GNG 2101
Design Project User and Product Manual

LARGE FONT PEDOMETER

Submitted by:
Team 10 - The Next Step
Andy Saber, 300166437
Best Osajie, 300163327
Caroline Tippins, 7691410
Eshaan Kunchur, 300176301
Yusuf Hilal, 300202318

December 5, 2021
University of Ottawa

Table of Contents

Tables of Figures and Tables

List of Acronyms and Glossary

Table 1. Acronyms

Acronym	Definition
UPM	User and Product Manual
3D	Three dimensional
UI	User interface
CIRCUITPY	Circuit python programming language

Table 2. Glossary

Term	Definition
Pedometer	Device to count steps.
TFT Gizmo	Display used to show the steps.
Circuit Playground Bluefruit	Microcontroller that has sensors, speakers and controls the display.
Microcontroller	Used to control the display and count steps.

1 Introduction

This User and Product Manual (UPM) provides the information necessary for all interested parties to use the Large Font Pedometer effectively and for prototype documentation. While anyone really can use the product, the focus is on people who suffer from bad/deteriorating eyesight, organizations that help the community, and future students who would like to expand on our work. This user and product manual looks to break down the Large Font Pedometer's use and functionalities. Also, the earlier prototypes and testing will be touched on mainly for students' use. Finally, there will be a section for recommended future work and some final points that may be added.

This manual is meant to be used for educational purposes and to understand how to use the pedometer. The design can be reproduced by anyone interested, but only to improve upon it in the case of future students.

2 Overview

Keratoconus is a disease that affects 1 in every 2000 people. Our client's brother currently suffers from this disease and urges him to lose weight and get in shape. You might be asking what Keratoconus is. Keratoconus is an eye disease that affects the structure of the human cornea and results in loss of vision; sadly, there is currently no cure for this disease. Covid-19 has also impacted many people and our users. As articles state, roughly 54% of people admitted they had gained lots of weight through the pandemic. Our user's primary needs are that the pedometer has a large and clear display while it is also user-friendly and not too heavy on the wrist. These needs are a must as this is the project's whole purpose, and we have enabled all these features and more in our final design.

By researching and benchmarking other pedometers and even comparing our pedometer to the other group who made the Large Font Pedometer, ours stands out by a mile away. To begin, our screen displays up to 5 digits of numbers, which can be up to a total of 99,999 steps on the display screen. Other pedometers have up to 4 numbers as a display that limits the user to doing under 10,000 steps, which is unmotivating. Our number font size is huge compared to the other pedometers with a friendly, easy-to-read font. We have a target bar at the top of the pedometer that keeps going up until the user's goal is met, and we then implemented a song that can be played to congratulate the user on meeting their goal. Our pedometer has a yellow and black screen/text color which is researched to be the best, and after self-testing it as a group, it was indeed the best.

Other pedometers use different color contrasts that can be terrible with glare in the sun. We implemented buttons that generally aren't included in basic pedometers, such as increasing the goal range and viewing how many steps you had previously before hitting the delete button. Our buttons have audible cues when clicked, which is good so that the user is self-aware of a button is clicked. Our pedometer is lightweight and comes with a safety case that keeps the display screen

secured. We also added a necklace variant if our client wants to wear the pedometer on the neck rather than the wrist.

Figure 1. View of the whole pedometer

The key features of our product include, as touched on earlier, a clear display controlled by a microcontroller. This microcontroller also includes the code that runs the motion detection and button functions. All this is powered by a battery that can be disconnected and recharged when needed. These internal electronics are housed in a water-resistant case.

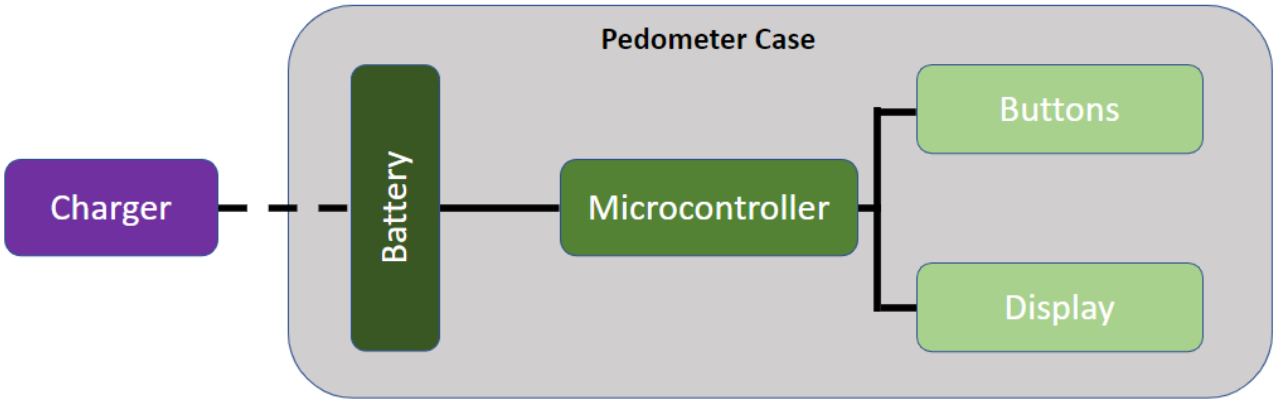

Figure 2. Block Diagram of the Module

2.1 Cautions & Warnings

Our final prototype does not require too many cautions, as we ensured that it is as safe as possible. One specific warning though, is that the product is water resistant but shouldn't be worn in very heavy rains or while swimming. Also, in the case of needing to access the internal

components, opening the case should be done gently, as to not damage and parts. While everything is replaceable, some parts are soldered on and may require extra tools.

3 Getting started

The design was made in a way that is easy to follow and understand, much like how the module is easy to use. The way it's broken down is pretty much following the power from the battery. The following main components are used in the design:

- The microcontroller: Adafruit Circuit Playground Bluefruit
- The display: Adafruit Circuit Playground TFT Gizmo
- The buttons: Waterproof 12mm Colored Buttons
- The battery: Lithium 3.7V 350mAh Battery
- A 3D printed Case

All these pieces work together to ensure proper functionality of the pedometer. The set of pictures below outline the running of the pedometer. The details dealing with each functionality will be elaborated on in ***Sections 3.3 and 3.4***.

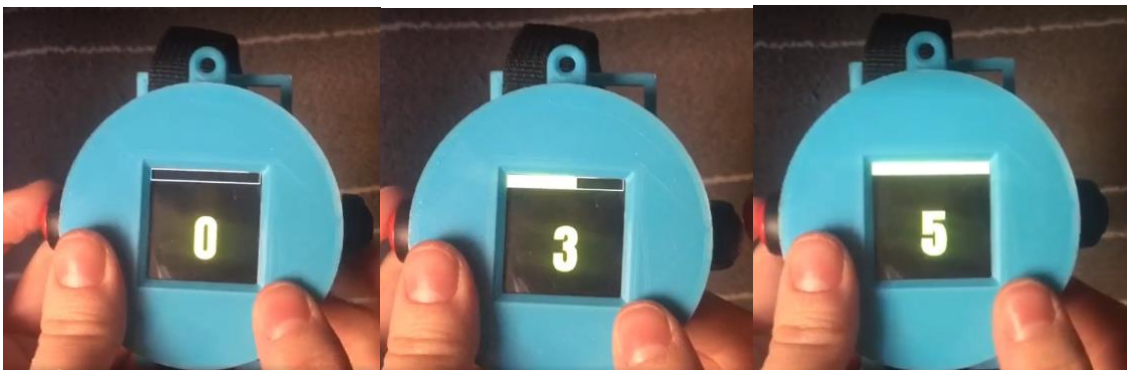

Figure 3. Walking with the Pedometer (Tune Plays when Goal is Reached)

Figure 4. Resetting the Pedometer (Pressing the Red button 3x)

Figure 5. Setting a Goal (Holding the Blue button)

Another feature that will be described below is the ability to show the previous step count by pressing the blue button.

3.1 Set-up Considerations

The way the system works to count steps is simple. While the details are described in ***Section 6.1***, this is a little overview of how it operates. Every time a shake of a specific amount is detected, a step is counted. This graph reflects a rough presentation of that process.

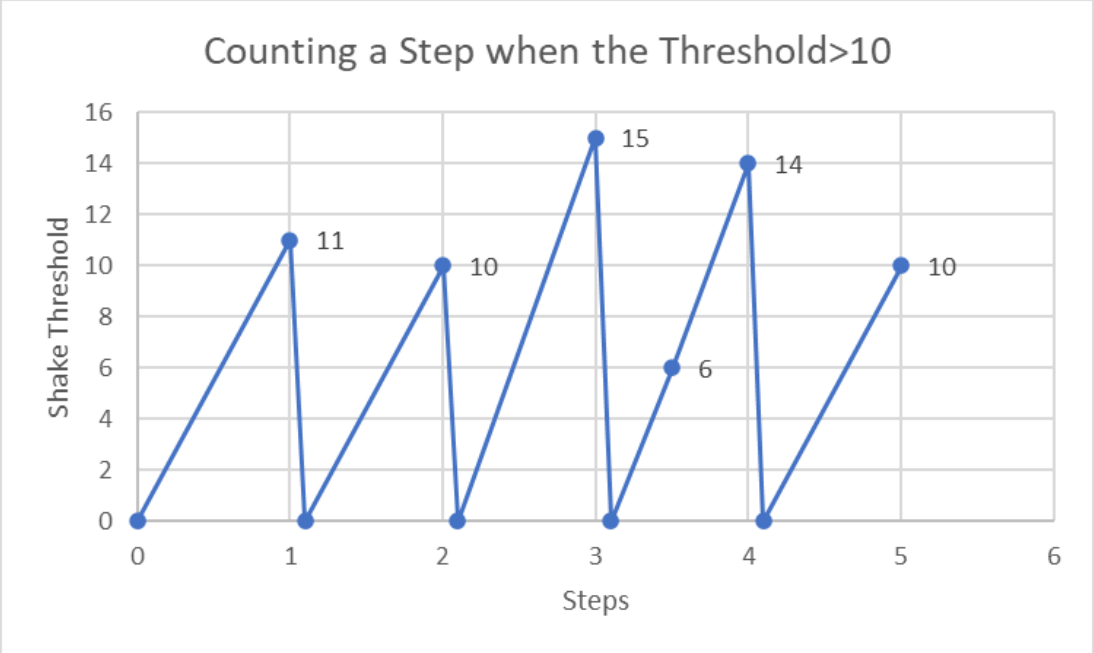

Figure 6. Step Count Graphical Representation

As seen in the graph, every time the threshold reaches 10 or more, a step is counted and displayed. The code part also takes care of the resetting and other processes.

3.2 User Access Considerations

Anyone that wants a bracing pedometer that is willing to lose weight will be considered fit for a user. As of now, we have targeted people with eye vision problems such as Keratoconus, but the pedometer is very user-friendly and there are no restrictions on the system of accessibility. This product is **not meant for children** though, as the electric components may be dangerous if toyed with.

3.2.1 Turning the Pedometer On/Off and Charging

The pedometer was built to not need to turn on or off, as it has a very long battery life (over 1000hrs). Then the pedometer can be accessed by:

- Connecting/Disconnecting the battery clip to/from the microcontroller. This can be done easily, as the clip doesn't present any electrical danger.

The most important part is to ensure that the case is gently put back together. This on/off process can also be utilized when wanting to charge the battery.

3.2.2 Using the Different Functions

The functions of this pedometer are controlled by the buttons. And while the button functions can be changed using the code, these are the current functions that are implemented. (Every button press also makes a sound to help with accessibility!)

3.2.3 Resetting the Step Count

The way the step count is reset is by **pressing the red button 3 times**. This is to make sure the user doesn't accidentally reset the step count and gets demotivated.

3.2.4 Show Previous Step Count

The way to show the previous step count is by **pressing the blue button**. This sets the pedometer to a kind of "standby mode". To return to the counting process, the blue button should be pressed again.

3.2.5 Setting a Goal

The way to show the previous step count is by **holding the blue button**. This increases the goal by 500 and can get the goal up to 10,000 steps. The pedometer can then be reset as mentioned above and the user can track steps again. For additional motivation, a little tune is played every time a goal is reached.

4 Using the Pedometer

The following sub-sections provide detailed, step-by-step instructions on how to use the various functions or features of the Large Font Pedometer. ***This part is very detailed so please return to section 3 for simple usage instructions***. The details about each feature will be discussed, as well as some key things to know if reproducing the design. And as this design deals with an electronic appliance, it's broken into 2 main parts: Hardware and Software.

4.1 Hardware

The hardware aspect of this project is not that hard to assemble and understand. It consists of 2 sections: the actual electronics, and the case that holds these electronics. Building the module is touched on here and in ***Section 6.4***.

4.1.1 Electronics

Making sure the electronics are properly set up is essential to having the system work properly. The main parts of the design need to be secured in place, to make sure that no random connection issues arise. In an ordered list, these are the main parts of the electronics that need stressing:

1. All the connections are secure and in the proper place.
2. The buttons and battery are placed before screwing the display onto the microcontroller.

The following figure shows the most important pieces of electronics:

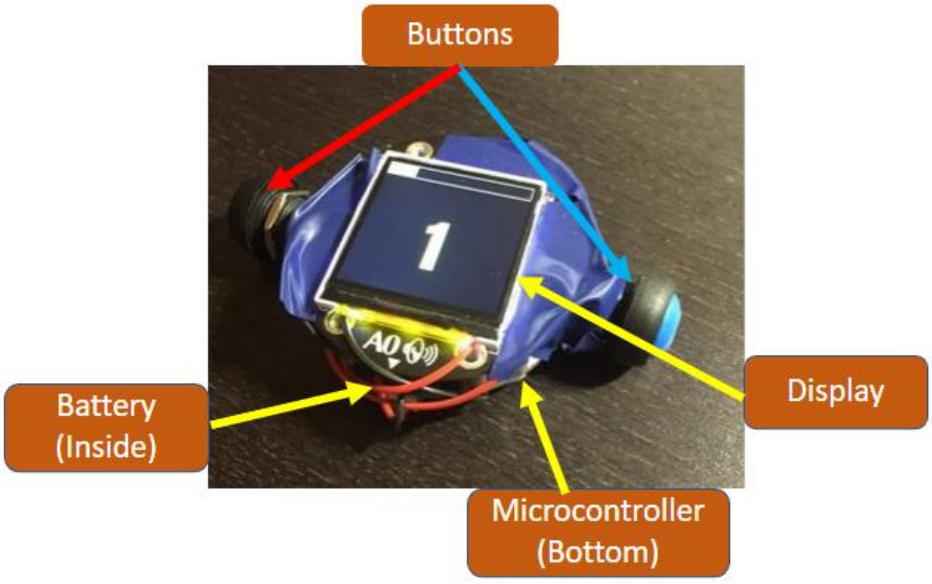

Figure 7. All the Electronics That are Used

4.1.2 Case Usage

The case is the other important hardware aspect. It was designed to allow the user to have multiple functionalities, while keeping the electronics secured. It's also meant to open easily to allow for turning on and off and recharging. The following image is the design of the case:

As seen above, the case was built to allow for the necklace and wrist attachments, as well as space for the buttons.

4.2 Software

While the bulk of this part is also mentioned in ***Sections 6.1 and 6.4***, the most important aspects will be gone over here. These are the most essential parts to be understood when using the pedometer.

4.2.1 Counting Steps

The main function of a pedometer is to count steps, as touched on in ***Section 3.1***, the code works by detecting a shake that is greater than a set threshold of 10. Every time this threshold is passed, a step is counted and displayed. This is done in a loop until power is turned off or one of the buttons is used to start another function.

4.2.2 Button Control

The buttons are also controlled using the code. Once a button receives input, the code translates that into a set function. Depending on the input, the output (display) shows a specific value. A description of the functions can be found in ***Section 3.4.***

5 Troubleshooting & Support

This section deals with the troubleshooting and support for the design if anything arises.

5.1 Error Messages or Behaviors

It is unlikely that any major problems arise with the function of the pedometer, but these are a few of the most probable situations

- If a wire disconnects, it can probably be easily reconnected. The most important part of doing this is disconnecting the battery first, then resecuring the connections.
- If the display stops working, this could be due to not securing the display. To fix this problem, disconnect the battery (***Section 3.3***) and tighten the screws on the module.
- Try to keep the module away from water as much as possible.

See Section 5.4 for additional support.

5.2 Special Considerations

Please keep in mind that the electronic components are very delicate. Thus, when turning off and on the pedometer, refrain from pulling the buttons or the wires. It is advisable to keep the pedometer on standby (pushing blue button) if one is planning on using it for multiple days as it can be charged for 1000 hours.

5.3 Maintenance

The only little maintenance required for our prototype is that the user should open the case and check if the wires are still properly soldered every time the battery is being recharged. This ensures the user that the pedometer is still intact and will function properly while keeping the user satisfied with the functioning pedometer, otherwise the pedometer could lose its signal and won't work. Also keeping the pedometer in a safe place when not being used helps ensure it doesn't accidentally break.

5.4 Support

In the case that the user requires emergency assistance/support from our team, they can email Andy Saber or Yusuf Hilal at their respective emails asabe042@uottawa.ca and yhila023@uottawa.ca .

It would be most beneficial if the support email is formatted as follows:

SUBJECT: Large Font Pedometer

Module PROBLEM: Describe the problem in as much detail as possible.

NOTES: Include any other notes (e.g. context, date that the problem started, etc.)that may be important in solving the problem. Please include a signature indicating exactly who is reaching out for assistance.

6 Product Documentation

The final prototype involved a variety of components and materials that each contributed to the overall functionality of the pedometer. The project's development was split into three major areas: software, electrical, and mechanical.

6.1 Software Design

The pedometer's software was built by leveraging CircuitPython, an open-source derivative of the MicroPython programming language. Adafruit Industries oversee CircuitPython's development. Therefore, many of their hardware products are well integrated with the software framework. When deciding upon a framework to develop our pedometer functionality on top of, our team took three main factors into account: flexibility, customizability, and the framework's documentation. When examining the market for flexible and customizable

solutions, our team looked for a framework that could allow us to incorporate our accessibility-centric designs into our final product. We initially considered taking advantage of the advanced hardware present in today's smartphones by building an extensive mobile app on top of Android or iOS.

However, having our product as a mobile app would limit its use to handheld form factors. It would add an extra barrier of accessibility to those without much smartphone experience (such as our client). Therefore, our team selected two other options: Arduino and CircuitPython. Both Arduino and CircuitPython are very similar in how they are used. Both frameworks support many hardware devices and electronic components, which was ideal for the project that we aimed to create. These two options allow programmers to program onto the microcontrollers and interact with the sensors directly. Therefore it gave our product ample flexibility and customizability.

Ultimately, our team decided to move forward with CircuitPython due to its support for much smaller circuit boards that better fit our desired product form factor. We were also pleased with the level of documentation that Adafruit provided for CircuitPython. For every Adafruit-manufactured hardware component we made us of in our project. There was extensive documentation on how it could be programmed on the CircuitPython website. Furthermore, specific methods within the framework allowed us to program complex interactions simply, as you may observe in the following image.

```
if cp.shake(shake_threshold=10):
    step_count = (step_count+1)%6
    count_label.text = "{:6.0f}".format(step_count)
    step_time = time.monotonic()
    clock = step_time - mono
```

Figure 8. Shake functionality allowed us to track any user movements

6.2 Electrical Design

Upon selecting our preferred programing framework, we then began to explore Adafruit's large catalog of hardware devices that could be implemented in our product. We centered our project's design on the Adafruit Circuit Playground Bluefruit, a Bluetooth Low Energy circuit board featuring the nRF52840 microcontroller.

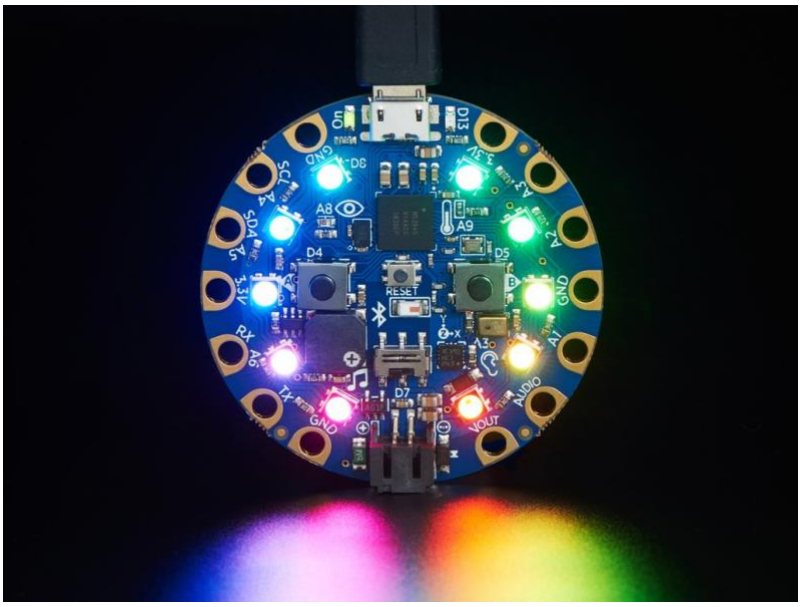

Figure 9. The Adafruit Circuit Playground Bluefruit

This circuit board is packed with functionality at a size that is ideal for a wrist-worn pedometer. While we considered other options such as the Arduino Pico and Beetle, they did not provide the same all-in-one functionality that the Adafruit Bluefruit presented. Some of the core components of the Bluefruit that we made use of in our project were the loud mini-speaker, LIS3DH triple-axis accelerometer, the slide switch, and input/output pins. To increase the accessibility of the design, we purchased separate large, colourful waterproof buttons. Pairing these easily visible buttons with the loud built-in speaker of the Bluefruit circuit board, the client can easily recognize when a button has been pressed.

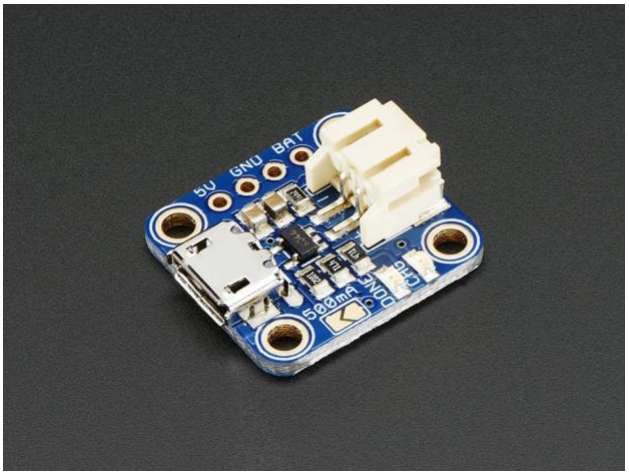

Figure 10. Adafruit MicroLipo and MiniLipo chargers

To allow for easy charging, our team opted to use the Adafruit MicroLipo and MiniLipo chargers alongside a 3.7V 400mAh Lithium-Ion Polymer Battery. This battery powers our device for an estimated 1000 hours on a single charge.

6.3 Mechanical Design

The design of the 3D case in this project was a tedious process that included a lot of testing and reprinting. The case is broken down into 3 parts to ease printing and to make the module detachable. Combined, all the pieces will look like this:

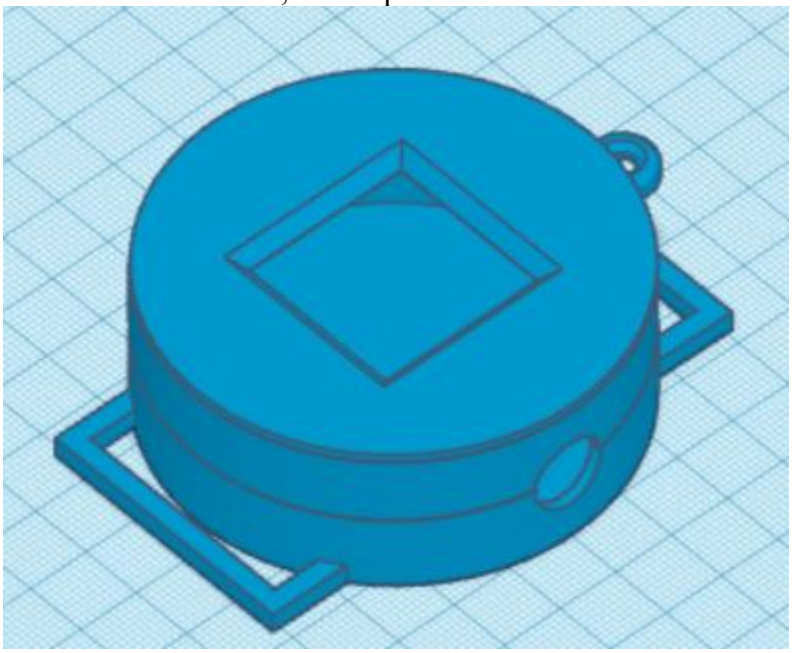

Figure 11. Complete CAD Design

6.3.1 Top Face

This part deals with the top face of the case. As seen in the picture below, this piece has clips that fit with the middle part, and a square for the display to be shown.

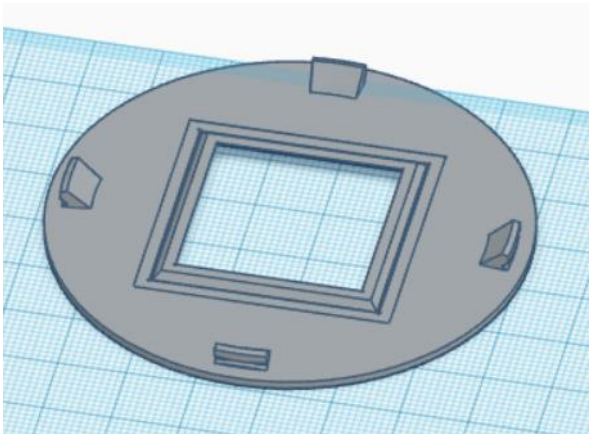

Figure 12. Top Face of CAD Design

6.3.2 Middle Piece

This part deals with the middle part of the case. As seen, the case clips on to both the top and bottom and has a place for the necklace attachment and the buttons.

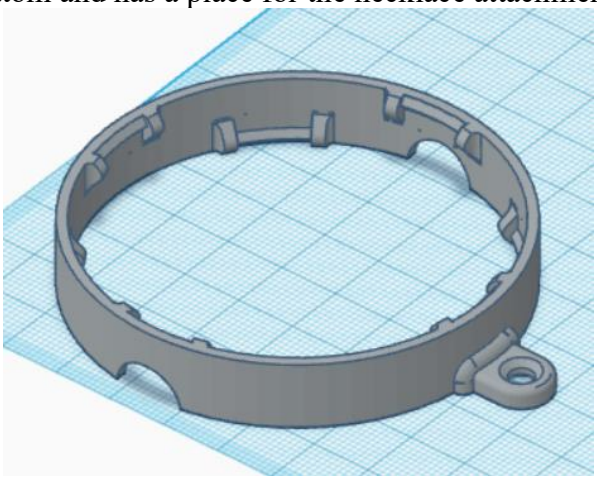

Figure 13. Middle Part of CAD Design

6.3.3 Bottom Piece

This part deals with the bottom part of the case. This part is a major part of the case, as the buttons and wrist attachment are there. The following figure shows the design of the bottom part.

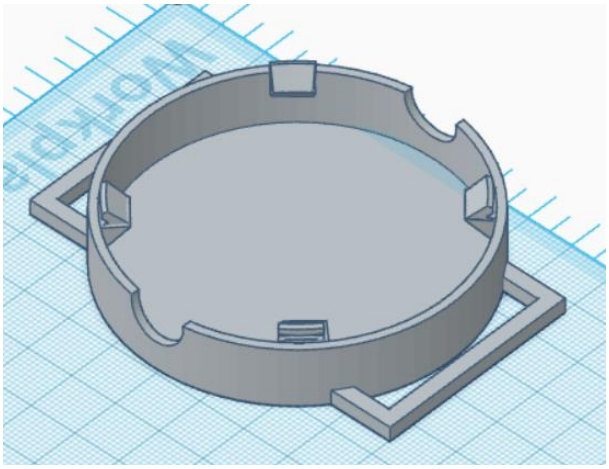

Figure 14. Middle Part of CAD Design

6.4 Building the Module

These are all the information pieces needed to build our module

BOM (Bill of Materials)

Software Platforms:

- CircuitPython
- Mu Editor

Table 3. Hardware Components

Materials	Dimensions	Unit	Quantity	Cost Per Unit (CAD)	Total Cost (CAD)	Cost with Tax(CAD)
Circuit Playground TFT Gizmo - Bolt-on Display + Audio Amplifier	53.3 x 53.3 x 9.4	mm	1	$25.04	$25.04	$28.30
Circuit Playground Bluefruit - Bluetooth Low Energy	50.6	mm	1	$31.32	$31.32	$35.39
Breadboard-friendly SPDT Slide Switch	33	g	1	$1.20	$1.20	$1.36

Lithium-Ion Polymer Battery Ideal for Feathers - 3.7V 400mAh	35.5 x 16.6 x 7.6	mm	1	$8.72	$8.72	$9.85
Fully Reversible Pink/Purple USB A to micro B Cable - 1m long	3.5	mm	1	$4.98	$4.98	$5.63
JST-PH Battery Extension Cable - 500mm	500	mm	1	$2.45	$2.45	$2.77
Adafruit Micro-Lipo Charger for LiPo/LiIon Batt w/MicroUSB Jack - v1	21 x 19 x 2	mm	1	$8.76	$8.76	$9.90

6.4.1 Equipment list

- Circuit Playground TFT Gizmo - Bolt-on Display + Audio Amplifier
- Circuit Playground Bluefruit - Bluetooth Low Energy
- Breadboard-friendly SPDT Slide Switch
- Lithium-Ion Polymer Battery Ideal for Feathers - 3.7V 400mAh
- Fully Reversible Pink/Purple USB A to micro B Cable - 1m long
- JST-PH Battery Extension Cable - 500mm
- Adafruit Micro-Lipo Charger for LiPo/LiIon Batt w/MicroUSB Jack - v1

6.4.2 Instructions

To build the step-counting functionality using CircuitPython and the Adafruit Bluefruit Circuit Playground, we first configure our development environment. We first install Mu Editor (https://codewith.mu), then we connect the circuit board to our desktop computer via a Micro-USB cable. You should now see a CPLAYBTBOOT drive appear in your file directory system. Next, we install the appropriate driver for the board, which can be found here: https://circuitpython.org/board/circuitplayground_bluefruit/. Once installed, the driver file must be dragged into the CPLAYBTBOOT drive. Upon completion, the circuit board will flash red once, and the CPLAYBTBOOT driver will be replaced with the CIRCUITPY drive.

Once the configuration steps have been completed, we may open up the code.py file found in the CIRCUITPY drive in Mu Editor. Please note, the whole code can be found in the appendix. To interact with the sensors, the following libraries must be added:

```
import time
import board
import displayio
import terminalio
from adafruit_gizmo import tft_gizmo
from adafruit_display_text import label
from adafruit_bitmap_font import bitmap_font
from adafruit_circuitplayground import cp
from adafruit_progressbar.progressbar import ProgressBar
from simpleio import map_range
from digitalio import DigitalInOut, Direction, Pull
```

Figure 15. Code libraries

Next, the following variables should be created.

```
button_a = DigitalInOut(board.A2)
button_a.direction = Direction.INPUT
button_a.pull = Pull.UP

button_b = DigitalInOut(board.A1)
button_b.direction = Direction.INPUT
button_b.pull = Pull.UP


#Set display constants
BACKGROUND_COLOR = 0x49523b  # Gray
TEXT_COLOR = 0xFFFF00  # Red
BORDER_COLOR = 0xAAAAAA  # Light Gray
STATUS_COLOR = BORDER_COLOR


countdown = 0 #  variable for the step goal progress bar
clock = 0 #  variable used to keep track of time for the steps per hour counter
clock_count = 0 #  holds the number of hours that the step counter has been running
clock_check = 0 #  holds the result of the clock divided by 3600 seconds (1 hour)
last_step = 0 #  state used to properly counter steps
mono = time.monotonic() #  time.monotonic() device
mode = 1 #  state used to track screen brightness
steps_log = 0 #  holds total steps to check for steps per hour
steps_remaining = 0 #  holds the remaining steps needed to reach the step goal
sph = 0 #  holds steps per hour
step_goal = 5
```

Figure 16. Code variables

Many of these variables access the direct functionality of the board and allow us to program interactions that leverage these functionalities later in the program.

The next step in programming the pedometer is rendering a UI to the TFT Gizmo screen. For this, we created a series of methods to load fonts, set labels, and set backgrounds. These methods are shown below:

```
#-------------------- FUNCTIONS FOR DISPLAY --------------------#
def wrap_in_tilegrid(filename:str):
    # CircuitPython 6 & 7 compatible
    odb = displayio.OnDiskBitmap(open(filename, "rb"))
    return displayio.TileGrid(
        odb, pixel_shader=getattr(odb, 'pixel_shader', displayio.ColorConverter())
    )

    # # CircuitPython 7+ compatible
    # odb = displayio.OnDiskBitmap(filename)
    # return displayio.TileGrid(odb, pixel_shader=odb.pixel_shader)

def make_background(width, height, color):
    color_bitmap = displayio.Bitmap(width, height, 1)
    color_palette = displayio.Palette(1)
    color_palette[0] = color

    return displayio.TileGrid(color_bitmap, pixel_shader=color_palette, x=0, y=0)

def load_font(fontname, text):
    font = bitmap_font.load_font(fontname)
    font.load_glyphs(text.encode('utf-8'))
    return font

def make_label(text, x, y, color, font=terminalio.FONT):
    if isinstance(font, str):
        font = load_font(font, "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789")
    text_area = Label(font, text=text, color=color)
    text_area.x = x
    text_area.y = y
    return text_area

def set_label(label, value, max_length):
    text = "{}".format(value)
    if len(text) > max_length:
        text = text[:max_length-3] + "..."
    label.text = text

def set_status(label, action_text, player):
    label.text = "{} on {}".format(action_text, player)
    _, _, label_width, _ = label.bounding_box
    label.x = display.width - 10 - label_width
```

Figure 17. User Interface code

To draw on the display, we call the following functions:

```
display = tft_gizmo.TFT_Gizmo()
group = displayio.Group()
display.show(group)
```

Figure 18. User Interface variables

Lastly, to implement the step-counting functionality of the pedometer, we encapsulate the tracking system within a while loop (to ensure steps are always being counted). Using control modifiers, we are able to dictate the output of various scenarios, such as when the user would like to reset their step count, adjust their goal, or view their past goal. Furthermore, a progress bar is implemented above the step count to visually indicate how much of the goal the user has accomplished. The remaining code for this functionality can be found here: https://github.com/EshaanK8/Pedometer

The case can also be printed using the **.stl file in the makerepo link**.

6.5 Testing & Validation

The bulk of our testing focused on the step count functionality of our pedometer and its physical fit with the 3D printed case. A series of tests were conducted when examining the accuracy of our step-counter. The threshold for step count sensitivity was adjusted programmatically. Therefore, we analyzed the accuracy of different thresholds through various tests. For instance, we measured the number of steps taken for thresholds 8, 9, 10, 11 and 12. For each threshold, 100 steps were taken, and counted manually. Through experimentation, we observed a threshold of 10 to produce the optimal result.

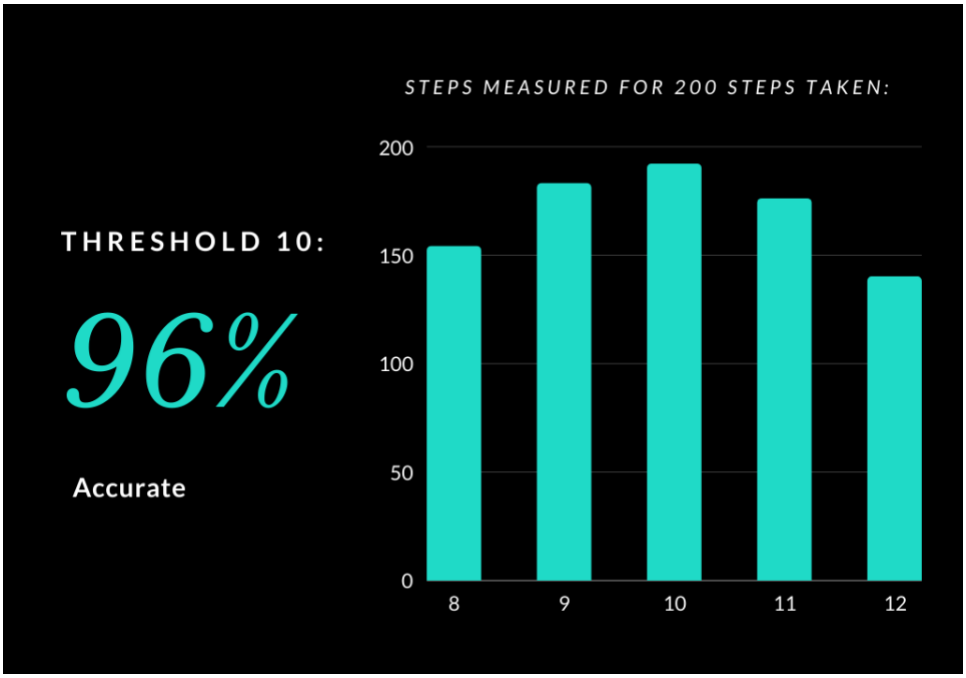

Figure 19. Graph indicating accuracy of various thresholds

Figure 20. Testing during the development phase

While testing the 3D printed case, our goal was to make it as compact as possible while still securely housing the electronic hardware. We conducted tests to ensure that the issue could be worn comfortably on the wrist without the risk of any hardware components becoming loose. For instance, we attempted to wear the pedometer on a walk, a jog, and a sprint. We felt that the device stayed firmly in place in all three tests and felt no different from a regular watch.

7 Conclusions and Recommendations for Future Work

Overall, we have learned a lot working together as a group. Firstly, during the Empathizing phase, we have learned how to have a deeper understanding of the client, which enabled us to identify the problem that needed to be solved. We interpreted and organized the needs that established our problem statement by asking client-specific questions about the project. We have learned that it is essential to deeply understand the task at hand, especially when creating a product, we would want people to buy.

In the next phase of Define, we learned how to handle conflict. Fortunately, our group did not have severe conflicts that could devastate the overall project task. This is since we all understood each other's weaknesses and strengths which enabled us to reach an agreement when planning out tasks. Learning how to handle conflict is vital when working in a group of people that one is not accustomed to. In doing this, we gain respect and communication skills which will help in future jobs or opportunities. In the same way, we also learned how to benchmark, which ultimately enabled our team to compare different products to target areas of the development that need improvement specifically.

After this Define phase, we learned different design techniques such as sketches and engineering analysis in the Ideate phase. For instance, we learned how to draw ideal prototypes individually and then compare those identical prototypes using the same technique learned previously: benchmarking. After comparing our concepts and selecting an ideal concept for our project, we realized more about programming. We learned to develop a code based on what we needed our design to do. As a result, doing this task challenged us to think like an engineer,

ultimately making us better understand a typical type of problem we could encounter in the future. In addition, we also learned how to use materials to good use, especially when there is a budget cut. By doing this, we were more aware of thinking creatively of using cheap and certain materials to good use.

In the next Prototyping phase, we learned how to design our project to make it physically feasible. In this stage, we also learned that the initial design might not be the final design; it's okay to make changes for the better of the project. For instance, our group initially decided to have the buttons as emojis; however, we decided to stick with regular buttons to comply with our design due to a slight shortfall.

This stage and the final stage of Testing enabled us to match our desired project and meet all the target specifications.

We have learned it is vital not to take time for granted for future awareness. Given a few extra months would have been of great use in the case of Testing or finding better materials to beat target specifications specifically. Overall, our group has unanimously agreed that we possibly could have had a more remarkable design given more time, although we are still grateful for how far we have come.

Bibliography

[1]Amazon.com, "CS1 Easy Pedometer for Walking | Step Counter with Large Display and Lanyard," [Online]. Available: https://www.amazon.ca/Pedometer-Walking-Display-Counter-Lanyard/dp/B07QBCY8XB/ref=sr_1_1_sspa?dchild=1&keywords=pedometer&qid=1632178010&s=electronics&sr=1-1-spons&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEyR1UwWEhYRDQ5WUMmZW5jcnlwdGVkSWQ9QTA2NDYxNThZR1pSQU9OQ1pGOD. [Accessed 20 September 2021].

[2] Amazon.com, "AK1980 Fitness Tracker, Activity Tracker Watch with Heart Rate Monitor Blood Pressure Blood Oxygen Sleep Monitor IP68 Waterproof Smart Watch Step Tracker Calorie Counter for Kids Women Men," [Online]. Available: https://www.amazon.ca/AK1980-Fitness-Activity-Pressure-Waterproof/dp/B07SR3R64J/ref=sr_1_29?dchild=1&keywords=pedometer&qid=1632179347&s=electronics&sr=1-29&th=1. [Accessed 20 September 2021].

8 APPENDICES

8.1 APPENDIX I: Design Files

MAKEREPO PROJECT LINK

Table 4. Referenced Documents

Document Name	Document Location and/or URL	Issuance Date
GNG 2101- Deliverable A: Team Contract, Client Meeting Preparation and Project Management Skeleton	file:///C:/Users/win%2010%20pro/Downloads/Team_Contract_GNG2101Group%2010.pdf	September 16, 2021
GNG 2101- Deliverable B: Needs, Problem Statement, Metrics, Benchmarking and Target Specifications	file:///C:/Users/win%2010%20pro/Downloads/GNG2101_Report_Final.pdf	September 23, 2021
GNG 2101- Deliverable C: Conceptual Design, Project Plan, and Feasibility Study	file:///C:/Users/win%2010%20pro/Downloads/GNG2101_DeliverableC.pdf	September 30, 2021
GNG 2101- Deliverable D: Detailed Design, Prototype 1, BOM, Peer Feedback and Team Dynamics	file:///C:/Users/win%2010%20pro/Downloads/Deliverable_D.pdf	October 7, 2021
GNG 2101- Deliverable E: Project Progress Presentation	file:///C:/Users/win%2010%20pro/Downloads/Deliverable%20E%20-%20Project%20Progress%20Presentation.pdf	October 13, 2021
GNG 2101- Deliverable F: Prototype 2	file:///C:/Users/win%2010%20pro/Downloads/Project%20Deliverable%20F%20Final.pdf	November 4 , 2021
GNG 2101- Deliverable G: Business Model and Economics Report	file:///C:/Users/win%2010%20pro/Downloads/GNG2101_Project_Deliverable_G.pdf	November 18, 2021
GNG 2101- Deliverable H: Design Day Pitch and Final Prototype Evaluation	file:///C:/Users/win%2010%20pro/Downloads/Design%20Day%20Presentation.pdf	December 1, 2021
GNG 2101- Deliverable J: Final Presentation	file:///C:/Users/win%2010%20pro/Downloads/Presentation.pdf	December 5, 2021

8.2 APPENDIX II: Other Appendices

```
import time
import board
import displayio
import terminalio
from adafruit_gizmo import tft_gizmo
from adafruit_display_text import label
from adafruit_bitmap_font import bitmap_font
from adafruit_circuitplayground import cp
from adafruit_progressbar.progressbar import ProgressBar
from simpleio import map_range
from digitalio import DigitalInOut, Direction, Pull
button_a = DigitalInOut(board.A1)
button_a.direction = Direction.INPUT
button_a.pull = Pull.UP

button_b = DigitalInOut(board.A2)
button_b.direction = Direction.INPUT
button_b.pull = Pull.UP


#Set display constants
BACKGROUND_COLOR = 0x49523b  # Gray
TEXT_COLOR = 0xFFFF00  # Red
BORDER_COLOR = 0xAAAAAA  # Light Gray
STATUS_COLOR = BORDER_COLOR


countdown = 0 #  variable for the step goal progress bar
clock = 0 #  variable used to keep track of time for the steps per hour counter
clock_count = 0 #  holds the number of hours that the step counter has been running
clock_check = 0 #  holds the result of the clock divided by 3600 seconds (1 hour)
last_step = 0 #  state used to properly counter steps
mono = time.monotonic() #  time.monotonic() device
mode = 1 #  state used to track screen brightness
steps_log = 0 #  holds total steps to check for steps per hour
steps_remaining = 0 #  holds the remaining steps needed to reach the step goal
sph = 0 #  holds steps per hour
step_goal = 5

#-------------------- FUNCTIONS FOR BUTTON --------------------#
def touch_a():
    return not button_a.value

def touch_b():
    return not button_b.value

#-------------------- FUNCTIONS FOR DISPLAY --------------------#
def wrap_in_tilegrid(filename:str):
    # CircuitPython 6 & 7 compatible
    odb = displayio.OnDiskBitmap(open(filename, "rb"))
```

```
    return displayio.TileGrid(
        odb, pixel_shader=getattr(odb, 'pixel_shader', displayio.ColorConverter())
    )

    # # CircuitPython 7+ compatible
    # odb = displayio.OnDiskBitmap(filename)
    # return displayio.TileGrid(odb, pixel_shader=odb.pixel_shader)

def make_background(width, height, color):
    color_bitmap = displayio.Bitmap(width, height, 1)
    color_palette = displayio.Palette(1)
    color_palette[0] = color

    return displayio.TileGrid(color_bitmap, pixel_shader=color_palette, x=0, y=0)

def load_font(fontname, text):
    font = bitmap_font.load_font(fontname)
    font.load_glyphs(text.encode('utf-8'))
    return font

def make_label(text, x, y, color, font=terminalio.FONT):
    if isinstance(font, str):
        font = load_font(font, "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789")
    text_area = Label(font, text=text, color=color)
    text_area.x = x
    text_area.y = y
    return text_area

def set_label(label, value, max_length):
    text = "{}".format(value)
    if len(text) > max_length:
        text = text[:max_length-3] + "..."
    label.text = text

def set_status(label, action_text, player):
    label.text = "{} on {}".format(action_text, player)
    _, _, label_width, _ = label.bounding_box
    label.x = display.width - 10 - label_width

display = tft_gizmo.TFT_Gizmo()
group = displayio.Group()
display.show(group)


# Draw the text fields
#goal_label = make_label("None", 12, 30, TEXT_COLOR, font="/fonts/LibreBodoniv2002-Bold-27.bdf")
goal_label = label.Label(bitmap_font.load_font("/fonts/LibreBodoniv2002-Bold-27.bdf"),text="",color=0xFFFF00)
goal_label.x=0
goal_label.y=40
#count_label = make_label("None", 12, 60, TEXT_COLOR, font="/fonts/Roboto-Black-48.bdf")
count_label = label.Label(bitmap_font.load_font("/fonts/Anton-Regular-104.bdf"),text="None",color=0xFFFF00)
count_label.x=-20
count_label.y=150
```

```
#title_label = make_label("None", 12, 120, TEXT_COLOR, font="/fonts/LibreBodoniv2002-Bold-27.bdf")
#sph_count = make_label("None", 12, 150, TEXT_COLOR, font="/fonts/LibreBodoniv2002-Bold-27.bdf")
#sph_label = make_label("None", 12, 180, TEXT_COLOR, font="/fonts/LibreBodoniv2002-Bold-27.bdf")
#group.pop()
#group.append(make_background(240, 240, BACKGROUND_COLOR))
#border = Rect(4, 4, 232, 200, outline=BORDER_COLOR, stroke=2)
group.append(goal_label)
group.append(count_label)
#group.append(title_label)
#group.append(sph_count)
#group.append(sph_label)
#group.append(border)
step_count = 0
previous_steps = 0
press_count=0
press_count_b=0
number_when_pressed=0
to_be_reset = False
showing_prev_steps = False
#set_label(goal_label, "B", 18)
count_label.text = "{:6.0f}".format(0)
#set_label(title_label, "Steps", 18)
#set_label(sph_count, "", 18)
#set_label(sph_label, "Steps Per Hour", 18)

#  creating the ProgressBar object
bar_group = displayio.Group()
prog_bar = ProgressBar(1, 1, 239, 25, bar_color=0xFFFF00)
bar_group.append(prog_bar)
group.append(bar_group)

while True:
    if(to_be_reset==False):
        #  creating the data for the ProgressBar
        countdown = map_range(step_count, 0, step_goal, 0.0, 1.0)

        #button stuff
        if touch_b():
            showing_prev_steps = True
            while touch_b():
                cp.play_tone(1400, 0.20) #40 for testing 4000 for actual
                time.sleep(0.1)
                while(showing_prev_steps):
                    #goal_label.text = "Old Steps"
                    count_label.text = "{:6.0f}".format(previous_steps)
                    if touch_b():
                        goal_label.text = ""
                        count_label.text = "{:6.0f}".format(step_count)
                        showing_prev_steps = False
        if touch_a() and step_count>0:
            if((press_count==0) or (number_when_pressed!=step_count)):
                press_count =1
                number_when_pressed=step_count
```

```
    else:
        press_count=press_count +1
    while touch_a():
        cp.play_tone(2000, 0.20) #40 for testing 4000 for actual
        time.sleep(0.1)
        if press_count==3:
            previous_steps = step_count
            step_count=0
            press_count=0
            count_label.text = "{:6.0f}".format(step_count)
        pass

if cp.shake(shake_threshold=10):
    #if step_goal - step_count > 0:
     #   step_count = 0
    #else:
    step_count = (step_count+1)
    #step_count = (step_count+1)%6
    #set_label(count_label, str(step_count), 18)
    count_label.text = "{:6.0f}".format(step_count)

    step_time = time.monotonic()
    clock = step_time - mono


    #  logging steps per hour
    if clock > 3600:
        #  gets number of hours to add to total
        clock_check = clock / 3600
        #  logs the step count as of that hour
        steps_log = step_count
        #  adds the hours to get a new hours total
        clock_count += round(clock_check)
        #  divides steps by hours to get steps per hour
        sph = steps_log / clock_count
        #  adds the sph to the display
        #set_label(sph_count,'%d' % sph,set_label,18)
        #  resets clock to count to the next hour again
        clock = 0
        mono = time.monotonic()

    #  adjusting countdown to step goal
    #prog_bar.progress = float(countdown)

#  displaying countdown to step goal
if step_goal - step_count > 0:
    prog_bar.progress=float(countdown)
    steps_remaining = step_goal - step_count
    string = str(steps_remaining)+' Steps Remaining'
    #set_label(goal_label , string,18)
else:
    countdown = map_range(step_count, 0, step_goal, 0.0, 1.0)
    prog_bar.progress=float(countdown)
```

```
            print(step_count)
            #set_label(goal_label,'Steps Goal Met!',18)
            #put button function here, and ADD A SOUND
            if(last_count != step_count):
                cp.play_tone(1240, 1)
                cp.play_tone(1240, 1)
                cp.play_tone(1400, 1)
                cp.stop_tone()
                to_be_reset = True
            #set_label(count_label, str(0), 18)
            #step_count = 0
            #time.sleep(5)
            #step_count = 0

        last_count = step_count
    else:
        while(to_be_reset):
            #goal_label.text = "Yay"
            if touch_a():
                press_count_b = press_count_b +1
                while touch_a():
                    cp.play_tone(2000, 0.20)
                    time.sleep(0.1)
                    if press_count_b==3:
                        previous_steps = step_count
                        step_count=0
                        press_count=0
                        press_count_b=0
                        count_label.text = "{:6.0f}".format(step_count)
                        goal_label.text = ""
                        to_be_reset = False
                    pass

            if touch_b():
                start_count = 0
                while touch_b():
                    start_count = (start_count+500)%10500
                    time.sleep(0.5)
                    goal_label.text = ""
                    count_label.text = "{:6.0f}".format(start_count)
                    step_goal = start_count

while len(group):
    group.pop()
```