Hand Trainer for Brain Injury Related Spasticity

B code	Name (first middle last)
B00690905	Joel Tyrone Gray

Background

The hand trainer is designed to be used as an aid in the health care industry for the rehabilitation of a person with muscle spasticity of the hand. It can be difficult for persons who have developed spasticity due to some form of brain injury, for example: violent impact victims and stroke victims, to stay motivated during their rehabilitation. This may be because the current rehabilitation exercises are not engaging, or that the patient is not seeing results fast enough. The hand trainer can be used to make rehab more engaging by having different levels to complete, also the data recorded from the training sessions will be collated after each session meaning the after just a few sessions the user can view their statistics and see their improvements if any.

Idea in brief

A series of buttons will be connected to the MKR1000 via a I/O expander chip, these buttons shall be arranged in a manner that sit under your fingers if you were to lay your hand naturally on a flat surface, and one button left and right of your thumb. This should cover the range of motion of all your digits.

Beside each button there will be an LED which will light up and indicate to the user which button they should press. The LEDs will light up in certain sequences, depending on the difficulty level that is selected, this is done using a rotary encoder to choose an option on the LCD. The user will follow the LEDs and press the corresponding buttons. Every time the user completes a session the time that it takes for them to complete a sequence will be sent via Wi-Fi to a web interface, where the data will be stored. This enables the health care professional/designated carer or the user themselves, to log on and compare their average session times from the start of their rehabilitation to their most recent session.

This will enable the user to see whether they have improved and to what degree. This should motivate the user to continue their rehabilitation if they are seeing tangible results, as opposed to simply gauging how well they are doing based on how they feel. The fact that the user must follow a sequence ensures that the user stays attentive and interested in the exercise.

Objectives

The main objective of this project was to create a useful tool or product that would solve a problem. Having seen the effects that muscle spasticity can have on the body and the monotony of standard physiotherapy exercises, I have come up with a way to improve the experience for the patient. I think this project has the potential to help many people improve the dexterity of their hands after a serious brain injury. The second objective was to create a functional embedded system that uses both inputs and outputs and relays this information over Wi-F. Using Push Buttons, a Rotary Encoder and capacitive sensor as inputs, and using LED's and an LCD screen as outputs, I think I have achieved this objective. Furthermore, I have incorporated a second chip the MCP23017 for extra IO pins.

If this project were to be used as the basis to create a start-up company, I think it would perform well in the market. There are other systems out there to help people with these issues, but many involve wearing an uncomfortable glove or a VR headset. These systems can be costly, whereas this hand trainer would be very cheap to produce, even in small quantities due to the simplicity of the design. The ability to market the product as a budget rehabilitation tool could attract more customers. The value of the product would come from the programme within and its speed tracking utility which I think would be a first in the market. Also, if I were able to select a different MCU, it could do the job of both the Arduino and the MCP23017 chip, thus saving money again.

Implementation

The development of this project was challenging at times and has taken multiple iterations to get to its current state. One of the main development decisions I had to make was whether to use tactile buttons or a matrix-based touchpad as the user input. Although a touchpad would have been a more elegant solution, I decided to use a series of push to make buttons. I made this decision as I feel that the users of the hand trainer would benefit more from the practice of exerting force while pressing on the button, rather than simply touching the touchpad. The inherent haptic feedback of a physical button may be a more familiar process for older users that may not be as accustomed to touchpads, additionally the colourful caps for the buttons may help engage small children.

I have wired the buttons in an active High orientation. This means that the MKR1000 will read a button press as "HIGH" or binary 1, I have also wired an LED between the negative side of the button and ground. This means that when I want to play a sequence on the LEDs I can simply set the respective I/O pin to HIGH, it also means however when the user is trying to follow the sequence and they press the button, it will connect VCC to Ground via the button and the LED, thus lighting the LED and again indicating that the button has been successfully pressed.

Incorporating the Rotary Encoder was a necessary element for the project, being able to detect clockwise and counter clockwise rotations as well as a button press made this a clear choice. It has been implemented to allow the user to navigate the menu displayed on the LCD screen. A rotary encoder is used to determine the angular position of a rotating shaft. Inside the encoder there are evenly spaced contact pads, that are connected to the common pin and two other contact pins A and B. As we turn the dial either A or B will contact the common pins and will both generate an output signal. If A comes in contact with the common pad first and then B comes in contact with the common pad, then we determine that the encoder has been rotated clockwise, and if B is to contact the common pads first and then followed by A we determine the dial to have been turned counter clockwise. Pin CLK is the output pulse of A and pin DT is the output pulse of B. The pulses A and B output will be out of phase by 90 degrees and we can infer that when A changes state if the logic level of the pulses do not match, then B != A, thus the dial was turned clockwise, and if they do match then B = A and the dial was turned counter clockwise. See Appendix 1.6 for graphical explanation. I decided that if the encoder was rotated clockwise that I would increment a counter, and if the current value of the counter was greater than it previous value, I would move the selected level on the LCD up by one, and conversely if the rotation was counter clockwise, I would decrement the counter and move the selected item on the LCD down by one. We can see this implemented in the "encoder" function in Appendix 2.2.

Before a sequence of LEDs can be played, the user is required to place their hand on a metal resting plate, which acts as a sensor to determine if the user if ready to begin. When the user moves their hand close to the sensor, the metal plate and their hand create a rudimentary capacitance. When the send pin changes state, it will change the state of the receive pin after some time, the delay between changes is defined by an RC time constant, R being the resistor value and C being any capacitance at the receive pin plus any capacitance present, i.e A hand on the sensor plate. The line *'Sensor.capacitiveSensor(30);'* will take a reading of 30 samples of how long it takes for the change in send pin to arrive at the receive pin, it should return a higher value if there is a higher capacitance or if there is a hand on the sensor. I've decided that a value higher than 2000 is returned from the function that there is definitely a hand present. The sequence of LEDs will then be played, and the level can begin.

Initially I had planned to use an Arduino Uno in conjunction with the MKR1000 communicating via Serial communications to ensure I had enough pins for all my peripherals. After some research I discovered the MCP23017 Input/Output (I/O) Expander, giving an additional 16 I/O pins with only two pins used to connect the chip to the MKR1000 via I2C. This enabled me to remove the Arduino Uno from the circuit and made the project look more polished. To use the MCP23017, I have tied the Reset pin to VCC, and pulled the 3 address pins to ground to set the I2C address to be 0. I have utilised the Wire.h library to facilitate the I2C communications from the chip to the MKR1000. I began by reading the datasheet for the IC and researching online how communications between these types of IC's normally work. I initially was able to communicate with the chip using the code in Appendix 3.1. Although it was educational to understand how accessing, reading, and writing to the chip's registers allowed me to control its operation. I decided it was more prudent to use a library in effort to keep the code as concise as possible. By interfacing the chips Interrupt pins with the Arduinos interrupt pin, I was able to trigger an interrupt on the MKR1000 via the MCP23017s I/O pins. Then calling the function *'mcp.getLastInterruptPin()'* I was able to determine which input triggered the interrupt, I could then check the results of *'mcp.getLastInterruptPin()'* with the expected button press, showing an X if incorrect and showing a Tick if correct.

In terms of displaying information to the user, I opted to go with a Thin Film Transistor Liquid Crystal Display (TFT LCD) Screen in colour. I thought this would look more modern than the common 16x2 LCD screens which do not provide colour displays and are less pixel dense. This screen communicates via SPI and is handled by including SPI.h library in the Arduino sketch. I originally purchased the LCD in an assorted package of components and unfortunately it was unbranded, with no markings or indication of the part number. After some research I discovered it was using a common ST7735 chip and I was able to find a library to assist me with incorporating it into my project. Using this library made it a lot easier to use the LCD however, it still took time to work out how to display shapes and designs, rather than just plain text. I was able after some time to create some animations of a Tick and 'X' and a rotating circular loading symbol, which I think really added professionalism and flair to the project.

The final aspect to implement was sending the data to my personal website via HTTPS. This was my first time manually creating and sending a HTTP(S) request. Initially I tried to send it via simple HTTP however as the website is HTTPS encrypted it would not accept a HTTP request. I was able to upload the WiFi101 Firmware Updater sketch and add my domain graycode.ie to the list of SSL domains in the Wi-Fi chips firmware. After this I was able to successful perform a 'GET' request from my website, to confirm that it was communicating correctly, I was then able to construct my 'POST' request and troubleshoot it until it worked, now that I knew the connection to the site was working. I created a PHP file that received and processed the data being posted to the website and saved this to a CSV file. Once I confirmed this was operating correctly, I then wrote a small PHP script on the webpage I wanted to display my data on. This script is able to read through the CSV file iteratively and print the values to the correct cells in a HTML table, this can be seen in Appendix 2.1. See below a flow chart of the entire system logic.

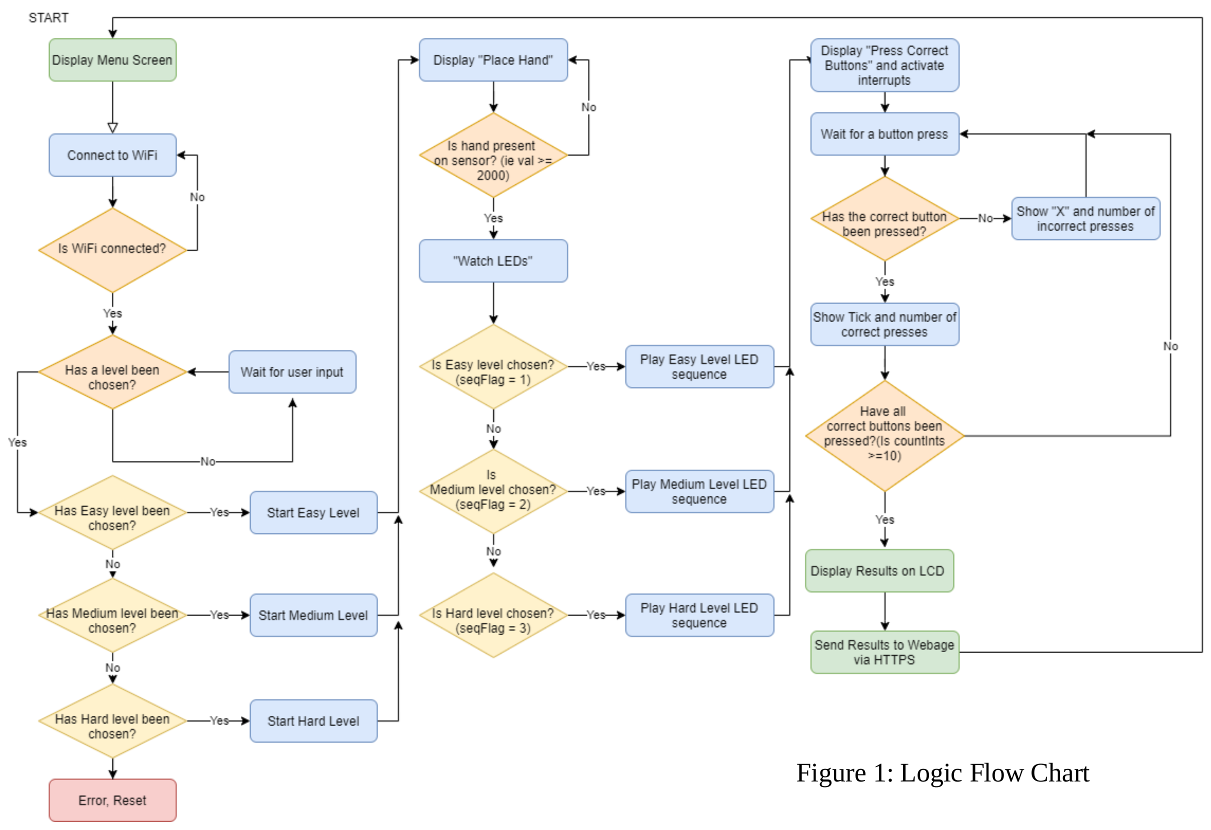

START
Display Menu Screen
Connect to WiFi
No
Is WiFi connected?
Yes
Has a level been chosen?
Wait for user input
No
Yes
Has Easy level been chosen?
Yes
Start Easy Level
No
Has Medium level been chosen?
Yes
Start Medium Level
No
Has Hard level been chosen?
Yes
Start Hard Level
Error, Reset
Display "Place Hand"
Is hand present on sensor? (ie val >= 2000)
No
Yes
"Watch LEDs"
Is Easy level chosen? (seqFlag = 1)
Yes
Play Easy Level LED sequence
No
Is Medium level chosen? (seqFlag = 2)
Yes
Play Medium Level LED sequence
No
Is Hard level chosen? (seqFlag = 3)
Yes
Play Hard Level LED sequence
Display "Press Correct Buttons" and activate interrupts
Wait for a button press
Has the correct button been pressed?
No
Show "X" and number of incorrect presses
Yes
Show Tick and number of correct presses
Have all correct buttons been pressed?(Is countInts >=10)
No
Yes
Display Results on LCD
Send Results to Webage via HTTPS

Figure 1: Logic Flow Chart

Block diagram of the system

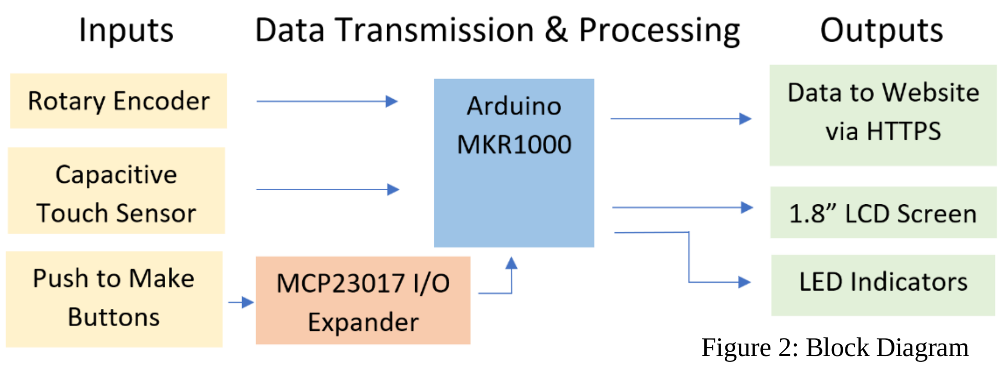


Figure 2: Block Diagram

The user is prompted by the LCD screen to select a level, they do this using the rotary encoder to highlight which level they desire which is displayed in the 1.8” LCD Screen. After this, the LCD screen prompts the user to place their hand on the metal sensor. Led indicators begin flashing in sequence depending on the selected Level. After each button press the MCP23017 triggers an interrupt on the MKR1000 and we then read the registers on the MCP23017 to determine which input button was pressed, this is checked against the expected button press, when all correct buttons are pressed the MKR1000 temporarily stores the data before pushing it to a web server via Wi-Fi. This completes a full usage case for the project. Beyond this the user can select another level or repeat the same one, and the same data transfer will complete.

Appendix 1.1

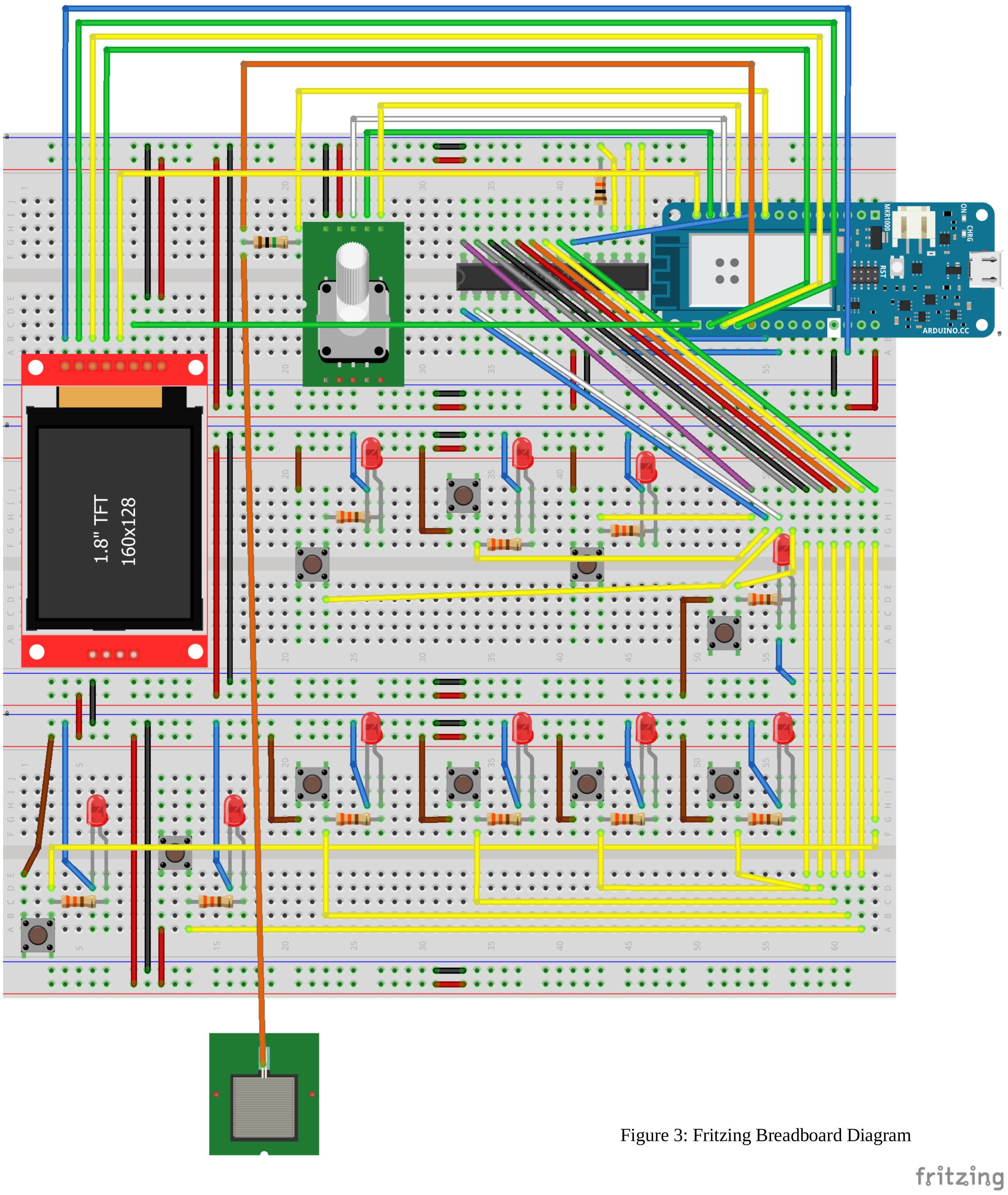


Figure 3: Fritzing Breadboard Diagram

Appendix 1.2

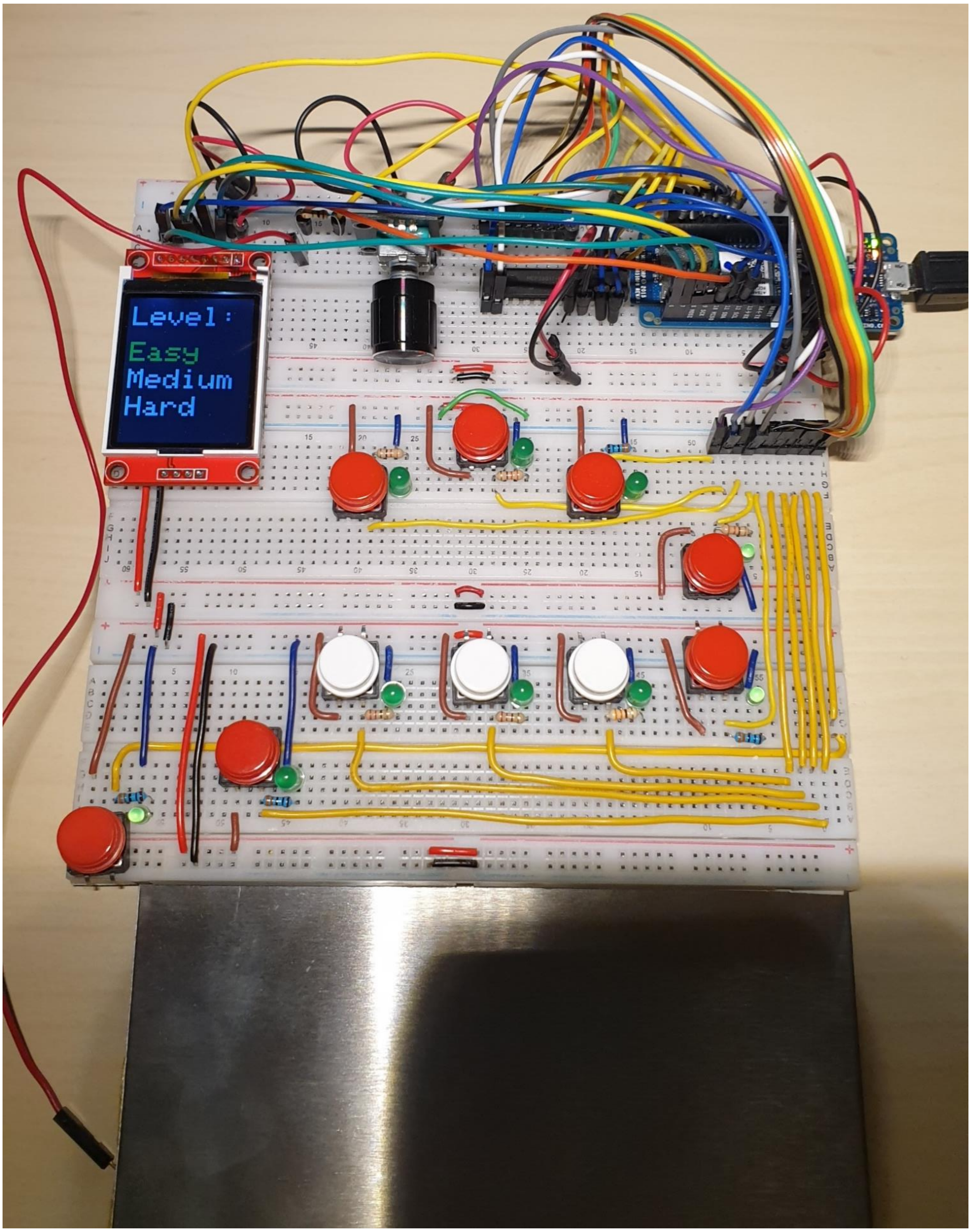


Figure 4: Photo of Entire circuit

Appendix 1.3

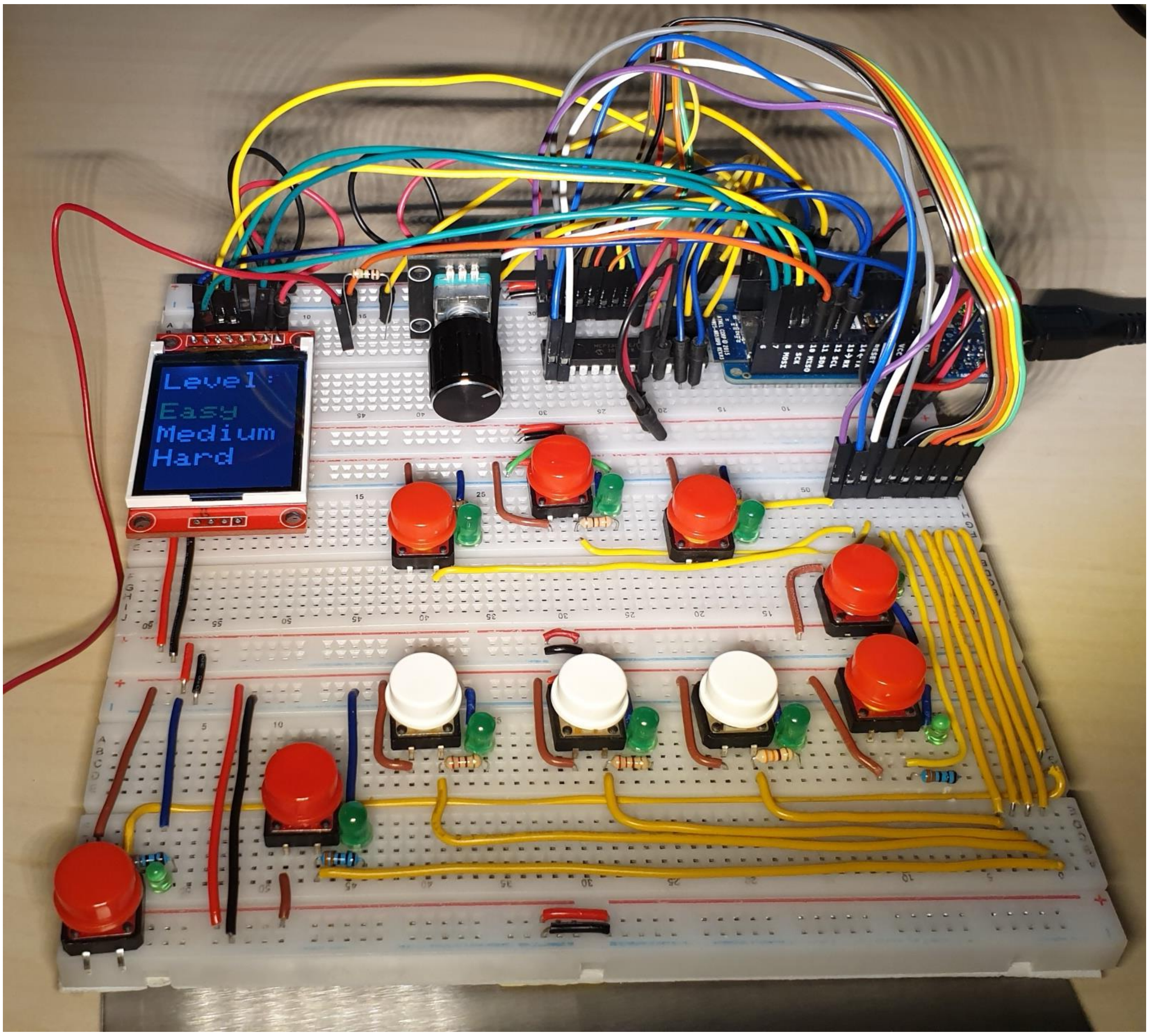
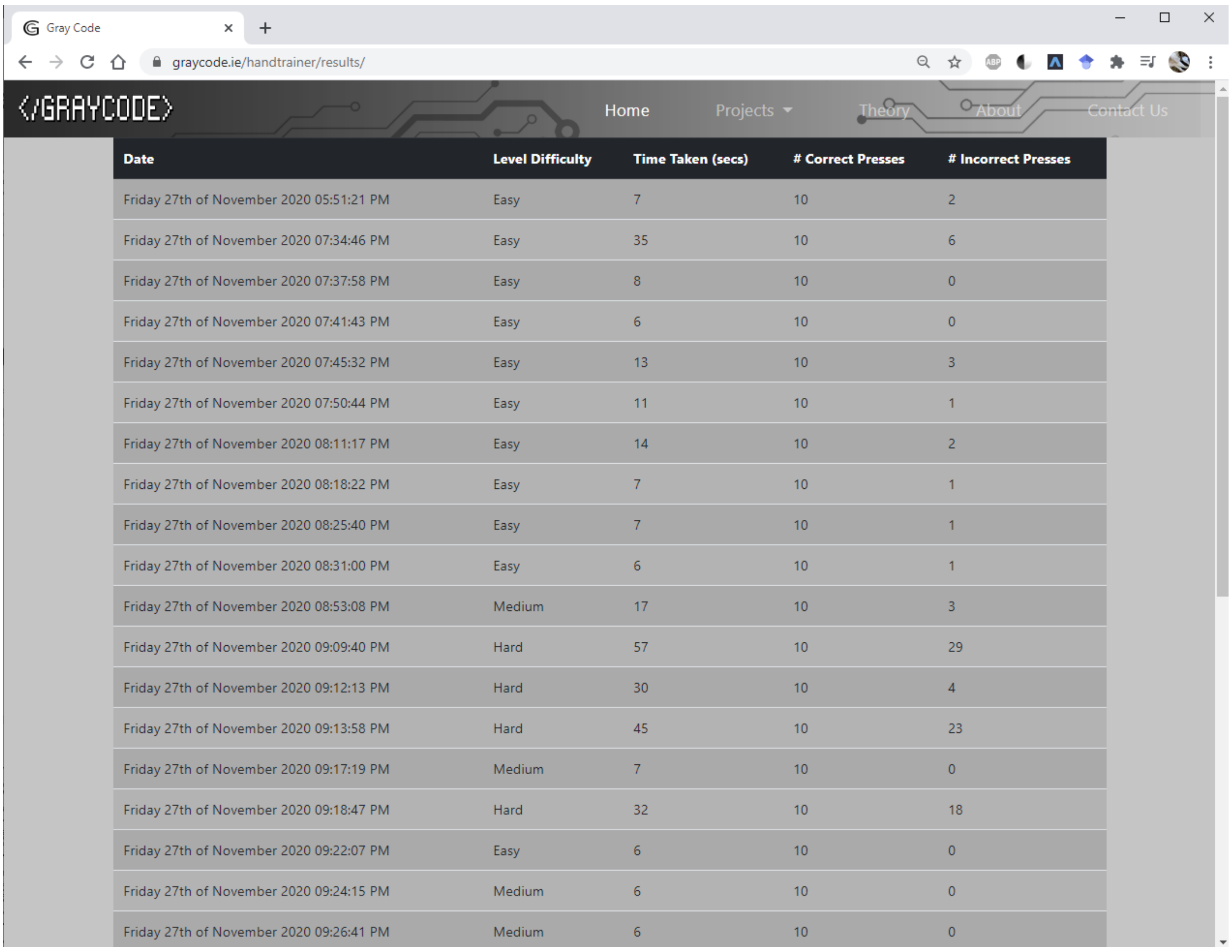


Figure 5: Up Close Photo of Circuit

Appendix 1.4



Date	Level Difficulty	Time Taken (secs)	# Correct Presses	# Incorrect Presses
Friday 27th of November 2020 05:51:21 PM	Easy	7	10	2
Friday 27th of November 2020 07:34:46 PM	Easy	35	10	6
Friday 27th of November 2020 07:37:58 PM	Easy	8	10	0
Friday 27th of November 2020 07:41:43 PM	Easy	6	10	0
Friday 27th of November 2020 07:45:32 PM	Easy	13	10	3
Friday 27th of November 2020 07:50:44 PM	Easy	11	10	1
Friday 27th of November 2020 08:11:17 PM	Easy	14	10	2
Friday 27th of November 2020 08:18:22 PM	Easy	7	10	1
Friday 27th of November 2020 08:25:40 PM	Easy	7	10	1
Friday 27th of November 2020 08:31:00 PM	Easy	6	10	1
Friday 27th of November 2020 08:53:08 PM	Medium	17	10	3
Friday 27th of November 2020 09:09:40 PM	Hard	57	10	29
Friday 27th of November 2020 09:12:13 PM	Hard	30	10	4
Friday 27th of November 2020 09:13:58 PM	Hard	45	10	23
Friday 27th of November 2020 09:17:19 PM	Medium	7	10	0
Friday 27th of November 2020 09:18:47 PM	Hard	32	10	18
Friday 27th of November 2020 09:22:07 PM	Easy	6	10	0
Friday 27th of November 2020 09:24:15 PM	Medium	6	10	0
Friday 27th of November 2020 09:26:41 PM	Medium	6	10	0

Figure 6: Screenshot of Results on Webpage

Appendix 1.5

Components List

Rotary Encoder (KY-040/HKT1062) with 15x16.5mm Cap

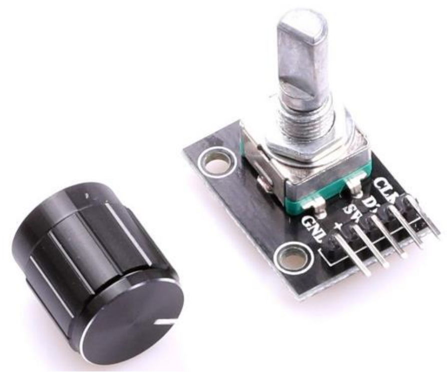


Figure 7

1.8” 128x160 Colour TFT Display SPI (ST7735)

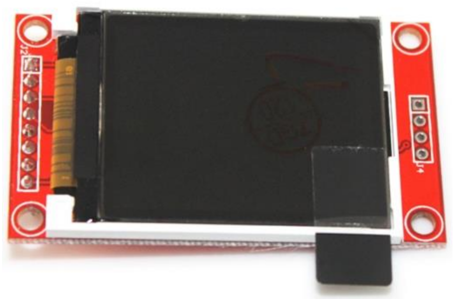


Figure 8

Push to Make Button Switch 12x12x7.3mm

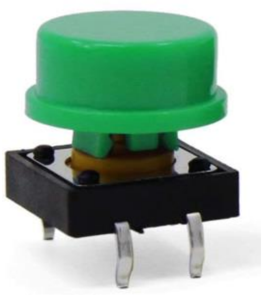


Figure 9

Arduino MKR1000

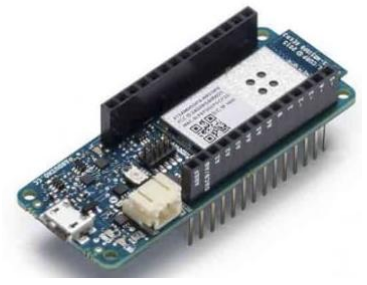


Figure 10

MCP23017 I/O Expander

Figure 11

Assorted Resistors (330Ω, 10k Ω, 1M Ω)

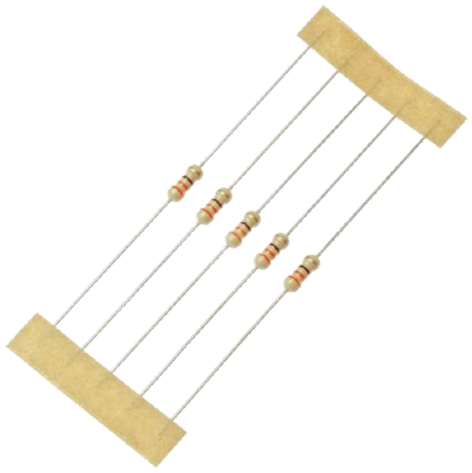


Figure 12

Green LEDs

Figure 13

Jumper Wires

Figure 14

Appendix 1.6

Understanding Rotary Encoder Decisions Based on Pulses

If B != A then the dial has turned Clockwise.

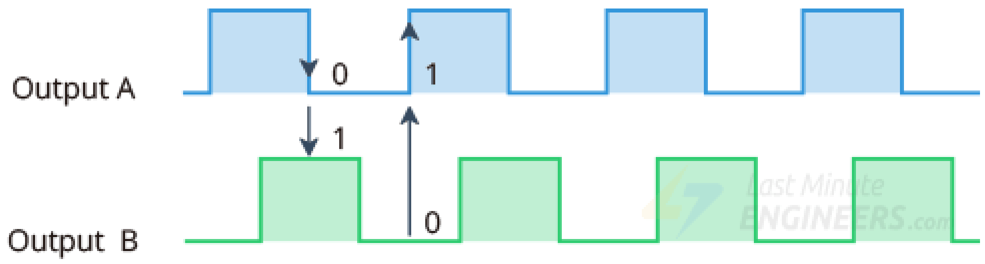


Figure 15: Rotary Encoder Clockwise Turn - https://lastminuteengineers.com/rotary-encoder-arduino-tutorial/

If B = A then the dial has turned Counter Clockwise

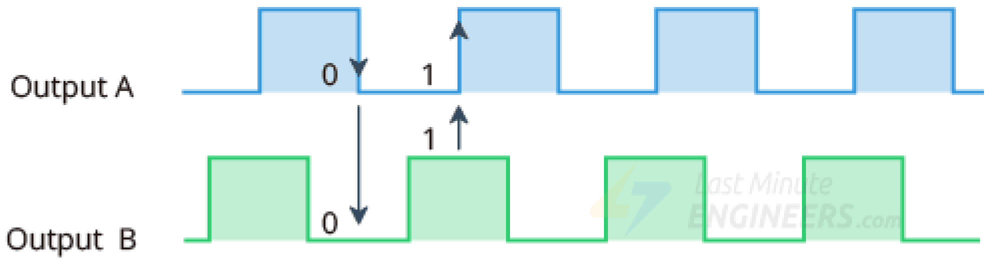


Figure 16: Rotary Encoder Counter Clockwise Turn - https://lastminuteengineers.com/rotary-encoder-arduino-tutorial/

Appendix 2.1

Website PHP POST Script

```
<?php
$time = date("l jS \of F Y h:i:s A");
$level = $_POST['level'];
$runTime = $_POST['result'];
$numCor = $_POST['right'];
$numRng = $_POST['wrong'];
$file = 'handTrainerResults.csv';
$data = $time.",".$level.",".$runTime.",".$numCor.",".$numRng;
file_put_contents($file, $data . PHP_EOL, FILE_APPEND);
?>
```

Website HTML to Display Results (Wordpress Based)

www.graycode.ie/handtrainer/results

```
<?php get_header();
$basePath = get_template_directory_uri();
$oldPath = "{$basePath}/old/"; ?>
<div>
    <div class="p-0 container" style="background-color: #b3b3b3;">
        <?php
        the_post();
        the_content();
        ?>
     <?php
echo "<html><body><table class=\"table table-striped\">\n\n";
echo "<thead class=\"thead-dark\">
<tr>
      <th scope=\"col\">Date</th>
      <th scope=\"col\">Level Difficulty</th>
      <th scope=\"col\">Time Taken (secs) </th>
      <th scope=\"col\"># Correct Presses</th>
      <th scope=\"col\"># Incorrect Presses</th>
</tr>
  </thead>";
  echo "<tbody>";
      $f = fopen("https://graycode.ie/wp-
      content/themes/graycode/handTrainerResults.csv", "r");
      while (($line = fgetcsv($f)) !== false) {
            echo "<tr>";
           foreach ($line as $cell) {
               echo "<td>" . htmlspecialchars($cell) . "</td>";}
           echo "</tr>\n";}
     fclose($f);
     echo "\n</tbody></table></body></html>";
  ?>
  </div>
</div>
<?php get_footer();?>
```

Appendix 2.2

Code for MKR1000

```
//Author:   Joel Gray
//Project:  EEE527 - Hand Trainer for Brain Injury Related Hand Spasticity
//Description:    Main code for MKR1000


//Include necessary Libraries
#include <Wire.h>                 // Library for I2C comms
#include <SPI.h>                  // Library to enable SPI comms
#include <WiFi101.h>              // Library for WiFi comms
#include <Adafruit_MCP23017.h>    // Library to include functions for MCP23017
I/O Expander
#include <Adafruit_ST7735.h>      // Hardware-specific library for ST7735 1.8”
TFT LCD Screen
#include <CapacitiveSensor.h>     //include library

//Assign Sensor prefix to reference Capacitive Sensor library
CapacitiveSensor Sensor = CapacitiveSensor(0, 10); //assign sensor pins

//TFT LCD Screen Set Up
#define TFT_CS 6        //CS
#define TFT_RST 5       //RES
#define TFT_DC 7        //DC
#define TFT_SCLK 9      // SCL
#define TFT_MOSI 8      // SDA
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_MOSI, TFT_SCLK,
TFT_RST);

// Rotary Encoder Inputs
#define CLK 2          //CLK
#define DT 4           //DT
#define SW 3           //SW

//Assign mcp prefix to reference MCP23017 library
Adafruit_MCP23017 mcp;

//Set up Wifi
char ssid[] = "TestNetwork";       // your network SSID (name)
char pass[] = "Password123";   // your network password (use for WPA, or use
as key for WEP)
int status = WL_IDLE_STATUS;       // the WiFi radio's status
char server[] = "graycode.ie";     // name address for Webstie (using DNS)
WiFiSSLClient client;              // Assign client prefix to reference
Wifi101 library

//Initalise Global Variable
int selLevFlag = 0;                  //Flag to indicate if a level has been
selected and that the level should then begin
int buttonFlag = 0;                  //Flag to indicate if the rotary encoder
button has been pressed
int listFlag = 0;                    //Flag to indicate which level in the
menu is currently selected
int counterPrev = 0;                 //Varaible to assign the previous number
of turns using the rotary encoder
int counter = 0;                     //Varaible to assign the current number
of turns using the rotary encoder
int countInts = 0;                   //Counter for number of correct button
presses
```

```
int currentStateCLK;               //Variable to save the current state of
the CLK pulse from Rotary Encoder
int lastStateCLK;                  //Variable to save the previous state of
the CLK pulse
String currentDir = "";            //String to save direction of rotary
encoder turn
unsigned long lastButtonPress = 0;  //Time of lastButtonPress on rotary
encoder to help create debounce
bool handPres = 0;                 //Detect if hand is present or not, hand
present = 1, hand no present = 0
int wrongPress = 0;                //Number of incorrect/wrong button
presses
int rightPress = 0;                //Number of correct/right button presses
unsigned long startMillis = 0;     //Start timer time
unsigned long finMillis = 0;       //Finish timer time
int interruptFlag = 0;             //Flag to indicate whether or not there
has been an interrupt triggered
int seqFlag = 0;                   //Flag to indicate which sequence has
been selected, 0 - Easy, 1 - Medium, 2 - Hard
byte arduinoIntPin = 1;            //Set interrupt pin on Arduino for the
MCP23017 to trigger

//Variables to determine the order buttons were pressed for Hard level -
necessary as hard level the sequence is randomised
int hfirst;
int hsecond;
int hthird;
int hfourth;
int hfifth;
int hsixth;
int hseventh;
int heighth;
int hninth;
int htenth;

void setup() {
    Serial.begin(9600);
    Serial.println("Serial Monitor Started Successfully");

    pinMode(CLK, INPUT);                 // Set encoder pins as inputs
    pinMode(DT, INPUT);                  // Set encoder pins as inputs
    pinMode(SW, INPUT_PULLUP);           // Set encoder pins as inputs
    pinMode(arduinoIntPin, INPUT);       // Set interrupt pin as an input
    mcp.begin();                         // use default address 0 and start
comms with MCP23017 chip

    // Initalise LCD screen
    tft.initR(INITR_BLACKTAB);           //Init ST7735S LCD chip, black tab
    Serial.println(F("Initialised"));    //Print Initalised to Serial
Monitor
    tft.setRotation(2);                  //Rotate the screen to fit our
usage case
    tft.fillScreen(ST77XX_BLACK);        //Set screen to black
    screenSetup();                       //Call screenSetup function

    // Read the initial state of CLK and print
    lastStateCLK = digitalRead(CLK);             //Set last CLK state to the
current CLK state for first initalisation
    Serial.print("Initial CLK State");           //Print to serial monitor
    Serial.println(lastStateCLK);                //Print value of
lastStateCLK
```

```
    // Check for the presence of the wifi chip
    if (WiFi.status() == WL_NO_SHIELD) {          //If there is no WiFi chip 
detected
        Serial.println("WiFi chip not present");  //Print chip not present
        while (true);                             //Loop forever as we need 
the WiFi funcationality to complete tests
    }

    // Attempt to connect to WiFi network:
    while (status != WL_CONNECTED) {                             //While we 
are not connected to the internet, keep trying
        Serial.print("Attempting to connect to WPA SSID: ");     //Print to 
serial monitor, that we're trying to connect
        Serial.println(ssid); // Connect to WPA/WPA2 network:    //Print name 
of network we're trying to connect to
        status = WiFi.begin(ssid, pass);                         //Update the 
connection status
        delay(10000);                                            //Wait 10 
seconds for connection:
    }

    // You're connected now, so print out the data:
    Serial.print("You're connected to the network");             //Print that 
we're connected
    printCurrentNet();                                           //Call 
function to print network stats of current network
    printWiFiData();                                             //Call 
function to print wifi data
}   //End Setup Function


void printWiFiData() {                            // Function to print wifi 
data
    IPAddress ip = WiFi.localIP();                // Assign ip to be the 
current IP address of device
    Serial.print("IP Address: ");                 // Print WiFi's IP address:
    Serial.println(ip);

    byte mac[6];                                  // Print your MAC address:
    WiFi.macAddress(mac);                         // Call wifi Mac Addres 
function
    Serial.print("MAC address: ");
    printMacAddress(mac);                         // Call function to print Mac 
address
}   //End printWiFiData function

void printCurrentNet() {                          // Function to print details 
of current network
    Serial.print("SSID: ");                       // Print "SSID"
    Serial.println(WiFi.SSID());                  // Print the SSID of the 
network we're connected to
    byte bssid[6];                                // BSSID is the same as Mac 
Address, creating variable
    WiFi.BSSID(bssid);                            // Call function to get Mac 
Address
    Serial.print("BSSID: ");                      // Print "BSSID"
    printMacAddress(bssid);                       // Call function to print the 
MAC address of the router were connected to

    long rssi = WiFi.RSSI();                      // Get Strength of singal and 
assign to 'rssi'
    Serial.print("signal strength (RSSI):");      // Print "signal strength"
    Serial.println(rssi);                        // Print value
    byte encryption = WiFi.encryptionType();     // Get encryption type
    Serial.print("Encryption Type:");            // Print "encryption type: "
    Serial.println(encryption, HEX);             // Print to serial monitor
    Serial.println();
}   //End printCurrentNet function

void printMacAddress(byte mac[]) {               // Function to print the MAC
Address which takes mac address in byte form
    for (int i = 5; i >= 0; i--) {               // Count down from 5
        if (mac[i] < 16) {                       // Print a 0 if array index
is less than 16
            Serial.print("0");
        }
        Serial.print(mac[i], HEX);               // Print current value of
array index in HEX
        if (i > 0) {
            Serial.print(":");                   // Print colon seperator for
MAC address
        }
    }
    Serial.println();                            // Print new line
}   //End printMacAddress function

void easyLev() {                                 // Function to light LEDs in
an incremental sequence for Easy Level
    for (int i = 0; i <= 9; i++) {               // Loop to assign all MCP
pins to outputs
        mcp.pinMode(i, OUTPUT);                  // Set mcp pin with value of
i as an output
        mcp.digitalWrite(i, LOW);                // Drive mcp pin with value
of i as LOW, to ensure it is off and no floating pins
    }

    for (int i = 0; i <= 9; i++) {               // Loop to light each LED 0 -
9 for 1 second
        mcp.digitalWrite(i, HIGH);               // Drive mcp pin with value
of i as HIGH
        timeDelay(1000);                         // Wait for 1 second (1000
milliseconds)
        mcp.digitalWrite(i, LOW);                // Drive mcp pin with value
of i as LOW again
    }
    readInputs();                                // Call readInputs function
}   //End easyLev function

void medLev() {                                  // Function to light LEDs in
a specific sequence for Medium Level
    for (int i = 0; i <= 9; i++) {               // Loop to assign all MCP
pins to outputs
        mcp.pinMode(i, OUTPUT);                  // Set mcp pin with value of
i as an output
        mcp.digitalWrite(i, LOW);                // Drive mcp pin with value
of i as LOW, to ensure it is off and no floating pins
    }
    for (int i = 0; i <= 9; i++) {               // Loop to light each LED 0 -
9 for 1 second
        int medSeq[10] = {1,0,9,2,8,3,7,4,6,5}; // Determine array to dictate
sequence for LEDs to light
        mcp.digitalWrite(medSeq[i], HIGH);       // Drive mcp pin with value
of i as HIGH
```

```
        timeDelay(1000);                          // Wait for 1 second (1000
milliseconds)
        mcp.digitalWrite(medSeq[i], LOW);         // Drive mcp pin with value
of i as LOW again
    }
    readInputs();                                 // Call readInputs function
}   //End medLev function

void hardLev() {                                  // Function to light LEDs in
a random sequence for Hard Level
    int leds[10] = {0,1,2,3,4,5,6,7,8,9};         // Creating array for all mcp
pins of LEDs
    for (int i = 0; i <= 9; i++) {                // Loop to assign all MCP
pins to outputs
        mcp.pinMode(i, OUTPUT);                   // Set mcp pin with value of
i as an output
        mcp.digitalWrite(i, LOW);                 // Drive mcp pin with value
of i as LOW, to ensure it is off and no floating pins
    }
    for (int i = 0; i <= 9; i++) {                // Loop to light each LED 0 -
9 for 1 second
        int j = leds[random(0, 9)];               // Set variable j to equal a
random number between 0-9 and that will be our pin to drive high to light its
respective LED
        switch (i) {                              // Switch case switching
based on i, will run through 10 times
          case 0: {                               // If i==0 then...
            hfirst = j;                           // hfirst == to current value
of j from our random function, will be used in checkHardLev function
          }
          break;                                  // break out of case
          case 1: {                               // If i==1 then...
              hsecond = j;                        // hsecond == to current
value of j from our random function, will be used in checkHardLev function
          }
          break;                                  // break out of case
          case 2: {                               // ""
              hthird = j;                         // ""
          }
          break;                                  // ""
          case 3: {
              hfourth = j;                        // ""
          }
          break;
          case 4: {
              hfifth = j;
          }
          break;
          case 5: {
              hsixth = j;
          }
          break;
          case 6: {
              hseventh = j;
          }
          break;
          case 7: {
              heighth = j;
          }
          break;
          case 8: {
              hninth = j;
          }
          break;
          case 9: {                             // If i==9 then...
              htenth = j;                       // htenth == to current value
of j from our random function
          }
          break;                                // break out of case
        }                                       //End switch statement

        mcp.digitalWrite(j, HIGH);              // Based on value of j from
random function, drive the respective pin/LED high
        timeDelay(1000);                        // Wait for 1 second (1000
milliseconds)
        mcp.digitalWrite(j, LOW);               // Drive the pin/LED low
again
    }
    readInputs();                               // Call readInputs function
}   //End hardLev function

void readInputs() {                             // Function to set up pins to
be ready to read button presses as inputs
    mcp.setupInterrupts(true, false, HIGH);     // First argument True means
Set up Mirroring both port A and B to be the same, Second argument False
means set Int Pin to open drain, pulling it to ground/low when not in use,
Third argument is polarity set to HIGH
    for (int i = 0; i <=9; i++){                // Loop from 0 - 9
        mcp.pinMode(i, INPUT);                  // Set pin value of i to be
an input
        mcp.pullUp(i, LOW);                     // Ensure internal 100K
pullup is NOT activated
        mcp.setupInterruptPin(i, RISING);       // Set each pin to trigger
interrupt on rising edge
    }
    mcp.readGPIOAB();                           // Function to read GPIO
rgisters on Ports A and B on the MCP23017 chip, resets the interrupt flags
    attachInterrupt(digitalPinToInterrupt(arduinoIntPin), handleInterrupt,
RISING);       //Attach interrupt to our chosen interrupt pin on the MKR1000

    tft.fillScreen(ST77XX_BLACK);               // Set LCD Background to
Black
    tft.setCursor(3, 40);                       // Set cursor on LCD screen
(x,y)
    tft.setTextSize(3);                         // Set Text Size to 3 (1-5)
    tft.setTextColor(ST77XX_GREEN);             // Set Text to Green
    tft.println("Press");                       // Print Text to LCD screen
    tft.setCursor(3, 70);                       // Set cursor on LCD screen
(x,y)
    tft.println("Correct");                     // Print Text to LCD screen
    tft.setCursor(3, 100);                      // Set cursor on LCD screen
(x,y)
    tft.println("Buttons");                     // Print Text to LCD screen
}   //End readInputs function

void handleInterrupt() {                        // Function for dealing with
a triggered interrupt
    interruptFlag = 1;                          // Functionality is limited
inside an interrupt function, thus we set interruptFlag =1 to indicate it has
been triggered and deal with the functionality in the main loop
}   //End handleInterrupt function

void checkEasyLev() {                 // Function for checking if the correct
buttons have been pressed for the Easy level
```

```
    int first;                      // Variable to saving the first button
press
    int second;                     // Variable to saving the second button
press
    int third;                      // """
    int fourth;                     // """
    int fifth;
    int sixth;
    int seventh;
    int eighth;
    int ninth;
    int tenth;                      // Variable to saving the tenth button
press

    if (countInts <= 0) {           // If countInts is less than or equal to
0 then this is the first time we've entered this function, thus it's the
start of the level
        startTimer();               // Start the timer to record time it
takes to complete leve;
    }

    if (seqFlag == 1) {             // Double check that seqFlag is equal to
1 and we are definitely checking for the Easy level
        Serial.println(countInts);  // Print current value of counter
        switch (countInts) {        // Switch statement based on the current
number of countInts
            case 0:
{                                                           // If countInts
== 0
                first =
mcp.getLastInterruptPin();                              // Assign 'first' to
equal the pin number of the button last pressed
                Serial.println(first);
   // Print value of pin
                if ((first == 0) && (mcp.getLastInterruptPinValue() == 1))
{    // If first == 0 (the expected pin number) and the value of that pin is
== 1 then that button was pressed correctly at the right time
                    rightPress++;
   // Add one to the number of correct button presses
                    showTick(rightPress);
   // Call showTick function and pass in the number of correct button presses
                    Serial.println("Correct
Button");                           // Print Correct button to serial monitor
                } else
{                                                          // Otherwise if this
is the wrong button pressed
                    wrongPress++;
   // Add one to the number of incorrect button presses
                    showX(wrongPress);
   // Call showX function and pass in the number of INcorrect button presses
                    Serial.println("Incorrect
Button");                         // Print Correct button to serial monitor
                    countInts--
;                                                  // Minus 1 from countInts to
ensure that we don't run out of presses before all correct buttons are
pressed
                }
                break;
   //Break out of case
            }
            case 1:
{                                                           // countInts == 1
```

```
                second =
mcp.getLastInterruptPin();                             // Assign 'second' to
equal the pin number of the button last pressed
                Serial.println(second);
   // Print value of pin
                if ((second == 1) && (mcp.getLastInterruptPinValue() == 1))
{   // If second == 1 (the expected pin number) and the value of that pin is
== 1 then that button was pressed correctly at the right time
                    rightPress++;
   // Add one to the number of correct button presses
                    showTick(rightPress);
   // Call showTick function and pass in the number of correct button presses
                    Serial.println("Correct
Button");                            // Print Correct button to serial monitor
                } else
{                                                         // Otherwise if this
is the wrong button pressed
                    wrongPress++;
   // Add one to the number of incorrect button presses
                    showX(wrongPress);
   // Call showX function and pass in the number of INcorrect button presses
                    Serial.println("Incorrect
Button");                          // Print Correct button to serial monitor
                    countInts--
;                                                 // Minus 1 from countInts to
ensure that we don't run out of presses before all correct buttons are
pressed
                }
                break;
   //Break out of case
            }
            case 2:
{                                                            // """
                third =
mcp.getLastInterruptPin();                              // """
                Serial.println(third);
   // """
                if ((third == 2) && (mcp.getLastInterruptPinValue() == 1))
{    // """
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 3: {
                fourth = mcp.getLastInterruptPin();
                Serial.println(fourth);
                if ((fourth == 3) && (mcp.getLastInterruptPinValue() == 1)) {
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 4: {
                fifth = mcp.getLastInterruptPin();
                Serial.println(fifth);
                if ((fifth == 4) && (mcp.getLastInterruptPinValue() == 1)) {
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 5: {
                sixth = mcp.getLastInterruptPin();
                Serial.println(sixth);
                if ((sixth == 5) && (mcp.getLastInterruptPinValue() == 1)) {
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 6: {
                seventh = mcp.getLastInterruptPin();
                Serial.println(seventh);
                if ((seventh == 6) && (mcp.getLastInterruptPinValue() == 1))
{
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 7: {
                eighth = mcp.getLastInterruptPin();
                Serial.println(eighth);
                if ((eighth == 7) && (mcp.getLastInterruptPinValue() == 1)) {
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 8: {
                ninth = mcp.getLastInterruptPin();
                Serial.println(ninth);
                if ((ninth == 8) && (mcp.getLastInterruptPinValue() == 1)) {
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 9: {
                tenth = mcp.getLastInterruptPin();
                Serial.println(tenth);
                if ((tenth == 9) && (mcp.getLastInterruptPinValue() == 1)) {
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            default:
{                                                              // Default case
                Serial.println("Error");
     // Print Error
                break;
     // Break out of case
            }
        }
        countInts++;
     // Increment countInts
    }
    if (countInts >= 10)
{                                                          // If number of
correct buttons to be pressed gets to 10
        finishLevel();
     // Then call the finishLevel function
    }
}   //End checkEasyLev function

void checkMediumLev() {                                          // Function
for checking if the correct buttons have been pressed for the Medium level
    int first;                                                   // Follows
same logic as easy level check, but expecting different values for each case
    int second;
    int third;
    int fourth;
    int fifth;
    int sixth;
    int seventh;
    int eighth;
    int ninth;
    int tenth;
    Serial.println("We're checking the inputs for Medium Level");
    if (countInts <= 0) {
        startTimer();
    }

    if (seqFlag == 2) {
        Serial.println("SeqFlag verified as 2 in if statement");
        Serial.println(countInts);
        switch (countInts) {
            case 0: {
                first = mcp.getLastInterruptPin();
                Serial.println(first);
                if ((first == 1) && (mcp.getLastInterruptPinValue() == 1))
{     // If first == 1 (the expected pin number) and the value of that pin is
== 1 then that button was pressed correctly at the right time
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 1: {
                second = mcp.getLastInterruptPin();
                Serial.println(second);
                if ((second == 0) && (mcp.getLastInterruptPinValue() == 1))
{    // If second == 0 (the expected pin number) and the value of that pin is
== 1 then that button was pressed correctly at the right time
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 2: {
                third = mcp.getLastInterruptPin();
                Serial.println(third);
                if ((third == 9) && (mcp.getLastInterruptPinValue() == 1))
{     // If third == 9 (the expected pin number) and the value of that pin is
== 1 then that button was pressed correctly at the right time
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 3: {
                fourth = mcp.getLastInterruptPin();
                Serial.println(fourth);
                if ((fourth == 2) && (mcp.getLastInterruptPinValue() == 1))
{   // If fourth == 2 (the expected pin number) and the value of that pin is
== 1 then that button was pressed correctly at the right time
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 4: {
                fifth = mcp.getLastInterruptPin();
                Serial.println(fifth);
                if ((fifth == 8) && (mcp.getLastInterruptPinValue() == 1))
{    // If fifth == 8 (the expected pin number) and the value of that pin is
== 1 then that button was pressed correctly at the right time
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 5: {
                sixth = mcp.getLastInterruptPin();
                Serial.println(sixth);
                if ((sixth == 3) && (mcp.getLastInterruptPinValue() == 1)) {
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 6: {
                seventh = mcp.getLastInterruptPin();
                Serial.println(seventh);
                if ((seventh == 7) && (mcp.getLastInterruptPinValue() == 1))
{
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 7: {
                eighth = mcp.getLastInterruptPin();
                Serial.println(eighth);
                if ((eighth == 4) && (mcp.getLastInterruptPinValue() == 1)) {
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 8: {
                ninth = mcp.getLastInterruptPin();
                Serial.println(ninth);
                if ((ninth == 6) && (mcp.getLastInterruptPinValue() == 1)) {
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 9: {
                tenth = mcp.getLastInterruptPin();
                Serial.println(tenth);
                if ((tenth == 5) && (mcp.getLastInterruptPinValue() == 1)) {
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            default: {
                Serial.println("Error");
                break;
            }
        }
        countInts++;
    }
    if (countInts >= 10) { //number of correct buttons to be pressed
        finishLevel();
    }
}   //End checkMediumLevel function
```

```
void checkHardLev()
{                                                                // Function
for checking if the correct buttons have been pressed for the Hard level
    Serial.println("We're checking the inputs for Hard
Level");                          // Similar logic as with the Easy and Medium
level however this time we check against values for hfirst... hsecond etc..
assigned in hardLev function
    if (countInts <= 0) {
        startTimer();
    }

    if (seqFlag == 3) {
        Serial.println("SeqFlag verified as 3 in if statement");
        Serial.println(countInts);
        switch (countInts) {
            case 0: {
                int first = mcp.getLastInterruptPin();
                Serial.println(first);
                if ((first == hfirst) && (mcp.getLastInterruptPinValue() ==
1)) {   // If first == hfirst (the expected pin number assigned in hardLev
function) and the value of that pin is == 1 then that button was pressed
correctly at the right time
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 1: {
                int second = mcp.getLastInterruptPin();
                Serial.println(second);
                if ((second == hsecond) && (mcp.getLastInterruptPinValue() ==
1)) {   // If second == hsecond (the expected pin number assigned in hardLev
function) and the value of that pin is == 1 then that button was pressed
correctly at the right time
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
                    Serial.println("Incorrect Button");
                    countInts--;
                }
                break;
            }
            case 2: {
                int third = mcp.getLastInterruptPin();
                Serial.println(third);
                if ((third == hthird) && (mcp.getLastInterruptPinValue() ==
1)) {
                    rightPress++;
                    showTick(rightPress);
                    Serial.println("Correct Button");
                } else {
                    wrongPress++;
                    showX(wrongPress);
```

```
                        Serial.println("Incorrect Button");
                        countInts--;
                    }
                    break;
                }
                case 3: {
                    int fourth = mcp.getLastInterruptPin();
                    Serial.println(fourth);
                    if ((fourth == hfourth) && (mcp.getLastInterruptPinValue() ==
1)) {
                        rightPress++;
                        showTick(rightPress);
                        Serial.println("Correct Button");
                    } else {
                        wrongPress++;
                        showX(wrongPress);
                        Serial.println("Incorrect Button");
                        countInts--;
                    }
                    break;
                }
                case 4: {
                    int fifth = mcp.getLastInterruptPin();
                    Serial.println(fifth);
                    if ((fifth == hfifth) && (mcp.getLastInterruptPinValue() ==
1)) {
                        rightPress++;
                        showTick(rightPress);
                        Serial.println("Correct Button");
                    } else {
                        wrongPress++;
                        showX(wrongPress);
                        Serial.println("Incorrect Button");
                        countInts--;
                    }
                    break;
                }
                case 5: {
                    int sixth = mcp.getLastInterruptPin();
                    Serial.println(sixth);
                    if ((sixth == hsixth) && (mcp.getLastInterruptPinValue() ==
1)) {
                        rightPress++;
                        showTick(rightPress);
                        Serial.println("Correct Button");
                    } else {
                        wrongPress++;
                        showX(wrongPress);
                        Serial.println("Incorrect Button");
                        countInts--;
                    }
                    break;
                }
                case 6: {
                    int seventh = mcp.getLastInterruptPin();
                    Serial.println(seventh);
                    if ((seventh == hseventh) && (mcp.getLastInterruptPinValue()
== 1)) {
                        rightPress++;
                        showTick(rightPress);
                        Serial.println("Correct Button");
                    } else {
```

```
                wrongPress++;
                showX(wrongPress);
                Serial.println("Incorrect Button");
                countInts--;
            }
            break;
        }
        case 7: {
            int eighth = mcp.getLastInterruptPin();
            Serial.println(eighth);
            if ((eighth == heighth) && (mcp.getLastInterruptPinValue() ==
1)) {
                rightPress++;
                showTick(rightPress);
                Serial.println("Correct Button");
            } else {
                wrongPress++;
                showX(wrongPress);
                Serial.println("Incorrect Button");
                countInts--;
            }
            break;
        }
        case 8: {
            int ninth = mcp.getLastInterruptPin();
            Serial.println(ninth);
            if ((ninth == hninth) && (mcp.getLastInterruptPinValue() ==
1)) {
                rightPress++;
                showTick(rightPress);
                Serial.println("Correct Button");
            } else {
                wrongPress++;
                showX(wrongPress);
                Serial.println("Incorrect Button");
                countInts--;
            }
            break;
        }
        case 9: {
            int tenth = mcp.getLastInterruptPin();
            Serial.println(tenth);
            if ((tenth == htenth) && (mcp.getLastInterruptPinValue() ==
1)) {
                rightPress++;
                showTick(rightPress);
                Serial.println("Correct Button");
            } else {
                wrongPress++;
                showX(wrongPress);
                Serial.println("Incorrect Button");
                countInts--;
            }
            break;
        }
        default: {
            Serial.println("Error");
            break;
        }
    }
    countInts++;
}
```

```
    if (countInts >= 10) { //number of correct buttons to be pressed
        finishLevel();
    }
}   //End checkHardLev function

void finishLevel() {                          // Function to be called when a
level has been completed
    selLevFlag = 0;                           // Set selLevFlag to be = 0, so
in our main loop it prompts us to select a new level again
    endTimer();                               // Call endTimer function to work
out the time taken to complete the level, this function calls another
function call displayResults also
    countInts = 0;                            // Reset countInts to be 0,so
when we start another level it will start the loop from 0 and ensure that
correct buttons 0 through to 9 are pressed
    wrongPress = 0;                           // Reset number of wrong button
presses
    rightPress = 0;                           // Reset number of right button
presses
    timeDelay(20000);                         // Delay time for 20 seconds so
the user can view their scores on the LCD
    screenSetup();                            // Call screenSetup to show the
user the level selection menu again
}   //End finishLevel function

void startTimer() {                           // Function to begin level timer
    startMillis = millis();                   // Set startMillis to equal the
current time, this is the level start time
    Serial.print("Level Start Time: ");       // Print to serial monitor
    Serial.println(startMillis);              // Print value of startMillis
}   //End startTimer function

void endTimer() {                                       // Function to end
level timer
    Serial.print("Level Finish Time: ");                // Print to serial
monitor
    Serial.println(millis());                           // Print current time
    finMillis = ((millis() - startMillis) / 1000);      //finMillis == the
current time at the time of the loop completing minus the start time
    Serial.print("Level took ");                        // Print to serial
monitor
    Serial.print(finMillis);                            // Print value of
finMillis
    Serial.print(" seconds ");                          // Print to serial
monitor
    displayResults(finMillis);                          // Call displayResults
function and pass in the variable finMillis
    finMillis = 0;                                      // Reset value of
finMillis to 0, to be fresh when we start the timer next time
    startMillis = 0;                                    // Reset value of
startMillis to 0, to be fresh when we start the timer next time
}   //End endTimer function

void displayResults(unsigned long timeTaken) {          // Function to display
the results of the level just played, which takes the argument timeTaken as
the time taken to complete the level
    String level = "";                                  // Create a blank
variable for the level
    tft.fillScreen(ST77XX_BLACK);                       // Set background to
Black
    tft.setCursor(5, 10);                               // Set cursor on LCD
screen (x,y)
```

```
    tft.setTextColor(ST77XX_WHITE);                     // Set Text to White
    tft.setTextSize(2);                                 // Set Text Size to 2
(1-5)
    tft.println("Completed");                           // Write text to LCD
screen
    tft.setCursor(5, 30);                               // Set cursor on LCD
screen (x,y)

    switch (seqFlag) {                                  // Switch statement to
print the Level name based on seqFlag which indicates which level is
currently selected
        case 1: {                                       // If seqFlag == 1
            tft.println("Easy");                        // Print to LCD Easy
            level = "Easy";                             // Set level to ==
"Easy"
            break;                                      // Break from case
        }
        case 2: {                                       // If seqFlag == 2
            tft.println("Medium");                      // Print to LCD Medium
            level = "Medium";                           // Set level to ==
"Medium"
            break;                                      // Break from case
        }
        case 3: {                                       // If seqFlag == 3
            tft.println("Hard");                        // Print to LCD Hard
            level = "Hard";                             // Set level to ==
"Hard"
            break;                                      // Break from case
        }
        default: {                                      // Default case
            tft.fillScreen(ST77XX_BLACK);               // Set background to
Black
            tft.setCursor(5, 10);                       // Set cursor on LCD
screen (x,y)
            tft.setTextColor(ST77XX_WHITE);             // Set Text to White
            tft.setTextSize(4);                         // Set Text Size to 4
(1-5)
            tft.println("ERROR");                       // Print to LCD ERROR
        }
    }

    tft.setCursor(5, 50);                               // Set cursor on LCD
screen (x,y)
    tft.print("Level");                                 // Print to LCD
    tft.setCursor(5, 70);                               // Set cursor on LCD
screen (x,y)
    tft.print("in ");                                   // Print to LCD
    tft.print(timeTaken);                               // Print to LCD
    tft.setCursor(5, 90);                               // Set cursor on LCD
screen (x,y)
    tft.println("Seconds");                             // Print to LCD
    tft.setCursor(5, 130);                              // Set cursor on LCD
screen (x,y)
    tft.setTextSize(1);                                 // Set Text Size to 1
(1-5)
    tft.setTextColor(ST77XX_GREEN);                     // Set Text to Green
    tft.print("# Correct: ");                           // Print to LCD
    tft.println(rightPress);                            // Print to LCD value
of correct presses
    tft.setCursor(5, 140);                              // Set cursor on LCD
screen (x,y)
    tft.setTextColor(ST77XX_RED);                       // Set Text to Red
    tft.print("# Incorrect: ");                         // Print to LCD
    tft.println(wrongPress);                            // Print to LCD value
of wrong presses
    sendResults(level, finMillis, rightPress, wrongPress);    // Call
sendResults function to send values to the internet and pass it the current
level, finish time, and number of right and wrong button presses
}   //End displayResults function

void readInterrupt() {                                        // Function to
read the current interrupt pin value
    Serial.println("\n We are handling an interrupt.");
    uint16_t x = 0b00000000;                                  // The 0b prefix
indicates a binary constant
    Serial.println(mcp.getLastInterruptPin());                // Print current
pin which triggered the interrupt

    for (int i = 0; i < 10; i++) {                            // Loop from 0-9
        uint16_t j = mcp.digitalRead(i);                      // To read each
pin on the MCP23017
        x = bitWrite(x, 0, j);                                // Write the
current value of j to the variable x, starting at bit 0
        Serial.print(x, BIN);                                 // Print value of
x in Binary form
    }

    switch (seqFlag) {                                        // Switch
statement to decide which function to call to validate button presses based
on seqFlag which indicates which level is currently selected
      case 1: {                                               // If seqFlag ==
1
          checkEasyLev();                                     // Call the
checkEasyLev function
          Serial.println("Checking Easy Level");              // Print to
Serial monitor
      }
      break;                                                  // Break from
case
      case 2: {                                               // If seqFlag ==
2
          checkMediumLev();                                   // Call the
checkMediumLev function
          Serial.println("Checking Medium Level");            // Print to
Serial monitor
      }
      break;                                                  // Break from
case
      case 3: {                                               // If seqFlag ==
3
          checkHardLev();                                     // Call the
checkEasyLev function
          Serial.println("Checking Hard Level");              // Print to
Serial monitor
      }
      break;                                                  // Break from
case
    }
    digitalWrite(6, LOW);                                     // Set internal
LED to LOW/off
    interruptFlag = 0;                                        // Set
interruptFlag to 0 to reset it as we've dealt with the interrupt
}   //End readInterrupt function
```

```
void screenSetup() {                     // Function to setup initial screen
menu
    tft.setTextWrap(false);              // Turn text wrapping off
    tft.fillScreen(ST77XX_BLACK);        // Set background to Black
    easy();                              // Call the easy function to set
the current selection to the easy level as it's at the top of the menu list
}   //End screenSetup function

void flash() {                           // Function to make menu screen
selection flash to show that the menu cannot move in the direction being
tried
    tft.setCursor(5, 10);                // Set cursor
    tft.setTextColor(ST77XX_WHITE);      // Set Text to White
    tft.setTextSize(3);                  // Set Text Size to "3"
    tft.println("Level: ");              // Write "Level" to LCD
    tft.setCursor(5, 50);
    tft.setTextSize(3);
    tft.println("Easy");
    tft.setCursor(5, 80);
    tft.println("Medium");
    tft.setCursor(5, 110);
    tft.println("Hard");
    tft.setCursor(0, 0);                 // Reset cursor to 0,0
}   //End flash function

void easy() {                            // Function to highlight the word
Easy in Green to show current selection in menu
    listFlag = 0;                        // Set listflag to 0 to indicate
Easy level being selected
    tft.setCursor(5, 10);                // Set cursor
    tft.setTextColor(ST77XX_WHITE);      // Set Text to White
    tft.setTextSize(3);                  // Set Text Size to "3"
    tft.println("Level: ");              // Write "Level"
    tft.setCursor(5, 50);
    tft.setTextColor(ST77XX_GREEN);      // Set Text to Green
    tft.println("Easy");
    tft.setCursor(5, 80);
    tft.setTextColor(ST77XX_WHITE);      // Set Text to White
    tft.println("Medium");
    tft.setCursor(5, 110);
    tft.println("Hard");
    tft.setCursor(0, 0);                 // Reset cursor to 0,0
}   //End easy function

void medium() {                          // Function to highlight the word
Medium in Green to show current selection in menu
    listFlag = 1;                        // Set listflag to 1 to indicate
Medium level being selected
    tft.setCursor(5, 10);                // Set cursor
    tft.setTextColor(ST77XX_WHITE);      // Set Text to White
    tft.setTextSize(3);                  // Set Text Size to "3"
    tft.println("Level: ");              // Write "Level" to LCD
    tft.setCursor(5, 50);
    tft.println("Easy");
    tft.setCursor(5, 80);
    tft.setTextColor(ST77XX_GREEN);      //Set Text to Green
    tft.println("Medium");
    tft.setCursor(5, 110);
    tft.setTextColor(ST77XX_WHITE);      //Set Text to White
    tft.println("Hard");
    tft.setCursor(0, 0);                 // Reset cursor to 0,0
}   //End medium function
```

```
void hard() {                               // Function to highlight the word
HArd in Green to show current selection in menu
    listFlag = 2;                           // Set listflag to 2 to indicate
MHArd level being selected
    tft.setCursor(5, 10);                   // Set cursor
    tft.setTextColor(ST77XX_WHITE);         // Set Text to White
    tft.setTextSize(3);                     // Set Text Size to "3"
    tft.println("Level: ");                 // Write "Level" to LCD
    tft.setCursor(5, 50);
    tft.println("Easy");
    tft.setCursor(5, 80);
    tft.println("Medium");
    tft.setCursor(5, 110);
    tft.setTextColor(ST77XX_GREEN);         // Set Text to Green
    tft.println("Hard");
    tft.setCursor(0, 0);                    // Reset cursor to 0,0
}   //End hard function

void error() {                              // Function to display ERROR is
there is an issue in selecting a level
    tft.setTextWrap(false);                 // Stop test from wrapping
    tft.fillScreen(ST77XX_BLACK);           // Set background to Black
    tft.setCursor(5, 10);                   // Set cursor
    tft.setTextColor(ST77XX_WHITE);         // Set Text to White
    tft.setTextSize(3);                     // Set Text Size to "3"
    tft.println("ERROR ");                  // Write "Level"
    tft.setCursor(5, 50);
    tft.println("PLEASE");
    tft.setCursor(5, 80);
    tft.println("RESET");
    tft.setCursor(0, 0);                    // Reset cursor to 0,0
    Serial.println("Error in choosing Level, issue with listFlag");     //
Print to Serial Monitor showing theres an error
}   //End error function

void moveDown() {                                 // Function to move down the
menu based on the listFlag which is changed by the encoder function
    switch (listFlag) {                           // Switch statement based on
listFlag
      case 0: {                                   // If listFlag == 0
          medium();                               // Call medium function to
highlight the medium text on the menu
          Serial.println("Medium Selected");      // Print to serial monitor
      }
      break;                                      // Break from case
      case 1: {                                   // If listFlag == 1
          hard();                                 // Call hard function to
highlight the hard text on the menu
          Serial.println("Hard Selected");        // Print to serial monitor
      }
      break;                                      // Break from case
      case 2: {                                   // If listFlag == 2
          flash();                                // Call flash function to
flash the menu to indicate you cant go any further down the menu
          timeDelay(100);                         // Time delay of 100
milliseconds
          hard();                                 // Call hard function to
highlight the hard text on the menu
      }
      break;                                      // Break from case
      default: {                                  // Default case
```

```
          error();                                      // Call error function to
display an error on screen as there much be an issue
      }
    }
}   //End moveDown function


void moveUp() {                                         // Function to move up the
menu based on the listFlag which is changed by the encoder function
    switch (listFlag) {                                 // Switch statement based on
listFlag
      case 0: {                                         // If listFlag == 0
          flash();                                      // Call flash function to
flash the menu to indicate you cant go any further up the menu
          timeDelay(100);                               // Time delay of 100
milliseconds
          easy();                                       // Call easy function to
highlight the easy text on the menu
          Serial.println("Easy Selected");              // Print to serial monitor
      }
      break;                                            // Break from case
      case 1: {                                         // If listFlag == 1
          easy();                                       // Call easy function to
highlight the easy text on the menu
          Serial.println("Easy Selected");              // Print to serial monitor
      }
      break;                                            // Break from case
      case 2: {                                         // If listFlag == 2
          medium();                                     // Call medium function to
highlight the medium text on the menu
          Serial.println("Medium Selected");            // Print to serial monitor
      }
      break;                                            // Break from case
      default: {                                        // Default case
          error();                                      // Call error function to
display an error on screen as there much be an issue
      }
    }
}

void encoder() {
    currentStateCLK = digitalRead(CLK);                                   // Read
the current state of CLK
    if (currentStateCLK != lastStateCLK && currentStateCLK == 1) {        // If
last and current state of CLK are different, then pulse occurred, react to
only 1 state change to avoid double count
        if (digitalRead(DT) != currentStateCLK) {                         // If the
DT state is different than the CLK state then the encoder is rotating CCW so
decrement
            counterPrev = counter;                                        // Set
counterPrev to == current value of counter
            counter--;                                                    // Minus
1 from counter
            currentDir = "CCW";                                           // Set
current direction to CCW (Counter Clockwise)
        } else {                                                          //
Encoder is rotating CW so increment
            counterPrev = counter;                                        // Set
counterPrev to == current value of counter
            counter++;                                                    // Plus 1
to counter
```

```
            currentDir = "CW";                                  // Set
current direction to CW (Clockwise)
        }

        Serial.print("Direction: ");                            // Print
direction to serial monitor
        Serial.print(currentDir);
        Serial.print(" | Counter: ");                           // Print
counter value to serial monitor
        Serial.println(counter);
        Serial.print("Previous Counter: ");                     // Print
previous counter value to serial monitor
        Serial.println(counterPrev);

        if (counter > counterPrev) {                            // If
current value of counter is greater then it's previous value..
            moveDown();                                         // Call
moveDown function to move down the selection in the menu
        } else if (counter < counterPrev) {                     // If
current value of counter is less than then it's previous value..
            moveUp();                                           // Call
moveUp function to move up the selection in the menu
        } else if (counter == counterPrev) {                    // If
counter and previous value of counter are the same then nothing
            Serial.println("Nothing Occured");                  // Print
to serial monitor
        }
    }

    lastStateCLK = currentStateCLK;                             //
Remember last CLK state
    int btnState = digitalRead(SW);                             // Read
the button state
    if (btnState == LOW) {                                      // If we
detect LOW signal, button is pressed
        if (millis() - lastButtonPress > 50) {                  // If
50ms have passed since last LOW pulse,and button state is still low then
button has been pressed again, 50ms debounce
            Serial.println("Button pressed!");                  // Print
to serial monitor so we see button has been pressed successfully
            buttonFlag = 1;                                     // Set
buttonFlag to be 1 to indicate the button has been pressed
        }
        lastButtonPress = millis();                             //
Remember the current time and assign it to the button press time
    } else {
        buttonFlag = 0;                                         // No
button press so set buttonFlag to == 0
    }
    timeDelay(1);                                               // Time
delay to help debounce
}   //End encoder function

void handWait() {                                   // Function to detect
if hand is present, wait until hand is present to begin level
    while (handPres == 0) {                         // While hand is NOT
present
        loading();                                  // Display loading
screen
        handPresent();                              // Call handPresent
function to check is the hand is present now
    }
    tft.fillScreen(ST77XX_BLACK);                       //Set bacground to 
Black
    tft.setCursor(5, 40);                               // Set cursor
    tft.setTextSize(4);                                 // Set Text Size
    tft.setTextColor(ST77XX_GREEN);                     // Set Text to Green
    tft.println("Watch");                               // Print to LCD
    tft.setCursor(5, 90);
    tft.println("LEDs");

    handPres = 0;                                       // Reset flag value 
for next level
}   //End handWait function

void selectLevel() {                                    // Function to start 
level based on the listFlag selected using the rotary encoder and the menu
    switch (listFlag) {                                 // Switch statement 
based on listFlag
      case 0: {                                         // If listFlag == 0
          int x = 0;
          start(x);                                     // Call start 
function to display starting text on LCD
          Serial.println("Starting Easy Level");
          selLevFlag = 1;                               // Set selLevFlag == 
1, to indicate a level is selected specifically the easy level
          handWait();                                   // Call handWait 
function, to wait until a hand is prevent on the sensor and person is ready 
to begin level
          easyLev();                                    // Call function to 
display LED sequence for easy level

      }
      break;                                            // Break from case
      case 1: {                                         // If listFlag == 0
          int x = 1;
          start(x);                                     // Call start 
function to display starting text on LCD
          Serial.println("Starting Medium Level");
          selLevFlag = 2;                               // Set selLevFlag == 
2, to indicate a level is selected specifically the medium level
          handWait();                                   // Call handWait 
function, to wait until a hand is prevent on the sensor and person is ready 
to begin level
          medLev();                                     // Call function to 
display LED sequence for medium level
      }
      break;                                            // Break from case
      case 2: {                                         // If listFlag == 0
          int x = 2;
          start(x);                                     // Call start 
function to display starting text on LCD
          Serial.println("Starting Hard Level");
          selLevFlag = 3;                               // Set selLevFlag == 
3, to indicate a level is selected specifically the hard level
          handWait();                                   // Call handWait 
function, to wait until a hand is prevent on the sensor and person is ready 
to begin level
          hardLev();                                    // Call function to 
display LED sequence for hard level
      }
      break;                                            // Break from case
      default: {                                        // Default case
```

```
          error();                                      // Display error
message
        }
      }
}   //End selectLevel function

void start(int lev) {                                   // Function to
display on LCD which
    tft.fillScreen(ST77XX_BLACK);                       // Set background to
Black
    tft.setCursor(5, 30);                               // Set cursor
    tft.setTextSize(3);                                 // Set Text Size to
"3"
    tft.setTextColor(ST77XX_GREEN);                     //Set Text to Green
    switch (lev) {                                      // Switch based on
value of lev
      case 0: {                                         // lev == 0
          tft.println("Easy");                          // Print to LCD
"easy"
          Serial.println("Starting Easy Level");
          seqFlag = 1;                                  // Set seqFlag to 1
for easy level
        }
        break;                                          // Break from case
      case 1: {                                         // lev == 1
          tft.println("Medium");                        // Print to LCD
"medium"
          Serial.println("Starting Medium Level");
          seqFlag = 2;                                  // Set seqFlag to 2
for medium level
        }
        break;                                          // Break from case
      case 2: {                                         // lev == 2
          tft.println("Hard");                          // Print to LCD
"hard"
          Serial.println("Starting Hard Level");
          seqFlag = 3;                                  // Set seqFlag to 3
for hard level
        }
        break;                                          // Break from case
      default: {                                        // Default case
          error();                                      // Display error
message
        }
      }
    tft.setTextColor(ST77XX_WHITE);                     // Set Text to White
    tft.setCursor(5, 70);                               // Set cursor
    tft.println("Level");                               // Write to LCD
    tft.setCursor(5, 110);
    tft.println("Chosen");
    tft.setCursor(0, 0);
    timeDelay(5000);                                    // Delay of 5 seconds
    tft.fillScreen(ST77XX_BLACK);                       // Set background to
Black
}   //End start function

void timeDelay(int time) {                              // Time delay
function, used instead of delay() function to avoid issues with interrupts as
delay()
    unsigned long startMillis = millis();               // Set start time to
be current time
    unsigned long finMillis = 0;                        // Initalise end time
    while (finMillis < time)                             // While finMillis is
less than the number of milliseconds passed into the function
    {
        finMillis = millis() - startMillis;              // finMillis == the
current time at the time of the loop completing minus the start time
    }
}   //End timeDelay function

void handPresent() {                                     // handPresent
function detects if a hand is resting on the capacitive sensor and indicates
if the person is ready to begin the level
    int val = Sensor.capacitiveSensor(30);               // Get val equal to
reading from sensor based on 30 samples
    Serial.println(val);                                 // Show value of val
in serial monitor
    if (val >= 2000){                                    // If value read is
greater that or equal to 2000
        handPres = 1;                                    // Set handPres == 1
to indicate a hand is present
    } else{                                              // If value read is
less that or equal to 2000
        handPres = 0;                                    // Set handPres == 0
as no hand is present
    }
}   //End handPresent function

void loading() {                                         // Function to
display loading screen on LCD while we wait for the person to present their
hand to the sensor
    tft.setCursor(5, 110);
    tft.setTextSize(3);                                  // Set Text Size
    tft.setTextColor(ST77XX_GREEN);                      // Set Text to Green
    tft.println(" Place");                               // Print on LCD
    tft.setCursor(5, 135);
    tft.println(" Hand");

    for (int i = 9; i > 0; i--)
{
             // Loop to count from 9 down to 1 which dictates the location of
the circle being printed
        tft.fillCircle(62 + 40 * (cos(-i * PI / 4)), 50 + 40 * (sin(-i * PI /
4)), 5, ST77XX_GREEN);                        // Create one small green filled
circle
        timeDelay(15);
                                               // Time delay of 15ms
        tft.fillCircle(62 + 40 * (cos(-(i + 1) * PI / 4)), 50 + 40 * (sin(-(i
+ 1) * PI / 4)), 5, ST77XX_GREEN);            // Create one small green filled
circle
        timeDelay(15);
                                               // Time delay of 15ms
        tft.fillCircle(62 + 40 * (cos(-(i + 2) * PI / 4)), 50 + 40 * (sin(-(i
+ 2) * PI / 4)), 5, ST77XX_GREEN);
        timeDelay(15);
        tft.fillCircle(62 + 40 * (cos(-(i + 3) * PI / 4)), 50 + 40 * (sin(-(i
+ 3) * PI / 4)), 5, ST77XX_WHITE);            // Create one small green filled
circle
        timeDelay(15);
                                               // Time delay of 15ms
        tft.fillCircle(62 + 40 * (cos(-(i + 4) * PI / 4)), 50 + 40 * (sin(-(i
+ 4) * PI / 4)), 5, ST77XX_WHITE);
        timeDelay(15);
        tft.fillCircle(62 + 40 * (cos(-(i + 5) * PI / 4)), 50 + 40 * (sin(-(i
+ 5) * PI / 4)), 5, ST77XX_WHITE);
        timeDelay(15);
        tft.fillCircle(62 + 40 * (cos(-(i + 6) * PI / 4)), 50 + 40 * (sin(-(i
+ 6) * PI / 4)), 5, ST77XX_WHITE);
        timeDelay(15);
        tft.fillCircle(62 + 40 * (cos(-(i + 7) * PI / 4)), 50 + 40 * (sin(-(i
+ 7) * PI / 4)), 5, ST77XX_WHITE);
        timeDelay(15);
    }
}   //End loading function

void showTick(int num) {                                // Function to
display an animated Tick on the screen for use when the correct button is
pressed
    tft.fillScreen(ST77XX_BLACK);                       // Set background to
Black
    tft.setCursor(3, 140);                              // Set cursor
    tft.setTextSize(1);                                 // Set Text Size = 1
    tft.setTextColor(ST77XX_GREEN);                     // Set Text to Green
    tft.print("Correct Presses: ");                     // Print number of
correct presses
    tft.println(num);
    int x1 = 40;                                        // Set up various x
and y begining co-ordinates for the Tick shape
    int x2 = 60;
    int x3 = 95;
    int x4 = 39;
    int y1 = 75;
    int y2 = 115;
    int y4 = 115;
    int y3 = 45;
    int i = 0;                                          // Variable for
number of times we want to print the tick
    while (i <= 20) {
        tft.drawLine(x1, y1, x2, y2, ST77XX_GREEN);     // Draw one half of
the tick with one line based on original co-ordinates
        tft.drawLine(x4, y4, x3, y3, ST77XX_GREEN);     // Draw the other
half of the tick with one line based on original co-ordinates
        x1--;                                           // Increase or
decrease the x co-ordinate by 1 each time the loop completes to make the tick
thicker and animated
        x2--;
        x4++;
        x3++;
        i++;                                            // Increate the loop
    }
}   //End showTick function

void showX(int num) {                                   // Function to
display an animated 'X' on the screen for use when the INcorrect button is
pressed
    tft.fillScreen(ST77XX_BLACK);                       // Set background to
Black
    tft.setCursor(3, 140);                              // Set Cursor
    tft.setTextSize(1);                                 // Set Text Size 1
    tft.setTextColor(ST77XX_RED);                       // Set Text to Red
    tft.print("Incorrect Presses: ");                   // Print number of
INcorrect presses
    tft.println(num);
    int x3 = 83;                                        // Set up various x
and y begining co-ordinates for the 'X' shape
```

```
    int x4 = 23;
    int y2 = 115;
    int y3 = 45;
    int i = 0;                                          // Variable for
number of times we want to print the 'X'
    while (i <= 20) {
        tft.drawLine(x4, y2, x3, y3, ST77XX_RED);       // Draw one half of
the 'X' with one line based on original co-ordinates
        tft.drawLine(x4, y3, x3, y2, ST77XX_RED);       // Draw the other
half of the 'X' with one line based on original co-ordinates
        x4++;                                           // Increment the x
co-ordinates by 1 each time the loop completes to make the 'X' appear thicker
and animated
        x3++;
        i++;                                            // Increate the loop
    }
}   //End showX function

void sendResults(String levelDiff, unsigned long runTime, int numCor, int
numRng) {           // Sends the level results to the web, takes 4 arguments,
level difficulty, time taken, and number of correct an incorrect button
presses
    if (client.connectSSL(server, 443))
{                                                     // Connect to the
server/website using port 443 which uses
SSL/HTTPS
        Serial.println("Connected to
server");                                               // Print connected
to server to let us know we connected to the site
        Serial.println(levelDiff);
                  // Print value of variables to be sent
        Serial.println(runTime);
        Serial.println(numCor);
        Serial.println(numRng);
        String field1 =
"level=";                                                             //
Create variables with names of the fields that my php file will recognise on
the server to successfully POST the data
        String field2 = "result=";
        String field3 = "right=";
        String field4 = "wrong=";
        String andNext = "&";
        String data = field1 + levelDiff + andNext + field2 + runTime +
andNext + field3 + numCor + andNext + field4 + numRng;      // Concatenate
strings together to be sent
        Serial.println(data);
                                                       // Print data to
serial monitor to viewing locally

        // Here we need to make a HTTP request, using client.println function
from the WiFi101 library to 'send' these HTTP headers to the site
        client.println("POST /wp-
content/themes/graycode/uploadTrainerResults.php HTTP/1.1");      // Define
request as POST to give link to the file we want to POST to, "HTTP/1.1" is
needed for HTTP header
        client.println("Host:
graycode.ie");                                                      // Define
host website
        client.println("Content-Type: application/x-www-form-
urlencoded");                        // Define content type
```

```
        client.print("Content-Length:
");                                                          // Define
expected data length
        client.println(data.length());
        client.println();
        client.print(data);
        client.println("Connection:
close");                                                       // Set
Connection to Close as request is completed
        Serial.println("Finished");
                     // Print finished to serial so we know the HTTP request
is completed
    } else {
        Serial.println("Failed to
Send");                                                          // Otherwise
print Failed to Send if we weren't able to connect to the server/website
    }
}   //End sendResults function

void loop() {                              // This is the main loop function,
this run continuously
    while (selLevFlag == 0) {              // While selLevFlag == 0, there is
no level selected
        encoder();                         // Keep calling the encoder
function to allow us to select a level by rotating and clicking the rotary
encoder
        if (buttonFlag == 1) {             // If the encoder button has been
clicked buttonFlag will == 1 (it is 0 by default) then a level is selected
            selectLevel();                 // This will begin the selected
level and set selLevFlag to > 0 to break out of while loop
        } else {}                          // Do nothing
    }
    if (interruptFlag == 1) {              // If interruptFlag == 1, this
means a button has been pressed and we can process the interrupt
        digitalWrite(6, HIGH);             // We set built in LED to HIGH for
debugging purposes
        readInterrupt();                   // We read the values of the
interrupts/button presses and verify they are correct
    }
    timeDelay(50);                         // Time delay before read MCP
registers
    mcp.readGPIOAB();                      // Reset MCP23017 Interrupt Flag
but reading GPIO registers on port A and B
}   //End loop function
```

Appendix 3.1

```
#include "Wire.h"

void setup() {
  Wire.begin();                       // wake up I2C Bus
  Wire.beginTransmission(0x20);       // MCP23017 Address
  Wire.write(0x00);                   // IODIRA register
  Wire.write(0x00);                   // set all port A to output
  Wire.endTransmission();

  Wire.beginTransmission(0x20);       // MCP23017 Address
  Wire.write(0x01);                   // IODIRB register
  Wire.write(0x00);                   // set all port B to output
  Wire.endTransmission();
}

// Function for Port A LED ON/OFF
void ledBlinkPortA(byte a)
{
  Wire.beginTransmission(0x20);
  Wire.write(0x12);                   // address bank A
  Wire.write(a);
  Wire.endTransmission();
  delay(100);
}

// Function for Port B LED ON/OFF
void ledBlinkPortB(byte a)
{
  Wire.beginTransmission(0x20);
  Wire.write(0x13);                   // address bank B
  Wire.write(a);
  Wire.endTransmission();
  delay(100);
}

void loop() {
  ledBlinkPortA((byte)0xfe);          // LED1 ON
  ledBlinkPortA((byte)0xfd);          // LED2 ON
  ledBlinkPortA((byte)0xfb);          // LED3 ON
  ledBlinkPortA((byte)0xf7);          // LED4 ON
  ledBlinkPortA((byte)0xef);          // LED5 ON
  ledBlinkPortA((byte)0xdf);          // LED6 ON
  ledBlinkPortA((byte)0xbf);          // LED7 ON
  ledBlinkPortA((byte)0x7f);          // LED8 ON

  ledBlinkPortB((byte)0xfe);          // LED9 ON
  ledBlinkPortB((byte)0xfd);          // LED10 ON
}
```