

”Twisted” SIMON

Arduino Ignition Grant

Zaza Soriano
Brian Taylor

May 4, 2014

Contents

1	Goals	1
2	References	1
3	Hardware	1
4	Discussion	1
4.1	SIMON	1
4.2	Wiring the Arduino	2
4.3	Making a Push Sensor	2
5	Procedure	3
5.1	"Not so twisted" SIMON	3
5.2	"Twisted" SIMON	3
5.3	Future SIMON	5
A		
	Arduino Code	5

Listings

5.1	Twisted SIMON Sensors	3
5.2	Twisted SIMON Debug	3
A.1	Twisted SIMON Code	5

1 Goals

To learn about various sensors and how they interact with the Arduino by making a game based off of the classic game of SIMON.

2 References

<http://www.instructables.com/id/Arduino-Simon-Says>

<http://www.instructables.com/id/Stickytape-Sensors/step5/Velostat>

3 Hardware

- Arduino Board
- USB Cable
- Breadboard
- Wires
- 5 x Resistor 220Ω
- 4 x LED
- 4 x Button
- Speaker
- Rotary Knob (Potentiometer)
- Light Sensor (Photocell)
- Motion Sensor (Tilt Switch)
- Temperature Sensor
- Conductive Fabric
- Conductive Thread
- Velostat
- Tape
- Scissors

4 Discussion

4.1 SIMON

"Think fast... SIMON, 'Chase my flashing lights and sounds'!"

The classic SIMON is a computer-controlled game that consists of a base unit with 4 color

lenses and a control panel. The challenge is to repeat the ever-increasing random signals that SIMON generates. If the pattern you respond is correct SIMON will add another light to the pattern.

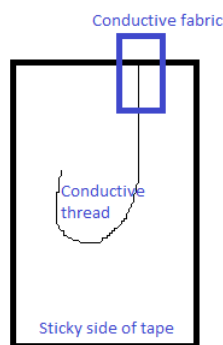
"Twisted" Simon is a game that is controlled by an Arduino. It will have 4 LEDs each with it's own way to activate it. The LEDs could be activated by the press of a button or a flashlight, just to name a few examples (as the options are endless). The Arduino will start by lighting up a random LED, at witch point the Player will activate the appropriate LED in response (hint: the correct LED is the one the Arduino lit up). If correct, the Arduino will add a random LED to the sequence (which is now 2 long) and the process repeats itself. If the Player enters the wrong sequence, the game is lost and the Player must start over.

4.2 Wiring the Arduino

4.3 Making a Push Sensor

1. Gather your materials
 - (a) 3 pieces of velostat (same size)
 - (b) 2 pieces of tape (slightly larger than the velostat)
 - (c) 2 strands of conductive thread (about the length of the velostat)
 - (d) 2 pieces of conductive fabric (enough for the alligator clip to work with)
2. Place a conductive thread and fabric onto the sticky side of a piece of tape, show in Figure 4.1

Figure 4.1: Push sensor insides



3. Repeat previous step
4. Sandwich velostat between tape making sure to overlap the conductive thread

5 Procedure

5.1 "Not so twisted" SIMON

1. Wire the Arduino with 4 LED's, 4 push buttons, and 1 speaker (See Section 4.2)
2. Load the Twisted_Simon_Says.ino file onto the Arduino (See Appendix A.1)

5.2 "Twisted" SIMON

1. Replace one of the push buttons with an analog sensor
2. Change the appropriate sensor information in the code, as an example look at line 21 in Listing 5.1.

Listing 5.1: Twisted SIMON Sensors

```
7  /*
   * IO: If all inputs are digital then this is sufficient. The format is
9  * { inputPort, outputPort, activeTone, threshold }. The output pins should be }
   *   ↳ digital,
   * the input pins can be analog or digital. Analog pins should be written as A#.
11 * The activeTone values can be found in the pitches.h file. threshold should be }
   *   ↳ the value
   * you want the sensor to be below to activate the LED. It will be 0 for digital }
   *   ↳ inputs.
13 */
   int P1[] = {
15     2, 2, NOTE_C4, 0};
   int P2[] = {
17     3, 3, NOTE_G3, 0};
   int P3[] = {
19     4, 4, NOTE_GS3, 0};
   int P4[] = {
21     A0, 5, NOTE_B3, 400};
```

To get correct threshold value, uncomment the debug lines (shown in Listing 5.2), run the program, and use the serial monitor to pick an appropriate threshold for the analog sensor.

Listing 5.2: Twisted SIMON Debug

```
49 int readPort(int n) {
   if(!IOports[n][3])
51     return !digitalRead(IOports[n][0]);
   else {
53     //Serial.println(analogRead(IOports[n][0])); //Uncomment this line to test }
   *   ↳ analog port
   return (analogRead(IOports[n][0]) < IOports[n][3]);
55 }
   }
57 *
   void setup() {
148 // put your setup code here, to run once:
   //Serial.begin(9600); //Uncomment this line to test analog port
150 randomSeed(analogRead(5));
   }
```

3. Load the modified Twisted_Simon_Says.ino file onto the Arduino

5.3 Future SIMON

A

Arduino Code

Listing A.1: Twisted SIMON Code

```
1 #include "Twisted_Simon_pitches.h"

3 #define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
   #define MAX_PATTERN 20
5 #define INITIAL_DELAY 1000

7 /*
   * IO: If all inputs are digital then this is sufficient. The format is
9   * { inputPort, outputPort, activeTone, threshold }. The output pins should be digital,
   * the input pins can be analog or digital. Analog pins should be written as A#.
11  * The activeTone values can be found in the pitches.h file. threshold should be the value
   * you want the sensor to be below to activate the LED. It will be 0 for digital inputs.
13 */
   int P1[] = {
15     2, 2, NOTE_C4, 0};
   int P2[] = {
17     3, 3, NOTE_G3, 0};
   int P3[] = {
19     4, 4, NOTE_GS3, 0};
   int P4[] = {
21     5, 5, NOTE_B3, 0};

23 int* IOports[] = {
   P1, P2, P3, P4};
26
   // port that outputs tone. Must be PWM capable
27 int tonePort = 6;

29 /*
   * PROGRAM STATE
31  * These global variables remember where we are in a pattern and if
   * we are reading or writing the pattern.
33 */
   enum {
35     INIT_PATTERN,
   SHOW_PATTERN,
37     READ_PATTERN
   };
40
   int state = INIT_PATTERN;
41 int pattern[MAX_PATTERN];
   int patternLength = 0;
43 int patternIndex = 0;
   int patternDelay = INITIAL_DELAY;
46
   /*
47  * Read from a virtual port.
   */
49 int readPort(int n) {
   if(!IOports[n][3])
51     return !digitalRead(IOports[n][0]);
   else {
53     //Serial.println(analogRead(IOports[n][0])); //Uncomment this line to test analog port
   return (analogRead(IOports[n][0]) < IOports[n][3]);
55 }
   }
58
```



```

/*
59  * Output the tone associated with a virtual port
*/
61 void writeTone(int n) {
    tone(tonePort, IOports[n][2]);
63 }

65 /*
    * Enable=1, Disable=0 a virtual port. Also outputs
67  * the tone associated with that port
*/
69 void writePort(int n, int v) {
    /* The reference design will short to ground if we ever write 1 to
71  * disable the light while the switch is depressed. Instead we put
    * the pin in high impedance (input) if OFF is requested
73  */
    if(v) {
75        pinMode(IOports[n][1], OUTPUT);
        digitalWrite(IOports[n][1], 0);
77        writeTone(n);
    }
79    else {
        pinMode(IOports[n][1], INPUT);
81    }
    }
84
/*
85  * Read the unique input available from any of the virtual ports.
    * Returns -1 if no input or if the input is not unique.
87  */
    int uniqueRead() {
89        int didRead = -1;
        int isUnique = 0;
91        for(int i = 0; i < ARRAY_SIZE(IOports); i++) {
            if(readPort(i)) {
93                if(didRead == -1) {
                    didRead = i;
                    isUnique = 1;
95                }
                else {
97                    isUnique = 0;
99                }
            }
101        }
        if(didRead != -1 && isUnique) {
103            return didRead;
        }
        else {
105            return -1;
107        }
    }
110
/*
111  * Enable only the given virtual port. All others are disabled.
*/
113 void uniqueWrite(int n) {
    for(int i = 0; i < ARRAY_SIZE(IOports); i++) {
115        writePort(i, i == n);
    }
117 }

119 /*
    * Disable all virtual ports and stop the tone.
121  */
    void clearOutput() {
123        uniqueWrite(-1);
        noTone(tonePort);
125    }

```

```

127 /*
    * Prepare all virtual ports for writing.
129 */
    void prepareWrite() {
131     for(int i = 0; i < ARRAY_SIZE(IOports); i++) {
        pinMode(IOports[i][0], INPUT);
133         pinMode(IOports[i][1], OUTPUT);
    }
135 }

137 /*
    * Prepare all virtual ports for reading.
139 */
    void prepareRead() {
141     for(int i = 0; i < ARRAY_SIZE(IOports); i++) {
        pinMode(IOports[i][0], INPUT);
143         pinMode(IOports[i][1], INPUT);
    }
145 }

147 void setup() {
    // put your setup code here, to run once:
149     //Serial.begin(9600); //Uncomment this line to test analog port
    randomSeed(analogRead(5));
151 }

153 /*
    void debugPrint() {
155     Serial.print(uniqueRead());
    Serial.print(" ");
158
    for(int i = 0; i < ARRAY_SIZE(IOports); i++) {
159         Serial.print(readPort(i));
    Serial.print(" ");
161     }

163     Serial.println(state);
    }
165 */

167 void loop() {
    //debugPrint();
169     if(state == INIT_PATTERN) {
        // create the next pattern. advance the length or the speed to
171         // make this pattern more challenging than the previous pattern
        if(patternLength > MAX_PATTERN) {
173             patternLength = 1;
            patternDelay = patternDelay/2;
175         }

177         pattern[patternLength] = random(ARRAY_SIZE(IOports));

179         patternLength = patternLength + 1;
        patternIndex = 0;
181         prepareWrite();
        clearOutput();
183         state = SHOW_PATTERN;
        delay(1000);
185     }
    else if(state == SHOW_PATTERN) {
187         // display the currently active pattern. If we've displayed it
        // completely then advance to the read state
189         if(patternIndex >= patternLength) {
            // clear display and audio
191             clearOutput();

193             patternIndex = 0;

```

```

195     state = READ_PATTERN;
196 }
197 else {
198     uniqueWrite(pattern[patternIndex]);
199     delay(patternDelay);
200
201     clearOutput();
202     delay(patternDelay / 2);
203     patternIndex++;
204 }
205 }
206 else if(state == READ_PATTERN) {
207     // read the currently active pattern. when the input matches the
208     // current token in the pattern we display it and then advance
209     // to the next token. When complete, move to INIT_PATTERN state
210     // if fail reset...
211     if(patternIndex == patternLength) {
212         state = INIT_PATTERN;
213         clearOutput();
214     }
215     else {
216         prepareRead();
217         int unique = uniqueRead();
218         if(unique == -1) {
219             clearOutput();
220         }
221         else {
222             prepareWrite();
223             uniqueWrite(unique);
224             if(unique == pattern[patternIndex]) {
225                 delay(patternDelay);
226                 patternIndex++;
227             }
228             else{ //wrong LED!
229                 writeTone(4);
230
231                 for(int n = 0; n < ARRAY_SIZE(IOports); n++){
232                     pinMode(IOports[n][1], OUTPUT);
233                     digitalWrite(IOports[n][1], 0);
234                 }
235
236                 delay(patternDelay);
237                 state = INIT_PATTERN;
238                 patternDelay = INITIAL_DELAY;
239                 patternLength = 0;
240             }
241         }
242     }
243 }
244 else {
245     // reset
246     patternLength = 0;
247     patternDelay = INITIAL_DELAY;
248     state = INIT_PATTERN;
249 }

```