NAIT

Edmonton, Alberta

**Silly Little Guy**

Submission to:

Gary Munro, Instructor

English and Communications Department

&

Kevin Moore, Instructor

Computer Engineering Department

Submitted By:

Aurora Fraser, Bryce Frey and Connor Marsh, Students

CMPE2965

Computer Engineering Technology

February 25, 2025

**Statement of Contribution**

This document was co-authored by Aurora Fraser, Bryce Frey, and Connor Marsh. The table below lists each contributing member’s contributions to the report and is viewed as an equal distribution of our work.

|  |  |  |
| --- | --- | --- |
| **Section** | **Author** | **Reviewer** |
| Title Page, Table of Contents, Cover Letter, List of Figures and Tables, and general Formatting | Bryce Frey | Aurora Fraser, Bryce Frey & Connor Marsh |
| Abstract | Connor Marsh | Aurora Fraser, Bryce Frey & Connor Marsh |
| Introduction | Bryce Frey | Aurora Fraser, Bryce Frey & Connor Marsh |
| Overview | Bryce Frey | Aurora Fraser, Bryce Frey & Connor Marsh |
| Design | Aurora Fraser, Bryce Frey & Connor Marsh | Aurora Fraser, Bryce Frey & Connor Marsh |
| Implementation | Aurora Fraser, Bryce Frey & Connor Marsh | Aurora Fraser, Bryce Frey & Connor Marsh |
| Project Results | Aurora Fraser | Aurora Fraser, Bryce Frey & Connor Marsh |
| Conclusion | Bryce Frey, Connor Marsh | Aurora Fraser, Bryce Frey & Connor Marsh |
| References | Aurora Fraser, Bryce Frey & Connor Marsh | Aurora Fraser, Bryce Frey & Connor Marsh |
| Appendices | Connor Marsh | Aurora Fraser, Bryce Frey & Connor Marsh |

11762 106 St  
Edmonton, Alberta  
T5G 2R1

April 14, 2025

Mr. Gary Munro, Mr. Kevin Moore  
Instructors  
NAIT  
11762 - 106 Street NW  
Edmonton, AB T5G 2R1

Dear Mr. Munro and Mr. Moore,

As per the requirements of CMPE2965, we are submitting this report titled Silly Little Guy for your evaluation.

This report goes into detail on the planning and design for the Silly Little Guy. Further outlined are the challenges and the successes encountered during the creation process. The report details the electronics used and the software that was implemented into the project.

We would like to take this opportunity to thank our instructors throughout our NAIT career for  
their patience and enthusiasm in educating their students. If it weren’t for our instructors’ feedback and support, we wouldn’t have been as successful as we have been. We would also like to thank our peers for providing us with a great learning environment and the motivation to improve.

Sincerely,

Aurora Fraser Bryce Frey Connor Marsh

Table of Contents

[List of Figures 5](#_Toc1028869690)

[Abstract 6](#_Toc1718004364)

[1.0 Introduction 7](#_Toc827678895)

[2.0 Overview 7](#_Toc248344713)

[3.0 Design 9](#_Toc568500957)

[3.1 Hardware 9](#_Toc1619946006)

[3.1.1 Pedometer Hardware 9](#_Toc101282659)

[3.1.2 GPS Hardware 10](#_Toc1229203622)

[3.1.3 LCD Screen Hardware 11](#_Toc713049462)

[3.1.4 General IO Hardware 11](#_Toc1117533577)

[3.2 Firmware 13](#_Toc995440265)

[3.2.1 Step Tracking Firmware 13](#_Toc1220190738)

[3.2.2 GPS Firmware 14](#_Toc220126654)

[3.2.3 LCD Firmware 15](#_Toc1045839884)

[3.2.4 Non-Volatile Memory Firmware 15](#_Toc843265058)

[3.2.5 Peripheral Firmware 16](#_Toc938743531)

[3.3 Software 17](#_Toc415767106)

[3.3.1 Backend Design 17](#_Toc2018560126)

[3.3.2 Computer Interface 18](#_Toc1858630405)

[3.3.3 Image Compression Software 21](#_Toc1762159533)

[4.0 Implementation 22](#_Toc1240060735)

[4.1 Hardware Implementation 22](#_Toc1305169812)

[4.2 Graphics 27](#_Toc1609797550)

[4.3 Gamification 27](#_Toc1141476887)

[5.0 Project Results 28](#_Toc1307829175)

[6.0 Conclusion 29](#_Toc628790702)

[References 32](#_Toc898878432)

[Appendices 34](#_Toc1442580082)

[Appendix A - Excerpt From Proposal: 34](#_Toc1472957116)

[Appendix B - Specification: 35](#_Toc654589154)

[Appendix C – Verification Tests: 35](#_Toc632474753)

[Appendix D – Work Breakdown Structure: 40](#_Toc359461876)

# List of Figures

Figure 1 ……………………………………..……………………………………………………………………… 11

Figure 2 ………………………………….………….……………………………………………………………… 13

Figure 3 ………………………………………..…………………………………………………………………… 14

Figure 4 …………………………………..………………………………………………………………………… 19

Figure 5 ……………………………….………………………………………………….………………………… 20

Figure 6 ……………………………….…………………………………………………….……………………… 20

Figure 7 …………………………….……………………………………………………………….……………… 21

Figure 8 …………………….…………………………………………………………………….………………… 23

Figure 9 ..…………………………………………………………………………………………………………… 25

Figure 10 …………………………………………………………………………………………………………… 26

Figure 11 …………………………………………………………………………………………………………… 27

# Abstract

The Silly Little Guy is an embedded system designed to gamify exercise and exploration without reliance on a smartphone. Taking inspiration from Tamagotchi, and the attachments people form with virtual pets, the device acts as a virtual pet the user can take for walks. The concept utilizes accelerometer and GPS technologies, which monitor the user’s steps and path taken and reward their activity. The Silly Little Guy can adapt to the user’s preferences, as different people require different difficulties for exercise. This means creating goals whose goals can be adjusted to suit the needs of the player.

The Silly Little Guy requires a means to persist saves in the case that the device is lost or damaged. This involves saving the game state to flash storage regularly. The user is also able to store the save to a database through a desktop application. The application makes web requests to an endpoint that processes and backs up the data. The larger scale of the desktop application and the easy access to web requests allow for additional functionality in displaying the user’s stats and showing their visited locations on an interactive map.

# Introduction

The idea for this project originated from the Tamagotchi toy invented by Aki Maita in 1996. The Tamagotchi was a device that contained a virtual pet which is taken care of by the user through feeding, playing, and training (Wikipedia, 2025). This project utilized this idea of caring for a virtual pet to motivate the user to exercise.

The purpose of this report is to document the experiences made and methods taken while under production. It will start with an overview of what the project entailed. It will then have an in-depth discussion of the design process, highlighting the hardware, firmware, and software elements selected for use in the project. After discussing the design process the report will go over how the team transformed the design elements into a playable game with visual elements that motivate the user. In conclusion, the report will go over what succeeded and what failed during development, and how the results hold up to the original goals that were set.

The goal of this project was to gamify going out for a walk to help motivate the user to exercise more consistently. The project accomplishes these goals by tracking various datapoints achieved during a walk and rewarding the user for reaching their set exercise goals.

# 2.0 Overview

The Silly Little Guy has three crucial sections that make the device functional. These sections are the hardware, software, and firmware.

The hardware can be split into four individual components that all work with each other to interface with the device. The pedometer tracks the steps taken by the user and sends the count to the Silly Little Guy. This number is stored on the device to be used for gameplay mechanics. The GPS sends latitude and longitude values as well as a Real Time Clock (RTC) reading to the device through UART. The latitude and longitude values are stored locally and provide multipliers towards evolution while the RTC is used to reset step counts every day and every week. The LCD screen provides the user with a visual interface as well as an easy way to see their current stats. The General-Purpose IO buttons provide the user with a more interactive experience by allowing them to interact with their Silly Little Guy.

Each hardware component has its respective firmware written to enable the components to send and receive data from the microcontroller. The pedometer uses an I2C interface for the sending of step data. The GPS uses UART to send its data as a text string. The LCD is communicated with via an SPI interface and the GPIO buttons send voltage signals to be read by the microcontroller. The firmware also contains code that allows the device to store data in Non-Volatile Memory. This keeps the data saved on the device as long as it remains powered.

The software can be split into two sections that provide the user with ease of access to their data. These sections are the database and the Windows Forms application. The database receives data from the windows forms application and stores it according to the user ID provided. This data can also be sent to the Windows Forms application upon request. The windows forms application provides the user with a way to easily transfer their data from device to database and vice versa. The most recent data provided is also displayed in two separate windows in a visually appealing manner.

# 3.0 Design

For the Silly Little Guy to be actualized, time was taken to design how it would look and operate. This section will discuss how and why the hardware components were selected and what the hardware components are used for. Then, it will discuss the firmware that allows the hardware to be used, and then it will discuss the software that the Silly Little Guy interfaces with.

## 3.1 Hardware

The first step in the project was deciding what hardware components to use to get the wanted results. The components decided on were an Accelerometer/Pedometer, a GPS, an LCD screen, and some buttons for user input.

## 3.1.1 Pedometer Hardware

Originally, the plan was to create a new step tracking algorithm using the values read from any accelerometer available, this plan was changed after finding the SparkFun Triple Axis Accelerometer Breakout, which featured a BMA400 accelerometer. This accelerometer came with step tracking bundled in with other functionalities. Using the datasheet provided by BOSCH was vitally important to the initialization of the BMA400 accelerometer. An initialization diagram from page 13 of this datasheet can be seen below:

Figure 1

A diagram of a computer program

AI-generated content may be incorrect.

BMA400 Initialization  
(BOSCH, 2022)

Following the commands in this diagram initialized the pedometer to a basic state, providing X, Y, and Z values when their respective registers were read. Further writing values into the STEP\_STAT and INT\_CONFIG1 registers enabled the step counter register to start tracking steps. Once all these registers were initialized, the current steps counted could be read from register STEP\_CNT\_0.

## 3.1.2 GPS Hardware

After reviewing prices of various GPS breakout boards, the Adafruit Mini GPS PA1010D sensor was chosen for its low price relative to most other boards, and the fact that it ran on 3.3v power, like the rest of the project. The GPS provided I2C and Serial communication, but the documentation on the I2C was limited and the best resources were in the Arduino vendor libraries. Fortunately, since the GPS required no configuration for the purposes of the project, and since it sent out constant serial data, I2C was not required. The GPS was connected to the MCU with standard Tx, Rx and power connections. No flow control or waking of the GPU was required.

## 3.1.3 LCD Screen Hardware

A 1.8” TFT LCD with a display resolution of 128x160 was chosen as the display. A larger, rectangular display was chosen over a smaller, squarish one, as a larger display could display more information at once, a factor that was essential to the project. In addition, a color display was prioritized to ensure the graphical elements would stand out and be endearing to a prospective user.

## 3.1.4 General IO Hardware

The inputs for the Silly Little Guy consist of three buttons, two to move between menu options, and one center button to confirm the selected option. Since most of the actual gameplay interacted with the sensors directly, three buttons were sufficient. The final design is shown in Figure 2:

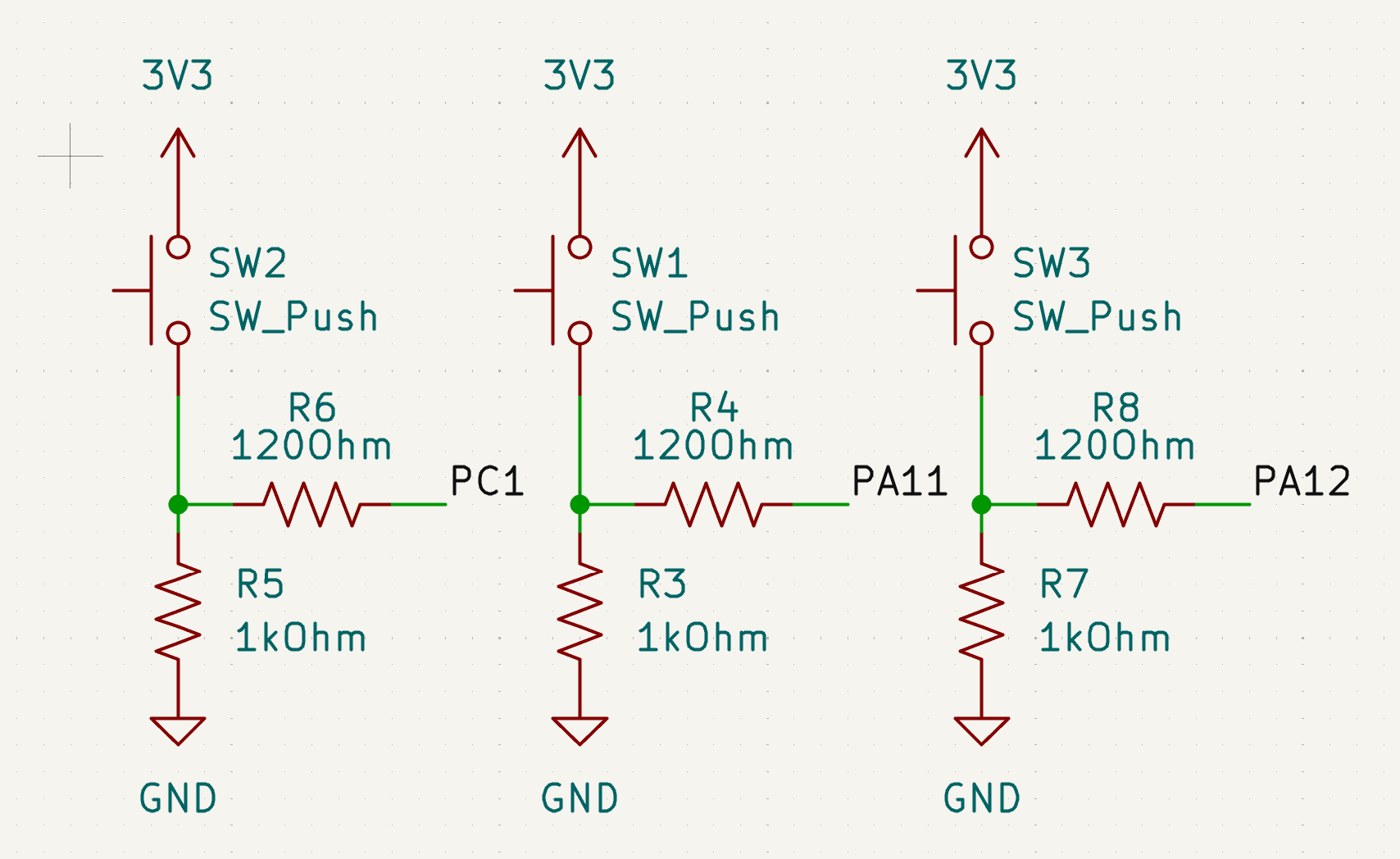
Figure 2 Button Circuits

Figure 2 shows the circuits for the buttons, including a pull-down resistor and a current limiting resistor for the input pins on the MCU.

To complete the Tamagotchi-inspired aesthetic of the Silly Little Guy, a single piezoelectric buzzer playing simple combinations of tones was used for creating sound effects reminiscent of the 90’s era. Simon Walker recommended that the buzzer be placed in a BJT driver circuit to reduce the amount of current being drawn from the output pins (Walker, 2025):

Figure 3

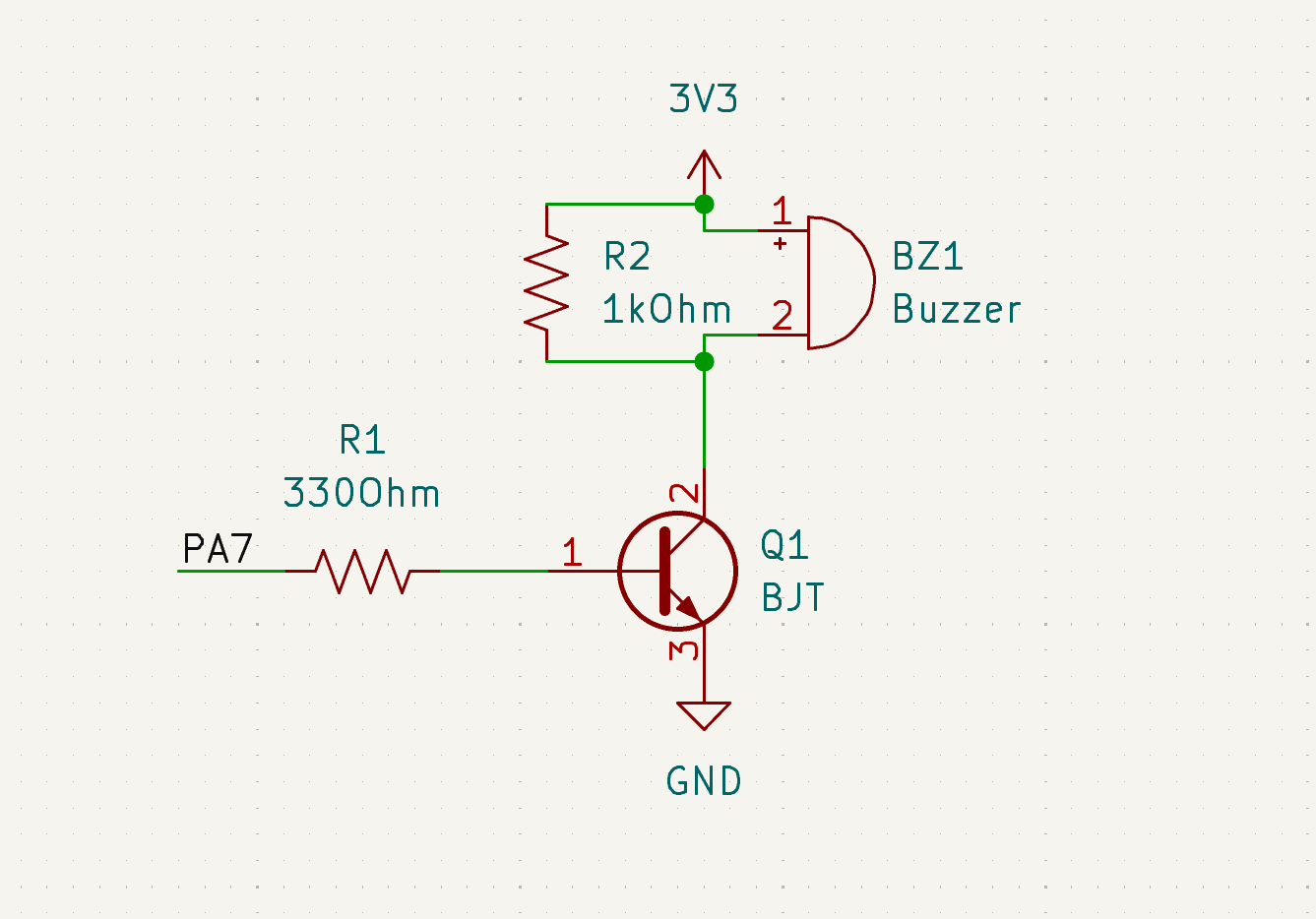
Buzzer Driver Circuit Design

Figure 3 shows the driver circuit for the buzzer, the current draw for PA7 limited to 8 milliamps, well below the maximum draw (STMicroelectronics, 2024b).

## 3.2 Firmware

For the selected hardware components to operate, they needed a microcontroller with specific firmware capabilities. The STM32G0B1RE microcontroller supplied the required UART, I2C, and SPI interfaces while also having enough memory to store all dataused within Silly Little Guy.

## 3.2.1 Step Tracking Firmware

In the BMA400 Accelerometer, the only register consistently read from is register 0x15, containing the current step count. In the main code loop, an I2C read command is sent to this register whenever the screen is updated, storing this value into a variable. However, this register often gave no values or an incorrect value back in the early stages of development. To ensure the BMA400 was still in working condition, registers 0x04, 0x06, and 0x08 were read from to read the X, Y, and Z acceleration values. Reading these registers and moving the BMA400 around allowed additional verification that the chip was sending values.

## 3.2.2 GPS Firmware

Since the only vendor libraries from Adafruit were for Arduino or CircuitPython, vendor libraries were not used (Adafruit, 2024). There was no need for libraries, however, since GPS data comes from the serial line in the form of strings, which are formatted in the NMEA protocol. The GPS generated a wide range of data that was intended for use in more advanced systems, which was unnecessary for the project, so the firmware only looked for successful latitude and longitude readings in the form of GPRMC readings. “Minmea”, a widely used C library for parsing NMEA strings into structs, was used to parse the data, and then the latitude and longitude readings were added to an array of previous readings (Kosma, 2014). The readings also provided RTC data since the GPS started up, which was convenient for other parts of the project as an alternative way to keep track of time.

## 3.2.3 LCD Firmware

The LCD was another component that only provided vendor libraries for Arduino (Adafruit, 2025). Having less experience with SPI than other protocols, a custom library was ruled out immediately. After looking through many libraries, one was selected that had built-in graphics functionality including drawing rudimentary shapes and text, as well as images, which were core to the project’s design (ControllersTech, 2020). The library required HAL, STM’s Hardware Abstraction Layer, to work, which meant porting existing code from bare metal libraries over to HAL libraries. Porting to HAL did also make it feasible to switch STM devices, which was done midway through the term when it was found that the initial Nucleo-G031 had only 8 kilobytes of RAM and 64 kilobytes of flash memory, not enough memory to store the images for display which took up over 8 kilobytes each (STMicroelectronics, 2024a, p1). This issue was resolved by purchasing a board with new memory, and by implementing a simple image compression algorithm that scanned the pixels in an image sequentially and grouped similar pixels that were lined up in a row. After buying the Nucleo-G0B1, which had 144 kilobytes of RAM and 512 kilobytes of flash memory, the LCD design choices were finished (STMicroelectronics, 2024b, p1).

## 3.2.4 Non-Volatile Memory Firmware

To allow the Silly Little Guy to be powered on and off without loss of data, the main game save data was stored in flash memory. Flash storage on the STM32G0b1RE required some research into the memory structure on the MCU, and the HAL functions used to read and write to the memory.

According to the reference manual, the flash memory structure on the STM32G0B1RE is composed of two banks, each containing 128 pages 2 kilobytes in size (STMicroelectronics, 2024c, ???). The final page from the first bank was selected after a cursory look at the system memory during runtime to ensure that there was no data overlap. The address for that location was found in the reference manual as 0x0807 F800.

The functions HAL provided for writing to the flash only allowed for writing to storage in 64-bit chunks. If the storage requirements for the game save were more demanding, multiple variables could have been stored in one 64-bit chunk. But for ease of programming, and because the game save did not exceed the page size of 2 kilobytes when all the variables were written as 64-bit values, one chunk per variable sufficed, except for strings, which were considered unreasonable to store as values 8 times the size.

## 3.2.5 Peripheral Firmware

The interfaces with GPIO for the buttons and PWM for the buzzer were both simple calls to the STM32 HAL. For the buttons, a state machine was made to decide what options were selected, and each button’s state was polled in the main loop. For the buzzer, custom sound effect functionality was implemented using collections of tones and delays to time the changes.

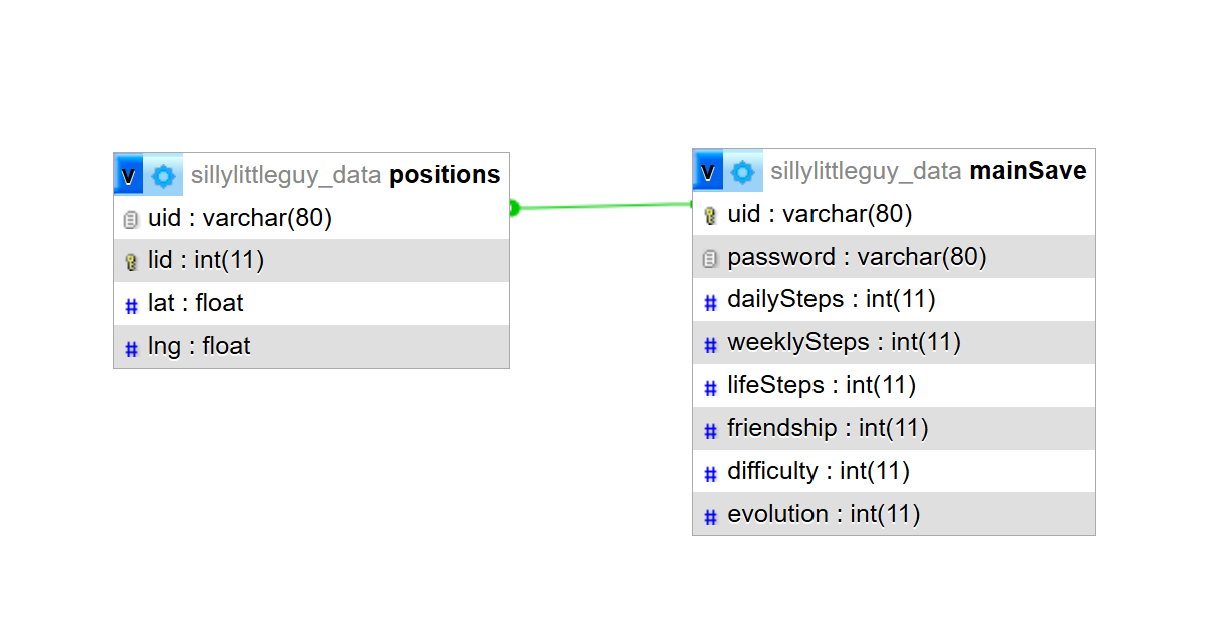
## 3.3 Software

To tie everything together, software applications were designed for the user to operate the device in ways that the microcontroller was incapable of doing, such as database storage and the management of data.

## 3.3.1 Backend Design

Since the main progression of the game would be linear in the form of steps counted, and progress would only be lost by rolling back to another save, it was decided that only one save per user would be necessary. Each time the user would persist their save using the app, it would update their progress on the backend. This simplified the database design. Two tables were required, one to store user statistics, and one to store the user’s accumulated positions. This is shown in Figure 4:

Figure 4



Silly Little Guy Database Design

Figure 4 shows the database design. User data was keyed under the username, which meant each username had to be unique. It was also what was used to secure the user’s data, as reading and writing to the database could only be done through the username. A password was considered for added security. The database was accessed through http requests to PHP endpoints hosted on an internal NAIT server.

## 3.3.2 Computer Interface

With the database designed, the user needed a way to access the database and a way to transfer data to and from the Silly Little Guy. To do this, a C# Windows Forms application was created. This application is used for the transference of data, as well as display the current save data in a more explicit manner. Figure 5 and Figure 6 show the Pet Menu and GPS Menu:

Figure 5

A screenshot of a computer

AI-generated content may be incorrect.

Pet Menu

The pet menu is responsible for displaying the currently stored pet specific stats as well as allowing the user to scroll through all sprites used for their pets’ current evolutionary state:

Figure 6

A screenshot of a computer

AI-generated content may be incorrect.

GPS Menu

The GPS menu has two sections, the left box displays the latitude and longitude of every location the Silly Little Guy has pinged while out for a walk, and the right GMAP window will put a pin on each of those locations as a visual representation of the Silly Little Guy’s adventures.

The user can access the database through the Save Menu (Figure 7). This menu allows the access to receive data from the database as well as send the received data to the Silly Little Guy Device:

Figure 7

**A screenshot of a computer

AI-generated content may be incorrect.**

Save Menu

The top buttons: Send Save, Receive Data and Update Username, are responsible for sending and receiving data between the database and the application. Once data has been received the other two pages will update their information with the most recent received data. For sending data to the Silly Little Guy, a com port must be connected before hitting the Update SLG button. This button sends a formatted text string to the microcontroller to be parsed and have its internal variables updated to match the save data pushed to it. This system allows the user to keep their data stored on the internet as well as allowing the device to be useable between multiple people without losing data.

## 3.3.3 Image Compression Software

Given the fact that images were stored in text, and that the Silly Little Guy used a custom encryption algorithm, an image processing application was created to convert images into compressed data and then exported that data in the form of text. The text was structured so that it could be copied and pasted as an implicit 2-dimensional array declaration in the main code on the Silly Little Guy device. Figure 8 shows the UI for the application:

Figure 8

Image Compression App

This app provided functionality useful for the development of the project and was not intended for the user. It added to the toolchain used for the graphic design on the Silly Little Guy.

# 4.0 Implementation

After the design process, Silly Little Guy required a way for a user to interact with it. This section will discuss the process from design to implementation, the graphical design process and how the graphics were used to provide a fun and interactive experience. Then, it will discuss the gamification of the Silly Little Guy, including the overall gameplay loop and how the hardware components affect the inner workings of the game.

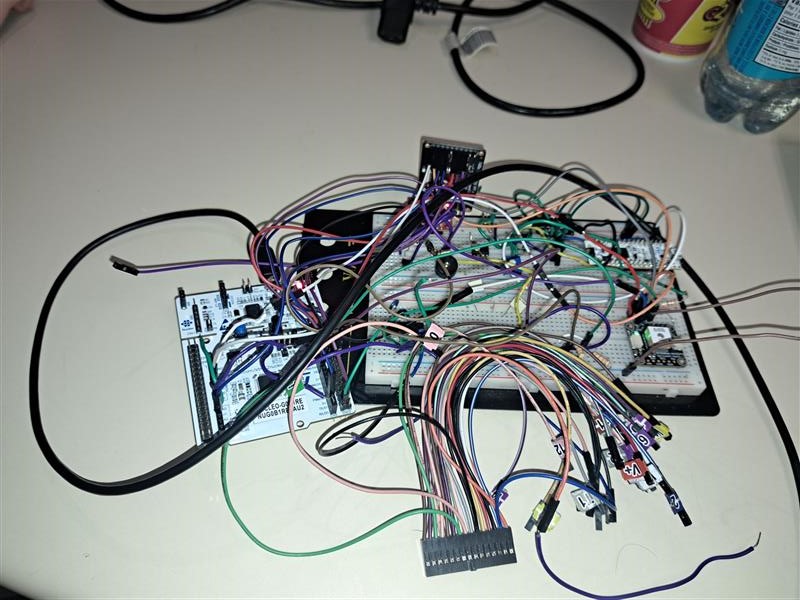
## 4.1 Hardware Implementation

The project was initially split into three parts, the GPS, the accelerometer and the LCD. Each person used Nucleo-G031K8 boards to test individual components. Initial tests went well once libraries were found for the LCD with relatively minimal issues. The circuitry was built on breadboards and comprised of direct connections from the pins to the microcontroller. No pull-up or pull-down resistors were required. Each device ran on the board’s 3.3V power supply with current draw within the limits stated by the datasheet (STMicroelectronics, 2024b). For the accelerometer, pull-up resistors for the I2C connections were internal, and simply required connecting to the board (Wade, 2022). For the GPS, testing without pullup resistors showed no errors. For the lcd, it had internal logic level shifting circuits that involved pullup resistors (Ada et al, 2025).

After testing the core components, the GPS circuit was expanded to contain peripheral elements like buttons and the piezoelectric buzzer and driver circuit. No other issues were found in the initial implementation of the peripheral elements, other than the cluttered state of the wiring, which made interacting with the buttons difficult.

After purchasing the Nucleo-G0B1RE board, the individual circuits were placed on one breadboard and were tested as a complete system. The first revision can be seen in Figure 9:

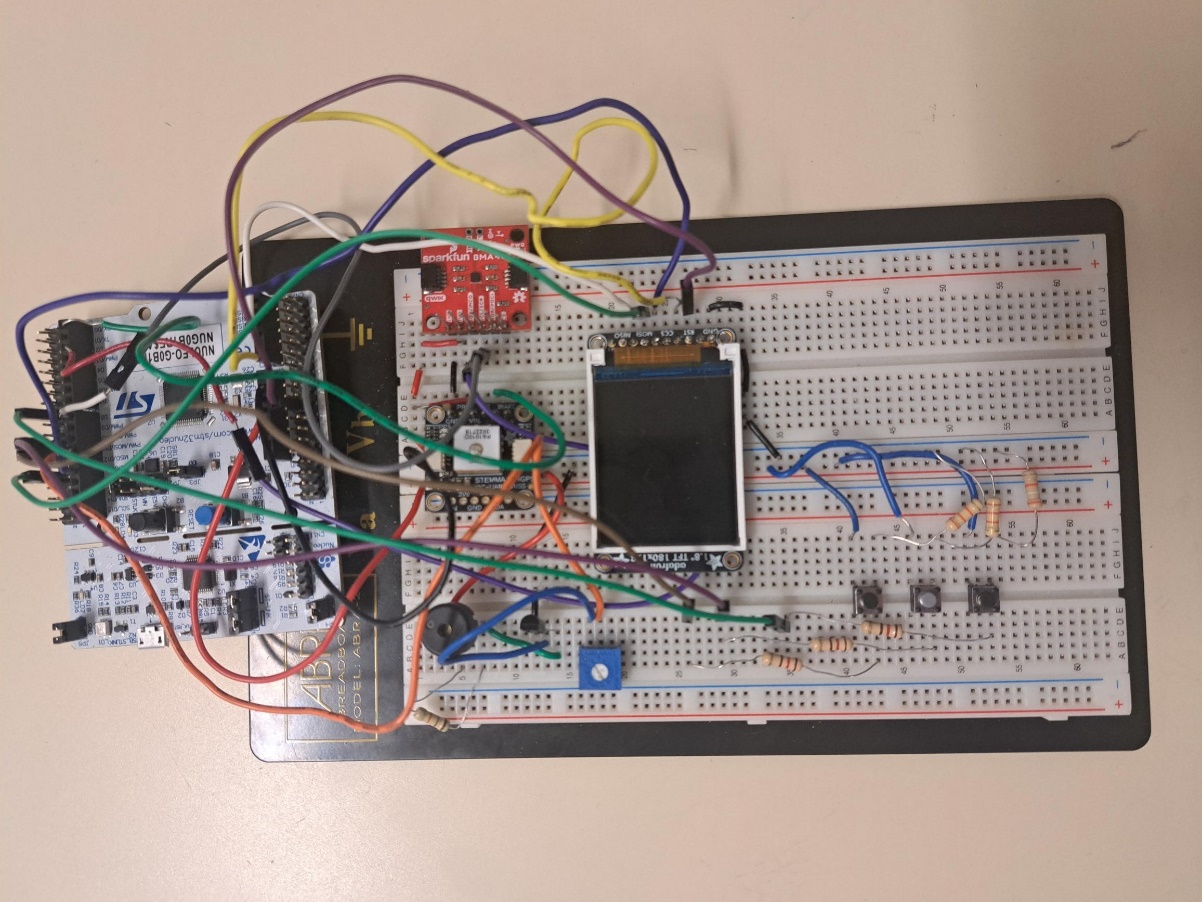
Figure 9



First Revision

The benefits of this revision revolved around the low time it took to prototype, the fact that it didn’t have any errors, and the fact that it allowed for testing of a complete Silly Little Guy. The downsides were difficulties in troubleshooting the circuit, and the lack of buttons. This revision used direct wired connections to pull down resistors to emulate buttons. The LCD was attached using wires and not fixed to the breadboard. It was decided this needed revision. The second revision attempted to clean the circuit up, as shown in Figure 10:

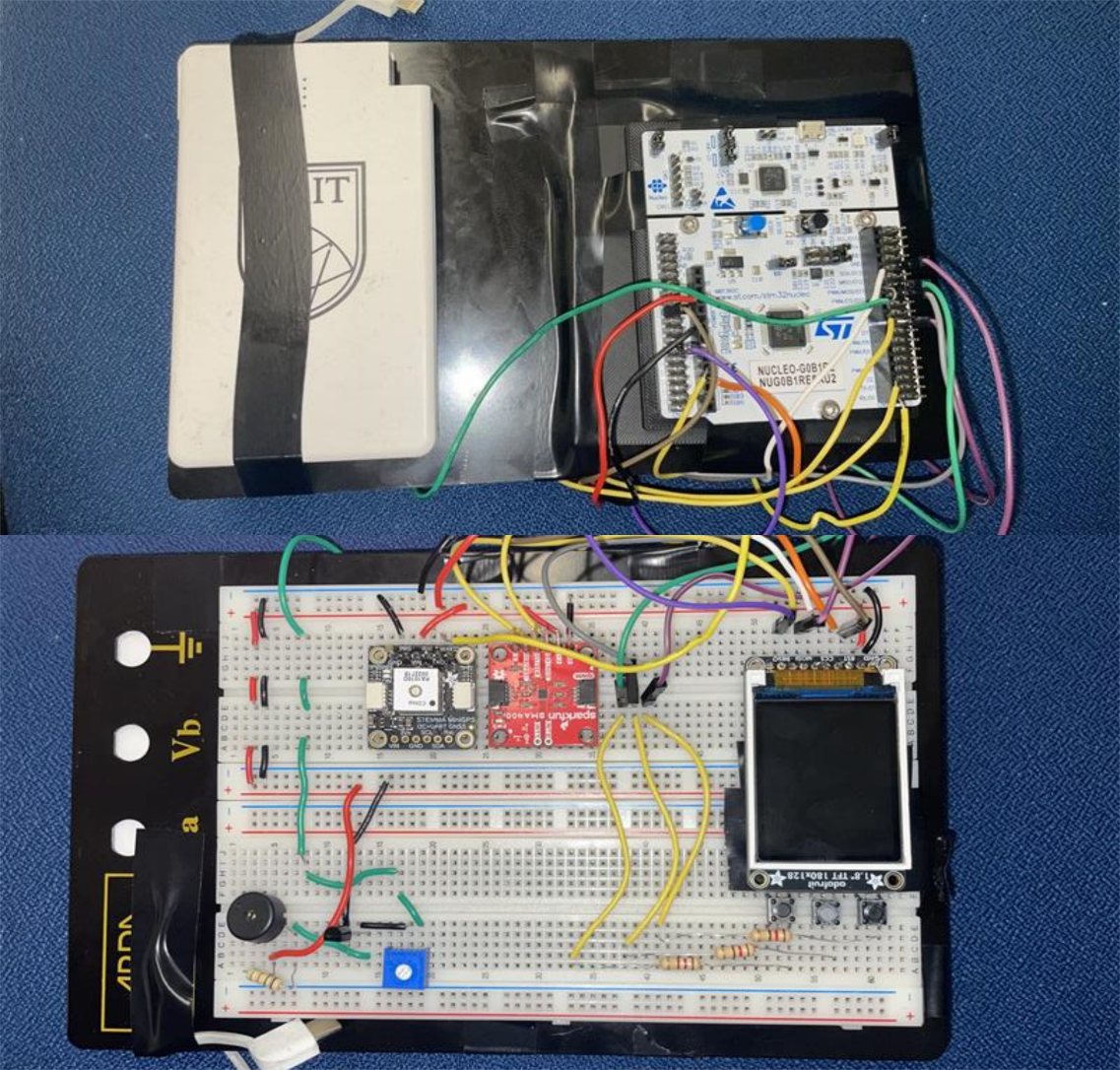
Figure 10



Second Revision

The second revision kept the microcontroller to the side of the breadboard, but it simplified the power and ground rails. It was the first attempt at managing wires so that they gave a clear view of the LCD and access to the buttons. At this stage, some trial-and-error was required to find an ideal resistance for the volume of the buzzer, and so a 10k Ohm potentiometer was used in place of a base resistor. This revision, however, did not meet the needs for portability and durability that was required, and it became necessary to test and rebuild portions of the system multiple times. The third revision focused on making a Silly Little Guy that could reliably be taken for walks. Figure 11 shows the final revision to the design:

Figure 11



Final Revision

The third revision placed the microcontroller on the back of the breadboard, as well as a portable USB power pack. It shifted the connections between the breadboard and the microcontroller to one side of the breadboard and wrapped around the side. The buttons were moved under the screen. Mounting brackets were sourced, 3D-printed, and taped to hold the microcontroller in place without scratching the metal plate on the back of the breadboard (Tymm, 2016).

Soldering the components to a prototyping board was an option, as well as printing a case. These would have drastically improved the durability, but it was not an option given the remaining time after meeting the minimum specifications.

## 4.2 Graphics

Strong emphasis was placed on creating a character design that would be appealing to users. To streamline the design process, it was decided early on to focus on a character that would make logical sense to hatch from an egg, as an egg would be simple to animate and illustrate. After some deliberation, a dragon was selected as the basis for the character.

Popular character designs from the 90s and 00s were looked to for guidance in creating appealing sprites. In particular, the design choices of the 00s video game “Hamtaro: Ham-Ham Heartbreak” were referenced while illustrating. Other sources of artistic inspiration included the “How to Train Your Dragon” book series, as well as the “Pokémon” and “Spyro the Dragon” video games.

The sprites were illustrated and animated using the software “Aseprite”, which is a popular tool for pixel art (Aseprite, 2025). The software was compiled from source using resources available on the Aseprite GitHub repository.

## 4.3 Gamification

The core gameplay loop revolves around the user’s step count tracked by the pedometer. In the settings menu they select their personal step goal, ranging from 1000 to 999000. Their step goal effects the Silly Little Guy’s mood, if it has been reached the Silly Little Guy will be happy, otherwise their mood will stay neutral if as least half of the step goal was reached, and sad if the step goal is less than fifty percent complete. The pet’s emotion was originally planned to be displayed with sprites, but to save time it was decided to use text emoticons [ :), :|, :( ] in the corner of the screen instead.

In addition to changing the Silly Little Guy’s emotion, the user’s actions can also cause the Silly Little Guy to evolve. As the user gets more steps, they gain experience equal to steps\*(number of locations/10). Once their total experience passes a threshold, their pet will evolve. An evolution animation plays, and the evolution goal gets set to a new, higher goal. The Silly Little Guy can evolve twice: Baby to Young, and Young to Adult.

# 5.0 Project Results

Ultimately, the project was successful. The final device meets the specification, and features many of the graphical and audio flourishes that were intended to make the project stand out. The development process necessary to bring the project to success was both trialling and enlightening.

The planning stage generated numerous ambitious ideas. Hindsight suggests the project might have benefited from increased focus during planning. While it was rewarding in its own way to develop so many ambitious features, it did require the dedication of time and intellectual labour to work that was both delicate and intricate. Planned features relating to the graphical display, as well as the real-time elements, could have been simplified to reduce the intricacy of the development process.

The hardware and embedded firmware design process was challenging, primarily regarding the integration of several discrete devices. The debugging process presented a particular challenge, as bugs would often manifest without clear indication of their source. This led to a process of “whack-a-mole", where limited progress could be made until bugs were eliminated, and eliminated bugs were often hiding other bugs that would “pop up” at inopportune times. However, it did make successful progress more rewarding when it occurred.

The software side proved to be simpler than the hardware side. The team’s familiarity with C# and the availability of software libraries targeting the .NET framework streamlined the development process. This was an asset, as it prevented software issues from impeding development during the period where the hardware required the most attention.

# 6.0 Conclusion

The development and management of project Silly Little Guy was an enlightening experience for all people involved. The months of time that went into research, prototyping, and mountains of bug fixing allowed the team to understand the struggles that come with creating a large-scale project as well as how to overcome those struggles to get a working prototype.

The learning curve for the technologies involved for this project was steeper than expected, but that only made it more rewarding when things started to come together. Working with the GPS developed skills not only with geolocation, but also with standardized ASCII communication. Displaying color graphics to an LCD on a microcontroller with kilobytes of memory was reminiscent of the creative limitations that led to the inventiveness of game developers in the early days of handheld games. Working with the pedometer’s multitude of registers and researching the initialization solidified the skills in I2C communication learned in earlier courses.

The idea of gamification, considering all the different paths that could’ve been taken to make the experience more enjoyable for the user, took some serious thought, even for such a simple game. The complexity of making a rewarding experience, and the complexity of designing art and graphics for the Silly Little Guy was eye-opening. It gave the team insight and inspiration into making more games like this.

Finally, coming together as a team, getting to know each other and how to work together, taught patience and perseverance. Coming into the second half of the term, the list of tasks to be done were long. But this wasn’t taken as something discouraging. The team put in the time and effort needed to meet specification. Everyone looks forward to where they can take the skills learned here moving forward.

# References

Dmitry Grinberg. (2020). PokéWalker hacking. <https://dmitry.gr/?r=05.Projects&proj=28.%20pokewalker>

Aseprite. (2025). <https://www.aseprite.org/>

Wikipedia. (2025). *Tamagotchi*. <https://en.wikipedia.org/wiki/Tamagotchi>

BOSCH. (2025). *BST Data Sheet*. <https://cdn.sparkfun.com/assets/e/9/e/6/0/BMA400-Datasheet.pdf>

STMicroelectronics. (2024a). STM32G0B1xB/xC/xE. <https://www.st.com/resource/en/datasheet/stm32g0b1re.pdf>

STMicroelectronics. (2024b). STM32G031x4/x6/x8.

<https://www.st.com/resource/en/datasheet/stm32g031k8.pdf>

ControllersTech. (2020). ST7735 1.8″ TFT Display with STM32. [How ro interface ST7735 1.8" TFT Display with STM32](https://controllerstech.com/st7735-1-8-tft-display-with-stm32/)

STMicroelectronics. (2024c). RM0444 Reference manual - STM32G0x1 advanced Arm®-based 32-bit MCUs. <https://www.st.com/resource/en/reference_manual/rm0444-stm32g0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf>

Lady Ada et al. (2025). 1.8" TFT Display Breakout and Shield. [Downloads | 1.8" TFT Display Breakout and Shield | Adafruit Learning System](https://learn.adafruit.com/1-8-tft-display/downloads)

Dryw Wade. (2022). SparkFun\_Qwiic\_Accelerometer\_BMA400. [SparkFun\_Qwiic\_Accelerometer\_BMA400.sch](https://cdn.sparkfun.com/assets/0/2/5/5/b/SparkFun_Triple_Axis_Accelerometer_Breakout_-_BMA400__Qwiic_.pdf)

Tymm. (2016). STM32 Nucleo Coaster. [STM32 Nucleo coaster by tymm - Thingiverse](https://www.thingiverse.com/thing:1589714)

Kosma. (2014). minmea, a lightweight GPS NMEA 0183 parser library. [kosma/minmea: a lightweight GPS NMEA 0183 parser library in pure C](https://github.com/kosma/minmea)

Adafruit. (2024). Adafruit\_GPS. [adafruit/Adafruit\_GPS: An interrupt-based GPS Arduino library for no-parsing-required use](https://github.com/adafruit/Adafruit_GPS)

Adafruit. (2025). Adafruit-ST7735-Library [adafruit/Adafruit-ST7735-Library: This is a library for the Adafruit 1.8" SPI display http://www.adafruit.com/products/358 and http://www.adafruit.com/products/618](https://github.com/adafruit/Adafruit-ST7735-Library)

# Appendices

## Appendix A - Excerpt From Proposal:

This project aims to create a step-counting device that will encourage people to explore outdoors and exercise more frequently by gamifying the walking process. It will sense the user's taken steps as well as their physical position and convert that information into game data. The solution centers around an embedded system featuring a “virtual pet” style game. A sensor detects steps taken by the carrier during the day and stores them in a lifetime step count. The “stats” of the virtual pet will be manipulated in response to the user’s physical activity. Geolocation hardware will record the position of the device and reward the user for visiting new positions. A clock system will track the current time. A save system on the device will persistently store user data. The device will be able to interface with a desktop application where the user can view, store, and manage their data. Save data will be synced to a database. The app will offer a more high-definition interface for the data, including a scaled-up version of the pet, and a map showcasing explored positions.

## Appendix B - Specification:

**This device will:**

1. Display graphics that will appear the same on the game screen as they would on any other computer, meaning they will have no pixels out of place.
2. Connect to an interface with a sensor and interpret sensor data to count individual steps within an error of 5 steps out of 100.
3. Track its position within 50 meters and be capable of comparing the current position to visited positions with minimal drift.
4. Produce sounds in response to user input and internal software triggers.
5. Utilize analog controls to interact with the game.
6. Store daily, weekly, and lifetime data to non-volatile memory, the first two categories resetting after their assigned timeframe has elapsed.
7. Evaluate daily, weekly and lifetime user data and provide positive or negative feedback through an interactive virtual pet.

**The Virtual pet will:**

1. Mature and provide increasingly rewarding cosmetic changes as the weekly steps taken reach a preset goal.
2. Apply modifiers on the change in its stats based on the visited and current position data.
3. Allow the user to change the goal/difficulty level.

**The Application will:**

1. Utilize a wired connection with the device to load the device’s data, display the user’s stats, and include a map that showcases visited locations.
2. Sync game data to a database, with explicit control over whether or not the positional data is synced.

## Appendix C – Verification Tests:

Project: Silly Little Guy (SLG)

Team: Aurora Fraser, Bryce Frey, Connor Marsh

Date: Jan 22, 2025

1. The device should display graphics that appear the same on the game screen as they would on any other computer, meaning they will have no pixels out of place.

|  |  |
| --- | --- |
| Test | Power the Silly Little Guy, verify that the graphical interface powers on and displays the starting image/text. The starting images and text should be fully legible and identical to the source files.  No pixels should be missing, moved, or added from the initial graphic design. |
| Results | Text the same as intended and images were undistorted. No errors noted. |
| Conclusion | The lcd and graphics system is capable of displaying images and text. |

1. The device should connect to an interface with a sensor and interpret sensor data to count 5 individual steps out of 100.

|  |  |
| --- | --- |
| Test | Go for a walk with the Silly Little Guy and count the number of steps the user takes. When the user has walked 100 steps, compare that empirical value with the values on the Silly Little Guy.  To account for possible systematic errors, Silly Little Guy should be within 5 steps out of 100. |
| Results | Silly Little Guy tracked 137 steps out of 139, or 98.5 steps out of 100. |
| Conclusion | Silly Little Guy can track steps within tolerance. |

1. The device should track its position within 50 meters and be capable of comparing the current position to visited positions.

|  |  |
| --- | --- |
| Test | Go for a walk with the Silly Little Guy. Track positional data from a cell phone. Whenever the sensor gets a fix parse record the positional data from both the sensor and the phone. Repeat this test 20 times.  16 out of 20 times the sensor should present positional data within 50 meters of the phone’s positional data. |
| Results | Did not repeat the test 20 times, repeated 3 times and positional data was within 50 meters 3 times. |
| Conclusion | GPS is accurate within given tolerance. |

1. The device should produce sounds in response to user input and internal software triggers.

|  |  |
| --- | --- |
| Test | Take the following actions and record the responsive sound effect:   * Select an item from the menu. * Induce a change in the pet’s emotional state * Evolve the pet to the next stage * Wait for the pet to emote   Each action should evoke a different sound effect. |
| Results | Each action made a different sound effect, the pet emote can take anywhere from 10 seconds to over a minute to play as it is pseudo-random, some sounds are similar but none of them are the same. Sound effects can get cut off sometimes when switching menus. The pet emote sound effect is responsive to the emotional state, but does not play when the state changes. |
| Conclusion | The device does provide sound effects in response to the player’s actions/game state. |

1. The device should utilize analog controls to interact with the game.

|  |  |
| --- | --- |
| Test | Interact with the 3 buttons on the Silly Little Guy, the left and right buttons will be used to navigate the menu(s). The center button will act as a select button.  These buttons will all perform one specific action, and their effects should not overlap with the other buttons. |
| Results | The left and right buttons navigate the menus, holding down the buttons long enough will move multiple menus over. Center button selects options in the settings menu. |
| Conclusion | The buttons function as intended. |

1. The device should store daily, weekly, and lifetime data to non-volatile memory, the first two categories resetting after their assigned timeframe has elapsed.

|  |  |
| --- | --- |
| Test | To test non-volatile memory storage, accumulate data into the Silly Little Guy and then power it down. Turn it back on and verify that the step counters were not reset.  To test the reset feature, programmatically replicate the end of a day/week. Then ensure that the data has been wiped from memory and reset to 0.  For the sake of brevity, the progression of time on these devices may be accelerated. |
| Results | Data programmatically cleared on pressing the center button in the main menu, and the device is reset using the reset button on the back of the device. Clearing nvm and reset has to be done within one second or device will autosave. Powering device off and on when autosaved will persist the game state. |
| Conclusion | Memory is persisted, and reset as needed. |

1. The device should evaluate daily, weekly and lifetime user data and provide positive or negative feedback through an interactive virtual pet.

|  |  |
| --- | --- |
| Test | Programmatically replicate a positive previous day into the Silly Little Guy. Then go to the next day and ensure the Silly Little Guy is visually happy.  Do the same but for a negative previous day. Silly Little Guy should now be visually upset. |
| Results | Time for day shortened to 24 minutes, Silly Little Guy got sad. Walked around, Silly Little Guy went to neutral, then happy. |
| Conclusion | Negative and positive feedback provided through emotional state. |

1. The virtual pet should mature and provide increasingly rewarding cosmetic changes as the weekly steps taken reach a preset goal.

|  |  |
| --- | --- |
| Test | Set the threshold value to 30 steps, walk until this goal is reached. Observe the changes in the virtual pet.  Once the goal is reached, the Silly Little Guy will undergo a visual change, getting bigger for each consecutive change. |
| Results | Silly little guy evolved after 50 steps, did not set the threshold value to 30. Threshold value in exp as a function of steps and number of locations instead of steps. |
| Conclusion | Silly little guy provides positive cosmetic changes when weekly goal is met. |

1. The virtual pet should apply modifiers on the change in its stats based on the visited and current position data.

|  |  |
| --- | --- |
| Test | Reset the device. Walk the virtual pet to 3 new positions, and verify those new locations are entered into the device. Make sure to record the number of steps taken. Artificially end the week and record step bonuses. Reset the device and repeat this process with position logging disabled, making sure to take the same number of steps as in the previous trial. Once again end the week and record step bonuses.  Two observations should be made, one, that the steps at the end of the test that involved new locations should be at least 40% higher than the steps at the end of the test with tracking off. |
| Results | Value in exp as a function of steps and number of locations instead of steps. Exp calculated constantly instead of end of week. Exp with 3 locations was 229. Exp with no locations was 131. This is consistent with the desired .25x bonus per location. |
| Conclusion | Locations have the desired bonus effect. |

1. The virtual pet should allow the user to change the goal/difficulty level.

|  |  |
| --- | --- |
| Test | Select two different difficulties and observe the change in the goal displayed in-game. Artificially increase the steps taken until the goal has been reached.  The goal should change in-game, and programmatically change, so that when met, they register as completed. |
| Results | Goal was originally set to 50, goal completed at 50. Device was reset, goal was changed to 0. Goal completed at 0. Goal value in exp as a function of steps and number of locations instead of steps. |
| Conclusion | Difficulty can be changed. |

1. The application should utilize a wired connection with the device to load the device’s data, display the user’s stats, and include a map that displays visited locations.

|  |  |
| --- | --- |
| Test | Ensure there is a valid save on the device. Walk the device to a new position and note the address of the position visited. Connect the device to a computer and open the application. Connect to the device in the application. Observe the changes to the display. Select the map option, and observe the locations noted on the map.  All the same stats should be visible, as well as the image the pet is displaying on the device. The registered address would be within 100 m of the noted address. |
| Results | Registered position within 100 meters of noted address. Weekly, daily, and lifetime steps displayed identical to display on device. Emotional state displayed. |
| Conclusion | Application successfully imports and displays user stats. |

1. The application should sync game data to a database, with explicit control over whether the positional data is synced.

|  |  |
| --- | --- |
| Test | Connect the device to the desktop computer and open the application. Connect to the device in the application. Select the option to sync data, and do not change any configuration options. Observe any changes in database management software. Make a note of all the device’s stats and reset the device. Open the application and connect the device on a separate computer, hit sync, and observe changes in the application and device.  The database management software should show that the data has been imported with the correct structure. Making use of two computers ensures none of the original data is saved locally, and if both the application and the device’s stats were restored completely, the syncing was successful. |
| Results | Only had access to one computer but pulled the application and rebuilt it before running the second test. Device stats accurately synced to and from database. No locational data options in form. |
| Conclusion | Database syncing implemented, but locational data autosyncs. |

## Appendix D – Work Breakdown Structure:

**Level 1**

1. Silly Little Guy (SLG) ... ......................... 100

**Level 2**

1. **Silly Little Guy ................................... 100**
   1. Step Tracking ............................. 10
   2. Location Data ............................. 15
   3. Audio Drivers ............................. 10
   4. Graphic/Audio design ...................... 15
   5. Game design ............................... 20
   6. Application/Database design ............... 15
   7. Integrated Product ........................ 10
   8. Project Management ........................ 05

**Level 3**

1. **Silly Little Guy ................................... 100**
   1. **Step Tracking ............................. 10**
      1. Pedometer ............................ 05
      2. Step Tracking Interface .............. 05
   2. **Location Data ............................. 15**
      1. GPS Interface ........................ 05
      2. Position Tracking/Storage system ..... 10
   3. **Audio Drivers ............................. 10**
      1. Piezoelectric buzzer ................. 05
      2. Sound storage system ................. 05
   4. **Graphic/Audio design ...................... 15**
      1. Pet Animations ....................... 05
      2. Audio/Sound Elements ................. 05
      3. User Interface ....................... 05
   5. **Game design ............................... 20**
      1. Evolution ............................ 05
      2. Stat Tracking ........................ 10
      3. Emotions ............................. 05
   6. **Application/Database design ............... 15**
      1. Serial Communication ................. 03
      2. Import and Display Upscaled Game ..... 04
      3. Display Map .......................... 04
      4. Persist Saves ........................ 04
   7. **Integrated Product......................... 10**
      1. Layout ............................... 04
      2. Casing ............................... 03
      3. Battery .............................. 03
   8. **Project Management ........................ 05**