**IRRIGATION SYSTEM**

**PROJECT PROPOSAL**

### Elizabeth Minty

Analysis

|  |
| --- |
|  |

Plants are living organisms, each with specific needs that must be timely met. Many plant owners see them wither before they can flourish. As people become busy with life, plants are often left unattended with their owners not around.

This is where the idea for an automatic smart home irrigation system came about.

A simple system that keeps track of a plant’s moisture and humidity levels with automatic watering to allow plant owners to ensure their plants stay well maintained regardless of where life takes them.

The simple, compact and, most importantly, portable design allows for consumers to use it for any plant, anywhere. In the home, the garden, the greenhouse – anywhere!

Design

|  |
| --- |
|  |

The smart irrigation system (SIS) had a flexible scope, allowing for many different versions to be ideated. Each version was based on features I could implement and time/material constraints.

**Version 1:**

- Measures the soil moisture for any given plant and, based on a threshold, determines if the plant needs watering

- Photoresistor used to detect light level OR humidity sensor

- Light turns on to let plant owner know that it needs watering/environment change

- LCD screen displays moisture and environment, if it's dispensing water, ON/OFF, and plant name

- Watchdog timer and FSM

**Version 2:**

- A pump dispenses water based on the threshold

- Buttons for user interation/changing info displayed on screen/on and off

**Version 3 (USB Serial communication OR wifi):**

- App that sends and receives plant data

- Arduinos sending/receiving plant info/commands based on plant info from plantapi

- Dispensing water based on received command/info

**Version 4:**

- 3D Printed arduino case that allows for clipping on and a pot

Version 1 is the first scope I decided on for the main project idea. Version 2 implemented user interaction and automatic dispensing. This was the fully fledged market-ready version that I aimed for.

Version 3 and 4 were more about the final product I wanted to produce.

The first 2 versions met the idea of a “simple system that can be used in the home that automates the process of taking care of a plant”. However, the convenience of this system was lacking. The system would not have durability, portable casing, nor perfect cable management. The system would also not be able to cater for extensive plant needs nor different watering times depending on the plant.

Since the first 2 versions would resolve the initial problem, and due to the time constraint, this was to be the final scope of the design.

Insights

|  |
| --- |
|  |

Looking for an aesthetic piece of tech to add to your collection or for wear? The GoMoji acts as a keychain as well as a badge to add to any outfit. The device itself offers a unique appearance for users to decorate and change to their preference, fitting with any aesthetic or outfit any day.

The GoMoji is a simple, yet rewarding device that gives users small doses of satisfaction throughout the day as they complete tasks to ensure the comfort of their virtual pet.

As it is small and portable, users can use it anywhere even in class or meetings! This offers the chance to share with friends and converse about their unique pets.

Implementation

|  |
| --- |
|  |

The solution that the GoMoji will offer after further iterations is as follows:

* + Evolutions and ability to dress the pet
  + A cute pet that rewards users when they take care of it
  + More states/needs like sickness, excitement, contentment, focus, etc
  + More options to interact with the GoMoji
  + Sound and music
  + Small, portable, and simple design to allow for user customization

Currently, the product offers the following solution:

* + A cute pet that changes design depending on state
  + Basic states/needs such as contentment, hunger, affection, and fatigue
  + 3 colours to represent these states (red = tired, yellow = hungry, green = affection needed)

Scope

|  |
| --- |
|  |

The scope for the project was simple: display 3 states of being at random intervals using 3 LEDs, an LCD screen for the character, and 3 buttons to revert those states to normal. This remained relatively the same for the duration of the project’s implementation. Ideally, a box to contain the Arduino and breadboards was to be made if there had been time.

There were some limitations as follows:

* + Tight schedule due to academic and personal commitments
  + This being a solo project
  + Materials

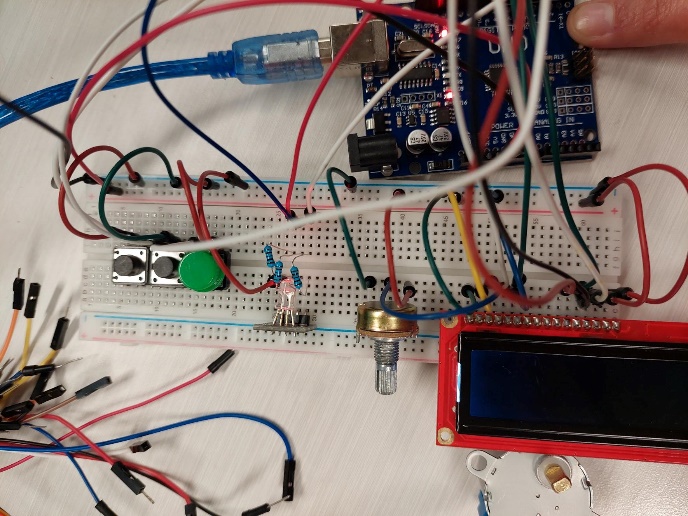
A lot of time went into research on hooking up each part, trying out different parts, methods for the LCD library, code examples, and how to display unique characters. I learned a lot about the pin mapping of different parts and further developed my interest in hardware.

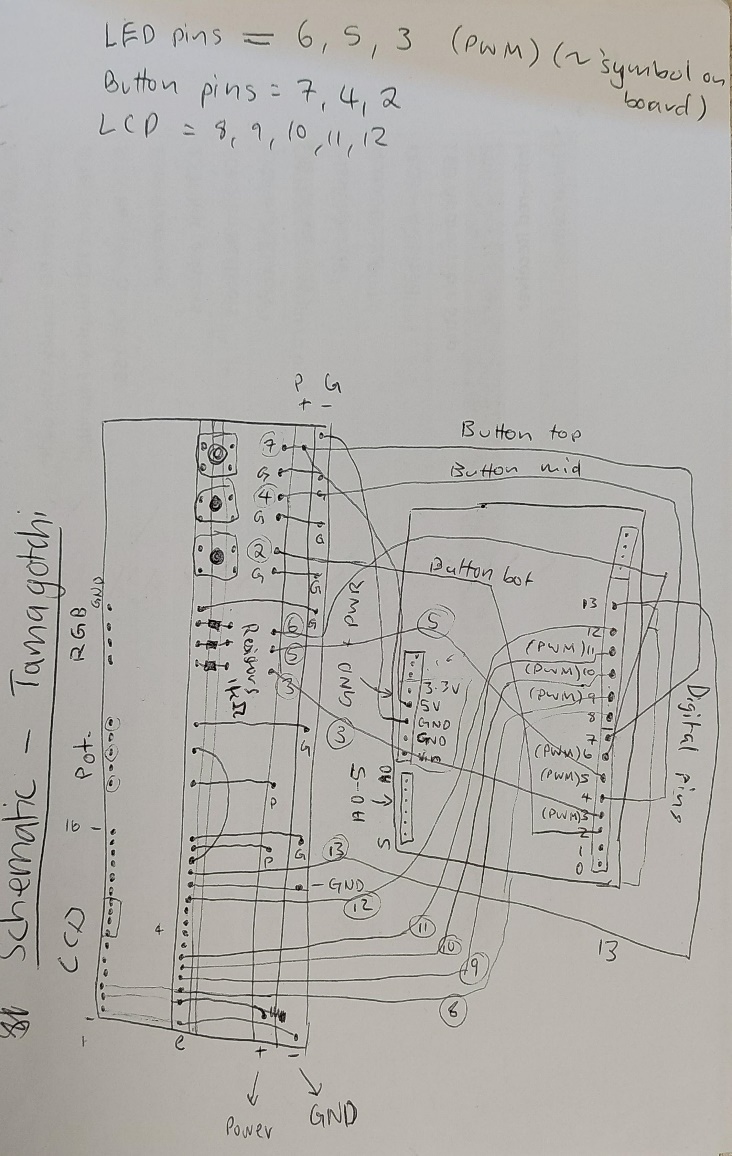
A major assumption that disrupted the flow of production was that the SparkFun guide and LCD screen would work. A big issue at the beginning of the project was that I used a SparkFun LCD screen along with the associated hookup guide to use for my GoMoji display. I tried a second SparkFun LCD and that also did not work. I then had to move on to an LCD screen with a different manufacturer which then worked with my hookup.

Design Process

|  |
| --- |
|  |

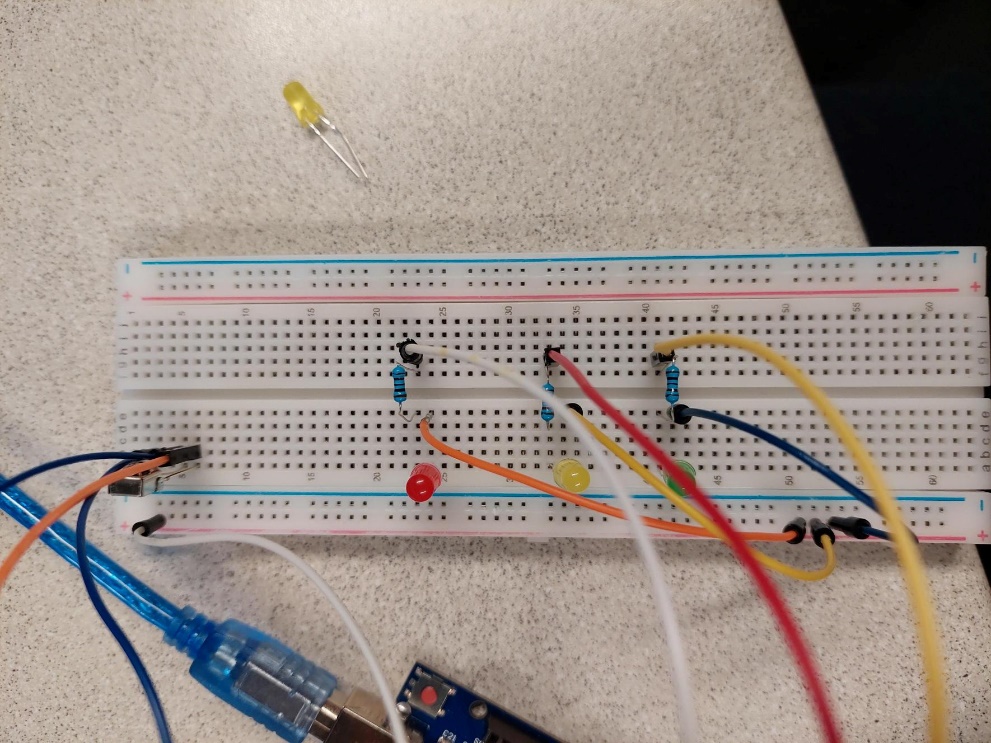
Due to the issues with the SparkFun LCD (SFLCD), the schematic changed pretty frequently. I started off with the SFLCD, a 10k potentiometer (pot), 1 breadboard, an RGB led, 3 buttons, and 3 resistors.

 *Fig 1.a. Initial Schematic with Pot, SFLCD, and RGB LED*

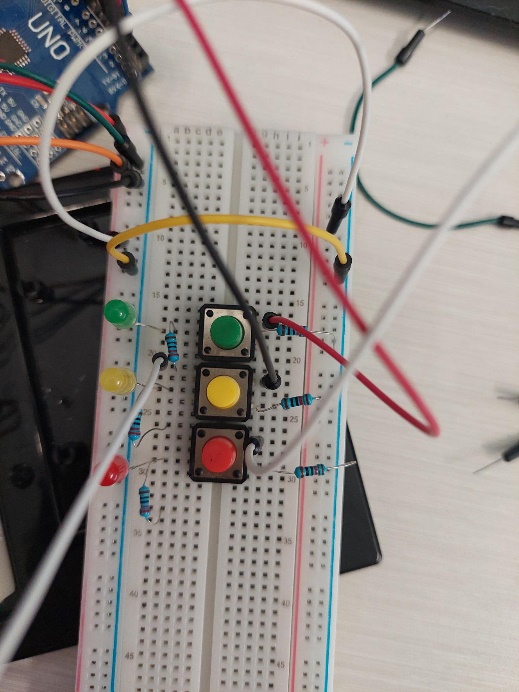
*Fig 1.b. Sketched schematic for initial design with SFLCD, pot, and RGB LED*

This was the first design I had in mind which, according to my sources, should have worked. I had to replace the SFLCD with a different manufacturer and decided to use individual LEDs rather than an RGB LED. The reason for this was that I only needed 3 colours for 3 states at the time and didn’t have a second potentiometer.

I asked Vaughn for help over the weekend. So, as I waited, I started planning for the LEDs and buttons.

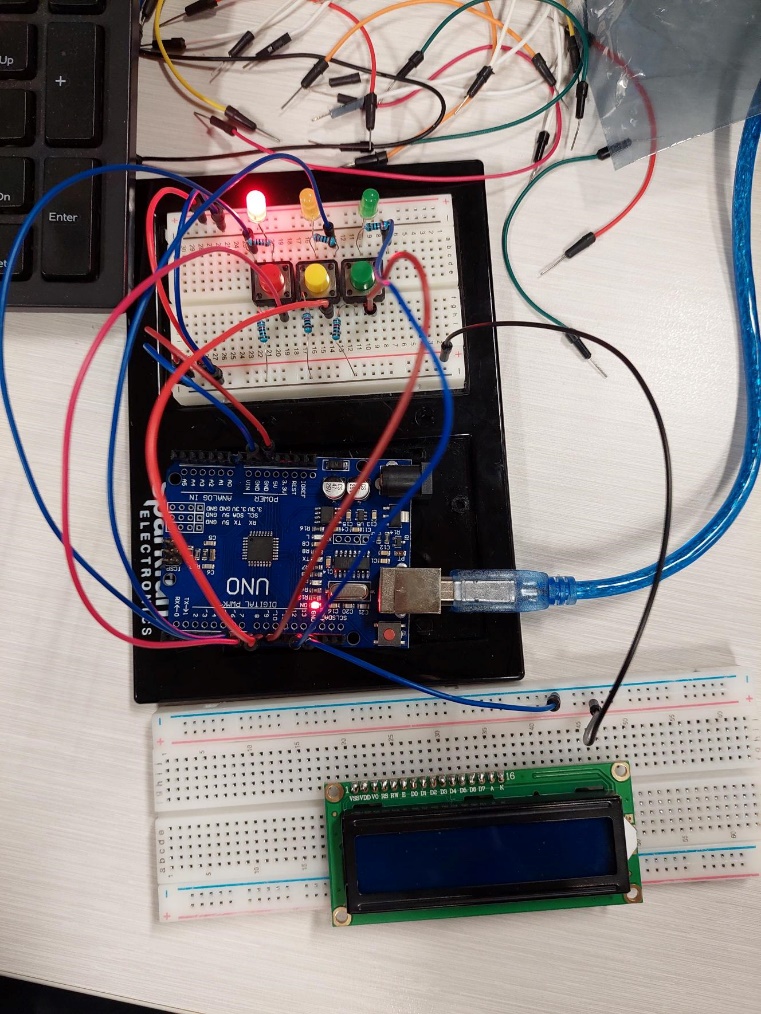
*Fig 2.a. Initial setup of 3 LEDs* 

I started with the LEDs. The scope required 3 LEDs (minimum) that would randomly light up to display a need/state of the pet. This was to happen at random and would only have 1 LED on at any time. The LED would stay on until turned off via the corresponding button press. This is the set up for the LEDs on their own in the final design as well. I initially coded the LEDs to turn on at random individually for this draft design. This helped with the final code by providing some setup as well as solidifying my knowledge on coding with LEDs.

*Fig 2.b. Initial setup of 3 buttons* 

Following the draft design of the LEDs, I implemented the same practice for the button design. I set each button up to ensure I had a good idea of how they need to be wired.

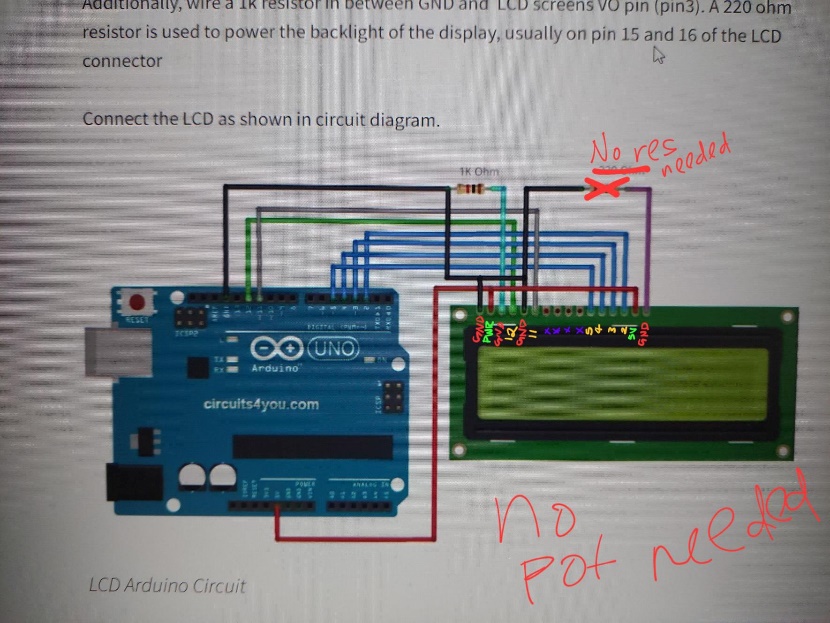
*Fig 3. Final design for buttons that turn off LEDs*



The last step I took before coding the LCD screen, I put the LED and button designs together to ensure they worked seamlessly. Then, I created code that turned on a randomly selected LED at a random interval that would not turn off until the corresponding-coloured button was pressed and released.

This step took a significant amount of time due to my assumption that I’d wired the resistors correctly. The LEDs would keep flickering, would change colour without a button press, and the read value of each button would change without being pressed. Turns out, I had connected the resistor for the LEDs to the buttons as well, so the buttons had 2 resistors. Fortunately, while I had been under the impression that it was a programming issue, I ended up improving my code along the way.

*Fig 4. Final LCD hookup schematic for LCD screen -* [*Normal LCD Hookup Reference*](https://circuits4you.com/2016/05/15/how-to-lcd-display-arduino-uno/)

Finally, after adding the LCD hookup with the button and LED hookup, the wiring worked. After testing the code for the buttons and LEDs, all that was left was to add code printing characters to the screen that changes depending on each LED state. This was a pretty seamless process. The extra part was attempting to add some form of double buffering, but this was out of scope for this project.

**Final Component List:**

* 1x LCD screen 2x16
* 3x buttons
* 3x LEDs
* 1x 1k Ohm resistor
* 3x 220 Ohm resistors
* 3x 10k Ohm resistors
* 20x wires

I decided to use a 2x16 LCD screen to keep to the idea of being small and portable for future iterations instead of a larger one, despite the improved graphics. I did start with a potentiometer and RGB LED but, as previously stated, it was deemed unnecessary for the scope of this iteration.

Some flaws with the current state of the product is the flickering of the LCD screen when the character design changes to represent states other than content.

I did try to implement some buffering using button interrupts, but the concept did not quite tackle the issue. Additionally, the project uses 3 buttons and there are only 2 interrupt pins.

I could try to connect all 3 to 1 interrupt pin, but that would not resolve the flickering as the character changes when the LED state changes, not when the button state changes. However, I believe this flickering could be improved with more research.

There is a lot of code that could be improved in terms of modularity. There are methods that could be added to make the code cleaner as well.

Code Defence

|  |
| --- |
| All code used is original. Although references were used, no code in these references was replicated and kept. References used are listed in the document titled “references.txt”.  These references were used for the following purposes:   * Hook-up guides * LCD Debugging * LCD Character displaying * Basic button with LED – turn on and off function * Datasheets and pin maps for each part * SparkFun LCD tutorial |

Future Improvements

|  |
| --- |
|  |

Clearly, the solution isn’t perfect. There needs to be a few more iterations before it can completely meet every possible use case in its entirety. There is flickering when the states change on the LCD screen, there are limited states, the code could use more modularity, and more. However, this is only the first iteration.

This first iteration solves the issue, but it’s only the beginning.

Ideally, I’d like to incorporate every aspect of the ideal solution mentioned previously such as more states/needs, music, pet evolution, accessory creation, and more.