ELEC2645 Embedded Systems Project

**Link to Project Files:**

**Project Brief**

This project aims to create Tamagotchi-style (egg pet) offering a modern twist to its nostalgic predecessor. The first stage of development involves brainstorming what exactly needs to and can be created. Given the task of creating an embedded systems-based game, with a minimum requirement. use of the ST Nucleo-L476RG development board, joystick, and LCD, I began exploring basic concepts that would provide a challenge while also falling in line with my expected skill and proficiency. After brainstorming and researching I fell into the niche of 3 options. The original bounce of Nokia, a Bullet Shooter game, and the option I decided on after cross examining my skills with my end goal. Tamagotchi was a popular hand-held game where one feeds and takes care of a virtual pet to keep it alive as long as possible.

*Features and Functions:*

1. Help Screen: An introductory screen providing guidance on how to interact with the virtual pet. Here one would find tooltips on what the buttons did and how to progress.
2. Customized Eggs: Users can select from a range of customized eggs to hatch their own unique virtual pet allowing for individuality between owners.
3. LED Indicators: LEDs will represent the virtual pet's hunger level, lighting up progressively as it gets hungrier.
4. Feeding Mini-Game: A mini-game where food would fall from the sky to feed the virtual pet, which from now on will be referred to as WEIR (reference to Egg Theory).

*Motivation:*

The inspiration for this project stems from an early childhood toy that I never really had but gave me my first introduction to the virtual landscape. By reimagining this concept on the Nucleo board, I aim to revamp this forlorn memory with the sparks of knowledge once privy to me. Hopefully this project showcases the leaps made since last I was face to face with the handheld mystery.

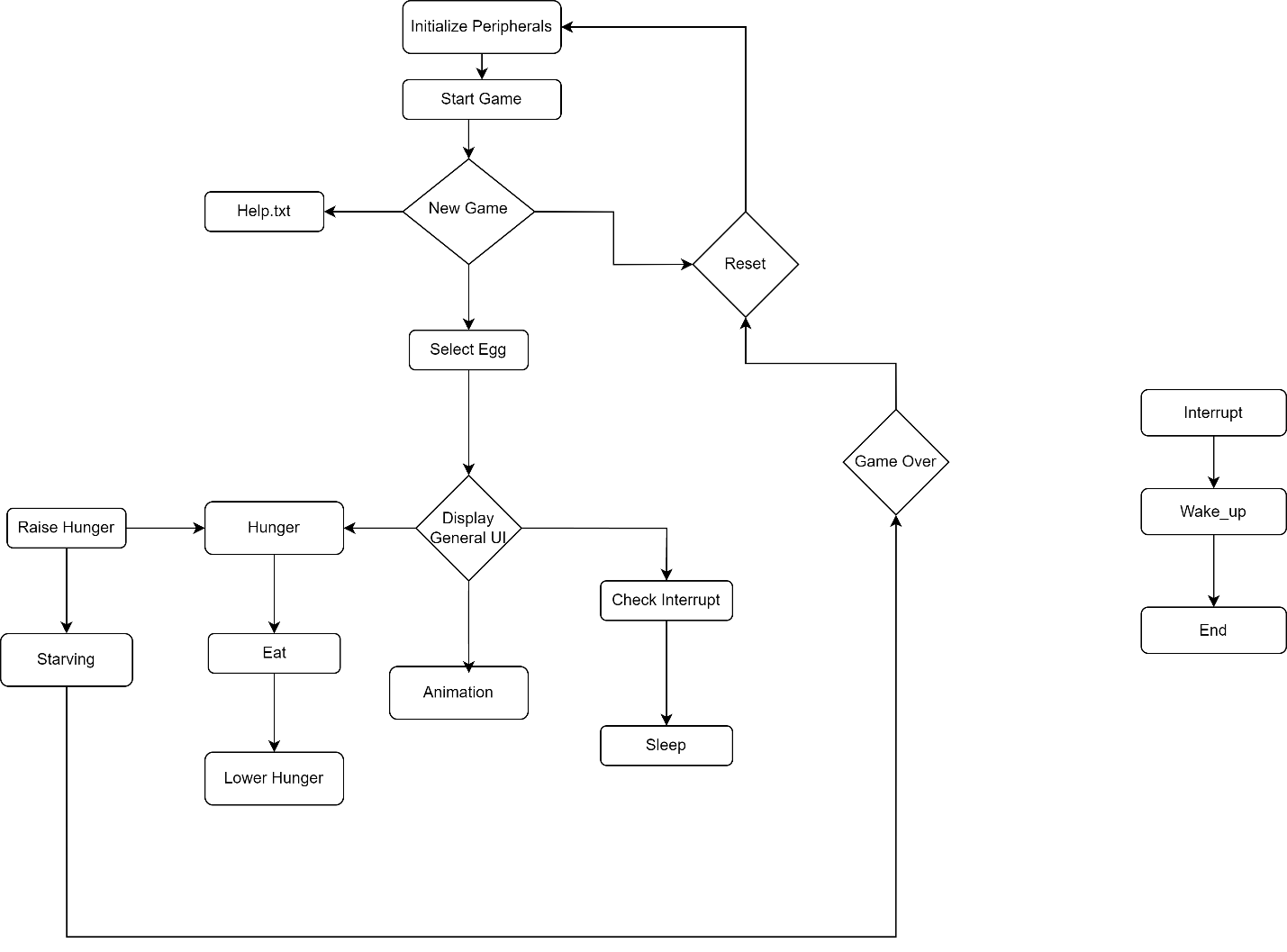
**Software Design**

Figure 1 Program Structure

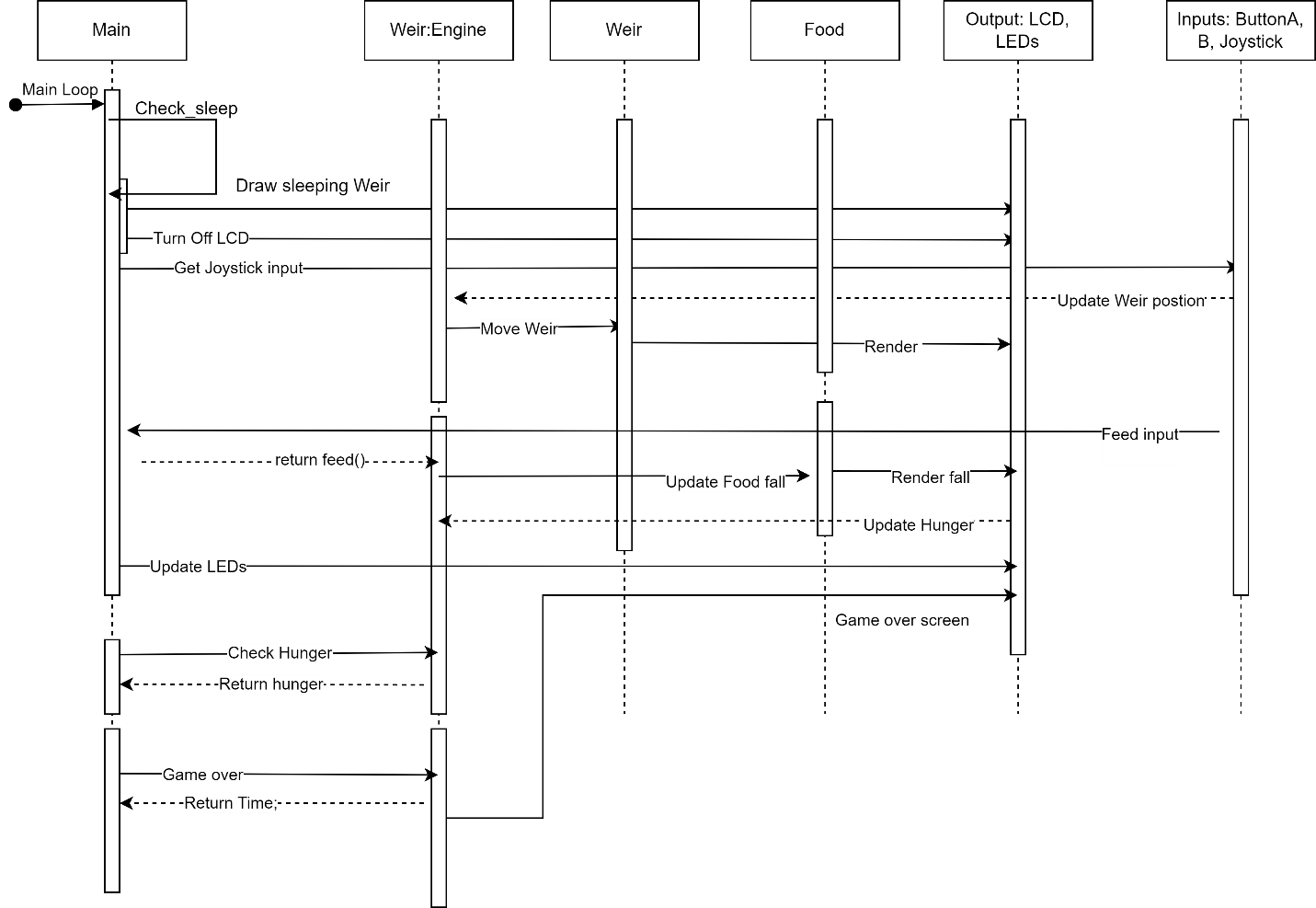


Figure 2 Sequence Diagram for Main loop

*Overview*

1. Initialize Peripherals and Objects:
   * The game initializes necessary components to be used in the code, including object components like button configurations and LEDs.
2. Start Game:
   * Starts the Main game function
3. Help Text:
   * Displays text providing information on how to play the game
4. Egg Selection:
   * Select an egg to decide what egg to go through with. A selection of eggs are presented for the user
5. Feeding Minigame:
   * Press a button to drop food for your pet
6. Sleep:
   * Turns the display off by checking for a flag that is triggered on button press
7. Game Over Screen
   * End of game display screen

*Early Ideas for Sprites*

A pixelated image of a grave

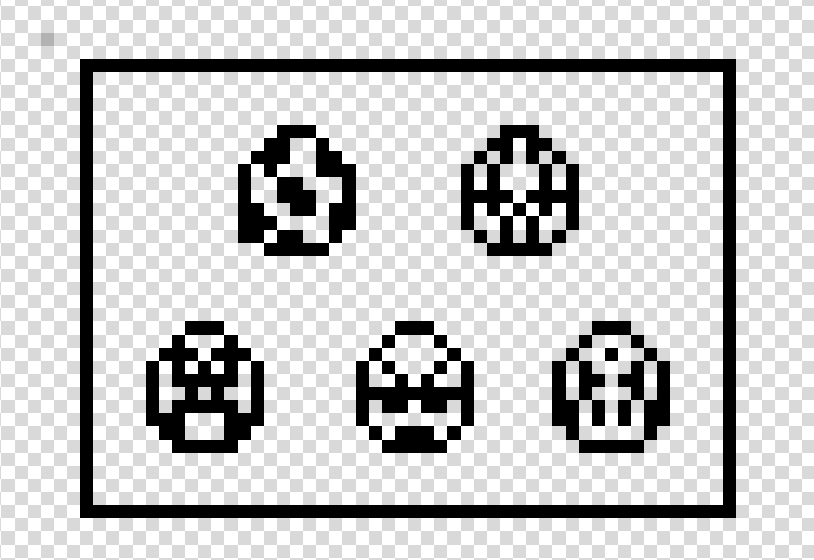
Description automatically generated

Figure 3 Early Game Over screen

Figure 4:EGG Designs for different customization

A black and white pixelated object

Description automatically generatedA black square with legs and a face

Description automatically generated

Figure 5 Original Food Design

Figure 6 Early Pet design

Implementation Checklist

* ~~Initialize Objects~~
* ~~Game-start~~
* ~~Controls scren~~
* ~~Help.txt~~
* ~~Egg-select~~
* ~~Egg Hatch Animation~~
* walking
* Game Music
* Life LED Hunger meter
* ~~Minigame/Feeding~~
* ~~Sleep!~~
* ~~Game over~~

**Week 22 Diary**

**Part 1 - Wake Up**

Completed the initialization of the components Used on the board. This includes initializing the button configurations to Pull None, creating the lcd object, and setting parameters for the display and joystick.

A computer screen shot of a program code

Description automatically generated

Figure 7 Initial Function

A circuit board with wires and a screen

Description automatically generatedWorked on the welcome splash-screen that displays the different possible options of preceding into the game or reading the help text.

Figure 8 Splash Display

A computer screen shot of code

Description automatically generated

Figure 9 Splash Screen Function

**Part 2 – Selecting a Choice**

Wrote up text to be printed as a when help function is run and provides a basic rundown on the games controls and modified the welcome function to loop and check when the help function ends to restore the splash screen.

A computer screen shot of code

Description automatically generated

Figure 10 Welcome Function

Egg Select box function. Developed a function that returns a chosen egg from a scalable array of eggs. The implementation of this took some time but I was able to accomplish the task exactly as I had envisioned. I created a map to store the egg sprite print locations relative to the box that would surround them. I then iterated over this map’s integer values and set the value of the x position of the printed box to this, allowing for a box that scales between eggs. The function then returns the string values corresponding to integer values gotten from the map.

A screen shot of a computer code

Description automatically generated

A computer screen shot of text

Description automatically generatedFigure 12 Map Holding integer and String Values

Figure 11 Returns Maps string corresponding to the Box value for X

A close up of a device

Description automatically generated

Figure 13 Egg Select Display

A close up of a device

Description automatically generatedDesigned the Sprite Screen for the Game Over screen.

**Part 3 – Living?**

Added the main pet with a blinking animation. As the engine code is continously run, I had to instantiate a different thread in order to simultaneously have animation and a game engine.

A screen shot of a computer code

Description automatically generated A computer screen shot of a program code

Description automatically generated

Figure 14 animation Function to design WEIR and game UI

Figure 15 Animate Function is run in a separate thread

A close up of a device

Description automatically generated

Figure 16 UI display and WEIR animated

In designing the Hatch functionality I decided to go with a pokemon like scatter screen. This was done by using the displays library containing the randomise buffer function, allowing me to create a unique changing display for a hatch animation.

A computer screen shot of code

Description automatically generated

Figure 17 Hatch animation code

**Part 4 – RE: Class**

Finalised the sprite design to display the controls of the game. The game is bi-buttoned with a singular joystick I represented using a D-pad on the display.

I kept on running into issues regarding the animation of multiple objects at once, causing it to not be animated correctly or to be animated with a trail. I decided that the cause had to be from my implementation of the animation logic but I could not pinpoint where exactly the problem stemmed from.

In the end I had to rebuild the program again.

I had originally written everything from scratch, slightly misunderstanding of how unfamiliar I was with the libraries provided. I decided to rewrite my previous functionality with the pong engine code working as my base. After re-writing all functions and creating classes, it now was animating correctly and seemed to no longer leave after images.

I think the cause of my initial error was the timing in which the draw and refresh functions were called in the thread. By having the different draw functions happen simultaneously in a class of its own, simulated a stable animation form. I was reluctant to use classes at first because I am greatly underskilled with Object based coding, but to achieve something worthwhile there must be some effort.

A screen shot of a computer program

Description automatically generated

Figure 18 Updated Animation function in engine class

A screen shot of a computer program

Description automatically generated

Figure 19 New Engine class

A screen shot of a computer code

Description automatically generated

Figure 20 Draw function for Weir

A computer code with white text

Description automatically generated

Figure 21 Draw function for Food

**Part 5 – Feeding**

To get the Feeding function to work and draw food falling and dissapearing on contact with weir, I decided to only initialise food on button press, and modified the parameters for food initialization so the x value would be random between 0 and 84. This causeed food to sometimes appear off screen, so I added logic to ensure food always spawned on screen

A computer screen shot of a program code

Description automatically generated

Figure 22 Food initial

A screen shot of a computer code

Description automatically generated

Figure 23 Straight fall

**Part 6 – Timed Rest and Song**

When researching how I could save energy usage while running the application, I decided my program would implement a pseudo sleep function. I wanted the Game to still run in the background while asleep to create a sense of responsibility and time management neccesary to own Weir. I implemented in the main loop a checker for a service routine value that would change on interrupt from the B button being pressed, and switch off the display(by setting the PWM value of the backlight to zero, and removing the contrast)

**A screenshot of a computer program

Description automatically generated**

Figure 24 Sleep Flag

A computer screen shot of a code

Description automatically generated

Figure 25 Sleep Function

I then worked on implementing the LED logic that act as a warning to the player when to feed and when not to. At first I wanted to have the red LED blink to convey a sence of urgency, but the flahing thread caused code to slow down as there would be traffic in the loop controlling the flashing.

Figure 26 LED indicator logic

I Implemented a Buzzer that plays sound, and I updated the main function with a ticker that depletes hunger every 8 seconds. I also implemented a time variable that displays time you kept Weir alive, like a score.

A computer screen shot of a program code

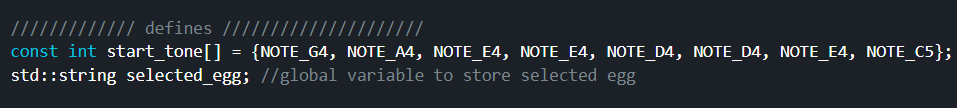
Description automatically generated 

Figure 27 Buzzer Code

 A computer screen shot of a computer code

Description automatically generated

Figure 28 Blurry From Flashing, updated game over screen

Figure 29 Ticker Code to lose hunger Every 8 seconds