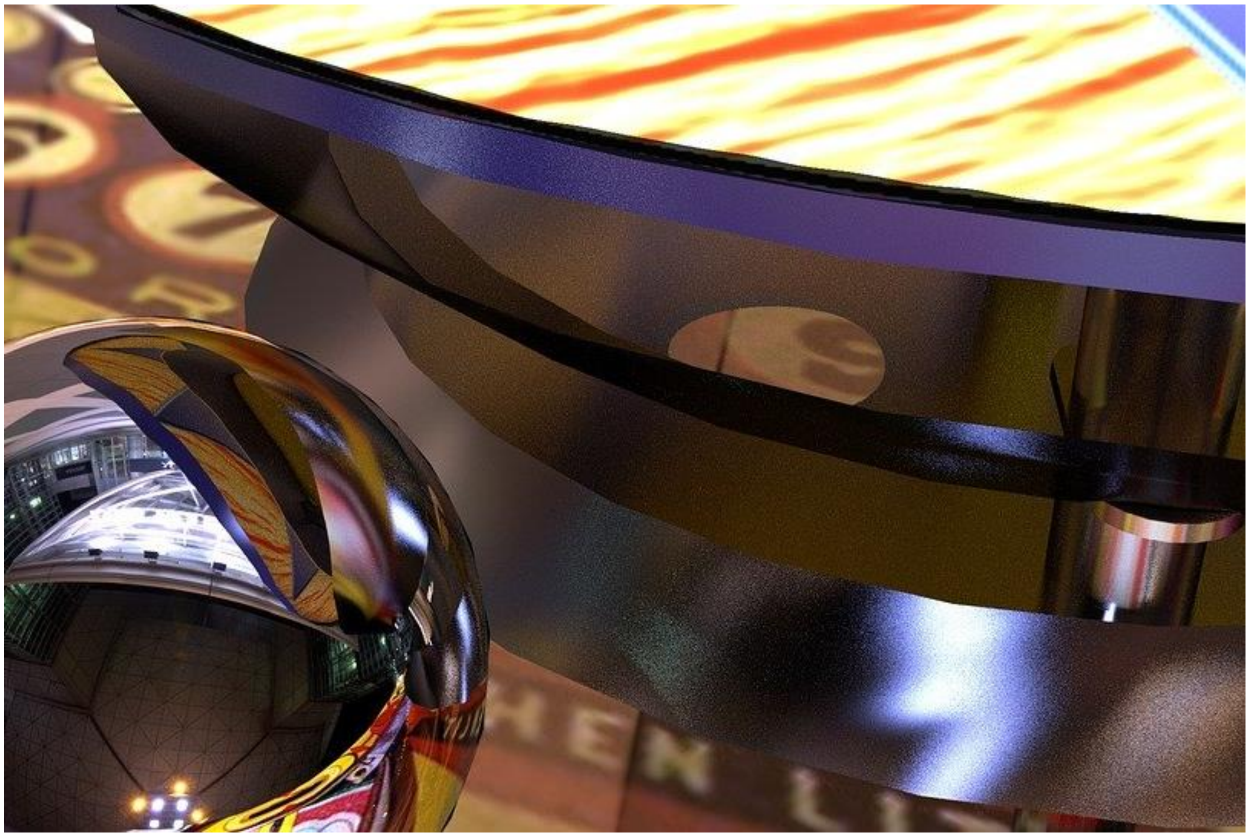


CISC 340 Project Report

26.11.2019

—

Team # 4

Luca Ciampi | 20193575 | 19crc2@queensu.ca

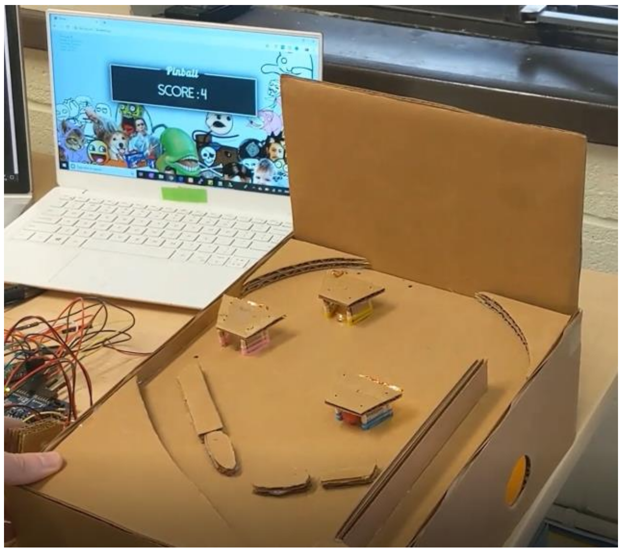
Matt Denny | 20065190 | 16mmmd1@queensu.ca

Ashley Drouillard | 10183354 | 14aad4@queensu.ca

Danielle Edward | 20062058 | 16dke@queensu.ca

Table of Contents

Outline



Entertainment can come in many shapes and forms, with varying levels of complexity. We decided to create a pinball machine that would integrate entertainment and digital systems. Pinball is a type of arcade game, in which points are scored by a player manipulating one or more balls on a play field. The goal of the game is to score as many points as possible.

Our pinball machine was created using photocell sensors, a piezo buzzer, the Basys3 pushbuttons and seven segment display, as well as a multitude of wires and other important pieces. The main objective of our project was to create an interactive object that showcased the ability of a simple logic function while creating an enjoyable experience for the user. There were many different segments used when building our contraption; we incorporated direct user input to trigger the sensors and run the machine, while the sensors and boards process the signals and direct the information to the website, keeping a real time score as the user interacts with the machine. When the user scores, (the ball rolls over one of the three point photocell sensors), the signal is sent to the Basys3 and from there through the raspberry pi to update the current score on the website. If the ball triggers the reset photocell, it resets the user’s current score to zero. The user’s previous score is saved and outputted back to the Basys3. At this point, the user can press the pushbutton on the Basys3 to display their previous score.

Problems Encountered

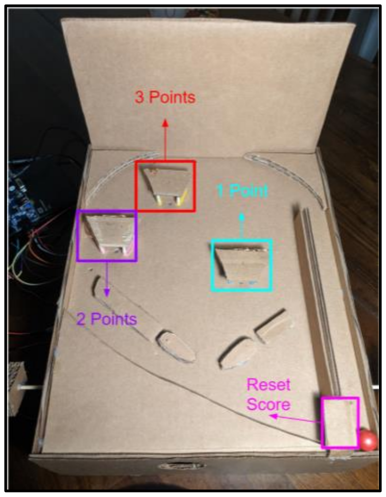
As noted in the project documentation, getting a combined hardware/software project that uses different software platforms and components to work properly is no easy task. Through the duration of our project, our team encountered various problems. Some notable issues involve communication to and from the raspberry pi and the Basys3 board.

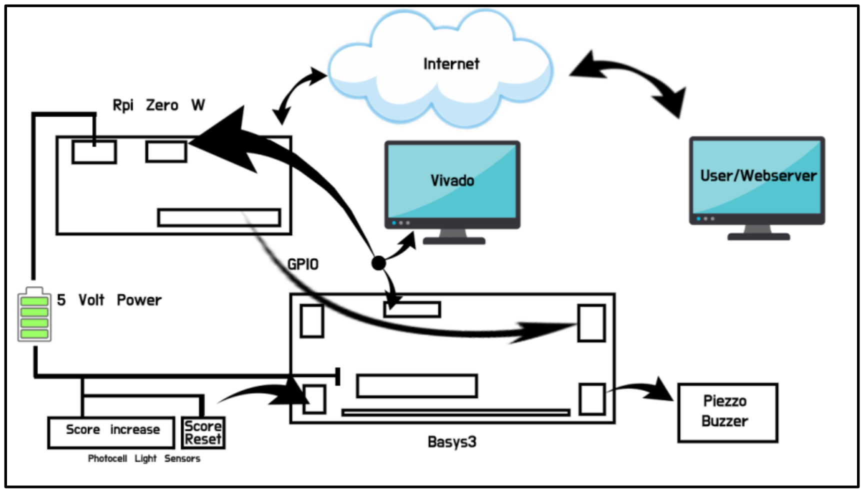
One of our first issues involved displaying output from the basys3 board to the raspberry pi via minicom. For some unknown reason, our raspberry pi had issues running minicom. After some assistance from the instructor, we replaced the SD card for the raspberry pi which seemed to fix the issue.

Another problem we ran into was displaying output from the raspberry pi to the basys3 board. Our raspberry pi was sending back an 8 bit value to the basys3. With all the wiring required to achieve this, it was very difficult to keep track of which wire was attached to which pin. When outputting data from the raspberry pi, there was a lot of wrong values getting passed back because the pins were not in the correct position. After making various tweaks to the GPIOHandler.py, we finally found the correct ordering of the pins to send the correct value back to the basys3. After this, we left the pins untouched and kept the basys3 board and rpi attached so we would not get the ordering mixed up.

Hardware

Schematic Diagram



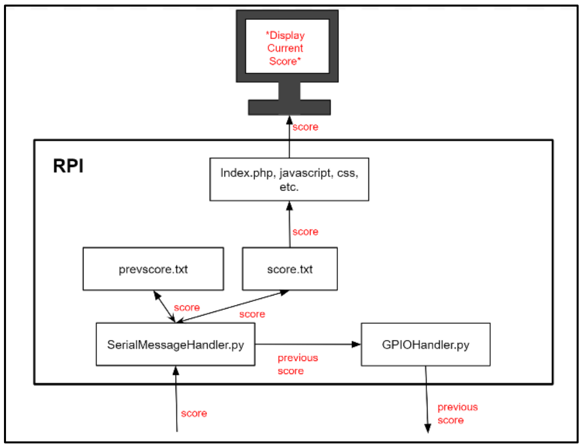


Test methods

When testing the Basys3 Board by itself, we displayed the score on the Basys3 Board to ensure the score was displaying correctly on the seven segment display. We tested the system in various lightings to ensure that the photocell sensors were consistent in capturing the light change when the ball rolls over a sensor. We also tested that the score would not exceed 255 points because we can only store up to 8 bits of information through the 8 GPIO pins from the raspberry pi. Additionally, we tested the output to Tera Term to ensure the correct serial messages would be sent to the raspberry pi.

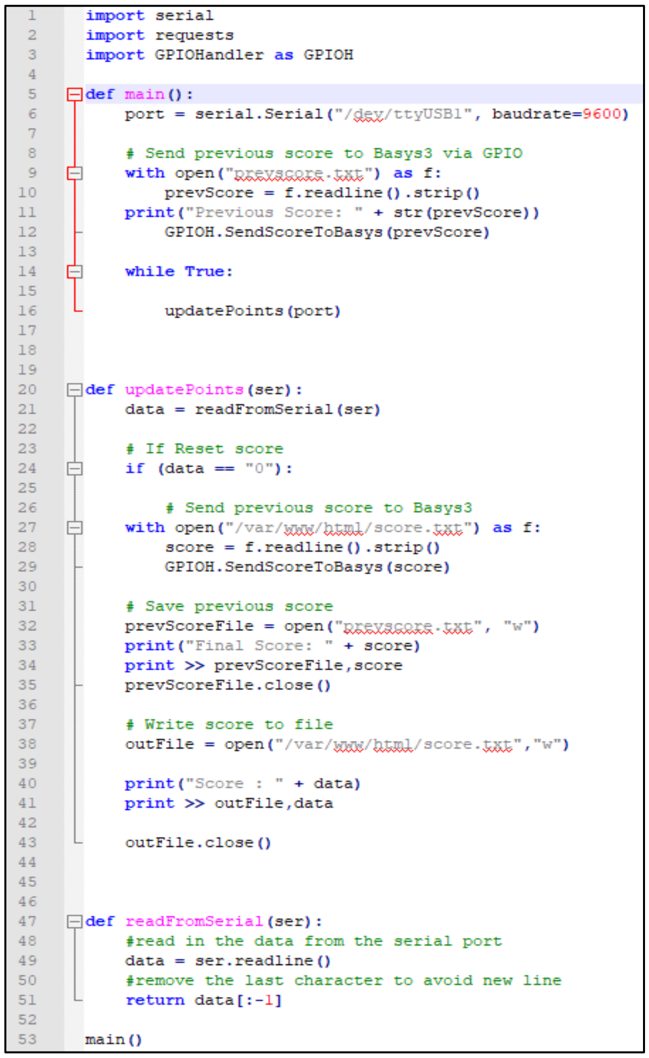
Raspberry Pi

The main purpose of the raspberry pi was to handle the incoming score from the Basys3 and ensure it reaches all the needed locations. When the score is updated, it is received by `SerialMessageHandler.py` and sent to `score.txt` to be used by the webserver. The previous score is sent to `prevscore.txt` to be stored for later and is also sent to `GPIOHandler.py` to be sent back to the Basys3.



Serial I/O

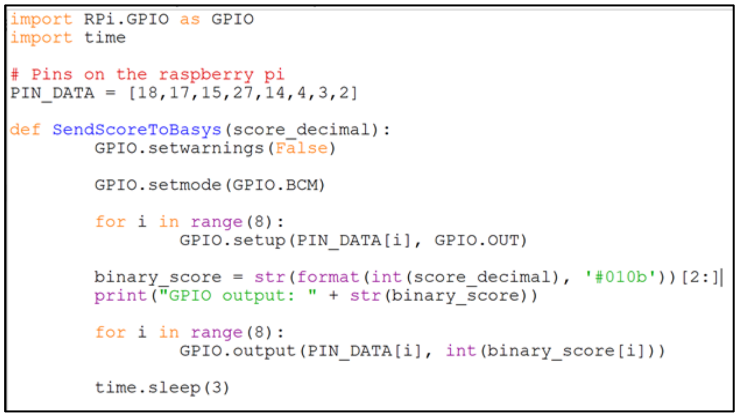
When it comes to receiving data from the Basys3, the script `SerialMessageHandler.py` is responsible for listening to the serial port `/dev/ttyUSB1` for all incoming score data from the Basys3. When new data is sent, the script reads the score from the serial port. It will then take the previous score and save it to the file `prevscore.txt` for storage. The current score will then be pushed to the `score.txt` so that it can be displayed on the web server.



```
import serial
import requests
import GPIOHandler as GPIOH

def main():
    port = serial.Serial("/dev/ttyUSB1", baudrate=9600)

    # Send previous score to Basys3 via GPIO
    with open("prevscore.txt") as f:
        prevScore = f.readline().strip()
    print("Previous Score: " + str(prevScore))
        GPIOH.SendScoreToBasys(prevScore)

    while True:

        updatePoints(port)



def updatePoints(ser):
    data = readFromSerial(ser)

    # If Reset score
    if (data == "0"):

        # Send previous score to Basys3
    with open("/var/www/html/score.txt") as f:
        score = f.readline().strip()
        GPIOH.SendScoreToBasys(score)

    # Save previous score
    prevScoreFile = open("prevscore.txt", "w")
    print("Final Score: " + score)
    print >> prevScoreFile,score
    prevScoreFile.close()

    # Write score to file
    outFile = open("/var/www/html/score.txt","w")

    print("Score : " + data)
    print >> outFile,data

    outFile.close()



def readFromSerial(ser):
    #read in the data from the serial port
    data = ser.readline()
    #remove the last character to avoid new line
    return data[:-1]

main()
```

GPIO Output

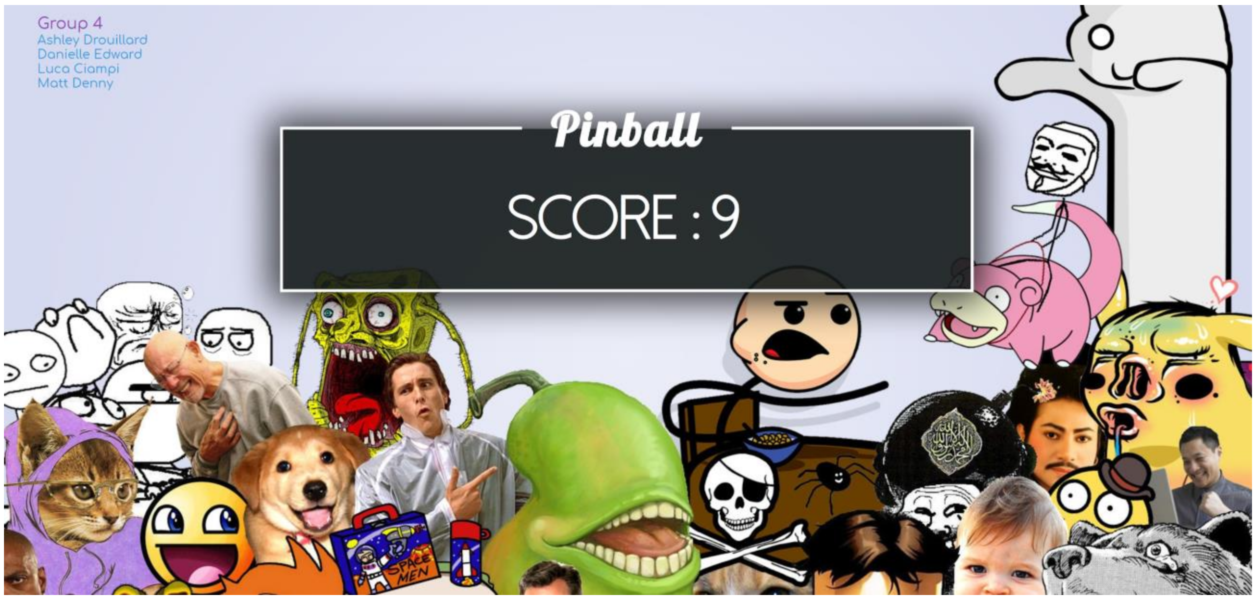
The Raspberry Pi outputs the previous score to the Basys3. The `GPIOHandler.py` file handles the output of the pi. It takes the previous score, converts it to 8 bits, and outputs a signal from each of the GPIO pins. This is done at the beginning of the execution of `SerialMessagesHandler.py`, and whenever the score is reset.



```
import RPi.GPIO as GPIO
import time

# Pins on the raspberry pi
PIN_DATA = [18,17,15,27,14,4,3,2]

def SendScoreToBasys(score_decimal):
        GPIO.setwarnings(False)

        GPIO.setmode(GPIO.BCM)

        for i in range(8):
                GPIO.setup(PIN_DATA[i], GPIO.OUT)

        binary_score = str(format(int(score_decimal), '#010b'))[2:]
        print("GPIO output: " + str(binary_score))

        for i in range(8):
                GPIO.output(PIN_DATA[i], int(binary_score[i]))

        time.sleep(3)
```

End Data Application

Web Browser



In order to track the score of the pinball game, we used the Raspberry Pi Zero W to host a web server and display the score on a web page.
In order to do so, the first thing we did was ensure that the Raspberry has a Wifi connection, and that SSH is enabled.

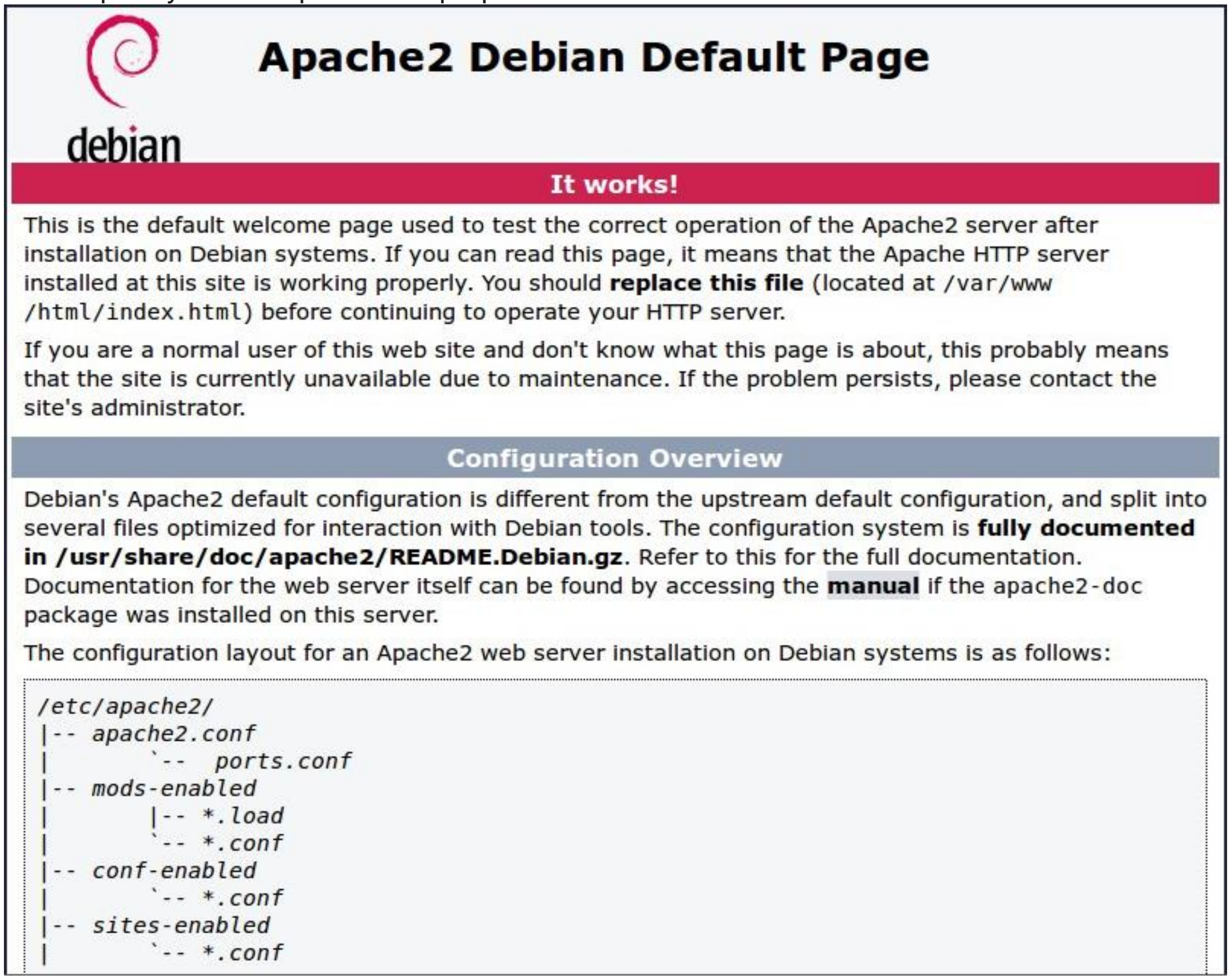
Installing Apache
Once the previously mentioned requirements are complete, we can install the web-server Apache, which is the most famous HTTP server.
First, we update the available packages by typing the following command into the Terminal:

```
sudo apt update
```

We can then install the Apache 2 package with this command:

```
sudo apt install apache2 -y
```

If the installation was successful, we should be able to see this page by typing the IP address of the Raspberry Pi in our personal laptop's browser **if** we are connected to the same Wifi.



Apache2 Debian Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at /var/www/html/index.html) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

Editing the default web page
The installation of Apache has created new folders. The files for the web page are located in this folder:

```
/var/www/html
```

We can navigate to this directory by using this command :

```
cd /var/www/html
```

We will then see a file named index.html. We have to make sure we have the rights to edit or delete the file.

```
ls -al
```

The command above will show us the rights on the files in our current folder. It looks like this:

```
total 12
drwxr-xr-x  2 root root 4096 Nov 2 08:12 .
drwxr-xr-x 12 root root 4096 Nov 2 08:12 ..
-rw-r--r--  1 root root  177 Nov 2 08:12 index.html
```

In order to edit the files, we need to either change the owner or the rights of the files. In this example we will just change the owner. Assuming we are using the default “pi” user, the command is then:

```
sudo chown pi:index.html
```

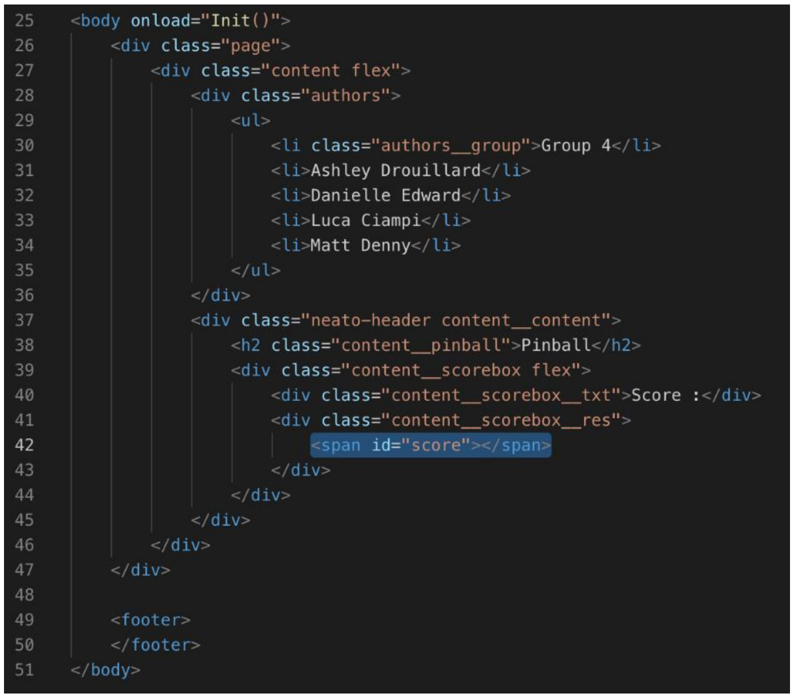
We can then edit the file and put our content. The file can be edited with this command:

```
Sudo nano index.html
```

Also, we can now move the default “*score.txt*” file containing our score in this folder. We will be using it later for our script.

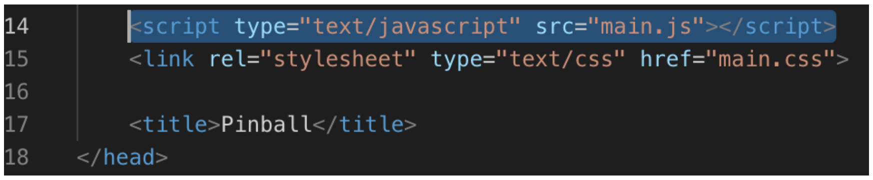
Displaying the score

Once we have written the text we wanted to appear on our page, we can work on the script. We just have to make sure we have created a tag for our score to show.



```html
<body onload="Init()">
    <div class="page">
        <div class="content flex">
            <div class="authors">
                <ul>
                    <li class="authors__group">Group 4</li>
                    <li>Ashley Drouillard</li>
                    <li>Danielle Edward</li>
                    <li>Luca Ciampi</li>
                    <li>Matt Denny</li>
                </ul>
            </div>
            <div class="neato-header content__content">
                <h2 class="content__pinball">Pinball</h2>
                <div class="content__scorebox flex">
                    <div class="content__scorebox__txt">Score :</div>
                    <div class="content__scorebox__res">
                        <span id="score"></span>
                    </div>
                </div>
            </div>
        </div>
    </div>

    <footer>
    </footer>
</body>
```

In this screenshot, you may see I have highlighted a "*<span>*" tag. This tag will be updated every 500ms to display the score. I have added an **ID** (*id="score"*)in order to use it in the JavaScript script we will be making.
We now need to make our script. In order to do so, we have created an "*main.js*" in the same folder as index.html . We then need to link the location of our JavaScript file to our html file.

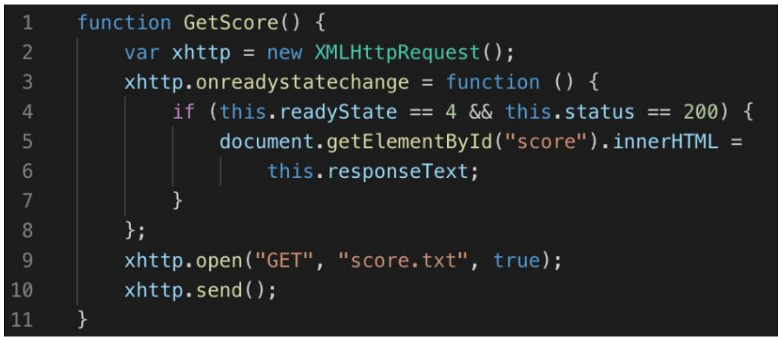


```html
    <script type="text/javascript" src="main.js"></script>
    <link rel="stylesheet" type="text/css" href="main.css">

    <title>Pinball</title>
</head>
```

You may also notice a link to the main.css file. "*main.css*" only enhances the beauty of the website and is not required. We have used it to change the fonts, some colours and font size for example.

The AJAX script

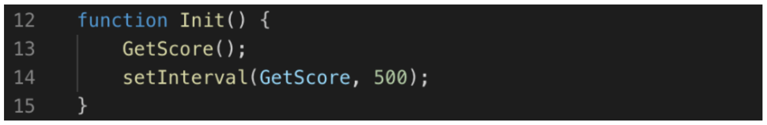
The final step is to edit our main.js file that we just created.
The score is displayed through a very simple AJAX (JavaScript) script. The function will be arbitrarily called *GetScore()*.



```
function GetScore() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function () {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("score").innerHTML =
                this.responseText;
        }
    };
    xhttp.open("GET", "score.txt", true);
    xhttp.send();
}
```

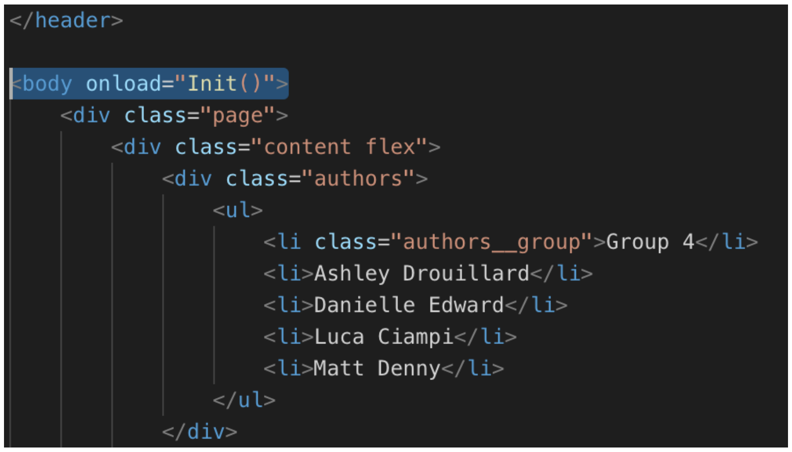
This script uses the basic guidelines of AJAX. Roughly, creating an XML HTTP request and we open our text file “*score.txt*” we previously moved into our folder containing our score. We then select the “*<span id=”score”>*” tag with the JavaScript method “*document.getElementById()*” and we tell it to modify the text of this tag to the XML HTTP response.

Then we can either create a function that will be executed when the page is loaded, or we can add a simple attribute in our body tag, which we will do. To do so, we will create a function Init(). Then we call the GetScore() function we just wrote every 500ms using the native JavaScript method “*setInterval(param, interval)*”.



```
function Init() {
    GetScore();
    setInterval(GetScore, 500);
}
```

Finally, we add an attribute to our “*<body>*” tag to tell it to load this method once it is loading.



```
</header>

<body onload="Init()">
    <div class="page">
        <div class="content flex">
            <div class="authors">
                <ul>
                    <li class="authors__group">Group 4</li>
                    <li>Ashley Drouillard</li>
                    <li>Danielle Edward</li>
                    <li>Luca Ciampi</li>
                    <li>Matt Denny</li>
                </ul>
            </div>
```

This is it. The tag will be updated every 500ms and it’s text will be the score from our “*score.txt*” file.