

Making an Arduino Memory Game

A Really Simple Cyber-Physical System

Zhiyang Ong

Department of Electrical and Computer Engineering
Dwight Look College of Engineering,
Texas A&M University
College Station, TX

February 21, 2018

- 1 Preamble
- 2 Expected Outcomes
- 3 Summary
- 4 Required Components and Software Applications/Services
- 5 Schematic for Hardware Implementation
- 6 Source Code for Software Implementation
- 7 Making and Playing the Memory Game

Table of Contents

- 1 Preamble
- 2 Expected Outcomes
- 3 Summary
- 4 Required Components and Software Applications/Services
- 5 Schematic for Hardware Implementation
- 6 Source Code for Software Implementation
- 7 Making and Playing the Memory Game

Acknowledgments

Dott. Francesco Stefanni, University of Verona

IEEE TAMU Officers

- Maria Theresia Tyas
- Millie Kriel
- Polina Golikova

Reference: Jerepondumie, “Make an Arduino Memory Game,” in *Hackster.io*, Hackster, Inc., San Francisco, CA, September 15, 2017. Available online from *Hackster.io* at: <https://www.hackster.io/Jerepondumie/make-an-arduino-memory-game-73f55e>; February 18, 2018 was the last accessed date.



Warnings!!!

“The code is rather buggy; use is at own risk.”

– Donald Chai

“Beware of bugs in the above code.

I have only proved it correct, not tried it.”

– Donald E. Knuth

Unlike the Fantastic and Fabulous Ms. Millie Kriel, I have yet to try out this tutorial.

Ice Breaker

Name

What suggestions do you have for freshmen who are deciding what they can do this summer?

What are some of your favorite study techniques for circuit analysis (ECEN 214) and logic design (ECEN 248)?

Table of Contents

- ① Preamble
- ② Expected Outcomes
- ③ Summary
- ④ Required Components and Software Applications/Services
- ⑤ Schematic for Hardware Implementation
- ⑥ Source Code for Software Implementation
- ⑦ Making and Playing the Memory Game

Expected Outcomes

- Learn to understand someone else's wiring diagram
- Understanding someone else's coding style
- Learn why microcontrollers are a useful tool

Table of Contents

- 1 Preamble
- 2 Expected Outcomes
- 3 Summary**
- 4 Required Components and Software Applications/Services
- 5 Schematic for Hardware Implementation
- 6 Source Code for Software Implementation
- 7 Making and Playing the Memory Game

Summary

- Cyber-physical system design
- Implement a memory game on the *Arduino* platform:
 - Implement simple analog circuit design on a breadboard
 - Connect breadboard to the *Arduino* platform
 - Program the microcontroller on the *Arduino* platform
- Play the memory game
- Repeat previous steps ad infinitum until you get bored.

Table of Contents

- 1 Preamble
- 2 Expected Outcomes
- 3 Summary
- 4 Required Components and Software Applications/Services
- 5 Schematic for Hardware Implementation
- 6 Source Code for Software Implementation
- 7 Making and Playing the Memory Game

Required Components and Software Applications/Services

- Software applications/services:
 - *Arduino* IDE

Table: Hardware components

<i>Arduino</i> UNO & <i>Genuino</i> UNO	1
<i>SparkFun</i> 330 ohm resistors	4
<i>SparkFun</i> 10k ohm resistors	4
<i>SparkFun</i> Assorted LEDs (4 LEDs of different colors)	4
<i>SparkFun</i> Mini speaker (Recommended, not required)	1
<i>SparkFun</i> breadboard (Full size)	1
Jumper wires (generic)	1

Table of Contents

- 1 Preamble
- 2 Expected Outcomes
- 3 Summary
- 4 Required Components and Software Applications/Services
- 5 Schematic for Hardware Implementation**
- 6 Source Code for Software Implementation
- 7 Making and Playing the Memory Game

Schematic for Hardware Implementation

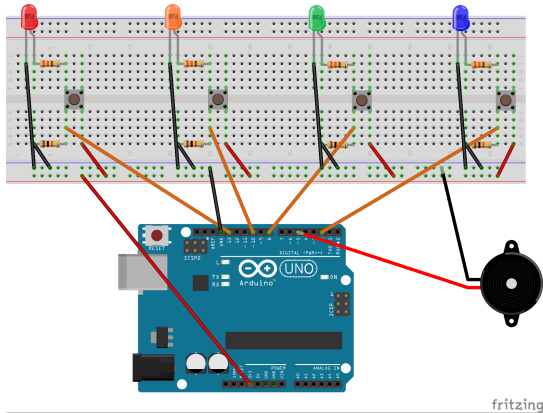


Figure: Schematic for Hardware Implementation on the Breadboard

Table of Contents

- 1 Preamble
- 2 Expected Outcomes
- 3 Summary
- 4 Required Components and Software Applications/Services
- 5 Schematic for Hardware Implementation
- 6 Source Code for Software Implementation**
- 7 Making and Playing the Memory Game

Source Code for Software Implementation

```

1  #define PLAYER_WAIT_TIME 2000 // The time allowed between button presses - 2s
2
3  byte sequence[100];           // Storage for the light sequence
4  byte curLen = 0;              // Current length of the sequence
5  byte inputCount = 0;          // The number of times that the player has pressed a (correct) button in a given turn
6  byte lastInput = 0;           // Last input from the player
7  byte expRd = 0;               // The LED that's suppose to be lit by the player
8  bool btnDwn = false;          // Used to check if a button is pressed
9  bool wait = false;            // Is the program waiting for the user to press a button
10 bool resetFlag = false;       // Used to indicate to the program that once the player lost
11
12 byte soundPin = 5;             // Speaker output
13
14 byte noPins = 4;               // Number of buttons/LEDs (While working on this, I was using only 2 LEDs)
15                               // You could make the game harder by adding an additional LED/button/resistors combination.
16 byte pins[] = {2, 13, 10, 8}; // Button input pins and LED output pins - change these values if you want to connect your buttons to other pins
17                               // The number of elements must match noPins below
18
19 long inputTime = 0;            // Timer variable for the delay between user inputs
20
21 void setup() {
22     delay(3000);                // This is to give me time to breathe after connection the arduino - can be removed if you want
23     Serial.begin(9600);         // Start Serial monitor. This can be removed too as long as you remove all references to Serial below
24     Reset();
25 }

```

Figure: Software to Set Up the Memory Game



Source Code for Software Implementation (2)

```
28  /// Sets all the pins as either INPUT or OUTPUT based on the value of 'dir'
29  ///
30  void setPinDirection(byte dir){
31      for(byte i = 0; i < noPins; i++){
32          pinMode(pins[i], dir);
33      }
34  }
35
36  //send the same value to all the LED pins
37  void writeAllPins(byte val){
38      for(byte i = 0; i < noPins; i++){
39          digitalWrite(pins[i], val);
40      }
41  }
```

Figure: Software Infrastructure for the Memory Game (2)

Source Code for Software Implementation (3)

```
43 //Makes a (very annoying :) beep sound
44 void beep(byte freq){
45     analogWrite(soundPin, 2);
46     delay(freq);
47     analogWrite(soundPin, 0);
48     delay(freq);
49 }
50
51 ///
52 /// Flashes all the LEDs together
53 /// freq is the blink speed - small number -> fast | big number -> slow
54 ///
55 void flash(short freq){
56     setPinDirection(OUTPUT); /// We're activating the LEDs now
57     for(int i = 0; i < 5; i++){
58         writeAllPins(HIGH);
59         beep(50);
60         delay(freq);
61         writeAllPins(LOW);
62         delay(freq);
63     }
64 }
```

Figure: Software Infrastructure for the Memory Game (3)

Source Code for Software Implementation (4)

```
67  ///This function resets all the game variables to their default values
68  ///
69  void Reset(){
70      flash(500);
71      curLen = 0;
72      inputCount = 0;
73      lastInput = 0;
74      expRd = 0;
75      btnDwn = false;
76      wait = false;
77      resetFlag = false;
78  }
79
80  ///
81  /// User lost
82  ///
83  void Lose(){
84      flash(50);
85  }
```

Figure: Software Infrastructure for the Memory Game (4)

Source Code for Software Implementation (5)

```
88  /// The arduino shows the user what must be memorized
89  /// Also called after losing to show you what you last sequence was
90  ///
91  void playSequence(){
92      //Loop through the stored sequence and light the appropriate LEDs in turn
93      for(int i = 0; i < curLen; i++){
94          Serial.print("Seq: ");
95          Serial.print(i);
96          Serial.print("Pin: ");
97          Serial.println(sequence[i]);
98          digitalWrite(sequence[i], HIGH);
99          delay(500);
100         digitalWrite(sequence[i], LOW);
101         delay(250);
102     }
103 }
```

Figure: Software Infrastructure for the Memory Game (5)



Source Code for Software Implementation (6)

```
106  /// The events that occur upon a loss
107  ///
108  void DoLoseProcess(){
109      Lose();           // Flash all the LEDS quickly (see Lose function)
110      delay(1000);
111      playSequence();   // Shows the user the last sequence - So you can count remember your best score - Mine's 22 by the way :)
112      delay(1000);
113      Reset();         // Reset everything for a new game
114  }
```

Figure: Software Infrastructure for the Memory Game (6)

Table of Contents

- 1 Preamble
- 2 Expected Outcomes
- 3 Summary
- 4 Required Components and Software Applications/Services
- 5 Schematic for Hardware Implementation
- 6 Source Code for Software Implementation
- 7 Making and Playing the Memory Game**

Making the Memory Game

- Implement the schematic on the breadboard
- Connect breadboard to the *Arduino* platform
- Software implementation
 - Copy source code from <https://www.hackster.io/Jerepondumie/make-an-arduino-memory-game-73f55e> to your *Arduino* IDE.
- Debug hardware implementation
 - Check if all LEDs flash as expected.
 - During initialization of the “Simple Simon” game, it takes 3 seconds to show “ 5 slow flashes of all the LEDs.”

How to Play the Memory Game

- Wait for the game to initialize
 - Takes 3 seconds to show “5 slow flashes of all the LEDs.”
- When a LED flashes, press the corresponding button for the flashed LED.
- You have 2 seconds to press each button.
- For a sequence of LED flashes, press their corresponding buttons in sequence.
- If all the LEDs flash rapidly, you have lost the game.
- When you lose the game,
 - The game will reset by showing “5 slow flashes of all the LEDs.”

Making the Memory Game More Challenging to Play

- Reduce the allowed reaction time for the player to press a button (for the corresponding LED that flashed).
- Add additional LEDs to the circuit to increase the sequence of flashing colored LEDs; for each additional LED:
 - Add a vertical stage in the circuit design.
 - Add a vertical stage on the breadboard.
 - Modify the array representing the input pins of the LEDs.