Noah Anderson, John Fernow
Dr. Kretchmar
7 May 2018

CS281: Introduction to Computer Systems
Final Project – *Simon*

Overview:

This circuit simulates the functionality of the 1978 electronic game named *Simon* (with some minor differences). The original game had four buttons that were illuminated from the bottom using LEDs. Originally, the game functioned by lighting up the LEDs in a random pattern, and the player has to enter the same pattern back by pressing the LED-lit buttons. If the user succeeds the pattern, they receive a new more complex pattern that builds upon the last one; otherwise, the game restarts from the beginning. In these basic respects, the circuit we have created is identical except in the manner that our pattern is random each time and that the LEDs do not double as buttons, but instead are separate.

Materials:

- (8) 330k resistors
- (4) LED's
- (4) Push Buttons
- (1) Breadboard
- (1) Piezo Buzzer
- (1) Arduino
- (1) 9v battery (with Arduino adapter)

- Numerous male-to-male wires

Before the Lab:

After writing the proposal, hardware-wise, we knew that we would need four push buttons and four LEDs. We also wanted each LED/button combination to have a unique sound when played or pressed. Beyond that, we knew we wanted to play different sound effects, either happy or sad sounds, for when the user entered a correct or incorrect pattern. As such, we decided to include a piezo buzzer in our design.

Software-wise, we created a level system. Within the system, players start with a basic pattern of four LEDs with corresponding tones. If they get the pattern correct, they progress to the next level where the pattern is one interval longer. This continues for patterns as high as 6,000 notes long (probably humanly impossible). If at any point they enter an incorrect pattern, they are forced back to the beginning. Each time the levels are different, as they are pseudo-randomly-generated. This prevents a player from memorizing the patterns in advance.

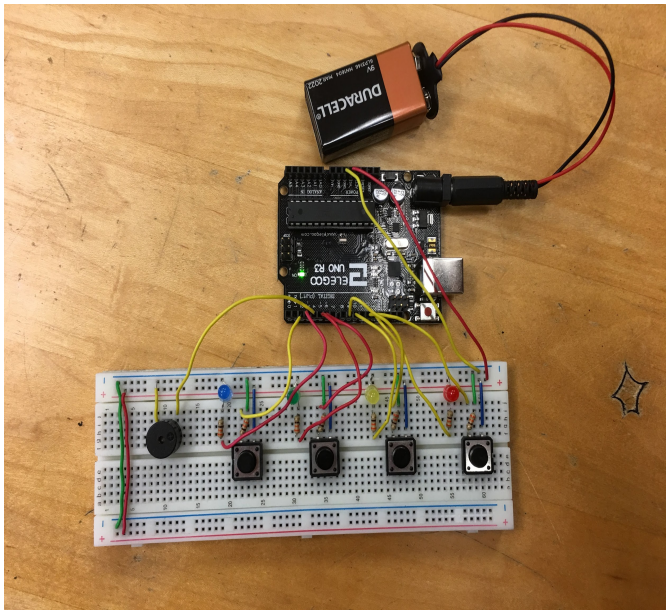
We debated how exactly difficulty should increase from one level to the next. At first, we believed increasing both pattern length and speed the notes would be a good way, but in testing, increasing frequency proved difficult enough.

Building Simon:

While planning out our game for the proposal, we thought of many "cases" that we would need to structure our build around. However, without actually building the circuits, it was

hard to catch all of the cases. Because of this, we ran into numerous problems when building our game. One problem we had was remembering which side of the LEDs were the side with the "long-leg" , as we cut them in order for to maximize the aesthetic of our breadboard. Additionally, we originally thought all of the data input ports on the Arduino were identical, so we gave no consideration to the number when hooking up the components. This later had to be changed as the Arduino data input ports are not all identical. After we got the hardware functioning, we trimmed all of the wires so it would clean up the overall design and maximize the playability of our game. In doing this, we made sure to leave enough slack so the cord would not accidentally pull out of the ports on the Arduino or breadboard.

When writing the software, we originally were writing as if it was any other C++ program. However, it turns out vectors are not included by default, and a simple "# include" statement does not work by itself. After a more careful read of the Arduino documentation, we discovered that Arduino has its own string class. Delving into the documentation even more, we later discovered that you cannot index using square brackets, rather, Arduino has its own syntax for indexing.



Picture of our Simon game

Hardware:

In order to achieve a clean design, we carefully planned out where each of the components were going to go. As you can see by the picture above, all of the buttons and LEDs are equally spaced. Making sure that our wires did not get in the way of the user's player-experience, we made sure to select wires from a wiring kit that were the exact length that we needed. In addition, our button placement was calculated to match the lengths of the wiring kit wires. Regarding the buzzer, we tried to place it as much out of the way as possible while still trying to account for uniformity of the level of our buttons. Lastly, we made sure to place the LEDs in a place where they would be visible and not get in the way of the user's playing experience.

Code:

```
void loop(){
  /*
   * Main loop of program. If the pattern is ever incorrect,
   * it will break the while loop, thus repeating function
   * (starting over from the beginning.)
   * Program works by playing a pattern (and saving that pattern as
   * a string), then saves user input, then compares if the user input
   * was the correct pattern, and then progresses them to the next level
   * or starts them over from the beginning.
   */
  bool patternCorrect = 1;
  int level = 1;
  while (patternCorrect == 1){
    // clear previous pattern and input
    String correctPattern = "";
    String userPattern = "";

    // play new pattern and gather input
    correctPattern = playPattern(level);
    userPattern = readInput(level);

    // verify input and proceed to next level
    level = checkInput(correctPattern, userPattern, level);
    if (!level)
      patternCorrect = 0;
  }
}
```

The main function of this program is called loop(). Loop calls all of the functions after setup() has been ran. It begins by calling playPattern(), which will generate a random pattern with a length based off the level number. Next, it plays this pattern by lighting up the LEDs and playing the corresponding sounds for each individual LED. After, playPattern() returns the generated pattern so it can be compared to the user's input later.

After playPattern() finishes, loop() then calls readInput(), which allows the user to enter a pattern by pressing the buttons. The corresponding LEDs and sounds will play when the buttons are pressed. The function stops taking input when the user has entered a pattern of the correct length. So if the pattern previously generated was of length six, the program

waits until the player has tapped a button six times. After they have pressed a sufficient number of buttons, the function will save what they entered and return it as a string.

Following the completion of readInput(), checkInput() will see if what the user entered is indeed the pattern that was played. If it was, the level will increase (and the pattern will become more difficult). In addition, a happy tune will play. If not, the variable level will be set to zero, which causes the loop to end and restart, putting the player back at level one.

If we had more time:

Given more time, we would have included a main-menu where different songs played. We also would have included lower, background music while playing the game. This would have required use of an SD card, as the Arduino does not have sufficient memory. Unfortunately, we found out that we ran out of ports for the SD enclosure to attach to the Arduino, so we would have needed to use a second Arduino and connect them together. The actual connection is not difficult, but it would have required quite a bit of restructuring the program we had written, as it requires specifying the master Arduino and the slave Arduino.

In addition to those improvements, we were considering 3D printing a case, but we needed somewhat precise measurements of the wires in order to do so. Additionally, without the wires being soldered to the Arduino, they come out somewhat easily, and have a case would obstruct us from reconnecting the wires.

It would have been nice if we could have designed the game in such a way the device could

be held in your hands. This likely would have required the Arduino to be attached to the back of the small breadboard and have the wires wrap around, as well as a case for it. Also, it would have been nice to have a seven-segment LED to keep track of what level the player is on, as well as the record high score. Overall, the game is easy to reset, as entering an incorrect pattern will reset it, but a dedicated reset button would have been a nice touch.

Conclusion:

Through the design of this project, we gained a deeper understanding of how various Arduino components function, as well as how to programatically tackle various issues, such as what to do if the player is pressing more than one button at the same time. We also learned what each of the different digital input ports on the Arduino can and cannot be used for. From a more generalized perspective, we learned that planning is everything. Not only is hardware design and placement essential, but so is brainstorming ideas for code algorithms.