CSE-379

Lab 7

Space Invaders

Christopher Long & Matthew Sixt
June 6, 2018

Contents

1 Overview **3**
1.1 Usage . . . 3
1.2 Division of Labor . . . 3

2 Assembly Subroutines **4**
2.1 lab7 . . . 4
2.1.1 Lab7 Flowsheet . . . 5
2.2 Interrupt Initiation . . . 6
2.2.1 Interrupt Init Flowchart . . . 7
2.3 FIQ_Handler . . . 8
2.3.1 UART . . . 8
2.3.2 timer0 . . . 8
2.3.3 timer1 . . . 8
2.3.4 EINT1 . . . 8
2.3.5 fiq_handler flowchart . . . 9
2.4 Read_character . . . 10
2.4.1 read_char Flowchart . . . 11
2.5 Read_string . . . 12
2.5.1 read_string Flowchart . . . 13
2.6 Output_character . . . 14
2.6.1 output_character flowchart . . . 15
2.7 Output_String . . . 16
2.7.1 Output String Flowchart . . . 17
2.8 UART_initialization . . . 18
2.8.1 UART initialization flowchart . . . 19
2.9 7 Segment Display . . . 20
2.9.1 Seven Segment Display Flowchart . . . 21
2.10 Illuminate RGB LED . . . 22
2.10.1 Illuminate RGB LED flowchart . . . 23
2.11 Illuminate LEDs . . . 24
2.11.1 Illuminate LEDs Flowsheet . . . 25
2.12 Update Level . . . 26
2.12.1 Update Level Flowsheet . . . 26
2.13 Update Score . . . 27
2.13.1 Update Score Flowsheet . . . 27
2.14 Start Timers . . . 28
2.14.1 Start Timer Flowsheet . . . 28
2.15 Stop Timers . . . 29
2.15.1 Stop Timer Flowsheet . . . 29
2.16 Handle Char . . . 30
2.16.1 Handle Char Flowsheet . . . 30

3 C Functions **31**
3.1 main . . . 31
3.1.1 main flowsheet . . . 32
3.2 Initiate Board . . . 33
3.2.1 Initiate Board Flowsheet . . . 34
3.3 Start Game . . . 35
3.3.1 Start Game Flowsheet . . . 36
3.4 New Game . . . 37
3.4.1 New Game Flowsheet . . . 38
3.5 newLevel . . . 39

3.5.1 New Level Flowsheet . . . 40
3.6 decTime . . . 41
3.6.1 decTime Flowsheet . . . 41
3.7 Death . . . 42
3.7.1 Death Flowsheet . . . 43
3.8 Update Board . . . 44
3.8.1 Update Board Flowsheet . . . 45
3.9 Print Board . . . 46
3.9.1 Print Board Flowsheet . . . 47
3.10 Move Player . . . 48
3.10.1 Move Player Flowsheet . . . 49
3.11 Move Mother . . . 50
3.11.1 Move Mother Flowsheet . . . 51
3.12 Generate Mother . . . 52
3.12.1 Generate Mother Flowsheet . . . 53
3.13 Generate Player Shot . . . 54
3.13.1 Generate Player Shot Flowsheet . . . 55
3.14 Generate Enemy Shot . . . 56
3.14.1 Generate Enemy Shot Flowsheet . . . 57
3.15 Move Shot . . . 58
3.15.1 Move Shot Flowsheet . . . 59
3.16 End Game . . . 60
3.16.1 End Game Flowsheet . . . 61

1 Overview

This program is written for Lab 7. This program is meant to showcase the learning we have done over the course of this semester by recreating the popular Space Invaders arcade game, with some modifications. The program contains all the components necessary to control the outputs, inputs, and interrupts generated and used by the program.

1.1 Usage

To use the program, first load it onto the NXP LCP2138 board. The program will ask if you would like to see the instructions, and will display them if requested. The program then starts the game at level 1 and a time of 2 minutes. The game continues for two minutes and then a screen is displayed showing details of the previous run.

1.2 Division of Labor

Division Of Labor		
Game Board	cmlong	
Movement	mwsixt	
Shooting	cmlong	
Timing	cmlong/mwsixt	
Score	cmlong	
Moral Support	mwsixt	
Motherships	mwsixt	

2 Assembly Subroutines

2.1 lab7

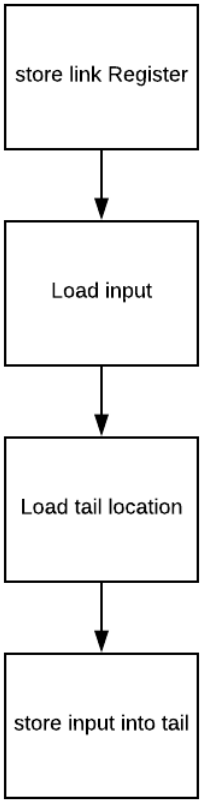
This subroutine sets up the memory locations for displaying prompts and reading inputs.

Usage
This subroutine is only for initializing memory locations for UART integration, and really does not have a use case further than that.

Exceptions

1. If there is an error in communication between the board and PuTTy the program will not display prompts or inputs

2.1.1 Lab7 Flowsheet



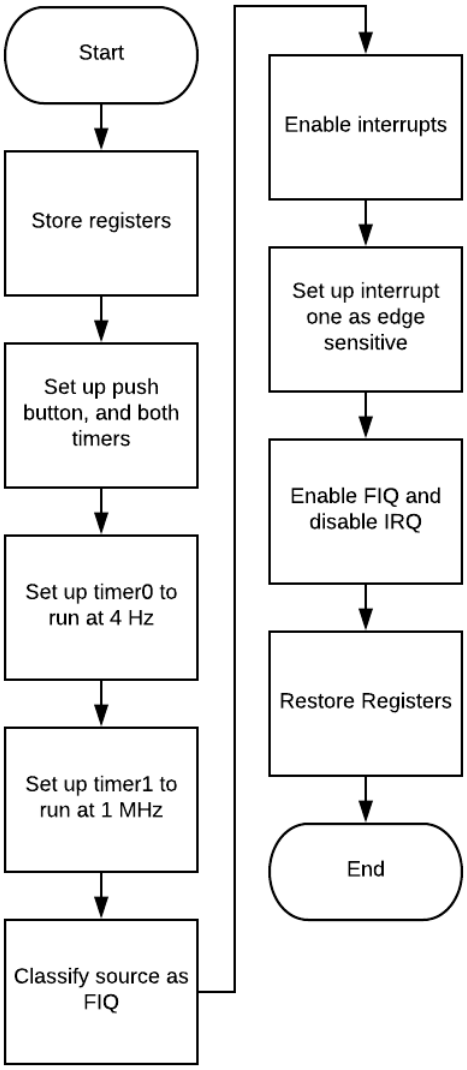
2.2 Interrupt Initiation

This subroutine is given to us in the skeleton code for lab 5 and reused in lab 7. It initializes the interrupt that we use to handle input, as well as the timers that allow us to drive the program and the external interrupt that lets us pause the game.

Usage

This subroutine is called in main, the only values that could need changing are the values for the timer counts, which dictate how often each timer triggers, everything else is fairly unnecessary for any user to worry about.

2.2.1 Interrupt Init Flowchart



2.3 FIQ_Handler

This subroutine serves to handle any interrupts generated by the UART, timers, and pin 0.14

Usage
The usage of this subroutine changes depending on which interrupt is given.

2.3.1 UART

If the UART is triggered, the subroutine reads the character entered, and stores it to a buffer for use in the handle_char subroutine.

2.3.2 timer0

If timer0 is triggered, the subroutine checks to see if a counter is equal to a designated number to decide if enemies should move, and handles enemy movement if so. It then moves the mothership, if one exists on the board, by checking another counter similar to the enemy counter listed above. Then the subroutine tries to generate a mothership, and enemy shots, both of which are subject to randomness in separate C functions. Next the subroutine handles any characters entered into the UART interface, moves any player shots and finally prints the game board with any changes and clears the timer.

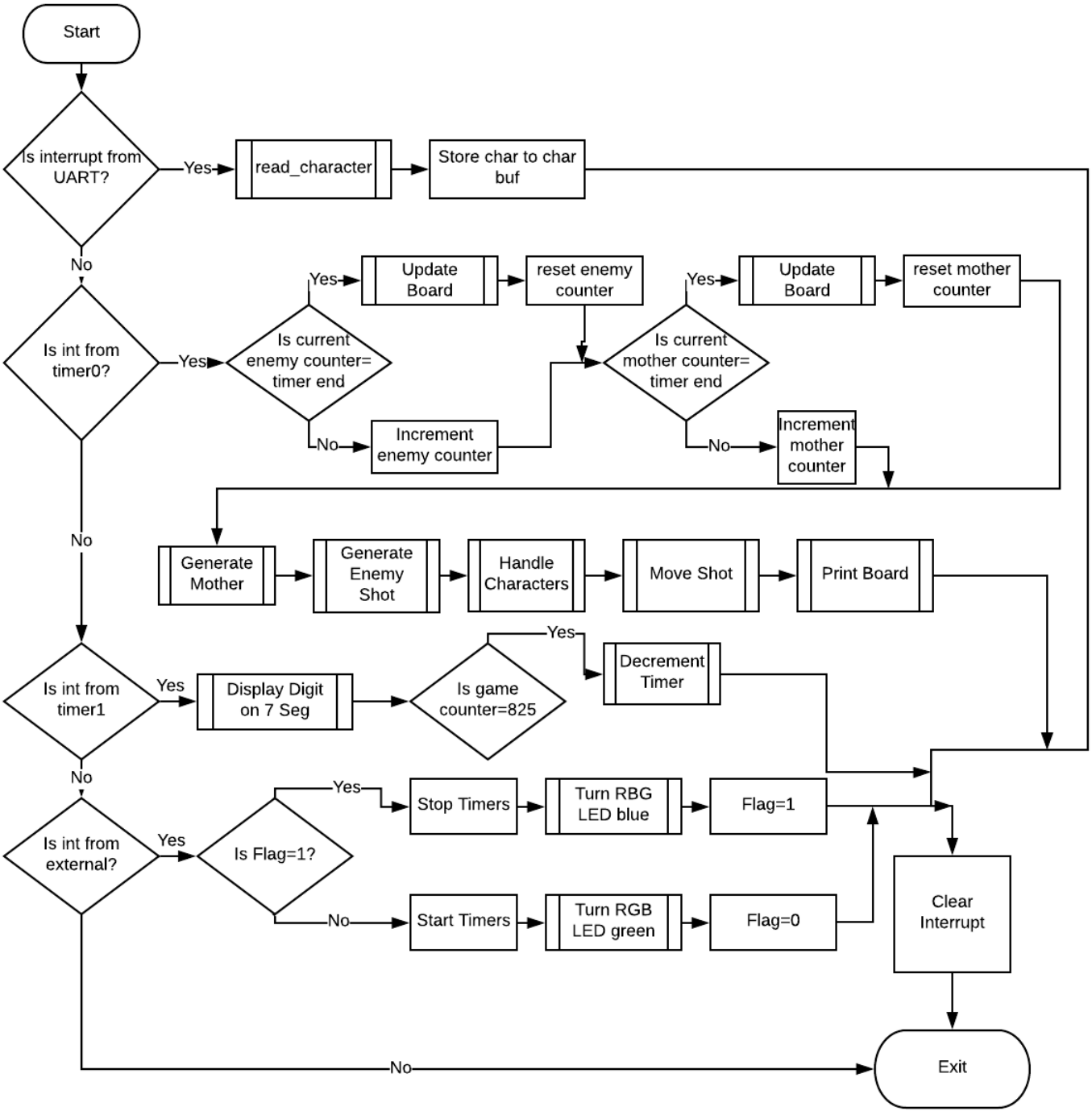
2.3.3 timer1

If the interrupt is from timer1, then the subroutine handles the strobing of the 7-segment display and the game timer. It does this by taking in the value of input, containing the players score and displaying one digit of it, each time changing based on separate calls to the interrupt. The subroutine also has a counter similar to the ones for the enemy and mothership movements, counting up to one second and then decrementing the game timer. Finally the subroutine clears the interrupt.

2.3.4 EINT1

If the interrupt is generated by the external interrupt, the program checks to see if the program is paused or not. If it is not paused then the subroutine turns the LED blue, stops the timers from running, and waits for the user to hit the button again. If the button is pressed again, the program turns the LED green and resumes the timers.

2.3.5 fiq_handler flowchart



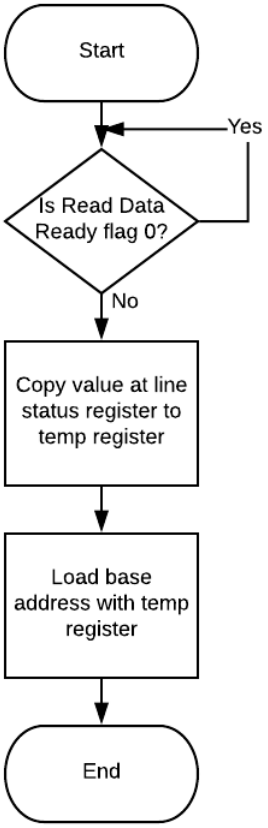
2.4 Read_character

This subroutine checks the Line status register and loops until the 1st bit of that register indicates that the processor is ready to read a character. The program then reads the last character that was sent through the UART. **Usage** This subroutine can be used on its own by calling it specifically, then entering characters over the serial interface, but it is more commonly used in conjunction with the read_string subroutine, which uses it to to store a character into memory.

Exceptions

1. characters that do not have an equivalent ascii value
2. Cases in which the routine is improperly used and will fail

2.4.1 read_char Flowchart



2.5 Read_string

This subroutine uses the read_character subroutine to read and store a string into memory, leaving a register with the head address to reference. It also uses the output_character subroutine to echo the characters entered to make the program easier to use. The subroutine only accepts certain characters, sending an error message if an incorrect character is entered.

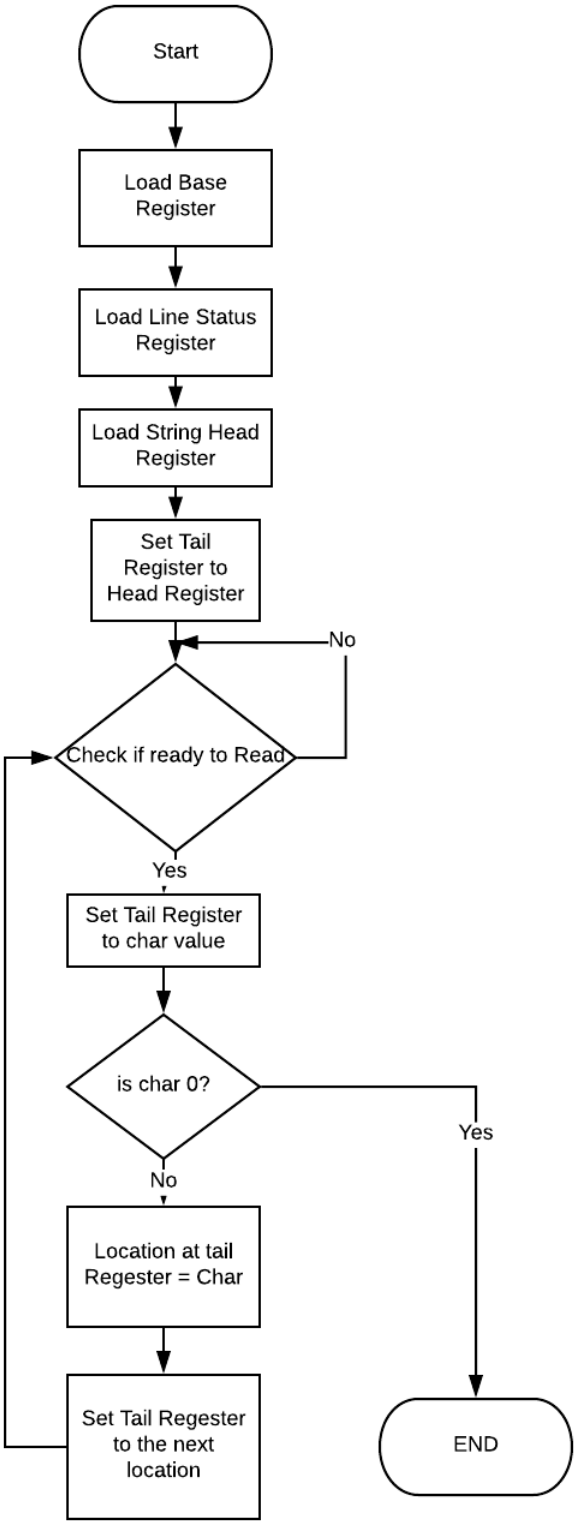
Usage

To use this subroutine, just call it and begin typing a string into the serial connection. You should see the characters types on the screen. Pressing enter ends the loop, and terminates the string with a null character

Exceptions

1. Doesn't work with backspace
2. Any character that does not fit within parameters
3. Cases in which the routine is improperly used and will fail

2.5.1 read_string Flowchart



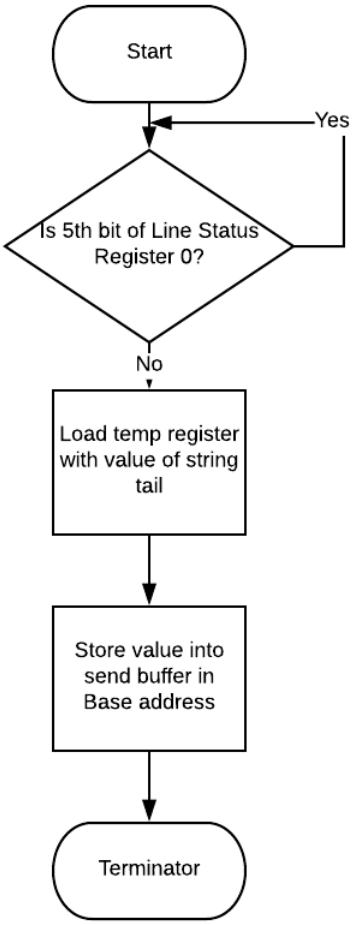
2.6 Output_character

This subroutine serves to output a character at a specific memory location to the serial bus.

Usage
This routine allows the user to see what they are typing into the computer when entering data into the bus, as well as allowing developers to prompt users using predefined strings. It is it used in every part of the program so that every input and output can be displayed to the user.

1. Any character that does not have an ascii value
2. Cases in which the routine is improperly used and will fail

2.6.1 output_character flowchart



2.7 Output_String

This subroutine takes in a value for the head of a string, and then outputs it character by character over the serial bus.

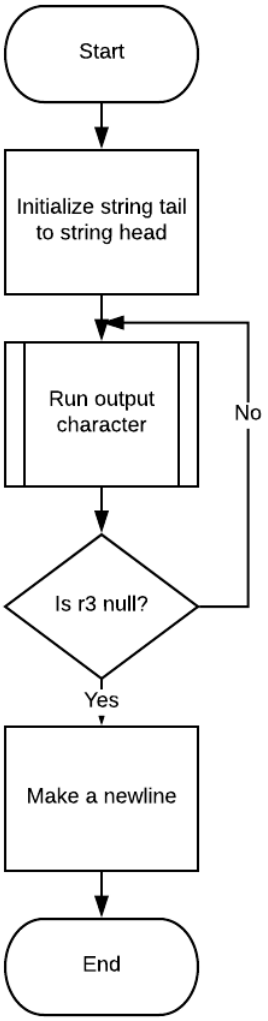
Usage

To use, set register r4 to a location in memory containing a string head, the subroutine will iterate through the string until it finds a null value, transmitting each value over the UART interface. This routine is used in read/writing the values that are put into our program in our different routines.

Exceptions

1. Cases in which the routine is improperly used and will fail

2.7.1 Output String Flowchart



2.8 UART_initialization

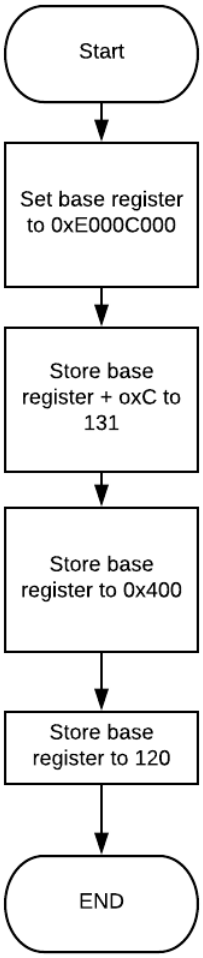
This subroutine is a constructor of sorts for use with the board. It initializes several key values that allow the board to perform in the way we want

Usage
This is used to initialize the LCP2138 board for use

Exceptions

1. Everything is buggy with Keil 4

2.8.1 UART initialization flowchart



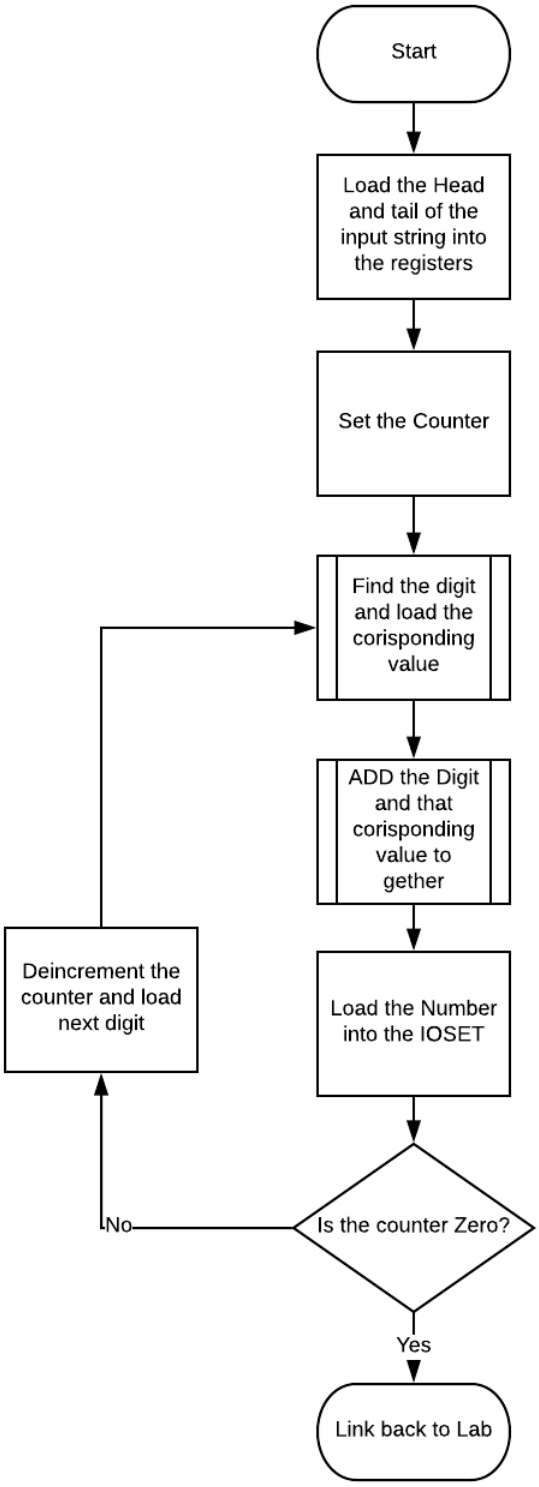
2.9 7 Segment Display

This subroutine communicates the input from UART to the LPC 2138 and allows the user to input and display the values(as long as their Hexadecimal) on the 2138 board. It does it by lighting the lights up individually at a pace of 60Hz. It happens so fast that the human eye can see it but slow enough to not have the light dimmed.

Usage

When first prompted you will be asked to put a in a string of Hexadecimal numbers and then hit enter. This sub routine is linked with lab 6. It sends the String to the routine then sends it to the board.

2.9.1 Seven Segment Display Flowchart



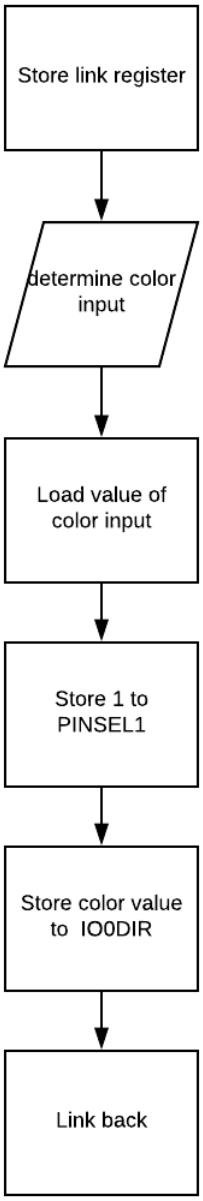
2.10 Illuminate RGB LED

This subroutine lights up the RGB LED on the board in the desired combination to produce the desired color.

Usage

This subroutine works by taking in a character representing the color desired by the user. It interprets this character and turns on the red, green, and/or blue LED to achieve the desired color.

2.10.1 Illuminate RGB LED flowchart



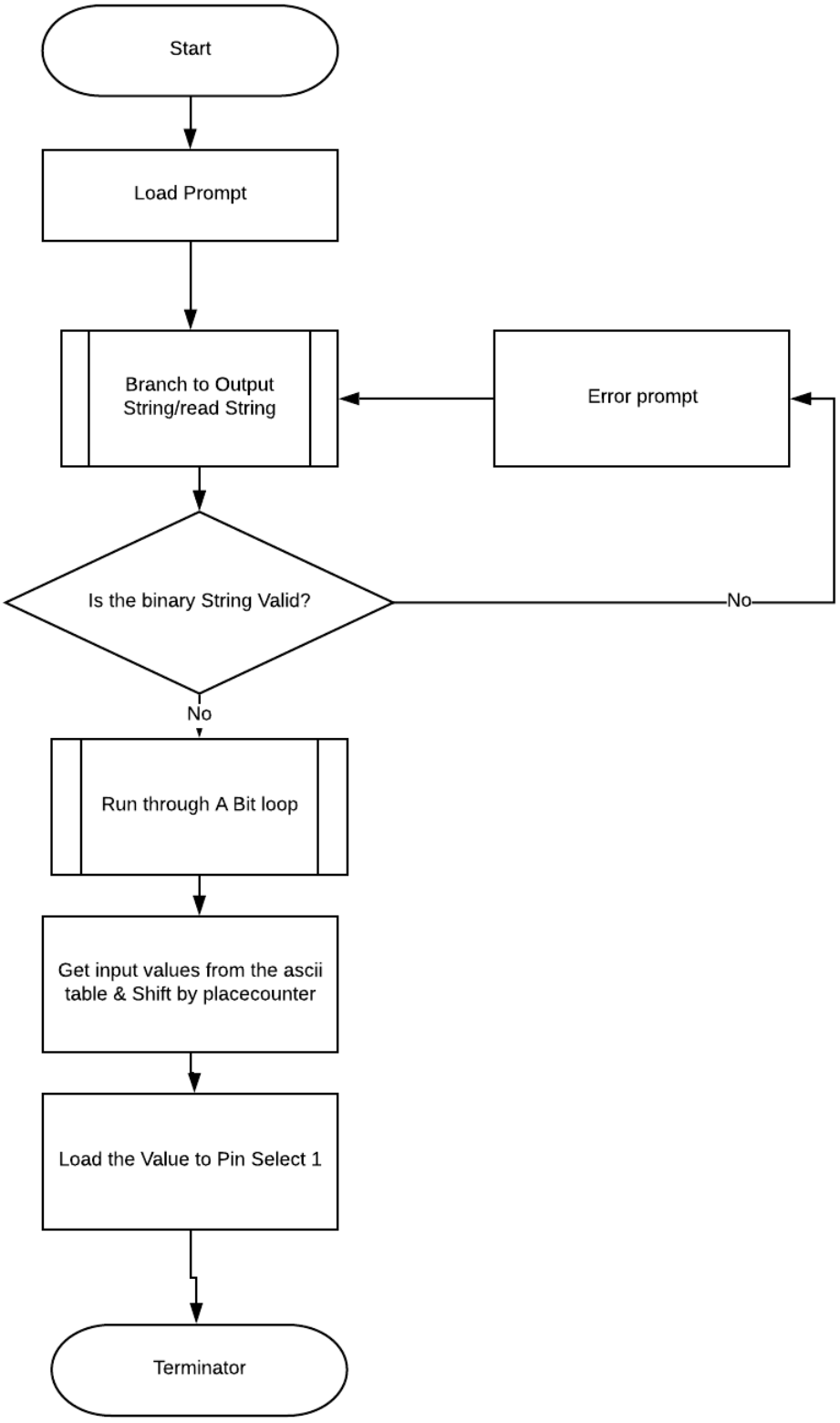
2.11 Illuminate LEDs

This subroutine takes in a string of 1 and 0 characters and lights up the four LEDS on the board according to that string.

Usage

The subroutine takes in a 4 character string of 1s and 0s, then turns on the LEDs representing that string.

2.11.1 Illuminate LEDs Flowsheet



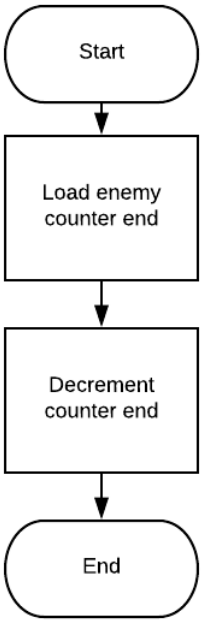
2.12 Update Level

This subroutine decreases the number needed by the enemy timer to move, making enemies move faster.

Usage

This subroutine is called when the user beats a level, and makes changing the difficulty of the game easier.

2.12.1 Update Level Flowsheet

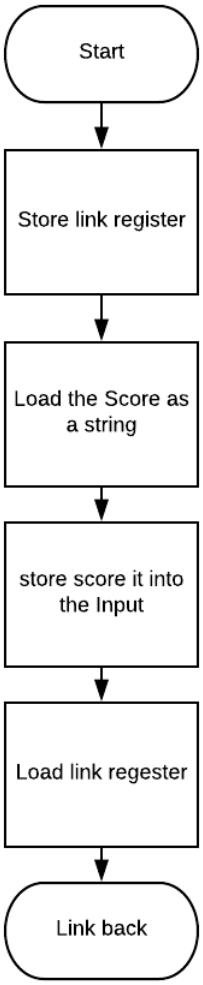


2.13 Update Score

This subroutine updates the memory location containing the string with the score in it

Usage
This subroutine is called whenever the clock cycles and makes sure that the 7 segment displays are showing the proper value.

2.13.1 Update Score Flowsheet

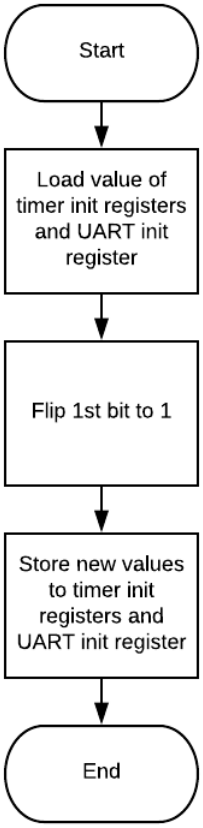


2.14 Start Timers

This subroutine starts each of the Timers and enables the UART interrupt.

Usage
This subroutine writes a one to the register containing the timer and UART interrupt enable bit.

2.14.1 Start Timer Flowsheet



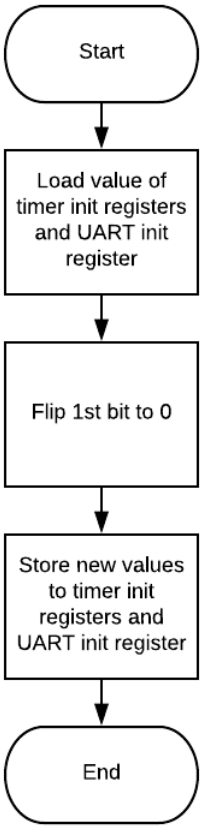
2.15 Stop Timers

This subroutine stops each of the Timers and disables the UART interrupt

Usage

This subroutine writes a zero to the register containing the timer and UART interrupt enable bit.

2.15.1 Stop Timer Flowsheet



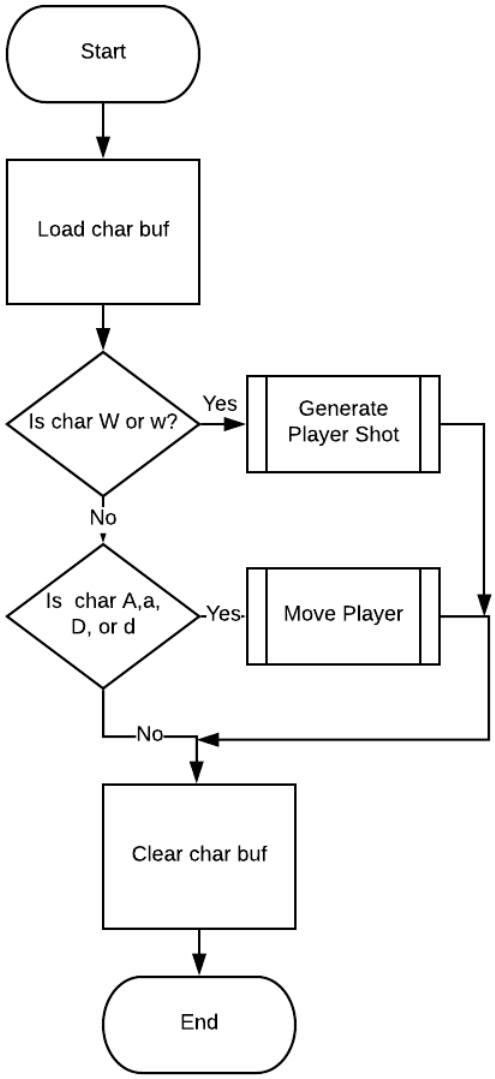
2.16 Handle Char

This subroutine reads the char buffer and interprets it to generate the proper response

Usage

This subroutine looks at the last input generated by the user and decides to shoot or move the player. The subroutine then clears the char buffer.

2.16.1 Handle Char Flowsheet



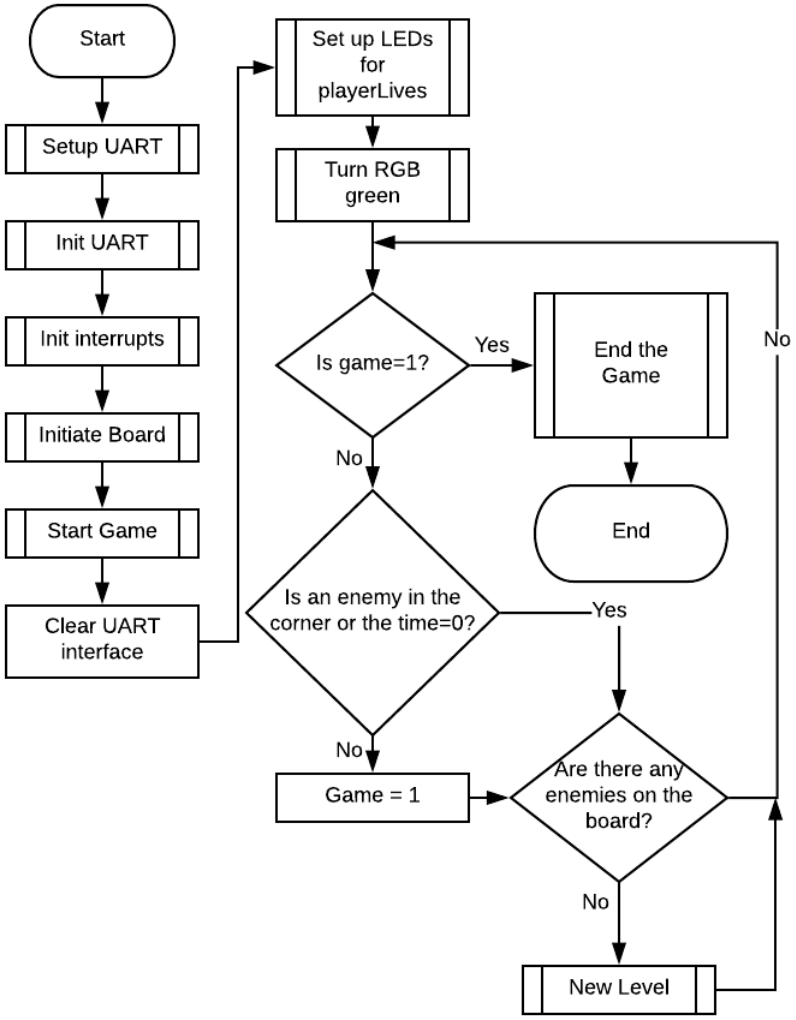
3 C Functions

3.1 main

This is the main function of the program, where the program begins. The function prepares many of the necessary initialization function, primes the UART interface by clearing the screen, turns the RGB LED to green, initializes the board, calls the game start function, starts the LEDs to the playerlives, and finally enters a loop that controls play.

Usage
This function is entered as soon as the program is begun. It controls all of the function listed above, then enters the main loop of the program. This loop checks to see the position of the enemies, and if they're too far down ends the game. If the user has defeated all of the enemies the game increments the level. After that loop is done, the function calls the end game function.

3.1.1 main flowsheet



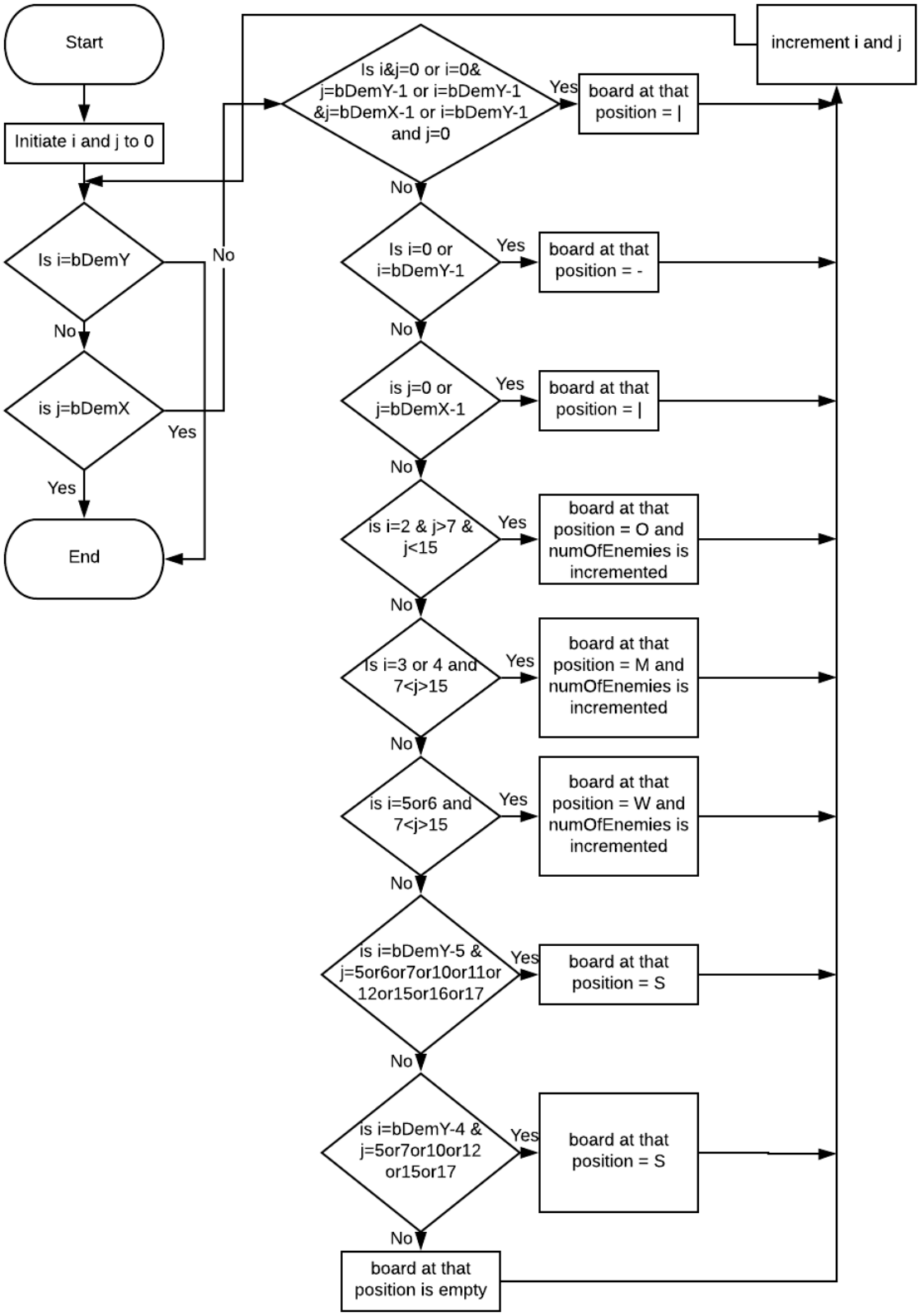
3.2 Initiate Board

This function sets up the game board array and fills it with all of the characters in the positions necessary at the beginning of a level

Usage

This function is called in the main function, and whenever a new level is called.

3.2.1 Initiate Board Flowsheet



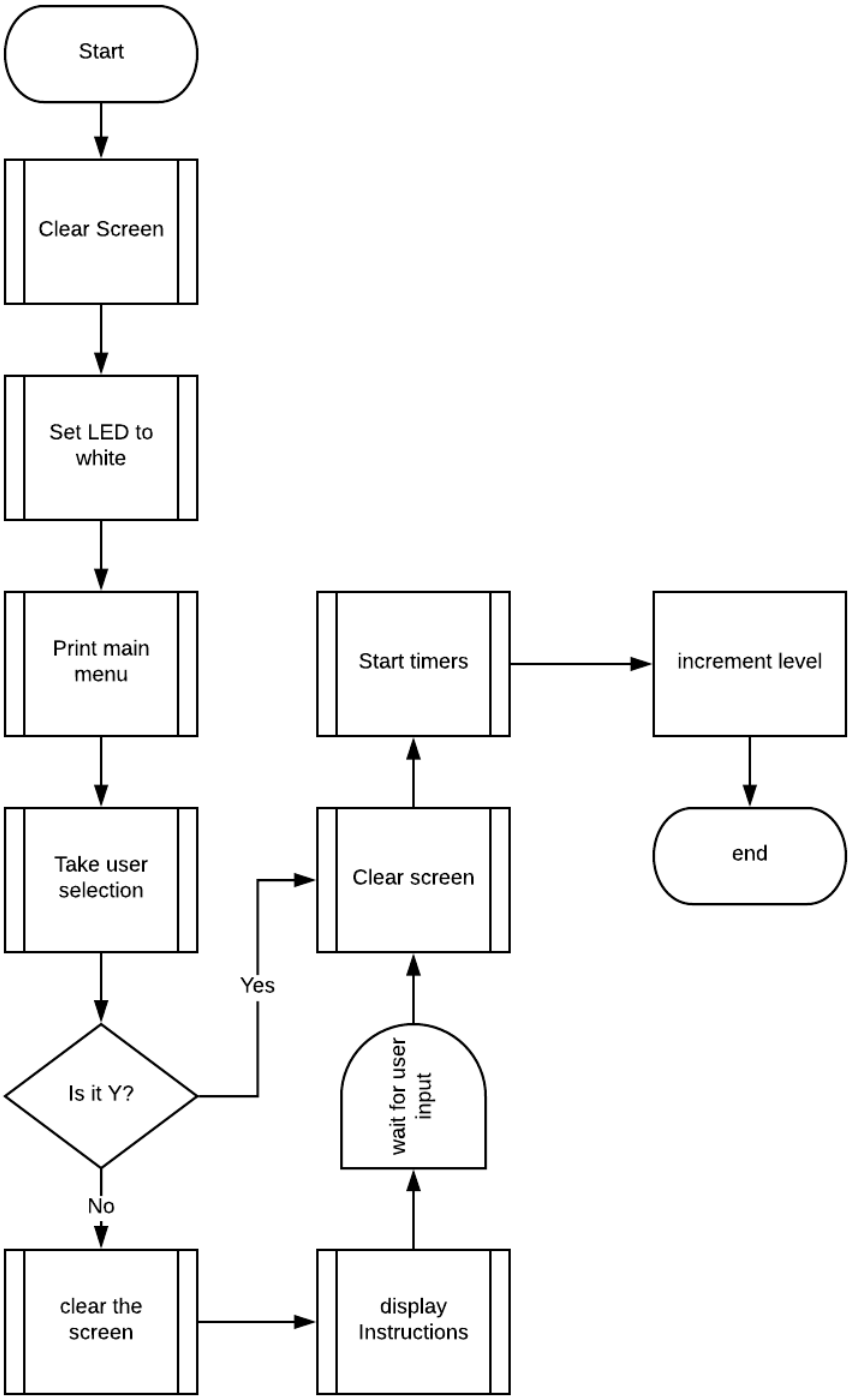
3.3 Start Game

This function puts the game into its introduction screen, where the user can see instructions for how the game is played, or the player can choose to skip this step.

Usage

This function serves to send the user into an introduction screen where the user can choose to view instructions on how to play the game. This function also turns the RGB LED white to show that its in intro mode, and starts the timers when the game is ready to begin

3.3.1 Start Game Flowsheet



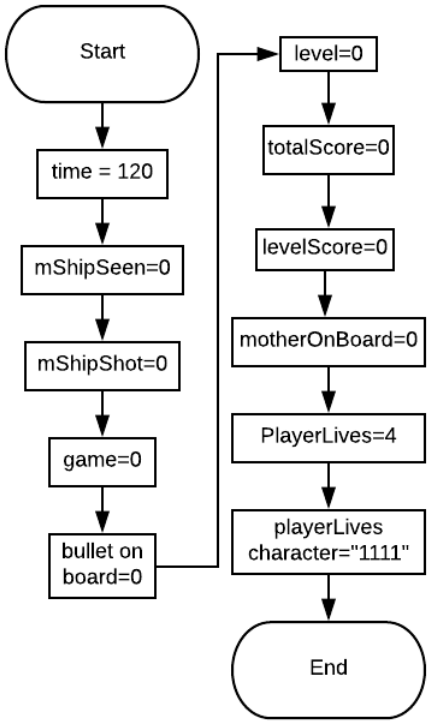
3.4 New Game

This function resets all of the values needed for the user to start a new game from scratch.

Usage

This function resets values such as the level, playerLives, number of enemies, and the score of the player. It is called when the user requests a new game.

3.4.1 New Game Flowsheet



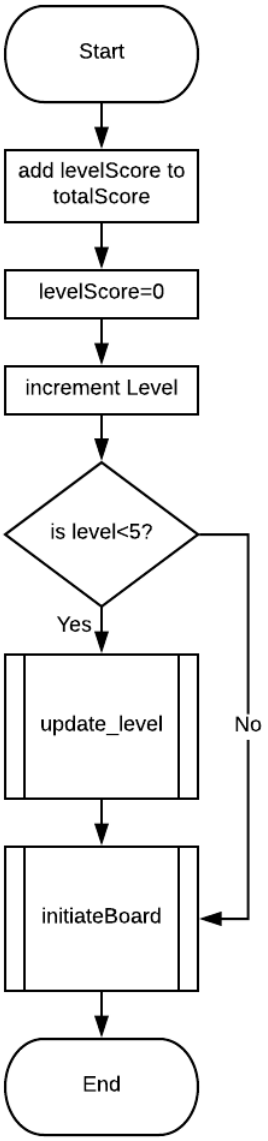
3.5 newLevel

This function sets up a new level for the user to play. It resets the level score and sets the total score to include the previous level's score. It reinitializes the board in the "home" position.

Usage

This function is called when the user shoots every enemy on the board, excluding the motherships.

3.5.1 New Level Flowsheet

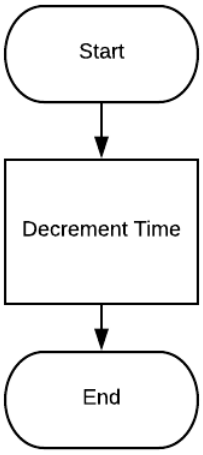


3.6 decTime

This function very simply serves to decrement the time count

Usage
This function is called by timer1 to keep a two minute counter.

3.6.1 decTime Flowsheet



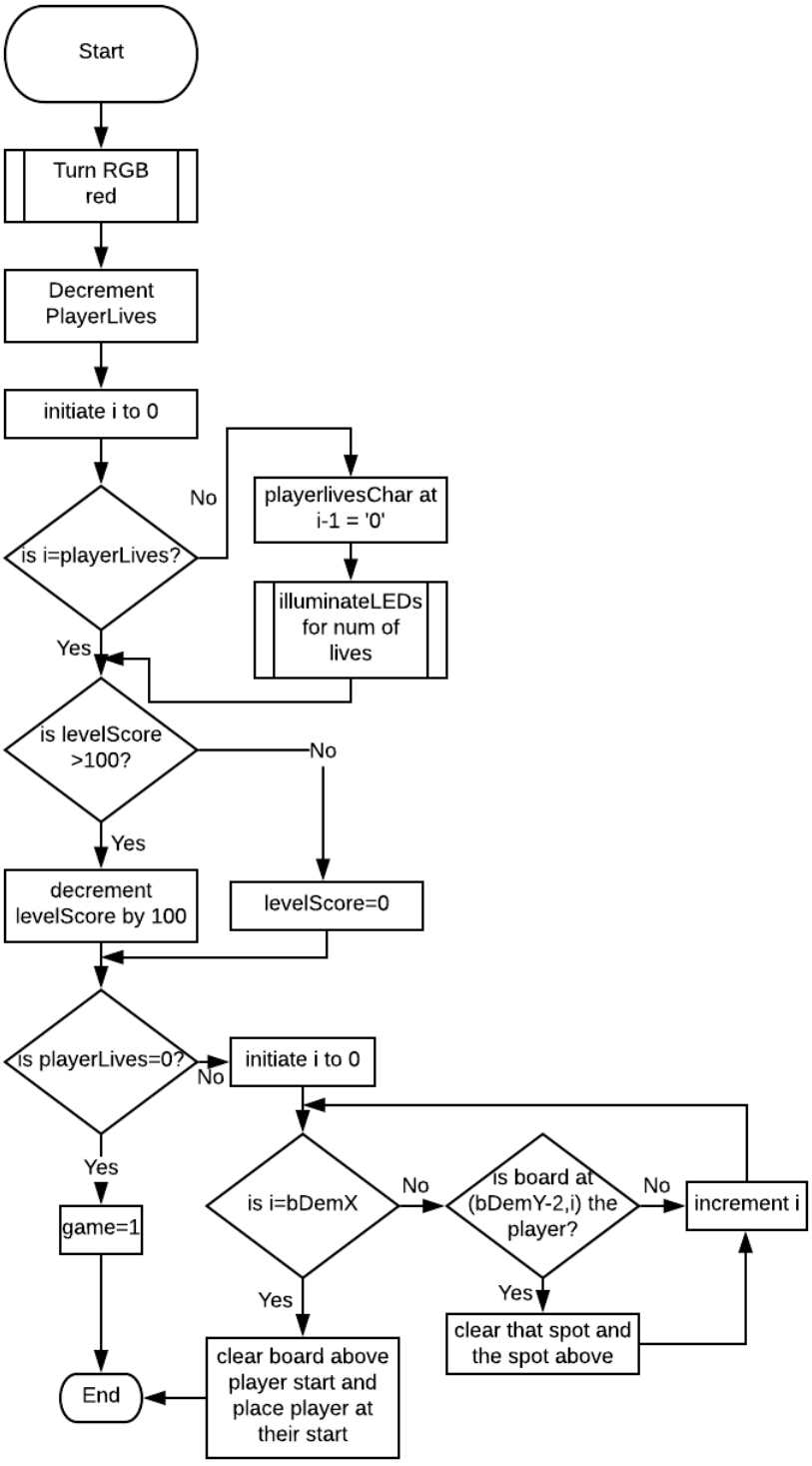
3.7 Death

This function handles what happens when a user is hit by an enemy shot. It flashes the RGB LED red, decreases the player's lives, lowers their score by 100, or to 0 if they don't have 100 points, and removes the enemy shot above the player and above the spot where the player regenerates.

Usage

This function is called whenever there is an enemy shot directly above the player.

3.7.1 Death Flowsheet



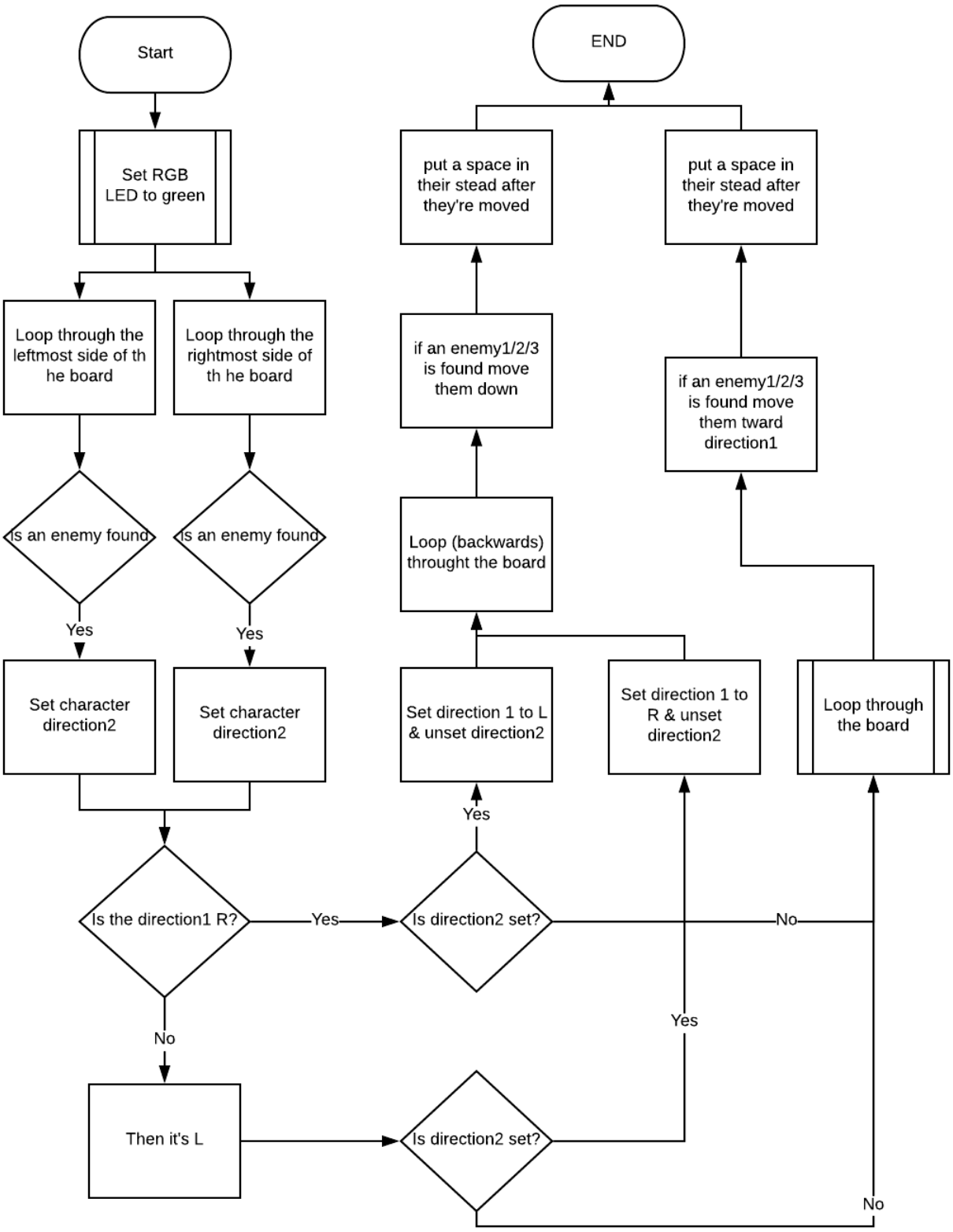
3.8 Update Board

This function handles all enemy movement, other than the motherships. The function checks to see if the enemies are approaching an edge, and if so moves them down and sets them moving to the other side. If not, the enemies continue in the direction they were going.

Usage

This function is called every time the enemy timer in the FIQ Handler satisfies its condition, this allows the program to control how fast the enemies move.

3.8.1 Update Board Flowsheet



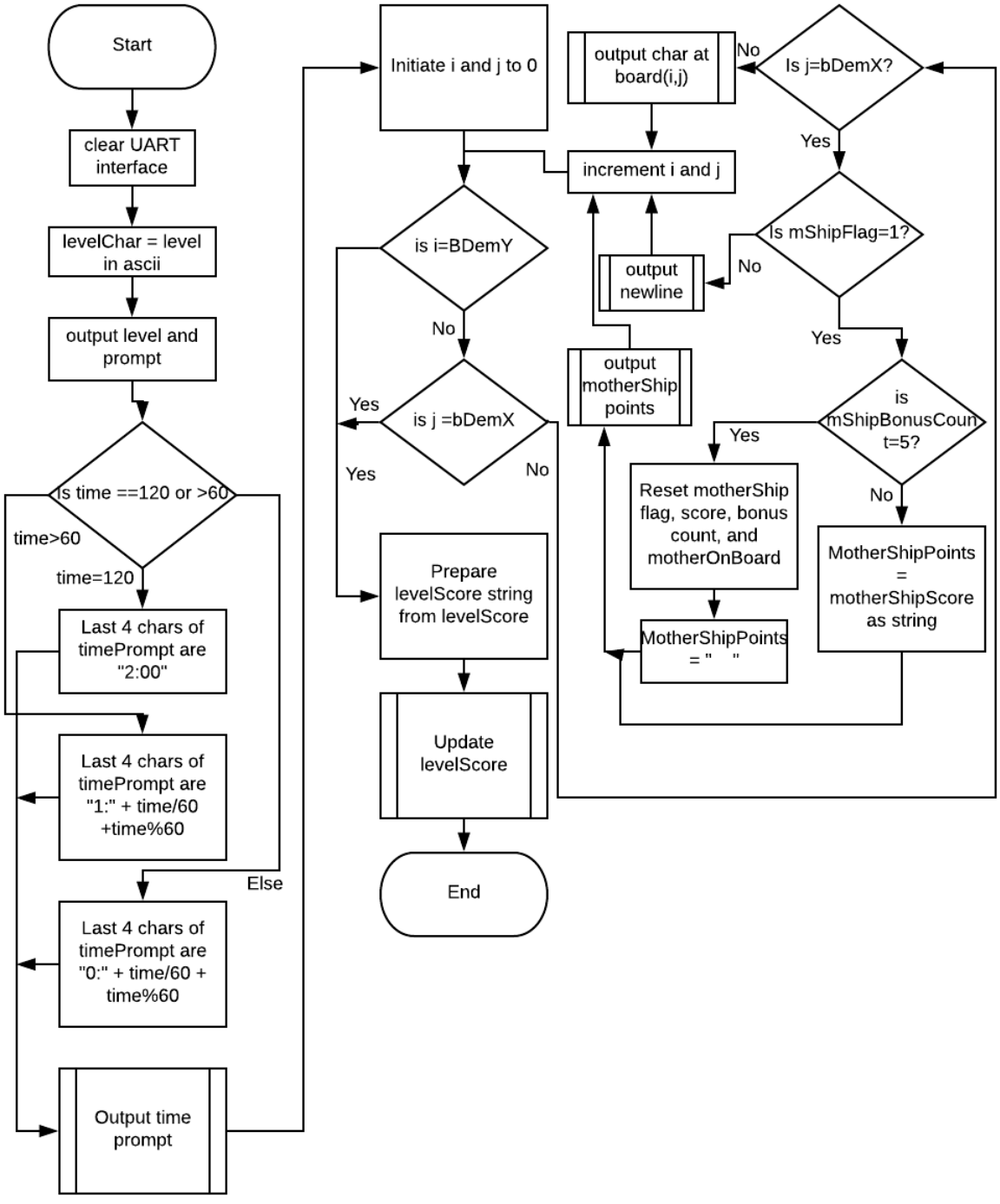
3.9 Print Board

This function handles the printing of the board on every refresh cycle. It clears the UART interface, prints the level and the state of the timer, and then loops through and prints the board at each position. It also handles the program freezing the board when a user hits a mother ship to let the user see the score they got. Finally the function updates the level score for the 7 segment display.

Usage

This function is called on every cycle of timer0, which means it updates as the refresh rate of the game overall.

3.9.1 Print Board Flowsheet



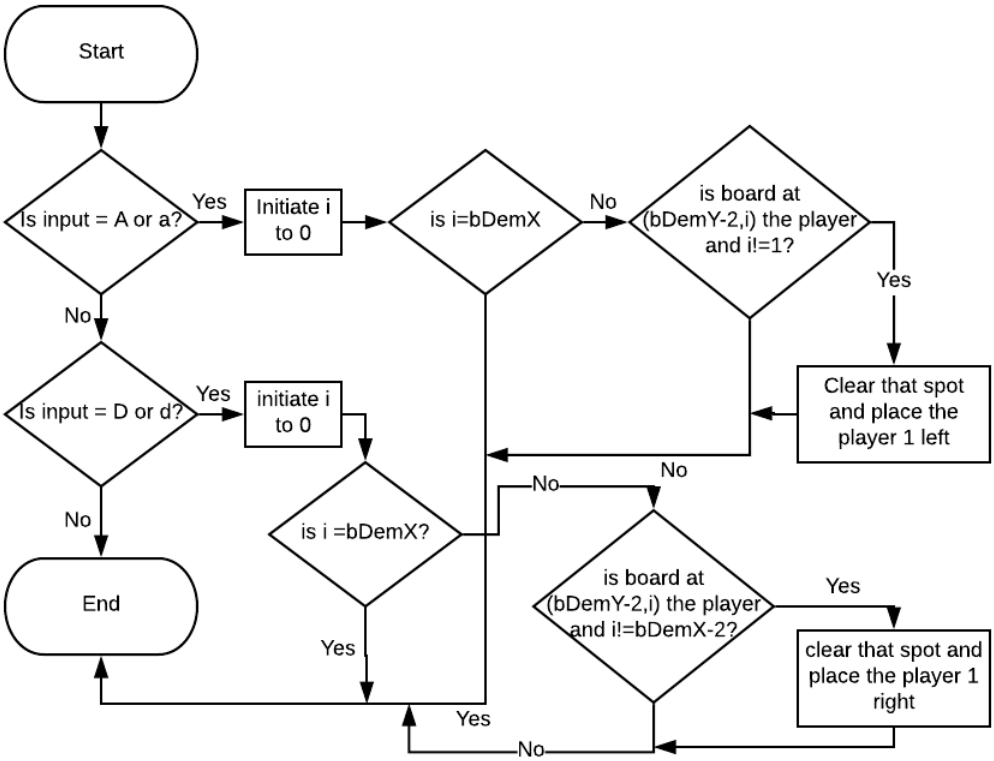
3.10 Move Player

This function moves the player to the right or left depending on the direction given in the UART interface

Usage

This function is called by the handle char subroutine when the input is determined to be a or d.

3.10.1 Move Player Flowsheet



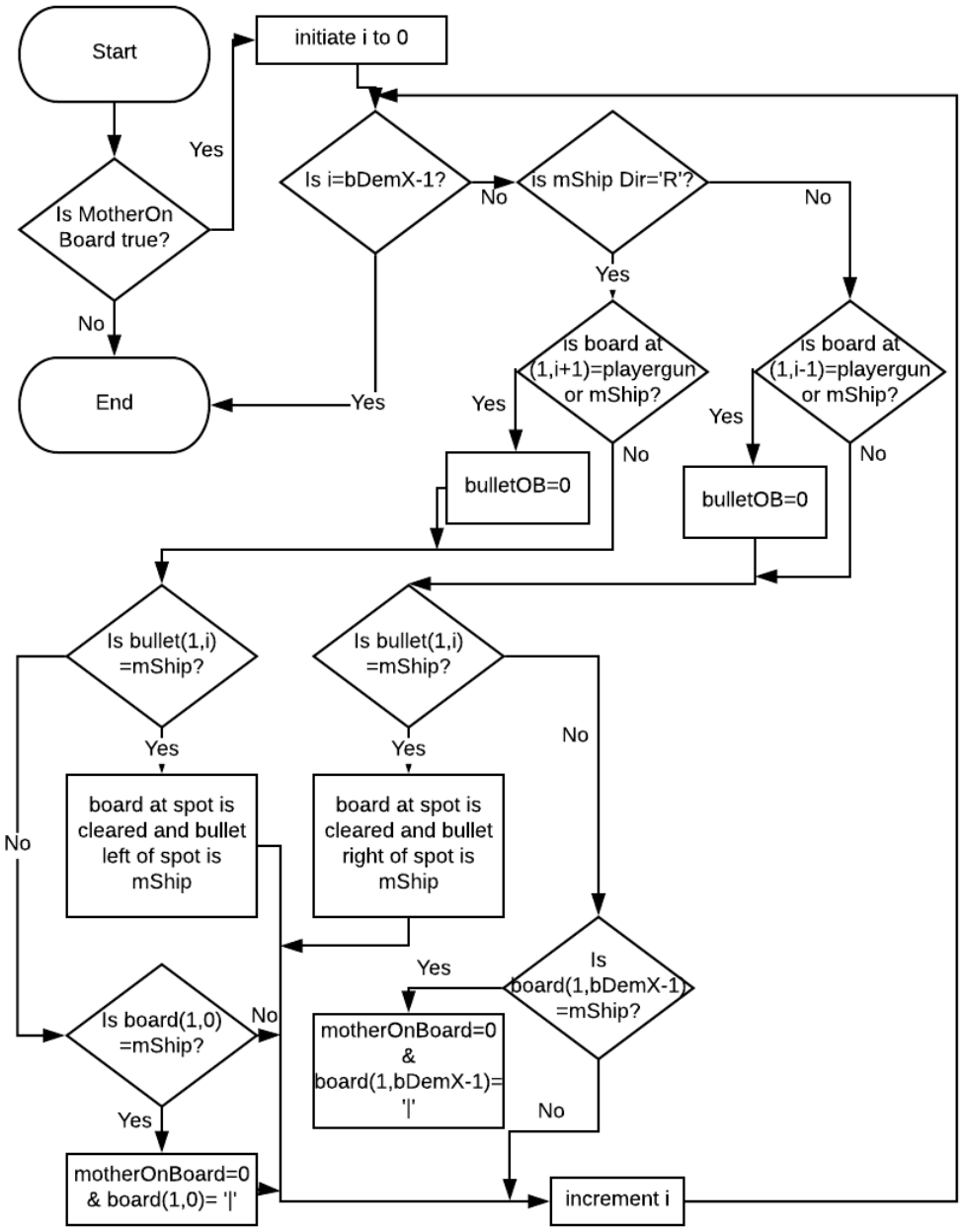
3.11 Move Mother

This function moves the mothership left or right depending on the direction given.

Usage

This function is called on every cycle of timer0, at the same speed as the refresh rate of the game.

3.11.1 Move Mother Flowsheet



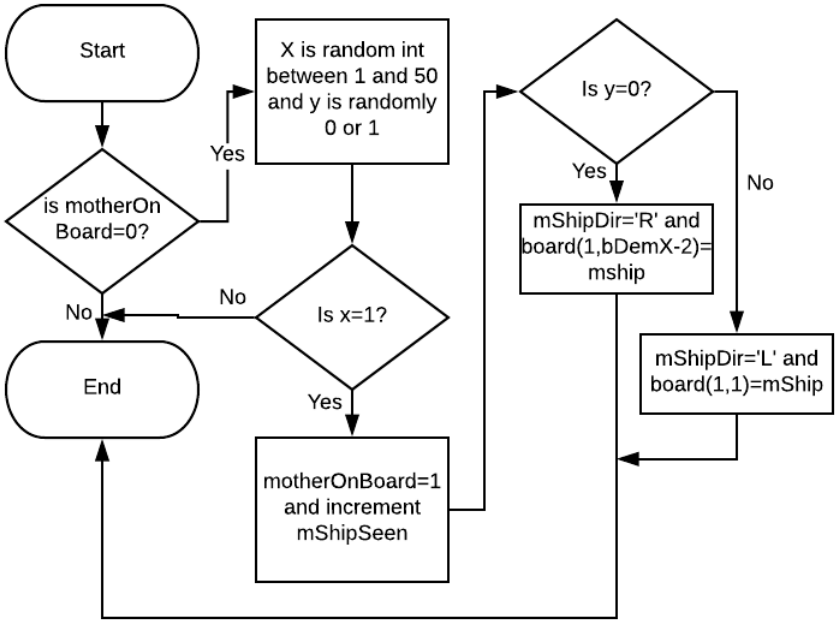
3.12 Generate Mother

This function determines whether or not to generate a mother ship based on a few conditions. First it checks to see if there is already a mother ship on the board, if there is it will not generate another. Then it generates a random number from 5-1, if that number is one then a mothership is generated. The mother ship's initial direction is determined by another random number.

Usage

This function is called on every clock cycle of timer0, but does not always generate a mothership.

3.12.1 Generate Mother Flowsheet



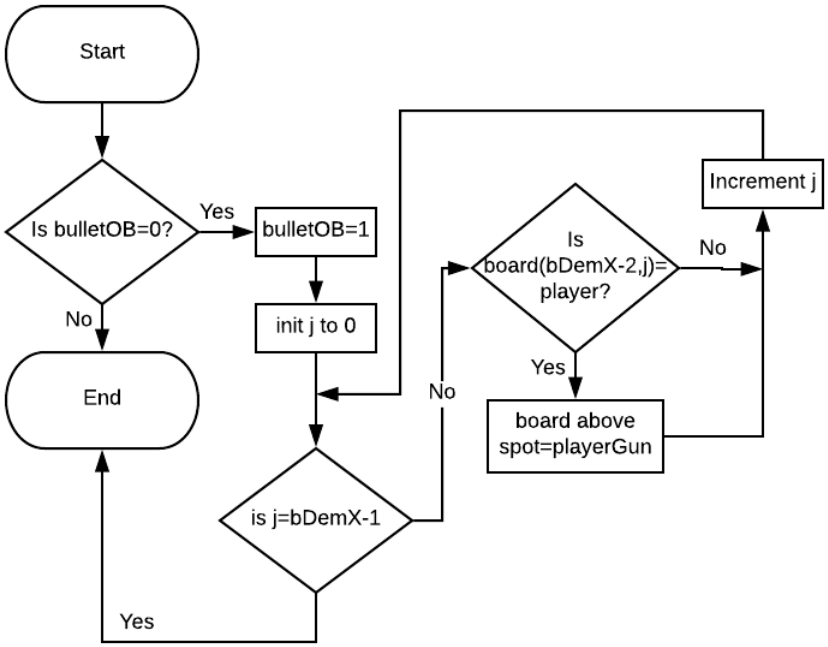
3.13 Generate Player Shot

This function checks to see if a player shot is already on the board. If not it places a player shot directly above the player

Usage

This function is called by handle char whenever the player's input is determined to be w or W.

3.13.1 Generate Player Shot Flowsheet



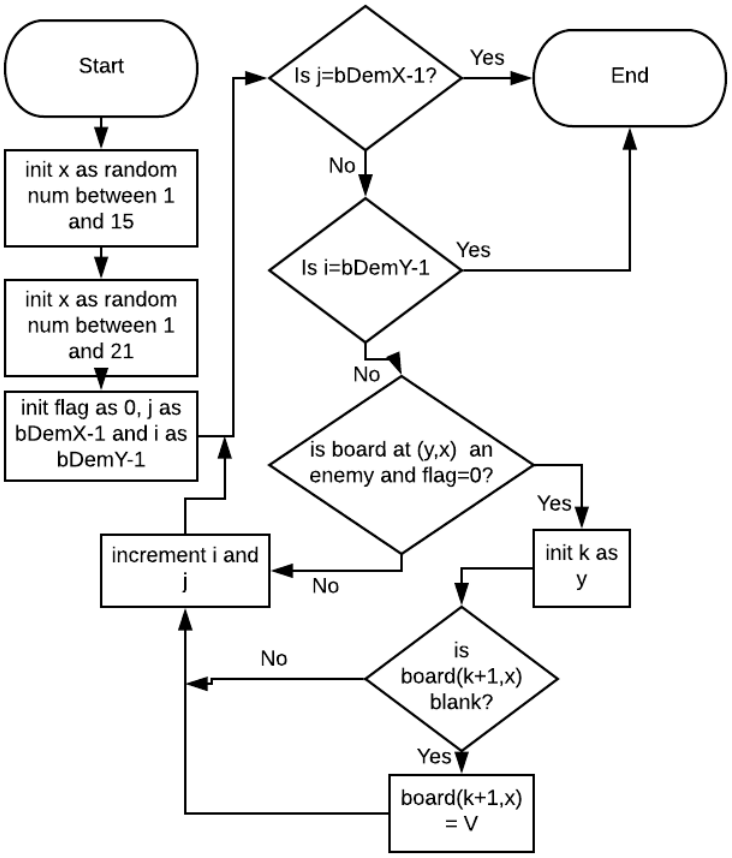
3.14 Generate Enemy Shot

This function checks a random spot on the board for an enemy. If there is an enemy at the spot the function will place an enemy shot below the furthest enemy down from the spot on the board.

Usage

This function is called on every cycle of timer0, but is only used when the random numbers hit on an enemy.

3.14.1 Generate Enemy Shot Flowsheet



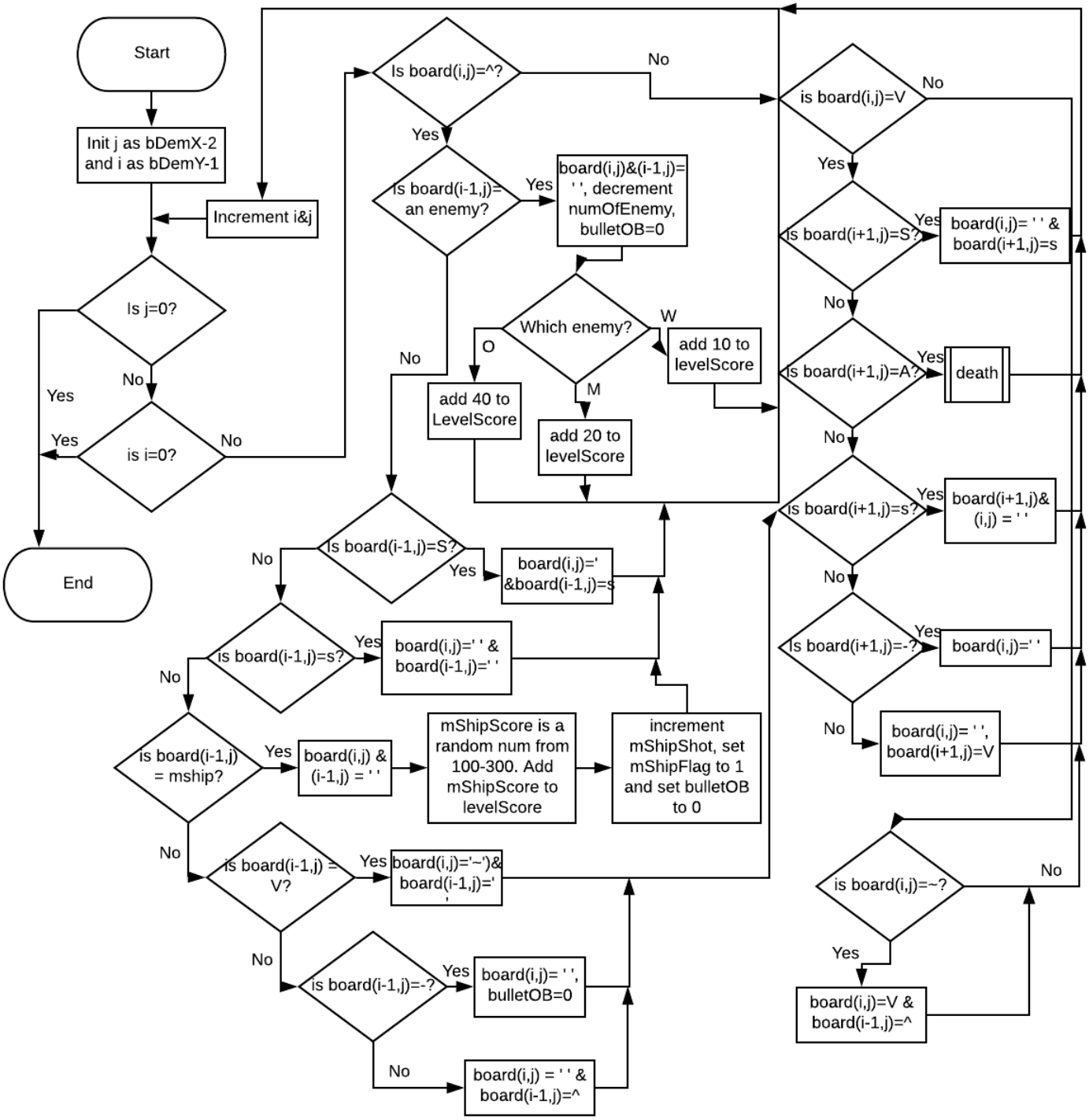
3.15 Move Shot

This function handles both enemy and player shots. The function searches the board for an enemy or player shot and looks at the spot below or above it, respectively. If either shot hits a Strong shield the shield is make into a weak shield. If they find a weak shield the shield is removed. If an enemy shot finds the player the death function is called. If a player shot finds any enemy the enemy and the shot are both removed. Finally, if the two shots find each other a tilde is placed where they met to represent the overlap.

Usage

This function is called on each cycle of timer0, it updates each shot on the board before exiting.

3.15.1 Move Shot Flowsheet



3.16 End Game

This function serves as the final screen in the game. It turns the RGB LED purple to reflect this. This function prints a bunch of stats and information regarding the players performance in the game for the user's knowledge. Finally it asks the user if they'd like to play again, restarting the game if so and ending the program if not.

Usage

This function is called when the player has lost all of their lives, or the space invaders have reached the bottom of the screen, or the timer has hit zero.

3.16.1 End Game Flowsheet