Introduction

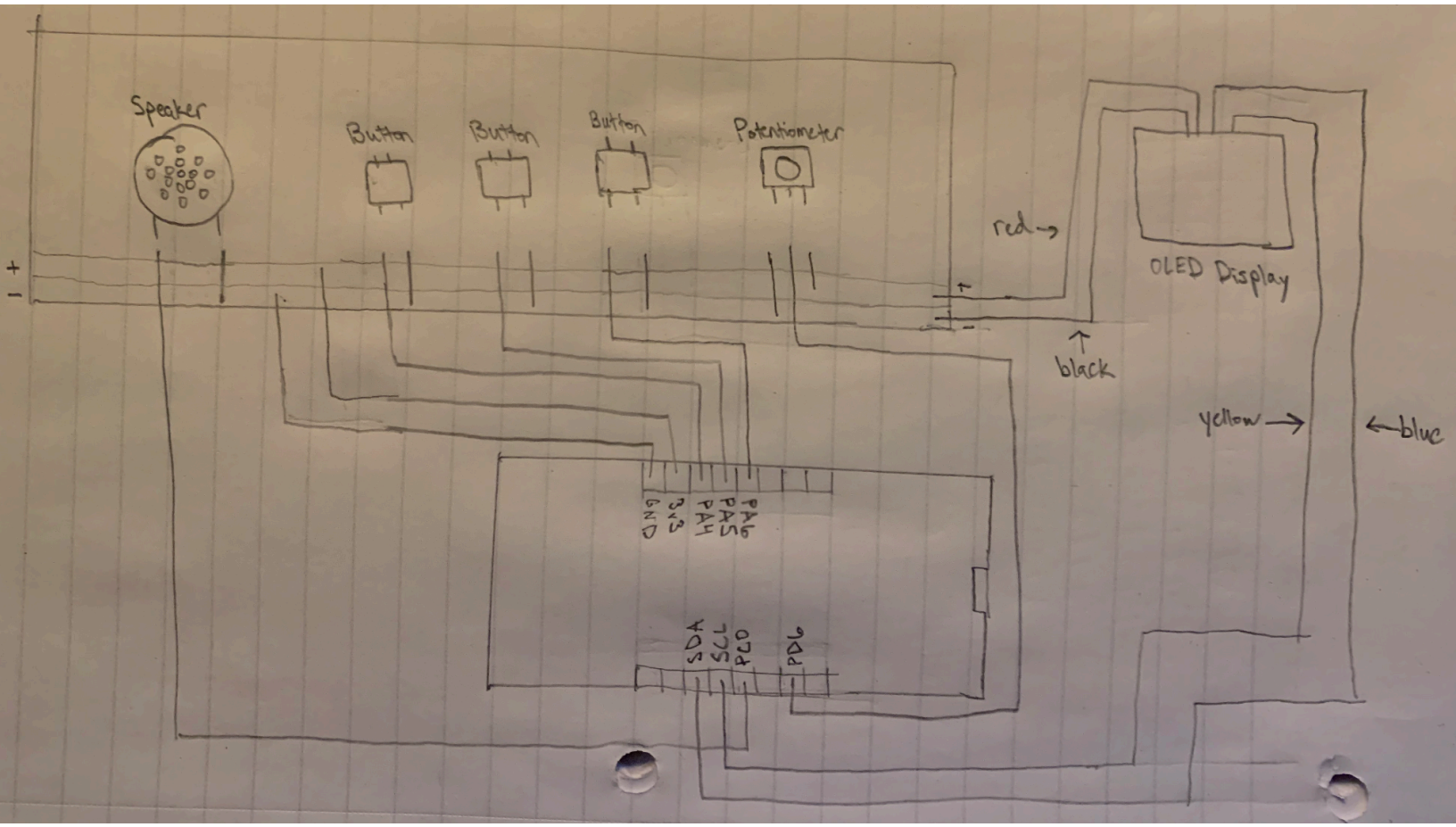
For my project, I created my own version of a Tamagotchi, which is a handheld digital pet that was incredibly popular in the late 1990s and early 2000s. I chose this project because of the many possibilities for features that would allow me to gain further experience with a variety of embedded concepts. My end product doesn't solve any problems, but is a fun toy that allowed me to gain practical experience with writing embedded code. The recreated Tamagotchi runs entirely on an embedded system, running on an AVR-BLE chip with an ATmega3208 microcontroller. There are several peripherals involved in the project that communicate over embedded protocols including GPIO, I2C, and USART. There are two buttons and a potentiometer that allow a user to interact with the system in real time through interrupt driven programming. Waveform generation is also used to play music through a speaker.

Design and Implementation

Hardware Components:

- AVR-BLE Development Board (1)
- Speaker (1)
- Button (3)
- Potentiometer (1)
- 48x64 Monochrome OLED Display (1)
- 4-pin Qwiic Breadboard Jumper Cable (1)

Hardware Setup: A central breadboard is used to connect all of the components. Directly on the breadboard lies the speaker, the three buttons, and the potentiometer, each of which have a direct connection to the AVR-BLE. The OLED display has a Qwiic connector port, and a flexible 4-pin Qwiic breadboard jumper cable connects it to the breadboard and the AVR-BLE.



OLED Display Description: Communication with the display is done over I2C. After specifying the device address, a command byte is sent that informs the display whether the following transmission contains data bytes or command bytes. Data bytes will automatically update the GDDRAM of the display and be reflected on the screen. The display is 48x64 pixels. The 48 rows are split into 6 groups of 8 rows called pages. The current page and column for updating can be specified through commands. Sending one byte of data updates the value of the 8 pixels at the current page and column, where a 1 indicates “on” and a 0 indicates “off”. Different addressing modes can be specified that determine what happens when the current column reaches the specified end column. All interactions used in the system use a horizontal addressing mode that makes it so the current column

is automatically incremented after each data byte, and when the specified end column is reached, the current column is reset to the start and the page is incremented.

Hardware Choices: The Qwiic enabled display was chosen for its quick and easy connection to the rest of the components, and the flexible connector cable allows it to be easily handled and moved around. The display is only 0.66” across, making it fairly small, and monochrome, limiting its display capabilities. This was chosen to minimize the necessary RAM for communication with the display. Additionally, an actual Tamagotchi is monochrome and fairly small, so it is true to the project inspiration. A more robust development board than the AVR-BLE with arduino capabilities may have made the project simpler, but its use was a requirement for the project and it was able to accomplish the requirements without issue.

Software File Breakdown:

- main.c - Performs all initialization and contains the main() function.
- i2c.c - Functions used for I2C communication with the OLED display. Also holds the uint8_t arrays that store the graphic designs.
- ble.c - Functions for communication with the AVR-BLE BLE peripheral over USART. Also contains the ISR for responding to bluetooth messages that trigger interrupts.
- buttons.c - ISRs for handling button interrupts.
- rtc.c - Initialization of the AVR-BLE RTC peripheral, as well as ISR for handling RTC overflow and compare interrupts.
- music.c - TCA0 initialization for waveform generation used to play music over the speaker.
- shapes.h - Header declaring the external arrays storing the graphic design data, allows methods outside of i2c.c to call these.

- globals.h - Header declaring variables used across several files.
- i2c.h - Header for i2c.c.
- ble.h - Header for ble.c.
- rtc.h - Header for rtc.c.

Software Requirements:

- Maintain track of real time, as a key component of a Tamagotchi is to have time dependent interactions and state modifications.
- Handle bluetooth communication, including both sending and receiving data, at any given moment.
- Respond to user input according to the current state of the device.
- Update the display in real time according to user interactions and the state of the system.
- Store all the necessary design data without exceeding 4KB of RAM.
- Generate music in some way.
- Make the current state of the pet clear to the user at all times.

Software Design:

- The system is entirely interrupt driven. The RTC peripheral is used to generate an overflow interrupt every .5 seconds. This interrupt is used to update the status of the pet, including its hunger, happiness, level, and sleep. The system makes use of global state variables to decide what steps to take at each interrupt. For example, there is a variable that indicates the pet is sleeping when it has a non-zero value, and this makes it so the RTC overflow interrupt won't toggle the position of the pet to give the jumping animation.
- Bluetooth communication is accomplished through USART communication with the BLE peripheral built into the AVR-BLE board. The system uses an ISR that triggers depending on the

USART.STATUS register to determine when there is new data available for reading. This ISR is triggered every time a new byte is available for reading. Received data always consists of more than one byte, so the interrupts are disabled at the beginning of the ISR, the entire transmission is read, and then the interrupts are re-enabled. Sending data will cause a response to be returned, so a state variable is used to indicate when data is being sent. This allows the system to differentiate between response transmissions and data transmissions so they can be handled accordingly.

- User input through the buttons and potentiometer is handled via ISRs. The function of the first two buttons depends on the current state of the system. Once again, state variables are used to determine the appropriate response. Depending on the state, one or both of the buttons may be temporarily disabled by disabling their interrupts. The third button is to be used in conjunction with the potentiometer. While pressing the third button, the potentiometer can be adjusted to change the contrast of the screen. The potentiometer is disabled when this button is not being pressed. The potentiometer input is read using the ADC peripheral and mapped to a value from 0-255 to be sent to the display as a command.
- Communication with the display is done over I2C. All I2C functions follow the same pattern - initiate transaction, send data, end transaction. Two core functions are used for updating the display, draw() and erase(). Draw() takes a pointer to an array storing the design data and simply loops through it to send to the display as data. Erase() sends a specified number of 0 bytes, which “erases” the screen by ensuring each of those pixels are turned off. Separate helper functions are called by these to send the commands for updating the column and page addresses. The draw() function provides a very simple and efficient method for performing most of the display updates. It allows for each design to be stored as its own array, and every design can utilize the same function. There are some

exceptions to this for designs that use their own functions for special cases. The majority of the time, only a small portion of the screen needs to be updated. Having a draw() function like this makes it so only the necessary portion is updated instead of redrawing the entire screen, which is much more efficient.

- The TCA0 peripheral is used for waveform generation directed to the speaker to produce music. Updating the CMP register changes the duty cycle, which changes the frequency of the sound being produced to play different notes. A “notes” array stores the CMP values that produce specific notes, and a separate “song” array stores indices into this notes array. Looping through the song array will give you the indices for the notes to play for that song. Music is only played during the “game” mode of the system. When “game” mode is entered, the RTC peripheral’s compare interrupt is also enabled to trigger every .25 seconds. The RTC peripheral then updates the CMP value at every compare and overflow interrupt to play a song.

Technical Challenges:

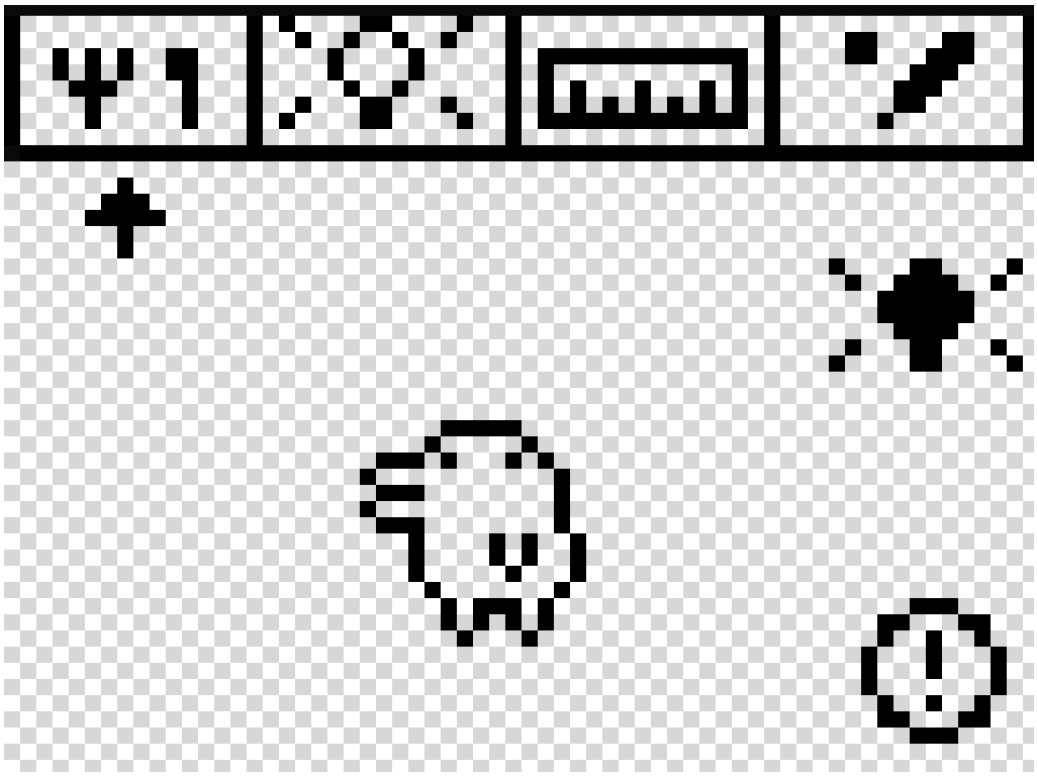
- Display: The process for initializing the display screen is somewhat confusing. It requires roughly 10 different commands, and the order in which they are sent matters. Additionally, despite the display only being 48x64, it actually stores 64x128 pixels of data. It is easy to be unknowingly writing to the non-displayed section and cause confusion when first getting the screen working. There is a command to set the starting column of the display, however I was unable to use it to reset this column to 0 for some reason. It was stuck at the first display column being column 32. In order to work around this, I add 32 to the values passed to the helper function that updates the column addresses.
- Memory: The AVR-BLE only has 4KB of RAM. Because of this relatively small amount, I wanted to be as efficient with my memory usage as possible. The setup for playing music I

described was an efficient method for saving memory. Instead of needing to store an entire song of 16 bit values for updating the TCA0 CMP value, a notes array stores 16 bit values for every possible note. Then, the song array stores 8 bit index values into this array. This method cuts the necessary memory for a song in half. Another way I saved memory was by writing methods that modify the already stored graphic data instead of storing unnecessary graphics. To perform the jumping animation, I wrote a function that upshifts the graphic instead of storing an upshifted version. This function is slightly more complex than just performing bit shifts to each value, since it is also necessary to check the page underneath for bits on the top row that would need to be included in the page above. I only use this function for upshifting the pet, but it is a versatile function that can be applied to any given graphic. Another method that saves memory in this way is the reverse() method that flips the direction of the graphic. If a graphic is split among multiple pages, more effort than just reversing the read direction of the graphic is necessary, and this function accomplishes that. Lastly, I experimented with limiting repetitive data for symmetric designs. For the lightbulb graphic, I stored only half of both the “on” and “off” graphic, and since they are symmetric, they can be reversed to obtain the second half.

- Timing: Since the TCA0 peripheral was being used for controlling the speaker, I couldn’t use it as a timer. I wanted to use it to control momentary events more precisely, like having the feeding animation display the apple for a certain amount of time. I attempted to use the TCB peripheral as a substitute, but I found that it does not allow for nearly as much prescaling as the TCA0 peripheral does. I attempted to still use it by only acting once it reaches a certain number of interrupts, but I found that it delivered unreliable results. Instead, I just used state variables that would then be checked by the RTC every overflow interrupt.

Final Product

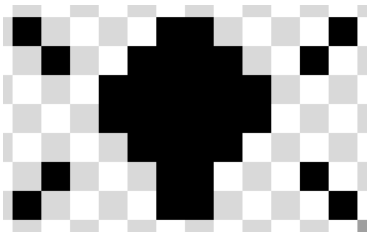
My project resulted in a fun, digital pet with 5 main interactable features. Besides the design for the sideways facing duck, which I got from the actual Tamagotchi website, I did all of the designs myself using a free online pixel drawing site. Here is a screenshot of what the normal home screen looks like.

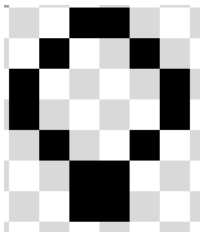


You can see that there is a top bar with four different options. Pressing the green button cycles the cursor through the options to show the currently chosen option. Pressing the blue button selects the currently hovered option. The pet is the duck-looking character in the center. Its position toggles up and down every half second to make it appear as though it's jumping. The alert circle in the bottom right appears when the pet’s hunger or happiness is below half. Getting both to over half will make the alert go away. The lightbulb on the right side of the screen indicates whether the light is currently on

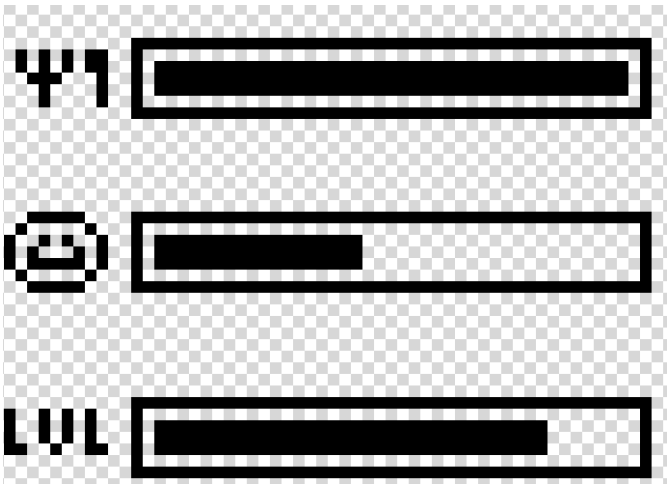
or off. The light needs to be on while the pet is awake, or their happiness will begin to drop.
Selecting the first option feeds the pet. When it is clicked, an apple will appear on the screen in front of the pet. If the duck was hungry, a heart will pop up on the screen, and they will become less hungry. If they were full, an ‘X’ will pop up on the screen, and their happiness will take a hit.

Selecting the second option toggles the light on or off.



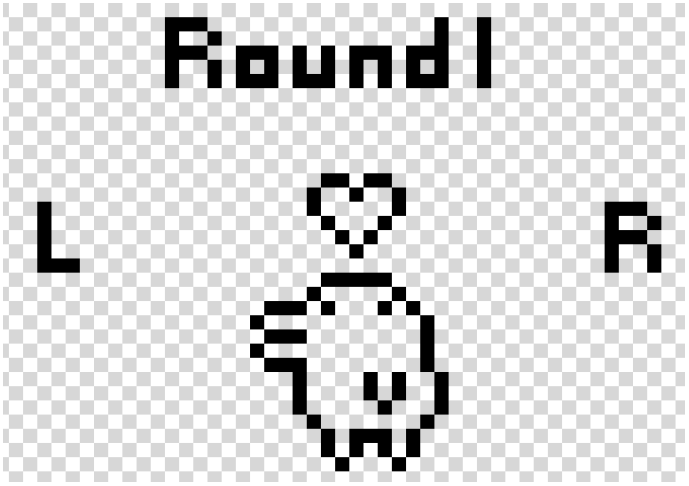


Selecting the third option will open up a stats screen for the pet that displays three lines indicating their level of hunger, happiness, and level.



Selecting the fourth option takes you to a mini game. The music is enabled while in the game. The top of the screen says “Round” and shows tallies corresponding to the current game round. A game consists of three rounds. The pet begins facing forward and is jumping. The goal is to guess which direction the pet will turn. The

green button selects left, and the blue button selects right. Once you make your pick, the pet will turn in the correct direction. A heart appears if you were right, and an ‘X’ appears if you were wrong. After three guesses, the game returns to the home screen, and the pet’s happiness goes up depending on how many guesses you got right.

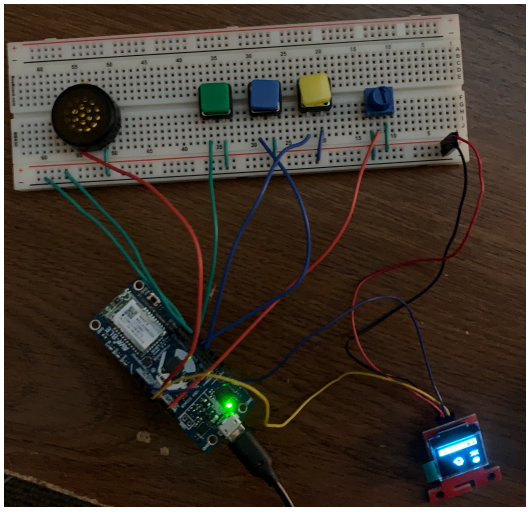


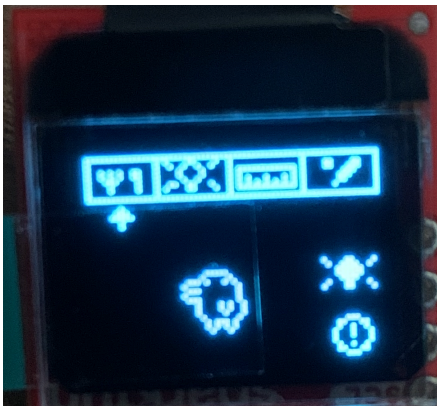
When the pet’s hunger or happiness drops below half and the alert circle shows up, a notification will be sent to the connected bluetooth device. A connected bluetooth device can also simulate “petting” your pet by updating the characteristic value of the system to a value ending in 0xAB. This will cause a heart to pop up in the bottom left corner of the screen for a few seconds.

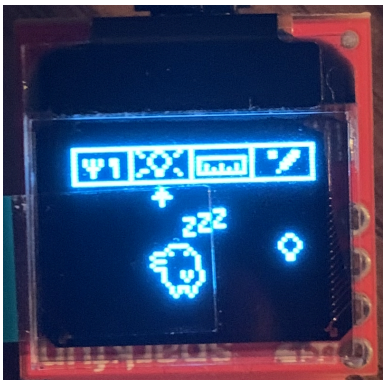
The contrast of the display screen can be adjusted using the potentiometer. The potentiometer is disabled unless the yellow button is being pressed. Hold down the yellow button and turn the knob on the potentiometer to increase or decrease the contrast, making the screen brighter or dimmer.

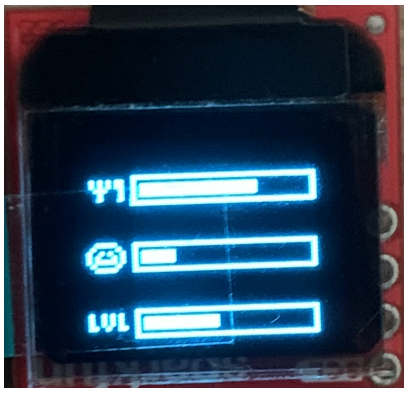
If the pet’s hunger or happiness reaches zero, the pet will die. A “PET DIED” screen will be displayed, and all user input will be disabled.

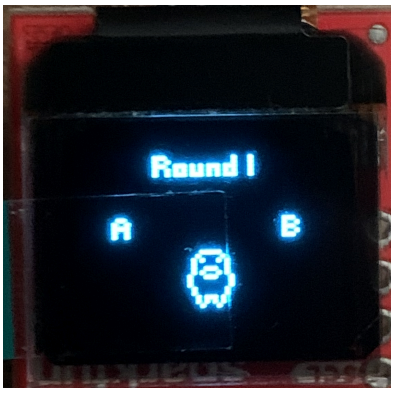
Here is a collection of pictures of the actual device.

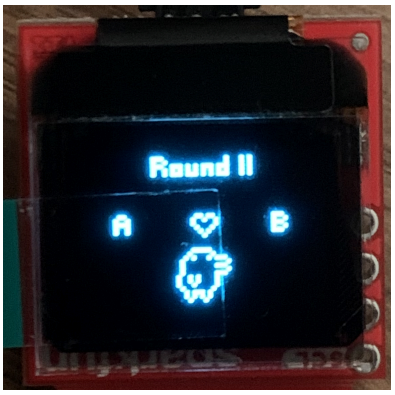


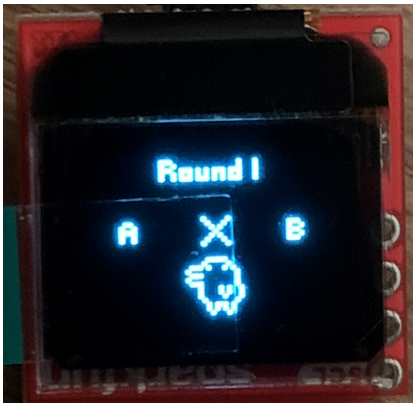












Why You Should Care: This project required combining many different embedded systems concepts. I had to learn how to communicate with the display screen entirely on my own and write my own methods for communicating with it instead of using already written libraries. It also required a deeper understanding of communication with the bluetooth peripheral in order to have both sending and receiving messages, as well as have it interrupt triggered instead of polling. Not only did the project require lots of embedded knowledge, it required lots of creativity and designing. Although the project is based on an existing product, I still had to come up with almost all of the designs and design the entire system from scratch. The final result was a unique and creative idea that allowed me to gain lots of practice developing my own embedded system.

Goals:

- Met: I was able to develop a working and interactable digital pet that supported several features. A big goal of mine was to incorporate bluetooth in some way, which I was able to do. Another large part of this project for me was interacting with a display screen of some sorts, which I was able to do, and come up with a framework for efficiently working with it. I was also able to play a song, which was something I wanted to do.
- Didn't Meet: I initially wanted to add a hatching and evolving animation into the system but did not get to. I found that once I learned how to communicate with the screen, adding new graphics became a repetitive task where I wasn't learning new embedded concepts. Instead of taking time to add these, I shifted my focus towards bluetooth so I could continue learning new concepts and add more variety to the functionality of the system.

Reflection

If I were to change the design of my system, I would include another unique piece of hardware. Figuring out how to interact with the display was something I learned a lot from, and I would have liked to learn about at least one more piece of hardware, such as a joystick for controls instead of just buttons. If I had more time, a feature I would have liked to add was non-volatile memory storage so that the device could lose power and maintain its current state. I believe this would be a useful feature and a good learning experience since we did not cover it in class, but unfortunately I did not have enough time.