HEART MONITORING SYSTEM

Robotics and Mechatronics

Phebe Le

Test Plans and Test Logs

Session 1.5: 5/6/2023-23/6/23

1. Full outline of design

Aim: The main design will be segmented into several different parts. The first being an armband which will store the Arduino and Ethernet shield. The second section is a 'smartwatch' contraption which contains a breadboard, LCD, 2 LEDs (red and green), a humidity sensor and a piezo.

2. Armband: Arduino and Ethernet Shield

Logical aim: To create the logical components of the armband, there will be several steps. These being:

1. Implementation of the webserver. This will be require the Ethernet shield to be configured to the network.
2. Integration of smartwatch data. The data from the smartwatch will need to be viewable in a web server
3. (If applicable) Creation of age profiles. Age profiles will be required to alert the user only when their age group BPM exceeds or beneath the average.

3. Smartwatch: Other components

Logical aim: To create the smartwatch, the logical design and implementation will be broken up into several different modules. These being:

1. Integration of sensors
    a. Pulse sensor: This will require the initial test to see if it works, changing the serial graph data to a set variable data (BPM)
    b. Humidity & temperature sensor: This will require an initial test to see if it works and is accurate (to a certain degree)
2. Integration of display components (LCD). The display should show the current BPM, temperature and humidity.
3. Integration of an alert system. The alert system will be triggered when an individual reaches a BPM that exceeds the average age range
    a. If applicable, create custom profiles for each age range group

8/7/23

Step 3.1-2. a: Smartwatch with Pulse Sensor and LCD

Aim: The first test will be only the integration of pulse sensor and the LCD screen, due to the current unavailability of a humidity sensor. The code which will be used to test this will be < https://techatronic.com/heart-beat-sensor-using-arduino-bpm-monitor/>. This code contains a simplistic architecture of the required components for this test. If this test were to work, optimisation of the code would be required to reduce the recursive and lengthy size of the sketch.

Test 1 Results:

The results for this test were favourable after debugging sections of the sketch. The test resulted in an extended amount of data through both the serial monitor and the LCD screen. Yet, these results were not accurate. The tested BPM ranged from 0-260 at various stages of the test. In attempting to resolve the varied BPM, I found that placing pressure on the sensor and moving it in different locations changed the BPM. The pulse sensor also seemed to require a strong artery pulse which resulted in the pulse from one's finger, neck, arm to have varied accuracy, whilst the heart had a steady accuracy of 250bpm, making all arteries invalid. The BPM may need to be reprogrammed as the <PulseSensorPlayground.h> was inaccurate. Due to the macro level of the code being successful, the code will be optimised through functions, a class and pointer for an LED and speaker.

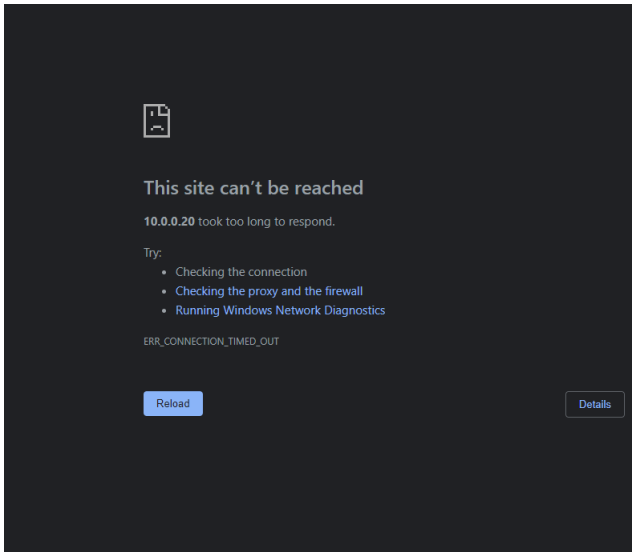
16/6/23

Step 2.1: Armband with Ethernet Shield web server

Aim: This test will be to test Ethernet Shield as well as create a functioning web server from the working Ethernet Shield. The starting code to test the Ethernet Shield and web server will be <https://startingelectronics.com/tutorials/arduino/ethernet-shield-web-server-tutorial/basic-web-server/>. If successful, the HTML of the web server will be edited to facilitate the output values of the sensors.

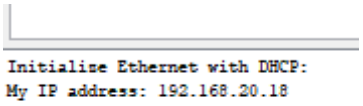
Test 1 Results:

The initial test was not favourable as the Arduino (that was correctly connected to my home router) was unable to create a web server. This test which showed flashing lights upon the shield showed that the shield was not broken but perhaps the Mac and Ip address (of the Arduino sketch) were incorrect. Based on this claim, the next test will be on changing the Mac address of the Arduino and the Ip address of the sketch to suite my home router.



Test 2 Results:

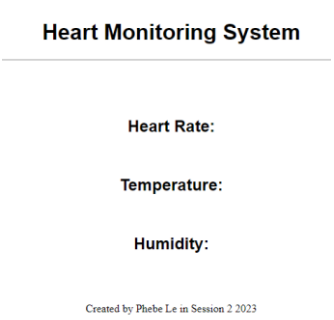
To initially find my Ip address, the use of an Arduino example sketch named “DhcpAddressPrinter” which helps to create an available Ip address for the Arduino based on the current network. The output of this was a printed serial message with a confirmation of the connection and the Ip address to use (as seen below).

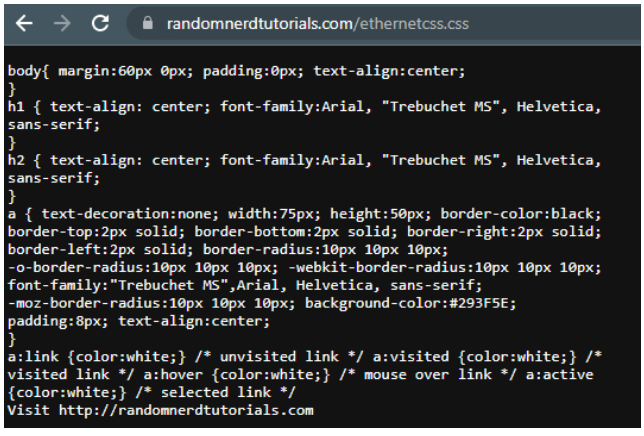


Due to this successful outcome, I replaced the Mac and Ip address of the previous failed sketch with the Mac and Ip address of this exemplar sketch. This resulted in a successful output of the intended web server (as seen on the right).

As a result, the next step will be to reiterate the current design of the web server and create a visual display for the output data of the smartwatch circuit. This screen should contain the heartbeat data, the humidity data and the temperature data.

In this image (See on right), the template for the sensors have been created. The HTML was created using basic knowledge of meta, heading, breaks etc. The style design of this web page was created from a CSS file from <Random Nerd Tutorials> (see CSS below).





This is what the current code looks like.

```
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };
IPAddress ip(192,168,5,81);
EthernetServer server(80);

void setup()
{
    Ethernet.begin(mac, ip);
    server.begin();
}

void loop()
{
    EthernetClient client = server.available();

    if (client) {  // got client?
        boolean currentLineIsBlank = true;
        while (client.connected()) {
            if (client.available()) {
                char c = client.read();
                if (c == '\n' && currentLineIsBlank) {
                    client.println("HTTP/1.1 200 OK");
                    client.println("Content-Type: text/html");
                    client.println();
                    client.println("<HTML>");
                    client.println("<HEAD>");
                    client.println("<meta charset='UTF-8'>");
                    client.println("<meta name='author' content='Phebe Le'>");
                    client.println("<meta http-equiv='X-UA-Compatiable' content='IE-edge'>");
                    client.println("<meta name='viewport' content='width=device-width, initial-scale=1.0'>");
                    client.println("<meta name='apple-mobile-web-app-capable' content='yes' />");
                    client.println("<meta name='description content='heartbeat, temperature, humidity'> ");
                    client.println("<meta name='apple-mobile-web-app-status-bar-style' content='black-translucent' />");
                    client.println("<link rel='stylesheet' type='text/css' href='https://randomnerdtutorials.com/ethernetcss.css' />");

                    client.println("<TITLE>Heart Monitoring System</TITLE>");
                    client.println("</HEAD>");
                    client.println("<BODY>");
                    client.println("<H1>Heart Monitoring System</H1>");
                    client.println("<br />");
                    client.println("<br />");
                    client.println("<H2>Heart Rate:</H2>");
                    client.println("<br />");
                    client.println("<H2>Temperature:</H2>");
                    client.println("<br />");
                    client.println("<H2>Humidity:</H2>");
                    client.println("<br />");
                    client.println("<br />");
                    client.println("<p>Created by Phebe Le in Session 2 2023</p>");
                    client.println("<br />");
                    client.println("</BODY>");
                    client.println("</HTML>");
                    break;
                }
                if (c == '\n') {
                    currentLineIsBlank = true;
                }
                else if (c != '\r') {
                    currentLineIsBlank = false;
                }
            }
        }
        delay(1);
        client.stop();
    }
}
```

Currently there are two more major steps for the logical infrastructure. These being:

1. Allow the pulse sensor to read and write to an SD card
2. Humidity sensor/temperature sensor
    a. Have a working sensor which can read data and display it on a Serial Monitor
    b. Add the humidity sensor to the <Smartwatch> code and be able to view data from it
    c. Allow the humidity sensor to read and write to an SD card
3. Combine the <Smartwatch> code and <Ethernet Armband> code

Currently, the physical design and infrastructure has not been completed yet.

18/6/23

Aim: This test will be to create a modular testing environment for the SD card to read and write data from the pulse sensor.

Test 1:

The implementation of the code was created based a combination of my previous knowledge of creating programs with SD cards and the heartbeat sensor code which was previously created. Due to my lacking knowledge this may contain errors and problems. On the left is the sketch.

Results of test 1:

This code was successful in creating a file and sensing the heartbeat (see results on the left). Based on presumption, the SD card should have written to the text file. Yet, this presumption will be tested before adding other functionalities such as malloc, vectors and pointers.

```
Initializing SD card...card initialized.
We created a heartMonitor Object !
♥  A HeartBeat Happened !
BPM: 99
sucess opening datalog.txt
♥  A HeartBeat Happened !
```

Test 2:

Aim: To test if the card wrote the text file and Arduino file named “ReadWrite” will be used. The code will be edited to only allow the reading function to be accepted. Initially the test will be testing an external file before the main file

Test 2 results: These results were favourable as it printed what was expected. See code below

Test 2.5:

When changing the name of the file to data txt., no data seemed to appear. This may a result of not closing with a SD card function named close.sd. As a result, the next step is to add the function and test. The Serial monitor on the left shows this result.

```
Initialising SD card...initialisation done.
done.
test.txt:
testing 1, 2, 3.
testing 1, 2, 3.
testing 1, 2, 3.
testing 1, 2, 3.
testing 1, 2, 3.
testing 1, 2, 3.
```

```
Initialising SD card...initialisation done.
done.
data.txt:
```

19.6.23

Aim: To fix the SD card and allow data to be transferred back and forth from the SD and sensors.

Test 1:

In the code below, the sd.close function is added and the pulse sensor code with the SD card is tested once more. This results in a strong entry of data. The correctness of this file was tested with the previous Arduino example code which resulted in success. Yet this was arranged in an illogical manner due to not adding a new line. As a result, this will be tested once more to arrange it nicely.

Pulse sensor with SD card code

Serial Monitor output of pulse sensor

```
#define USE_ARDUINO_INTERRUPTS true
#include <SD.h>
#include <PulseSensorPlayground.h>
#include <SPI.h>
#include <Ethernet.h>

byte mac[] = { 0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };
IPAddress ip(192, 168, 5, 81);
EthernetServer server(80);
const int chipSelect = 4;

const int PulseWire = 0;
int Threshold = 547;
PulseSensorPlayground heartMonitor;

void setup()
{
  Serial.begin(9600);
  Serial.print("Initialising SD card");
  if (!SD.begin(chipSelect)) {
    Serial.println("Card error");
    return;
  }
  Serial.println("card initialised.");
  heartMonitor.analogInput(PulseWire);
  heartMonitor.setThreshold(Threshold);

  if (heartMonitor.begin()) {
    Serial.println("We created a heartMonitor Object !"); //initialisation
  }
  delay(50);
}
void loop()
{
  int myBPM = heartMonitor.getBeatsPerMinute();
  if (heartMonitor.sawStartOfBeat()) {
    if (myBPM >= 130) {
      myBPM = myBPM / 2;
    }
    else if (myBPM <= 45) {
      myBPM = myBPM * 2;
    }

    Serial.println(myBPM);
    File sdcard_file = SD.open("data.txt", FILE_WRITE);

    if (sdcard_file) {
      sdcard_file.print("BPM: ");
      sdcard_file.print(myBPM);
      Serial.println("♥  A HeartBeat Happened ! ");
    Serial.print("BPM: ");
      Serial.println("sucess opening datalog.txt");
      sdcard_file.close();
      Serial.println("closed");
    }
    else {
      Serial.println("error opening datalog.txt");
    }
    delay(1000);
  }
}
```

```
Initialising SD cardcard initialised.
We created a heartMonitor Object !
107
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
110
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
112
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
92
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
83
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
81
```

The output of the initial test(successful), yet arranged oddly

```
BPM: sucess opening datalog.txt
closed
Initialising SD card...initialisation done.
data.txt:
BPM: 75BPM: 82BPM: 84BPM: 87BPM: 93BPM: 118BPM: 84BPM: 114BPM: 97BPM: 86BPM: 76BPM: 75BPM: 75BPM: 72BPM: 78BPM: 71BPM: 129BPM: 69BPM: 102BPM: 94BPM: 89BPM: 81BPM: 78BPM: 83BPM: 103BPM: 69BPM: 70BPM: 80BPM: 91BPM: 100BPM: 96BPM: 83BPM: 83BPM: 89BPM: 114B
```

Deleting the file to reset the contents of the file

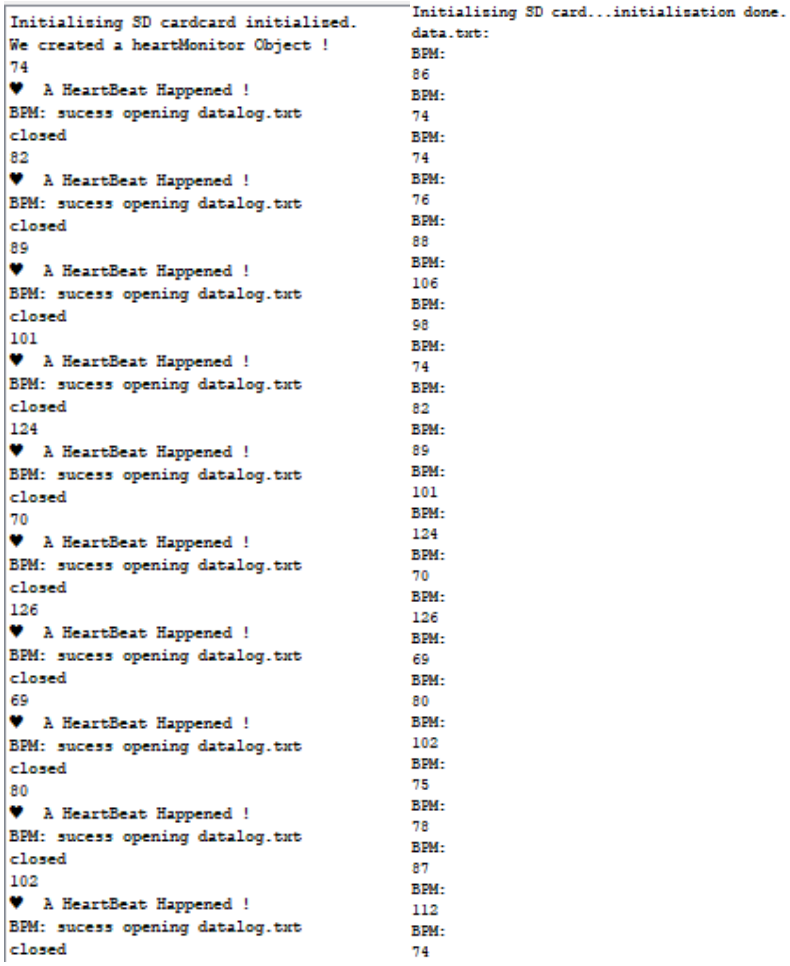
```
Initialising SD card...initialisation done.
data.txt doesn't exist.
data.txt doesn't exist.
Removing data.txt...
data.txt doesn't exist.
```

In this final test regarding the SD and pulse sensor, the results created in the same method as previously.

1. Upload pulse and SD code (with sd.close!)
2. Check Serial Monitor to make sure the results are valid

3. Upload the Arduino example code for reading the file
4. Check to see if both results are accurate

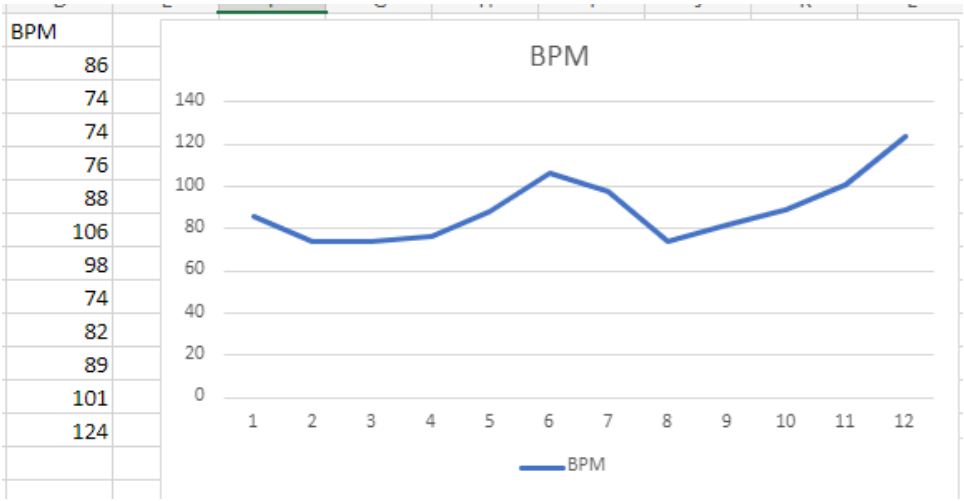
These results showed to be accurate which is successful. Although at this stage, the different initial number was confusing, yet this was explained by the order which the code was structures, SD being first and Serial being next.

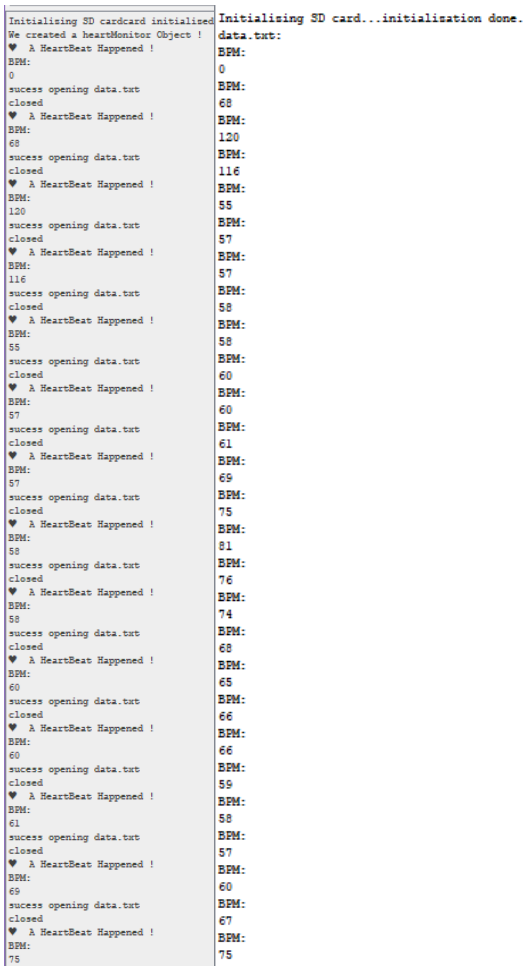


```
Initialising SD cardcard initialised.
We created a heartMonitor Object !
74
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
82
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
89
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
101
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
124
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
70
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
126
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
69
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
80
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
102
♥  A HeartBeat Happened !
BPM: sucess opening datalog.txt
closed
```

```
Initialising SD card...initialisation done.
data.txt:
BPM:
86
BPM:
74
BPM:
74
BPM:
76
BPM:
88
BPM:
106
BPM:
98
BPM:
74
BPM:
82
BPM:
89
BPM:
101
BPM:
124
BPM:
70
BPM:
126
BPM:
69
BPM:
80
BPM:
102
BPM:
75
BPM:
78
BPM:
87
BPM:
112
BPM:
74
```

The next step is to input this data into a graph to test it. This was done by using excel. The previous results were copied and the BPM line was removed so there would be one line.





This is another example of the working heart rate monitoring and data reading from the Sd file.

Below is the current code.

```
#define USE_ARDUINO_INTERRUPTS true
#include <SD.h>
#include <PulseSensorPlayground.h>
#include <LiquidCrystal_I2C.h>
#include "myClass.h"
LiquidCrystal_I2C lcd(0x27,16,2);

int speaker = 9;
const int chipSelect = 4;
MyClass redLED(5);
MyClass greenLED(6);


void setup() {
  pinMode(speaker, OUTPUT);

  Serial.begin(9600);
   Serial.print("Initialising SD card");
  if (!SD.begin(chipSelect)) {
    Serial.println("Card error");
    return;
  }
  Serial.println("card initialized.");

  File sdcard_file = SD.open("data.txt", FILE_WRITE);

  if (sdcard_file) {
    sdcard_file.println("BPM: ");
    sdcard_file.println(myBPM);
    Serial.println("♥  A HeartBeat Happened ! ");
    Serial.println("BPM: ");
    Serial.println(myBPM);
    Serial.println("sucess opening data.txt");
    sdcard_file.close();
    Serial.println("closed");
  }
  else {
    Serial.println("error opening data.txt");
  }
```

Test 2:

The next major aim is the integration of all major aspects. This being the Sd card and the smartwatch, then the ethernet shield code with the full code.

Test 2 results:

The first test showed to be successful after several tests. Yet the integration of Wi-Fi was not. The major problem with this test was the lack of storage in the Arduino. As seen below the storage overwhelms the limitations significantly

```
data section exceeds available space in board
Sketch uses 27594 bytes (85%) of program storage space. Maximum is 32256 bytes.
Global variables use 2444 bytes (119%) of dynamic memory, leaving -396 bytes for local variables
Not enough memory; see https://support.arduino.cc/hc/en-us/articles/360013825179 for tips on re
Error compiling for board Arduino Uno.
```

As a result, several unnecessary lines such as the implementation of graphics, specific html meta tags, and other external things were removed yet this continuously failed in Serial and through physical implementation. Yet the Arduino repeatedly flashed whilst

the pulse sensor was handled. Whilst this could possibly be solved by creating a library, creating more functions, decreasing the memory of strings ad other variables, the lack of time is a major impediment to doing so.

Due to this, the task will now be divided into two major sections. The first section is the current smartwatch, this will contain the main code (LCD, pulse sensor, SD card, LEDs, piezo buzzer). Logically this will be the 'mobile version' of the heart-rate monitoring system. The second section will be the ethernet-based code where the pulse sensor and the ethernet shield will work together to simultaneously display the user's heartrate through the Serial Monitor and a webserver. This will be the 'stationary version', based in a hospital.

Test 3:

Aim: the test will be centred on allowing the webserver to display the inputs of the pulse sensor.

Test 3 results:

The results for this first test were successful is displaying the BPM of the pulse sensor, yet it did not update the output data consistently. As a result, the next test for this section will be based around adding a refresh rate.

Heart Monitoring System

Heart Rate:

87

Temperature:

Humidity:

Created by Phebe Le in Session 2 2023

The code below this, is not another test. It is a result of optimising the efficiency of the code by creating functions.

20/6/23

Aim: The aim will be heavily based on implementing the required functions for the smartwatch code. These being malloc, vectors and pointers. These will be integrated to dynamically allocate memory to store within the SD. The vectors will be created through the c++ vector class. Below is the integration of the all required aspects.

```
15  const int PulseWire = 0;
16  const int BUFFER = 20; //dyunamically stores 20
17  std::vector<int> bpmBuffer;;
```

```
51  void setup() {
52    pinMode(speaker, OUTPUT);
53    Serial.begin(9600);
54    bpmBuffer.reserve(BUFFER);
55    Serial.print("Initializing SD card");
56    if (!SD.begin(chipSelect)) {
```

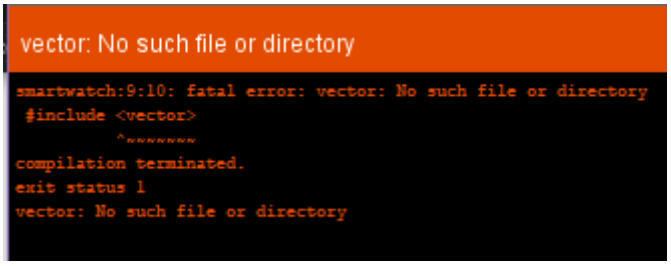
```
    bpmBuffer.push_back(myBPM);

    if (bpmBuffer.size() >= BUFFER) {
      int* dynamicBuffer = (int*)malloc(BUFFER * sizeof(int));
      if (dynamicBuffer == nullptr) {
        Serial.println("Error allocating memory");
      return;
    }
    for (int i = 0; i < BUFFER; i++) {
      dynamicBuffer[i] = bpmBuffer[i];
    }
    File sdcard_file = SD.open("data.txt", FILE_WRITE);

    if (sdcard_file) {
      for (int i = 0; i < BUFFER; i++) {
        sdcard_file.println("BPM: ");
        sdcard_file.println(dynamicBuffer[i]);
        Serial.println("♥  A HeartBeat Happened ! ");
        Serial.println("BPM: ");
        Serial.println(dynamicBuffer[i]);
        Serial.println("sucess opening data.txt");
      }
      sdcard_file.close();
      Serial.println("closed");
    }
    else {
      Serial.println("error opening data.txt");
    }
    delay(1000);
    LCDHeartView(myBPM);
    Instructions_view = 0;
    alertSystem(myBPM);
    free(dynamicBuffer);
    bpmBuffer.clear();
  }
  delay(20);
}
```

Below is the main code with a full integration.

```ino
#include <vector>
#define USE_ARDUINO_INTERRUPTS true
#include <SD.h>
#include <PulseSensorPlayground.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>

#include "myClass.h"
LiquidCrystal_I2C lcd(0x27, 16, 2);
PulseSensorPlayground heartMonitor;
int speaker = 9;
int Instructions_view = 500; //--> Variable for waiting time to display instructions on LCD.
int Threshold = 547;
const int chipSelect = 4;
const int PulseWire = 0;
const int BUFFER = 20; //dynamically stores 20
std::vector<int> bpmBuffer;

MyClass redLED(5);
MyClass greenLED(6);

MyClass::MyClass(int pin) {
  pinMode(pin, OUTPUT);
  _pin = pin;
}
void MyClass::on() {
  digitalWrite(_pin, HIGH);
}
void MyClass::off() {
  digitalWrite(_pin, LOW);
}

void MyClass::blink(int blinking) {
  digitalWrite(_pin, HIGH);
  delay(blinking);
  digitalWrite(_pin, LOW);
  delay(blinking);
}

//This section was copied froman external website to create the the heart displayed on the LCD scree
byte heart1[8] = {B11111, B11111, B11111, B11111, B01111, B00111, B00011, B00001};
byte heart2[8] = {B00011, B00001, B00000, B00000, B00000, B00000, B00000, B00000};
byte heart3[8] = {B00011, B00111, B01111, B11111, B11111, B11111, B11111, B01111};
byte heart4[8] = {B11000, B11100, B11110, B11111, B11111, B11111, B11111, B11111};
byte heart5[8] = {B00011, B00111, B01111, B11111, B11111, B11111, B11111, B11111};
byte heart6[8] = {B11000, B11100, B11110, B11111, B11111, B11111, B11111, B11110};
byte heart7[8] = {B11000, B10000, B00000, B00000, B00000, B00000, B00000, B00000};
byte heart8[8] = {B11111, B11111, B11111, B11111, B11110, B11100, B11000, B10000};
//External section ends

void setup() {
  pinMode(speaker, OUTPUT);
  Serial.begin(9600);
  bpmBuffer.reserve(BUFFER);
  Serial.print("Initialising SD card");
  if (!SD.begin(chipSelect)) {
    Serial.println("Card error");
    return;
  }
  Serial.println("card initialised.");
  lcd.init();
  lcd.backlight();
  pulsing();
  delay(2000);
  lcd.clear();
}
  lcd.clear();
}
int pulsing() {
  //This section was copied froman external website to create tl
  lcd.createChar(1, heart1);
  lcd.createChar(2, heart2);
  lcd.createChar(3, heart3);
  lcd.createChar(4, heart4);
  lcd.createChar(5, heart5);
  lcd.createChar(6, heart6);
  lcd.createChar(7, heart7);
  lcd.createChar(8, heart8);
  //External section ends

  lcd.setCursor(0, 0);
  lcd.print(" Heart Monitor ");
  lcd.setCursor(0, 1);
  lcd.print("    System     ");
  heartMonitor.analogInput(PulseWire);
  heartMonitor.setThreshold(Threshold);

  if (heartMonitor.begin()) {
    Serial.println("We created a heartMonitor Object !"); //ini
  }
}
int initial(int a) {
  if (a < 500) {
    a++;
  }
  else if (a > 499) {
    lcd.setCursor(0, 0);
    lcd.print("Put your finger ");
    lcd.setCursor(0, 1);
    lcd.print("on the sensor   ");
    delay(1000);
    lcd.clear();
    delay(500);
  }
}

int LCDHeartView(int bpm) {
  lcd.setCursor(1, 1);
  lcd.write(byte(1));
  lcd.setCursor(0, 1);
  lcd.write(byte(2));
  lcd.setCursor(0, 0);
  lcd.write(byte(3));
  lcd.setCursor(1, 0);
  lcd.write(byte(4));
  lcd.setCursor(2, 0);
  lcd.write(byte(5));
  lcd.setCursor(2, 0);
  lcd.write(byte(6));
  lcd.setCursor(2, 1);
  lcd.write(byte(7));
  lcd.setCursor(2, 1);
  lcd.write(byte(8));
  lcd.setCursor(5, 0);
  lcd.print("Heart Rate");
  lcd.setCursor(5, 1);
  lcd.print(": ");
  lcd.print(bpm);
  lcd.print(" ");
  lcd.print("BPM     ");
}
int alertSystem(int bpm) {
```

```ino
  lcd.print(bpm);
  lcd.print(" ");
  lcd.print("BPM     ");
}
int alertSystem(int bpm) {
  if (bpm > 100 || bpm < 55 ) {
    int* p_speaker = &speaker; // create a pointer to the pin v
    greenLED.off();
    redLED.blink(500);
    tone(*p_speaker, 262);
    delay(200);
    noTone(*p_speaker);
    delay(200);
  }
  else {
    greenLED.on();
  }
}
void loop() {
  int myBPM = heartMonitor.getBeatsPerMinute();

  initial(Instructions_view);
  if (heartMonitor.sawStartOfBeat()) {
    if (myBPM >= 120) {
      myBPM = myBPM / 2;
    }
    else if (myBPM <= 50) {
      myBPM = myBPM * 2;
    }
    bpmBuffer.push_back(myBPM);

    if (bpmBuffer.size() >= BUFFER) {
      int* dynamicBuffer = (int*)malloc(BUFFER * sizeof(int));
      if (dynamicBuffer == nullptr) {
        Serial.println("Error allocating memory");
        return;
      }
      for (int i = 0; i < BUFFER; i++) {
        dynamicBuffer[i] = bpmBuffer[i];
      }
      File sdcard_file = SD.open("data.txt", FILE_WRITE);

      if (sdcard_file) {
        for (int i = 0; i < BUFFER; i++) {
          sdcard_file.println("BPM: ");
          sdcard_file.println(dynamicBuffer[i]);
          Serial.println("♥  A HeartBeat Happened ! ");
          Serial.println("BPM: ");
          Serial.println(dynamicBuffer[i]);
          Serial.println("sucess opening data.txt");
        }
        sdcard_file.close();
        Serial.println("closed");
      }
      else {
        Serial.println("error opening data.txt");
      }
      delay(1000);
      LCDHeartView(myBPM);
      Instructions_view = 0;
      alertSystem(myBPM);
      free(dynamicBuffer);
      bpmBuffer.clear();
    }
    delay(20);
  }
```

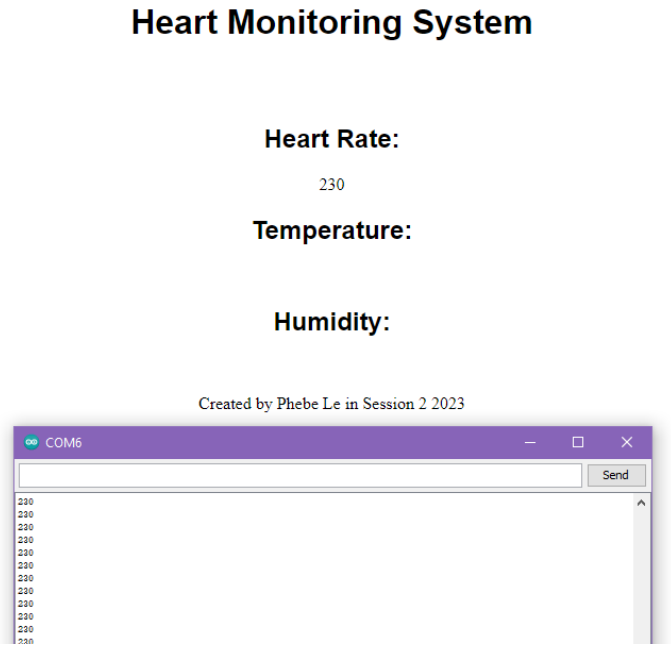
```ino
#ifndef MyClass_h
#define MyClass_h
#include <Arduino.h>

class MyClass {
  public:
    MyClass(int pin);
    void on();
    void off();
    void blink(int blinking);
  private:
    int _pin;
};

#endif
```



When debugging, there was an error due to the lack of proper inclusion of vector. A strange concept which was discovered was when changing from the stopwatch to the ethernet there were errors. This may be due to the previous code continuously running and not allowing the code to be completely uploaded. As a result, I found to run the ethernet code, the board had to be unplugged and replugged to successfully work. This had to be applied for the stopwatch code as well.

Test 1: The aim for this test is displaying the pulse sensor output into webserver.

Test 1 result: This result was recreating the previous response and this was successful.



The code for this test is below.

wifi2

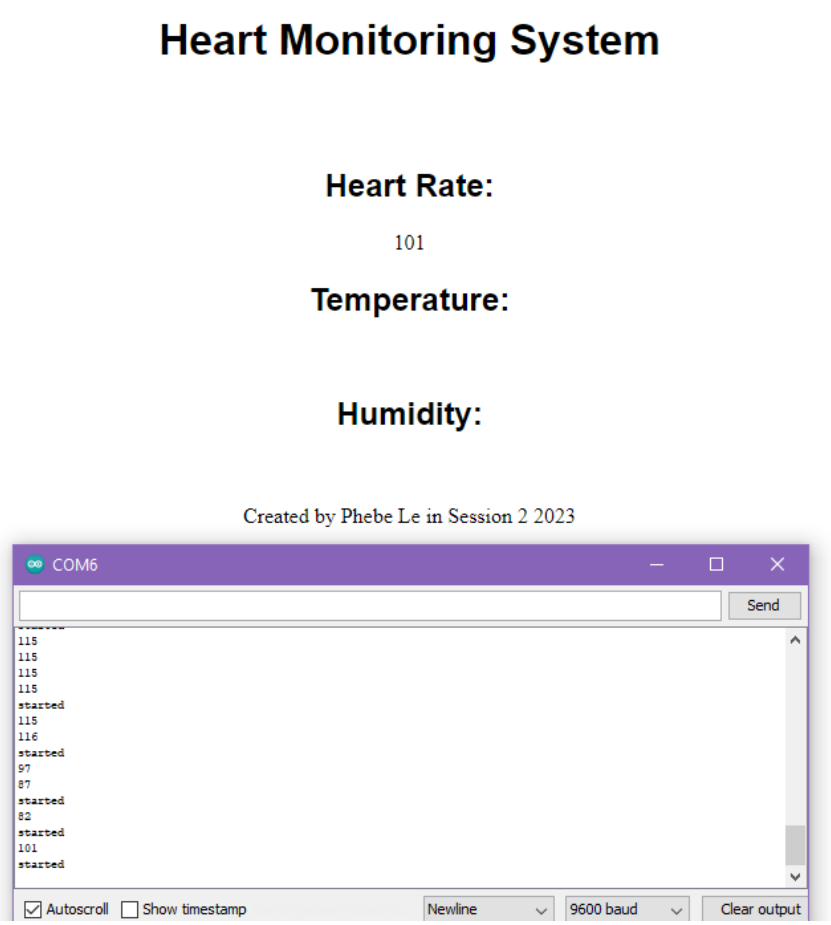
```
#define USE_ARDUINO_INTERRUPTS true
#include <SPI.h>
#include <Ethernet.h>
#include <PulseSensorPlayground.h>

byte mac[] = { 0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };
IPAddress ip(192, 168, 5, 81);
EthernetServer server(80);

const int PulseWire = 0;
int Threshold = 547;
PulseSensorPlayground heartMonitor;

void setup()
{
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.begin(9600);
  heartMonitor.analogInput(PulseWire);
  heartMonitor.setThreshold(Threshold);
    if (heartMonitor.begin()) {
    Serial.println("We created a heartMonitor Object !"); //initialisation
  }delay(50);
}

void loop()
{
  EthernetClient client = server.available();
  int myBPM = heartMonitor.getBeatsPerMinute();
  Serial.println(myBPM);
if (heartMonitor.sawStartOfBeat()) {
  if (client) {  // got client?
    boolean currentLineIsBlank = true;
    Serial.print("started");
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        if (c == '\n' && currentLineIsBlank) {
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println();
          client.println("<HTML>");
          client.println("<HEAD>");
          client.println("<meta charset='UTF-8'>");
          client.println("<meta name='author' content='Phebe Le'>");
          client.println("<meta http-equiv='X-UA-Compatiable' content='IE-edge'>");
          client.println("<meta name='viewport' content='width=device-width, initial-scale=1.0'>");
          client.println("<meta name='apple-mobile-web-app-capable' content='yes' />");
          client.println("<meta name='description content='heartbeat, temperature, humidity'> ");
          client.println("<meta name='apple-mobile-web-app-status-bar-style' content='black-translucent' />");
          client.println("<link rel='stylesheet' type='text/css' href='https://randomnerdtutorials.com/ethernetcss.css' />");

          client.println("<TITLE>Heart Monitoring System</TITLE>");
          client.println("</HEAD>");
          client.println("<BODY>");
          client.println("<H1>Heart Monitoring System</H1>");
          client.println("<br />");
          client.println("<br />");
          client.println("<H2>Heart Rate:</H2>");

             client.println(myBPM);
          client.println("<TITLE>Heart Monitoring System</TITLE>");
          client.println("</HEAD>");
          client.println("<BODY>");
          client.println("<H1>Heart Monitoring System</H1>");
          client.println("<br />");
          client.println("<br />");
          client.println("<H2>Heart Rate:</H2>");

              client.println(myBPM);


          client.println("<br />");
          client.println("<H2>Temperature:</H2>");
          client.println("<br />");
          client.println("<H2>Humidity:</H2>");
          client.println("<br />");
          client.println("<p>Created by Phebe Le in Session 2 2023</p>");
          client.println("<br />");
          client.println("</BODY>");
          client.println("</HTML>");
          break;
        }

        if (c == '\n') {
          currentLineIsBlank = true;
        }
        else if (c != '\r') {
          currentLineIsBlank = false;
        }
      }
    }
    delay(1);
    client.stop();
  }
  }
}
```

Test 2:

Aim: The next test will be creating a refresh rate for the webserver to update the data values and adding restrictions to limit the BPM range.

Test 2 results:

These results showed to be successful yet inconsistent. As seen from the image below the data from the webserver is accurate with the current version of the Serial monitor. The started entry in the Serial Monitor is an indication of the webserver's refresh. Yet there was a delay due to the speed of the internet.



Test 3:

Aim: To add restrictions to the pulse sensor's BPM. This will be copied from the smartwatch code.

The finished ethernet and pulse sensor code is below.

```
if (heartMonitor.sawStartOfBeat()) {
      if (myBPM >= 110) {
      myBPM = myBPM / 2;
    }
    else if (myBPM <= 50) {
      myBPM = myBPM * 2;
    } else if (myBPM == 0){
      myBPM = myBPM+50;
      }
   Serial.println(myBPM);
  if (client) {  // got client?
    boolean currentLineIsBlank = true;
    Serial.println("started");
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        if (c == '\n' && currentLineIsBlank) {
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Refresh: 1");
```

```
/*Created by Phebe Le | Session 2 | Heart Monitoring System
 *
 * This project is replicating a heart-rate watch system that uses output: LEDs,
 * a busser, LCD display screen. Inputs: Pulse-Sensor. It also includes an SD
 * card module which reads and writes data
 *
 * Due to the lack of space in the Arduino, the system is split up into two different modules
 * a stationary system and mobile system.
 *
 * This section is Part 2: The stationary system consisting of a webserver with Ethernet connection
 * and a pulse sensor
 */


#define USE_ARDUINO_INTERRUPTS true
#include <SPI.h>
#include <Ethernet.h>
#include <PulseSensorPlayground.h>

byte mac[] = { 0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02 };
IPAddress ip(192, 168, 5, 81);
EthernetServer server(80);

const int PulseWire = 0;
int Threshold = 547;
PulseSensorPlayground heartMonitor;

void setup()
{
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.begin(9600);
  heartMonitor.analogInput(PulseWire);
  heartMonitor.setThreshold(Threshold);
  if (heartMonitor.begin()) {
    Serial.println("We created a heartMonitor Object !"); //initialisation
  } delay(50);
}

void loop()
{
  EthernetClient client = server.available();
  int myBPM = heartMonitor.getBeatsPerMinute();

  if (heartMonitor.sawStartOfBeat()) {
    if (myBPM >= 110) {
      myBPM = myBPM / 2;
    }
    else if (myBPM <= 50) {
      myBPM = myBPM * 2;
    } else if (myBPM == 0) {
      myBPM = myBPM + 50;
    }
    Serial.println(myBPM);
    if (client) {  // got client?
      boolean currentLineIsBlank = true;
      Serial.println("started");
      while (client.connected()) {
        if (client.available()) {
          char c = client.read();
          if (c == '\n' && currentLineIsBlank) {
            client.println("HTTP/1.1 200 OK");
            client.println("Content-Type: text/html");
            client.println("Refresh: 1");
            client.println();
```

```
    Serial.println(myBPM);
    if (client) {  // got client?
      boolean currentLineIsBlank = true;
      Serial.println("started");
      while (client.connected()) {
        if (client.available()) {
          char c = client.read();
          if (c == '\n' && currentLineIsBlank) {
            client.println("HTTP/1.1 200 OK");
            client.println("Content-Type: text/html");
            client.println("Refresh: 1");
            client.println();
            client.println("<HTML>");
            client.println("<HEAD>");
            client.println("<meta charset='UTF-8'>");
            client.println("<meta name='author' content='Phebe Le'>");
            client.println("<meta http-equiv='X-UA-Compatiable' content='IE-edge'>");
            client.println("<meta name='viewport' content='width=device-width, initial-scale=1.0'>");
            client.println("<meta name='apple-mobile-web-app-capable' content='yes' />");
            client.println("<meta name='description content='heartbeat, temperature, humidity'> ");
            client.println("<meta name='apple-mobile-web-app-status-bar-style' content='black-translucent' />");
            client.println("<link rel='stylesheet' type='text/css' href='https://randomnerdtutorials.com/ethernetcss.css' />");

            client.println("<TITLE>Heart Monitoring System</TITLE>");
            client.println("</HEAD>");
            client.println("<BODY>");
            client.println("<H1>Heart Monitoring System</H1>");
            client.println("<br />");
            client.println("<br />");
            client.println("<H2>Heart Rate:</H2>");
            client.println(myBPM);
            client.println("<br />");
            client.println("<br />");
            client.println("<p>Created by Phebe Le in Session 2 2023</p>");
            client.println("<br />");
            client.println("</BODY>");
            client.println("</HTML>");
            break;
          }
          if (c == '\n') {
            currentLineIsBlank = true;
          }
          else if (c != '\r') {
            currentLineIsBlank = false;
          }
        }
      }
      delay(1);
      client.stop();
    }
  }
}
```

21/3/23

Aim: To run the vector library and allow the smartwatch code to run

```
}
void loop() {
  int myBPM = heartMonitor.get
  initial(Instructions_view);

  int memory[BUFFER];
  bpmBuffer vector;
  vector.setStorage(memory);
  Serial.print("1");
  if (heartMonitor.sawStartOfB
    if (myBPM >= 120) {
      myBPM = myBPM / 2;
    }
    else if (myBPM <= 50) {
      myBPM = myBPM * 2;
    }
    Serial.print("1.5");
    vector.push_back(BUFFER);
    Serial.print("2");
    if (vector.size() >= BUFFE
      Serial.print("3");
      int* dynamicBuffer = (in
      Serial.print("4");
      if (dynamicBuffer == nul
        Serial.println("Error
```

Output   Serial Monitor ×

Message (Enter to send message to 'Arduino Uno'

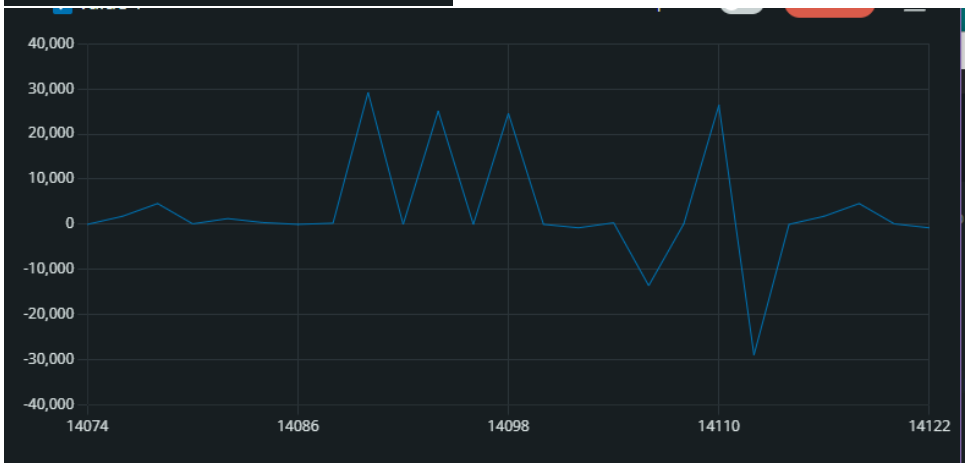
```
Initializing SD cardcard initialized.
We created a heartMonitor Object !
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
We created a heartMonitor Object !
11111111111111111111111111111111111□□□□□
We created a heartMonitor Object !
11111111111111111111111111111111
```

```
}
void loop() {
  int myBPM = heartMonitor.getBeatsPerMinute();
  initial(Instructions_view);

  int memory[BUFFER];
  bpmBuffer vector;
  vector.setStorage(memory);
  Serial.println("1");
  if (heartMonitor.sawStartOfBeat()) {
    if (myBPM >= 120) {
      myBPM = myBPM / 2;
    } else if (myBPM <= 50) {
      myBPM = myBPM * 2;
    }
    Serial.println("1.5");

    Serial.println("2");
    if (vector.size() >= BUFFER) {
      Serial.println("3");
      int* dynamicBuffer = (int*)malloc(BUFFER * sizeof(int));
      Serial.println("3-4");
      Serial.println("4");
      if (dynamicBuffer == nullptr) {
        Serial.println("Error allocating memory");
        return;
      }
```

```
        }
        sdcard_file.close();
        Serial.println("closed");
      } else {
        Serial.println("error opening data.txt");
      }
      delay(1000);
      LCDHeartView(myBPM);
      Instructions_view = 0;
      alertSystem(myBPM);
      free(memory);
      // return 0;
      // return;
    }
    delay(20);
  }
}
```

```
1
Initializing SD cardcard initialized.
We created a heartMonitor Object !
```

```
7
8
♥  A HeartBeat Happened !
BPM:
25960
sucess opening data.txt
7
8
♥  A HeartBeat Happened !
BPM:
29472
sucess opening data.txt
7
8
♥  A HeartBeat Happened !
BPM:
28261
sucess opening data.txt
7
8
♥  A HeartBeat Happened !
BPM:
28531
```



Within these several pieces of codes and outputs. There has been one major focus which is to allow the vectors to dynamically store the allocated malloc of data needed. These results whilst are accurate in storing them dynamically, the BPM value is absurdly inaccurate. This may be due to the use of vectors, whilst being dynamically allocated. Alongside this the alert system and the LCD functions do not work. As a result, the first requirement will be to create stronger restrictions for this data range, and then change the location of the functions to be within the SD writing loop.

After editing and taking these steps the results show to be favourable (see below). The inclusion of serial print 7 and 8 are also included due to testing the accuracy and debugging errors. In counting this, I also found that the buffer function also works correctly as when it reaches 20 data entries the vector resets and begins the process

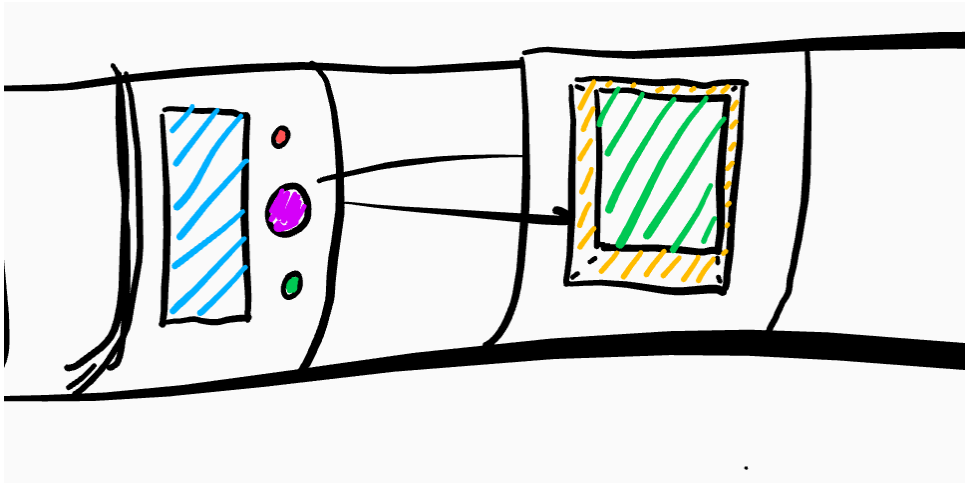
once again (see the 'closed' line in the first image). The restrictions also worked after changing different numbers and testing it (see second image). The third image is reading the file that had stored the data values and plotting it onto a graph.

```
8
♥  A HeartBeat Happened !
BPM:
1792
sucess opening data.txt
7
8
♥  A HeartBeat Happened !
BPM:
4609
sucess opening data.txt
7
8
♥  A HeartBeat Happened !
BPM:
114
sucess opening data.txt
7
8
♥  A HeartBeat Happened !
BPM:
15616
sucess opening data.txt
closed
```

```
    memory[i] = memory[i] - 80;
  } else if (memory[i] <= 50 && memory[i]
    //Serial.println("50/40");
    memory[i] = memory[i] * 2;
  } else if (memory[i] <= 39) {
    //Serial.println("39");
    memory[i] = memory[i] + 70;
  }
  // } else if (memory[i] < 10) {
  //   memory[i] = 50 + memory[i];
```

Housing Development

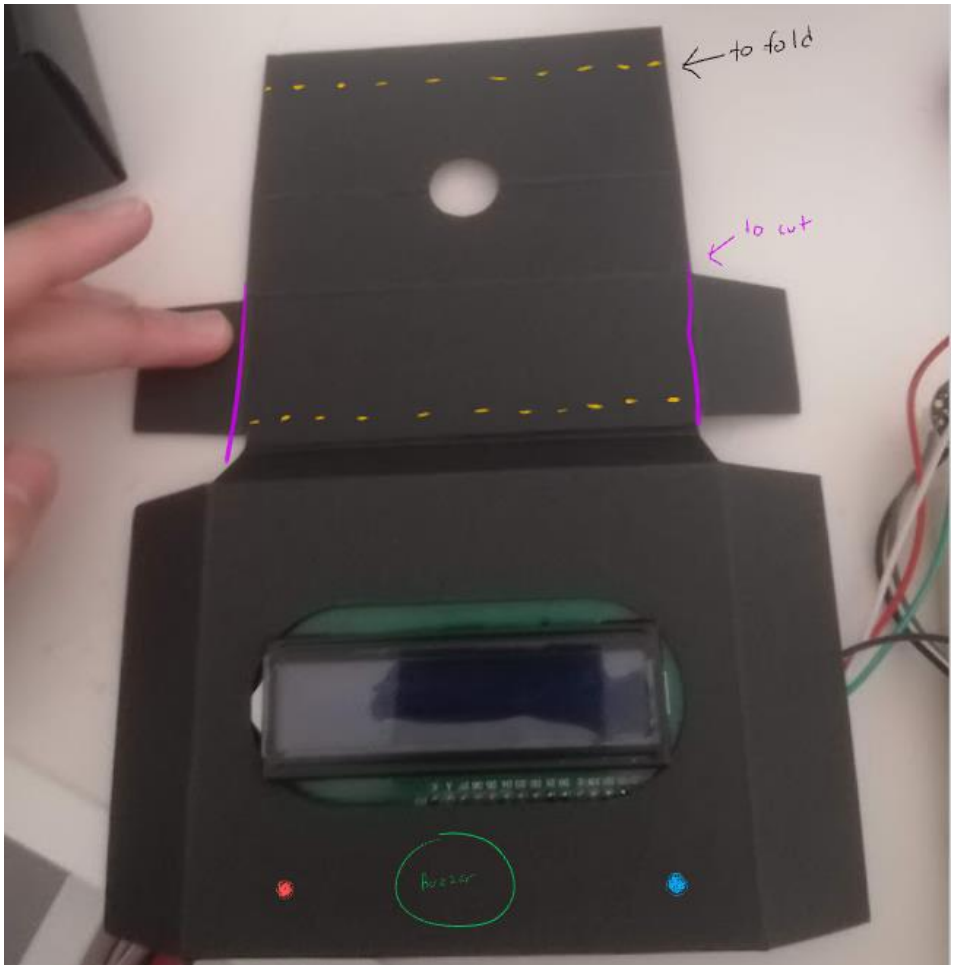
To develop the two sections, I will segregate the design into two pieces. The stopwatch and the ‘arm piece’. The arm piece will contain cabling, the Arduino shield and Arduino uno. The smartwatch will contain all other components. The initial sketch of the current design is shown below.



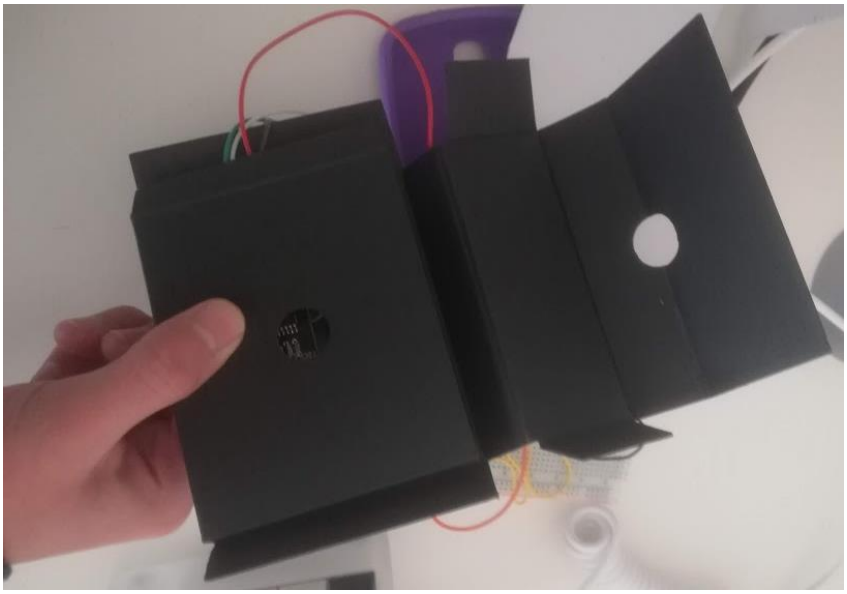
The smartwatch

Currently the main goals for this section is to make the smart device manageable and wearable whilst containing all required elements.

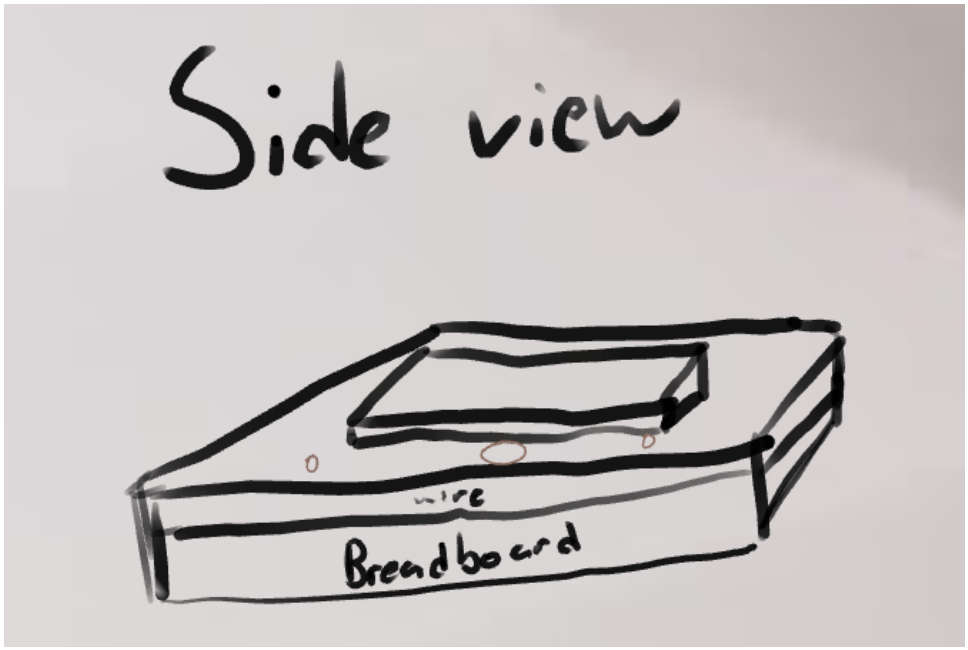
Using old carboard from smart devices, a piece that fit the LCD screen was found. This may be the main template of the design. This will be attached to elastic or a strong material which withstands heavy weights



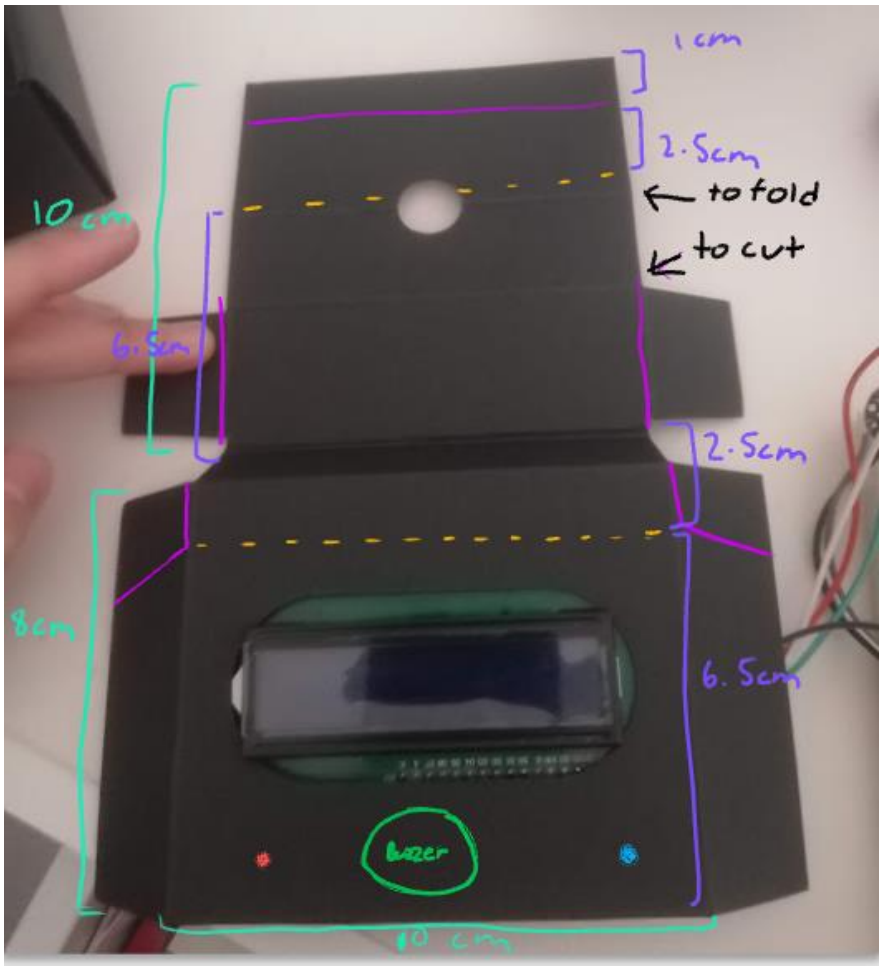
For this design the main plan is to use this box like shape to create the top of the smartwatch that is viewable to others. This will contain an LCD screen, a buzzer and two LEDs. Due to the design of this piece of carboard a two-layer box will be applicable (see the carboard fold on the right)



A drawing of this visualisation can be seen below



Yet despite this material, the size is significantly too large to be placed on a wrist. Due to this, I created a plan to edit the design.



The green lines with the measurements show the cardboards measurements. Whilst the purple is used to show the measurement that will be implemented onto the cardboard. The pink line will be places to cut. The yellow line illustrates the places to bend. These will be created tomorrow.

Wristband

To create the wristband another piece of cardboard will be required. The top facing part of the wristband of this will be glued to the bottom of the breadboard's cardboard. This is what it will look like when folded correctly. Whilst, this design is lacking in regards to aesthetics, due to time constraints this will be the best solution. If there was to be more

time, I would use a softer material such as an elastic band and attach a fabric-like material beneath the smartwatch to create a more comfortable design.

Wristband

1. Cut out the cardboard

2. Make a hole and stick a metal cable tie through it

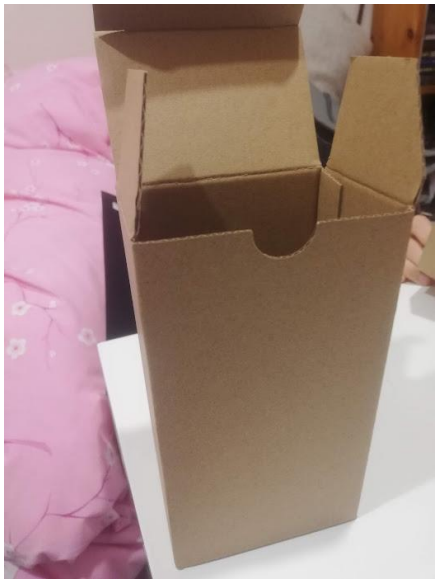
3.On the other side place a clip to slot the table tie through

4.Connect the cable tie with the clip

End result:

- I dislike my result so I will opt for just using a cardboard box and cut off the back and front. To be able to sense heartbeat through the contraption, a hold was added.

Initial result



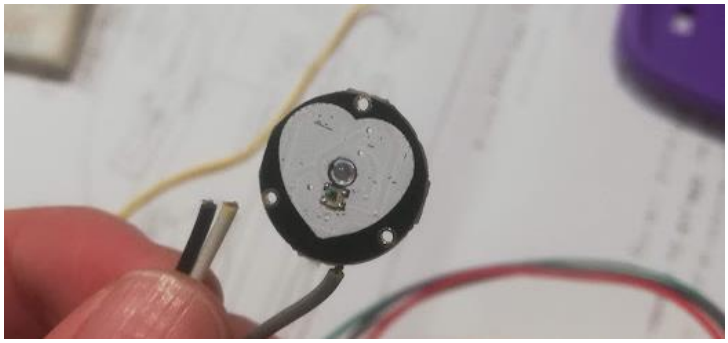
New concept

22/3/23

The initial structure of the smartwatch was adhered to as previous instructions state and this was the result. The concept the design has also changed as the Arduino, Ethernet shield and breadboard will be on one major piece instead of two separate pieces

This result was better than expected as it allowed room for the ethernet shield and the Arduino. This led me to assemble the main circuit on the smaller breadboard. Yet whilst assembling a major issue occurred: the heart sensor broke. If possible, this may be fixed

tomorrow but to temporarily (and unreliably) solve this I have sticky taped the wires to the pulse sensors which allows it to work 30% of the time.



Despite this issue, I continued to build the housing which resulted in this. Whilst I am happy with the general design of the watch, I also think that this design is significantly larger than the average watch.

The wristband-like concept was then added and aligned with the smartwatch housings hole.