // if all pressed buttons are correct, return true
}

// Light LED and play tone corresponding to the button pressed
void lightLEDandPlayTone(byte ledNumber) {
  digitalWrite(LED_PIN[ledNumber], HIGH);
  tone(BUZZER_PIN, TONES[ledNumber]);
}

// Turn off all LEDs and Buzzer
void allLEDOff() {
  for (byte i = 0; i < 4; i++) {
    digitalWrite(LED_PIN[i], LOW);
  }
  noTone(BUZZER_PIN);
}

// Turn on all LEDs
void allLEDOn() {
  for (byte i = 0; i < 4; i++) {
    digitalWrite(LED_PIN[i], HIGH);
  }
}

void playGameStartTone() {
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[72], 200);  // C6
  delay(250);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[76], 200);  // E6
  delay(250);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[79], 200);  // G6
  delay(250);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[84], 200);  // C7
  delay(250);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[72], 200);  // C6
  delay(250);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[76], 200);  // E6
  delay(250);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[79], 200);  // G6
  delay(250);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[84], 200);  // C7
  delay(250);
}
```

```
void playLevelUpTone() {
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[82], 100);  // E6
  delay(125);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[97], 100);  // G6
  delay(125);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[108], 100); // E7
  delay(125);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[84], 100);  // C7
  delay(125);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[86], 100);  // D7
  delay(125);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[99], 300);  // G7
  delay(375);
}

void playGameOverTone() {
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[61]);
  delay(300);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[59]);
  delay(300);
  tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[58]);
  delay(300);

  for (byte i = 0; i < 10; i++) {
    for (int pitch = -10; pitch <= 10; pitch++) {
      tone(BUZZER_PIN, MUSICAL_NOTE_FREQUENCIES[60] + pitch);
      delay(5);
    }
  }
  noTone(BUZZER_PIN);
  delay(500);
}

void playWinningTone() {
  // Melody notes for winning sound
  int melody[] = {
    784, 880, 988, 1174, 988, 784, // G5, A5, B5, D6, B5, G5
    698, 784, 880, 1046, 880, 698, // F5, G5, A5, C6, A5, F5
    587, 659, 784, 932, 784, 587, // D5, E5, G5, B5, G5, D5
    523, 587, 698, 880, 698, 523  // C5, D5, F5, A5, F5, C5
  };

  // Duration of each note
  int noteDurations[] = {
    8, 8, 8, 8, 8, 8, // 1/8 note
    8, 8, 8, 8, 8, 8, // 1/8 note
    8, 8, 8, 8, 8, 8, // 1/8 note
    8, 8, 8, 8, 8, 8  // 1/8 note
  };
```

```
  // Play the melody
  for (int i = 0; i < sizeof(melody) / sizeof(melody[0]); i++) {
    int noteDuration = 1000 / noteDurations[i];
    tone(BUZZER_PIN, melody[i], noteDuration);

    // Pause between notes
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);

    // Stop the tone playing
    noTone(BUZZER_PIN);
  }
}

// Read button inputs
byte readButtons() {
  for (byte i = 0; i < 4; i++) {
    if (digitalRead(BUTTON_PIN[i]) == LOW) {  // checks the button pins if
they have been pressed
      return i; // returns the index of the button pressed
    } // 0 = first button, 1 = second button, 2 = third button, 3 = fourth
button
  }
  return 4;  // No button pressed
}

// Halts the program until a button is pressed
void waitForButtonPress() {
  do {
    pressedButton = readButtons();
  } while (pressedButton == 4);
}

void handleCorrectInput() {
  score++; // Increase the score
  roundCounter++; // Move to the next round

  // Reset the score if it's the first round
  // P.S. This is a lazy fix for a bug that makes it so that the game only
plays on the second round
  if (firstRound) {
    score = 0;
    firstRound = false;
  } else {
    playLevelUpTone(); // Play level up tone for subsequent rounds
  }

  // Execute winning sequence if all rounds completed
  if (score == MAX_ROUNDS) {
```

```
    winSequence();
  }
}

// Executes the losing sequence
void loseSequence() {
  allLEDOn();

  // Displays game over text on LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("You Lose :(");
  lcd.setCursor(7, 1);
  lcd.print("Game Over");

  playGameOverTone();

  // Displays the current score and high score
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("High: " + String(highScore));
  lcd.setCursor(9, 0);
  lcd.print("You: " + String(score));
  lcd.setCursor(0, 1);
  lcd.print("<-Press to play again!");

  waitForButtonPress();

  // Reset the game
  score = 0;
  firstRound = true;
  gameStarted = false;
}

// Executes the wining sequence
void winSequence() {
  // Displays the congratulation text on LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Congratulations!");
  lcd.setCursor(4, 1);
  lcd.print("You Won!!");

  playWinningTone();
  showMenuScreen();

  // Reset the game
  score = 0;
  firstRound = true;
```

```
  gameStarted = false;
}
```

Section V. Troubleshooting

If you are encountering issues during the setup or gameplay of the Arduino Simon Game, take a look at these common problems and solutions to help you troubleshoot:

LCD Display Issues:

- If the LCD display is not showing any text or is displaying gibberish:
- Check the wiring connections between the LCD display and the Arduino board.
- Ensure that the I2C interface is properly connected.
- Verify that the LiquidCrystal_I2C library is correctly installed in the Arduino IDE.

Button and LED Functionality:

- If buttons or LEDs are not responding as expected:
- Double-check the wiring connections for the buttons and LEDs.
- Ensure that the pins are correctly mapped in the code.
- Test each component individually to identify any faulty connections or components.

Buzzer Sound Issues:

- If the buzzer is not producing any sound or producing inconsistent tones:
- Verify the wiring connections for the buzzer.
- Ensure that the buzzer pin is correctly configured in the code.
- Check the buzzer's polarity and connection to the breadboard.
- Adjust the tone frequencies in the code if necessary.

Code Compilation Errors:

- If compilation errors occur when uploading the code to the Arduino IDE:
- Review the code for syntax errors or missing library dependencies.
- Ensure that all required libraries are properly installed in the Arduino IDE.
- Check for any conflicts between library versions or outdated code references.

Game Logic Issues:

- If the game does not progress as expected or ends prematurely:
- Debug the game logic by adding serial print statements to track the flow of execution.
- Verify that the button sequence generation, user input validation, and score calculation functions are working correctly.
- Test different scenarios to identify potential bugs or edge cases.

Power Supply Problems:

- If the Arduino board resets unexpectedly or behaves erratically during gameplay:
- Ensure that the Arduino board is receiving a stable power supply.

- Check the USB cable connection or power source for any loose connections or voltage fluctuations.
- Consider using an external power supply or battery pack for consistent power delivery.

If you encounter problems beyond what is covered in this section, you can consult online forums, Arduino community groups, or reference materials specific to your hardware components.

Section VI. How to Play the Game

Once you have successfully set up the game and has uploaded the code into the Arduino, you only need to plug the Arduino into a PC/laptop or into a battery to keep it running.

As the program starts, you are greeted by the main menu. Press any button to start the game. The game starts with a single LED lighting up and producing a sound. The player must repeat the sequence by pressing the corresponding buttons. If the player successfully repeats the sequence, the game adds another random LED to the sequence and will proceed to the next round. If the player presses the wrong button, the game ends and the player's score is displayed. The player can restart the game by pressing any button. The player will win if they can successfully finish all 10 rounds. After this the congratulation message will be displayed and they can restart the game by pressing any button.

Alternatively, you can restart the whole program by pressing on the tiny red button on the Arduino itself. Doing so will restart everything, including the high score that you have set!

Section VII. Credits

I’d like to acknowledge these websites and resources that served as an inspiration and guide in making this game:

- https://www.tinkercad.com/
- https://goodarduinocode.com/projects/simon#pin-connections
- https://chat.openai.com/
- https://www.arduino.cc/
- https://docs.arduino.cc/
- https://www.youtube.com/
- https://www.youtube.com/watch?v=JnJIKX5J0Cc&list=PLwWF-ICTWmB7-b9bsE3UcQzz-7ipI5tbR

Have fun playing the game!