# Contents

# 1 Overview

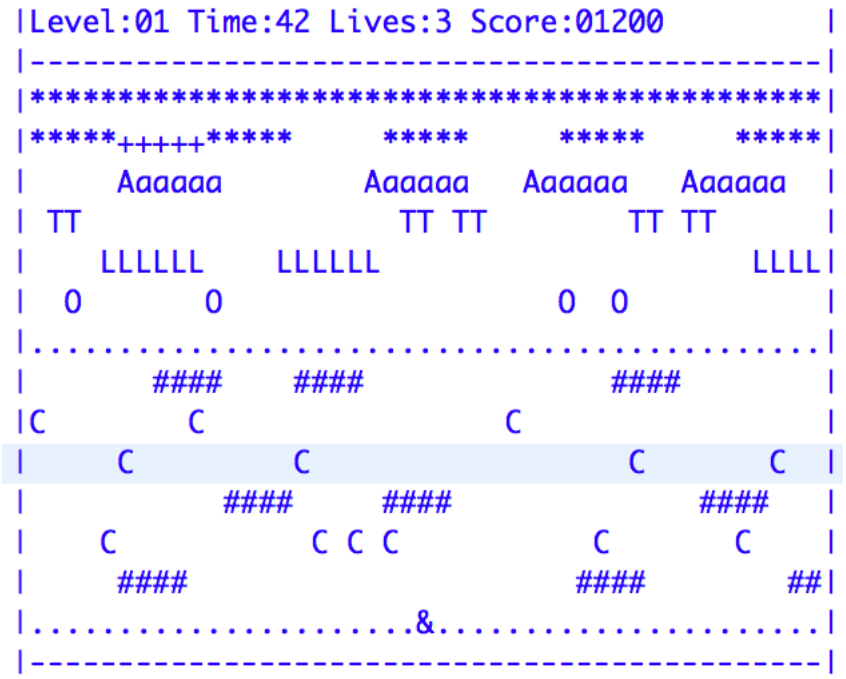
## 1.1 Division of work

This is what parts both of us worked on.

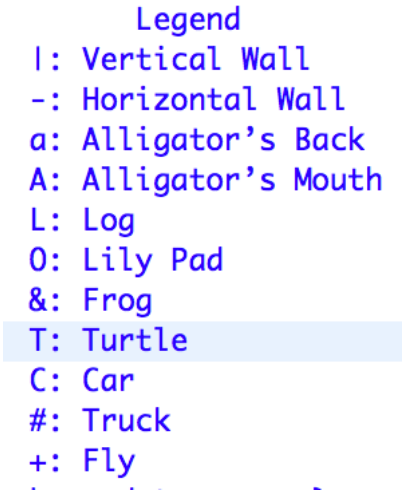
| Tijmen | Sakar |
|---|---|
| Board Init | Hazard detection |
| Main Menu | Collusion |
| Timers | Endgame menu |
| Random number | Pause screen |
| Keypad | RGB |
| Score keeping | LED |
| New level | Frog movement |

## 1.2 Purpose of program

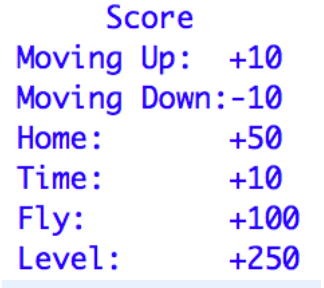
The purpose of this program was to make frogger. The objective of this program is to cross the street while avoiding obstacles and crass the river without falling in the water and get to the other side before time runs out or you run out of lives.




The ” is the frog that you want to get home. The bottom half of the board is the street in this half you want to stay in the whitespace. The topo half of the board is the river here you want to avoid the white space and cross the river using the platforms given. The top non-asterisks area is the home area where you want your frog to be. Once 4 frogs are brought home the level will increment.




This is the legend. Walls, Alligator's Mouth, Cars and Trucks are hazards. Flies are bonus points. Logs, Lily pads, Alligator's back and turtles are platforms.




Moving up one space is worth 10 points. Moving the frog down one space is worth -10 points. Getting a frog home is worth 50 points. Every second left on the time is worth 10 points each. The fly is worth 100 points and completing a level is worth 250 points.

### 1.3 Controls

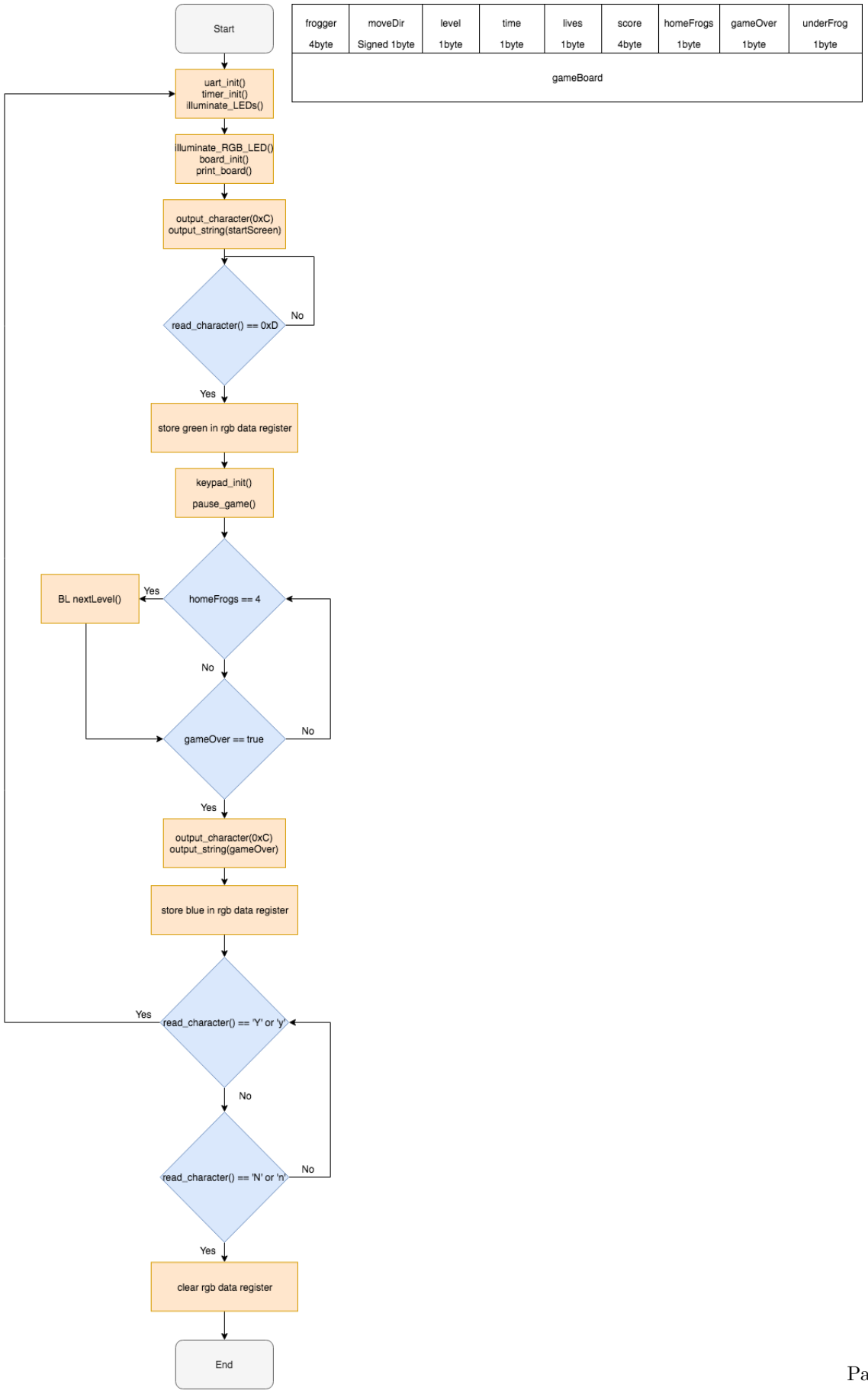
To play this game you use WASD to move the frog up, down, left, and right. To pause and unpause the game press anybutton on the keypad.

### 1.4 Outside material and debugging steps

We used the ARM Reference card, the lecture notes, and the Tiva$^{TM}$ C Series TM4C123GH6PM Microcontroller Data Sheet to make this program.We debugged this program by testing all the subroutines before creating the main program.

## 1.5 Program Flow Chart

The program starts by initializing uart, timers, leds, and the board. Then it prints the start screen. Once the player presses enter on the keyboard the game start by initializing the keypad unpausing the game and and making the rgb green. Once the game has started teh program will go into a infinite loop if homeFrogs gets to 4 it increases the level. If there is a game over it will print the gameover screen and turn the rgb blue. Once the game over screen is printed the user will be asked to press y or n. if n is pressed the rgb turns off and the program ends. If y is pressed the program restarts from the top.
The rest of the game is handled in the interrupt handlers. The Uart handler changes the moveDirection for the frog. The Timer0A handler updates the slower elements on the board and the fly generation. The Timer0B handler moves the frog updates the fast elements on the board and checks if the frog dies or got home. The timer1A handler decrements the level time. The portA handelr pauses and unpauses the game when entered.


Start
uart_init()
timer_init()
illuminate_LEDs()
illuminate_RGB_LED()
board_init()
print_board()
output_character(0xC)
output_string(startScreen)
read_character() == 0xD
No
Yes
store green in rgb data register
keypad_init()
pause_game()
homeFrogs == 4
Yes
BL nextLevel()
No
gameOver == true
No
Yes
output_character(0xC)
output_string(gameOver)
store blue in rgb data register
read_character() == 'Y' or 'y'
Yes
No
read_character() == 'N' or 'n'
No
Yes
clear rgb data register
End
frogger 4byte
moveDir Signed 1byte
level 1byte
time 1byte
lives 1byte
score 4byte
homeFrogs 1byte
gameOver 1byte
underFrog 1byte
gameBoard

# 2 Subroutines

## 2.1 output_character

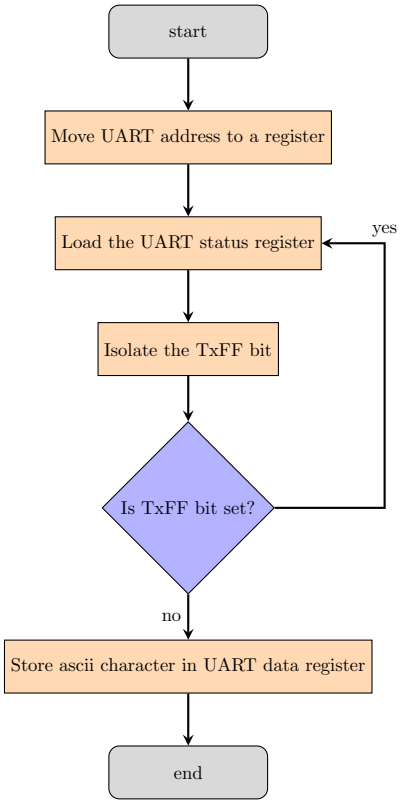
output_character takes an ascii value from r0 and transmits it using UART

**Usage**
To use this subroutine first initialize UART with the uart_init subroutine. Then pass the ascii value using r0.

**Exceptions**

1. This transmits one 8bit ascii value.
2. This will fail if you do not initialize UART properly.

### 2.1.1 output_character Flow Chart



## 2.2 output_string

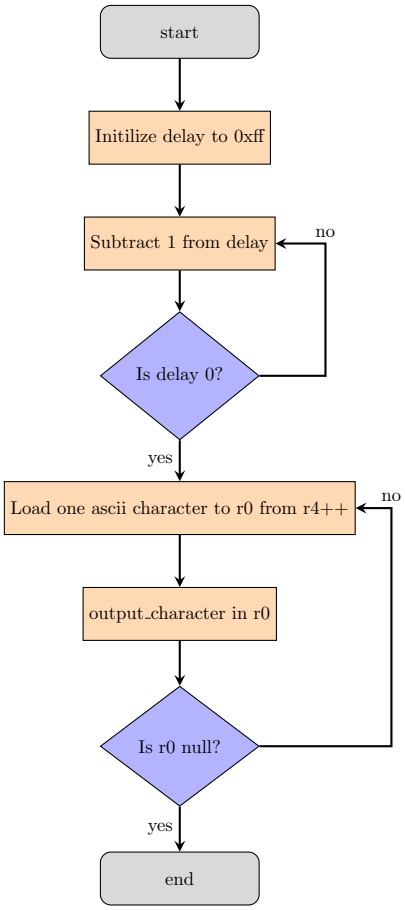
output_string transmits a null terminated string using UART. The base address for the string should be passed though r4.

**Usage**
To use this subroutine you need initialize UART using the uart_art subroutine. Then pass the base address of a null terminated string using r4.

**Exceptions**

1. It can pass an indefinite amount of 8bit ascii characters.
2. This can fail if the string isn't null terminated.
3. This can fail if UART isn't initialized.

### 2.2.1 output_string Flow Chart

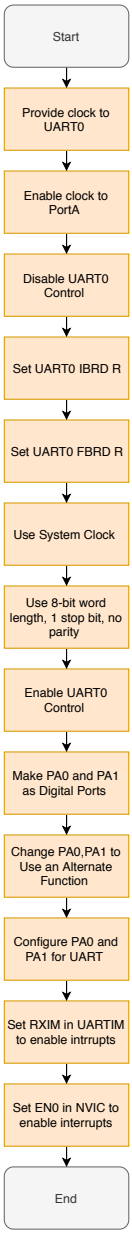


## 2.3 uart_init

uart_init initializes UART0 for interrupts. Interrupts are handled in Uart0Handler.

**Usage**
Use this subroutine before using subroutines that rely on UART.

### 2.3.1 uart_init Flow Chart

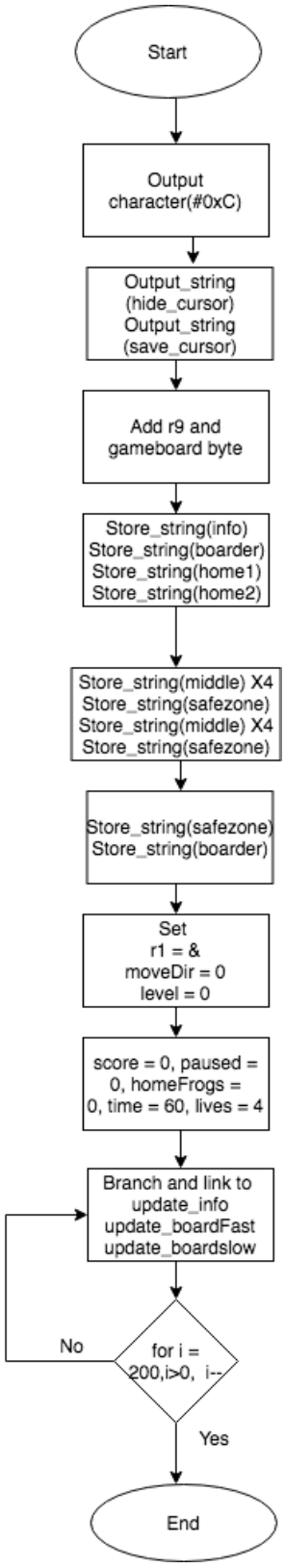


## 2.4 board_init

board_init initializes the board and all the variables in memory.

**Usage**
Use this subroutine to initialize the board and variables for the frogger game.

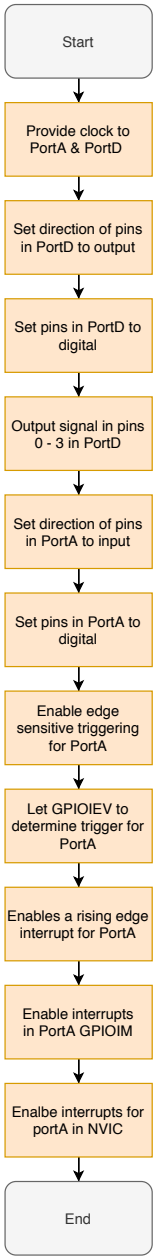
### 2.4.1 board_init Flow Chart



## 2.5 timer_init

timer_init initializes the timer to interrupt for the program.

### 2.5.1 timer_init Flow Chart

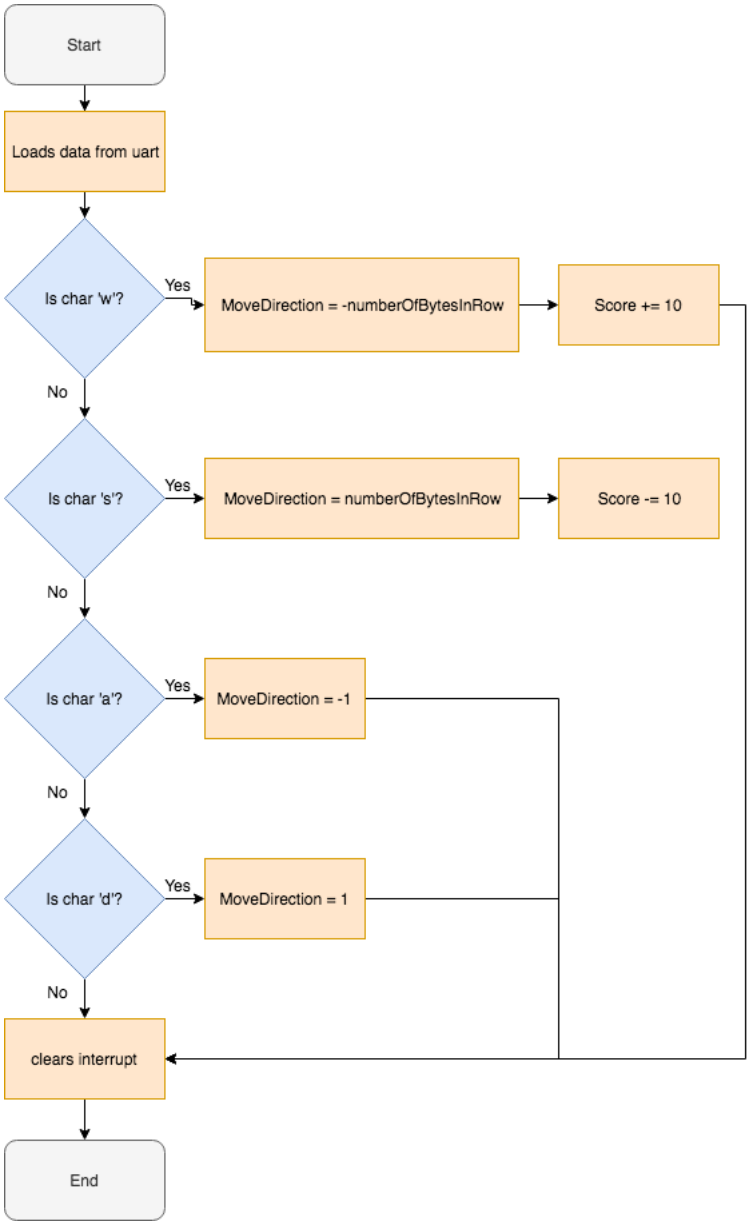


## 2.6 Uart0Handler

Uart0Handler changes the moveDirection of the frog changes the score appropriately when entered

**Usage**
This interrupt handler handles interrupts from the keyboard

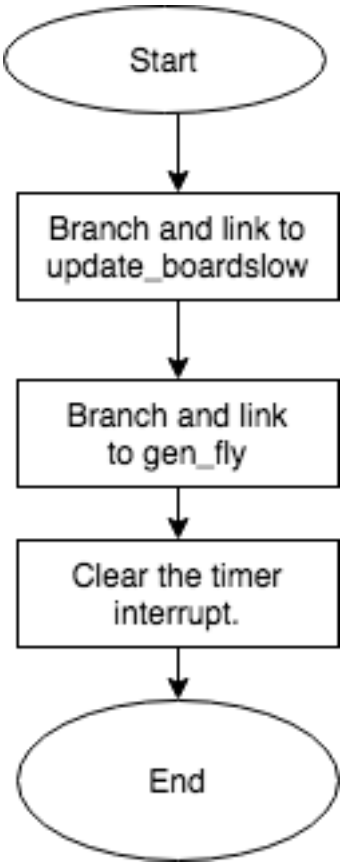
### 2.6.1 Uart0Handler Flow Chart



## 2.7 Timer0AHandler

This subroutine is called when the slow timer runs out. This subroutine is responsible to update the board and generate the fly.

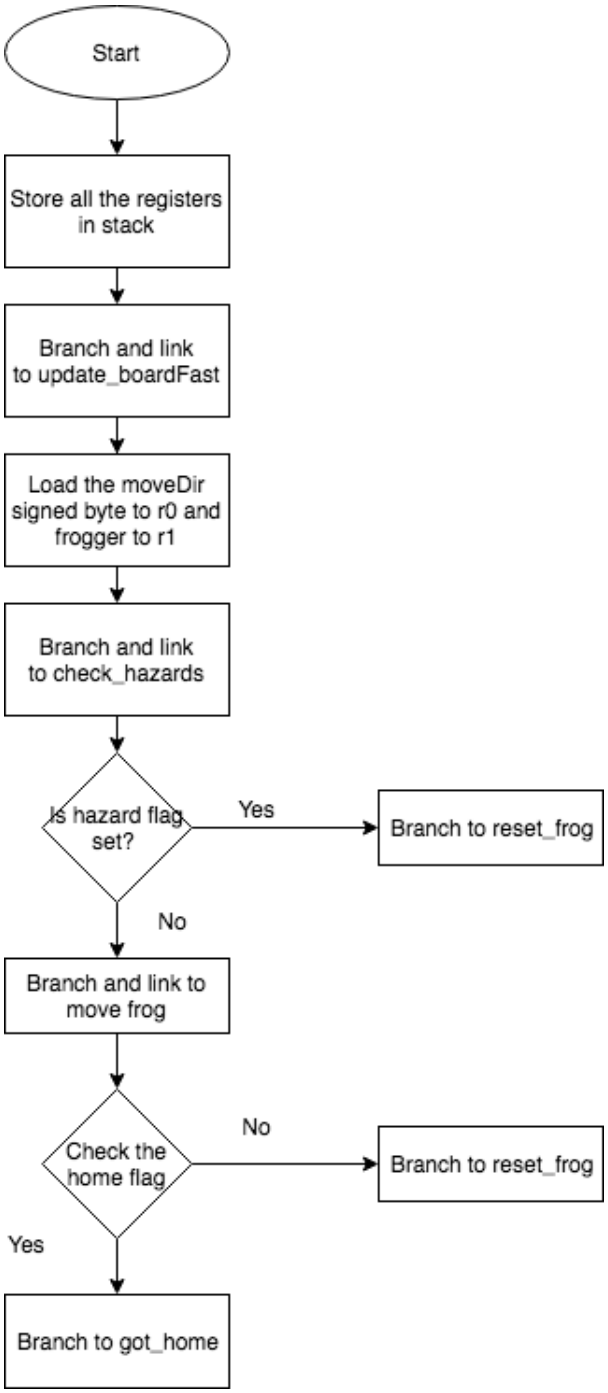
### 2.7.1 Timer0AHandler Flow Chart



## 2.8 Timer0BHandler

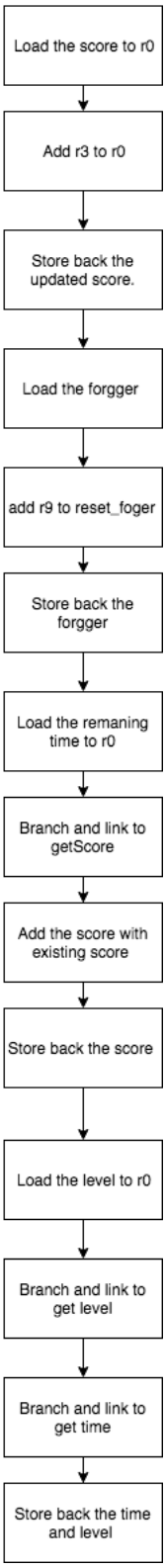
This subroutine is called when the Fast timer runs out. Frog movement and board updates happens from this subroutine.

### 2.8.1 Timer0BHandler Flow Chart



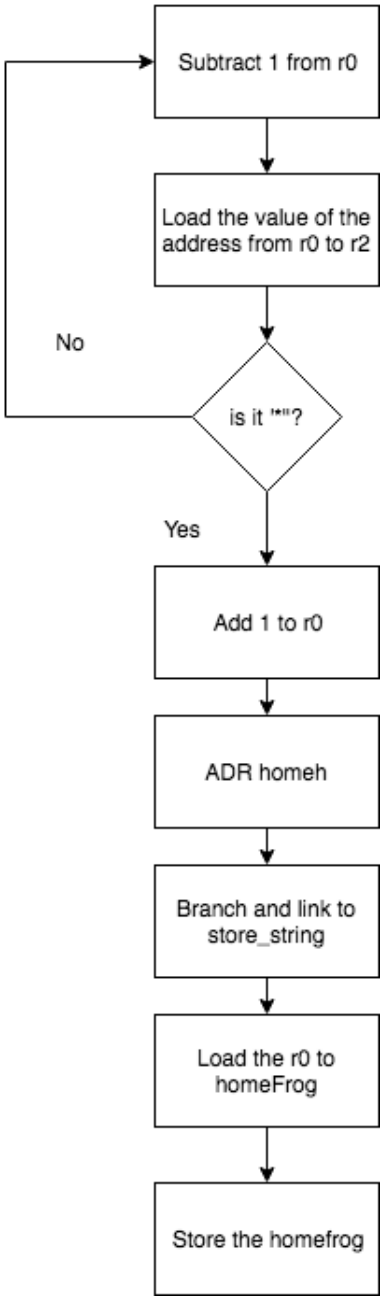
## got_home

This subroutine is called when the frog gets home row.



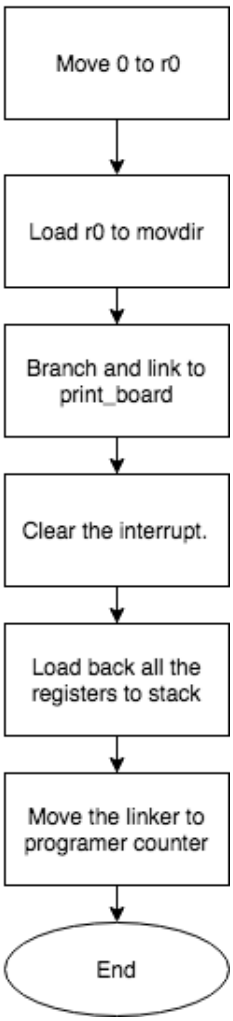
# move_left

This subroutine moves the pointer left on the home row to check if the home is taken or not.



# skip_home

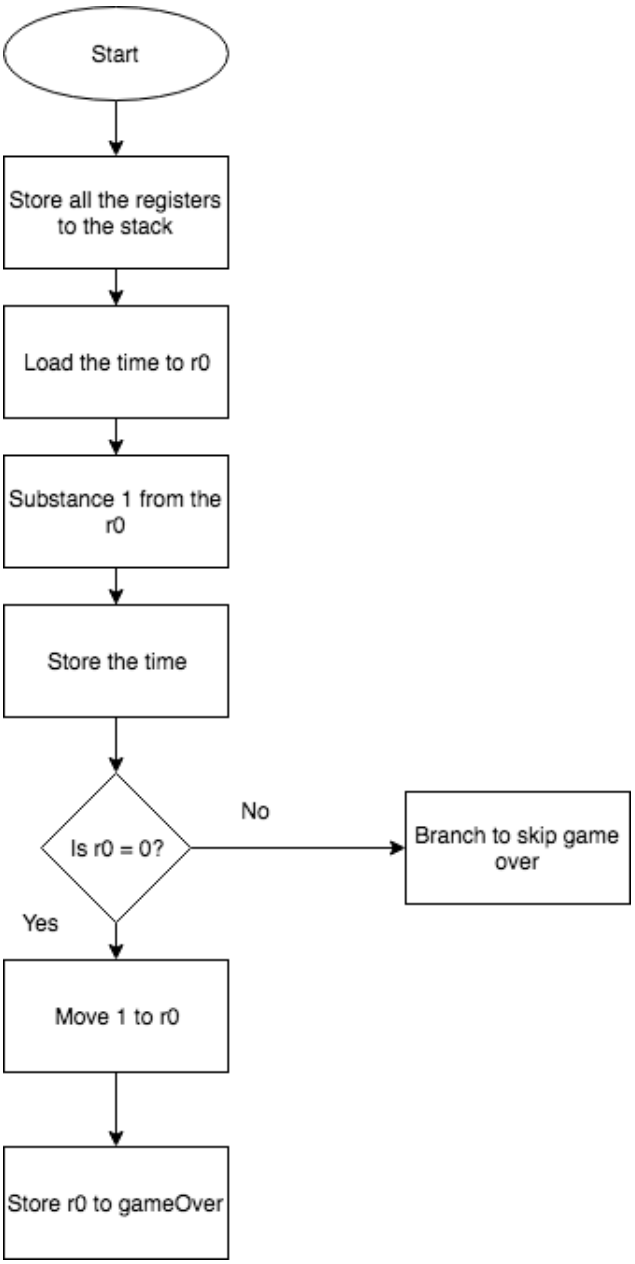
This subroutine is called when we want to skip the home subroutine



## 2.9 Timer1AHandler

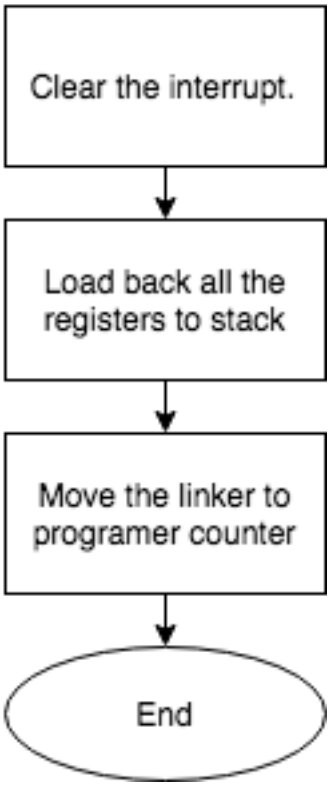
This subroutine is called when the game timer interrupt. it keep track of game timer.

### 2.9.1 Timer1AHandler Flow Chart



# skipGameOver

This subroutine is called when the game timer is interrupted but has run out yet.



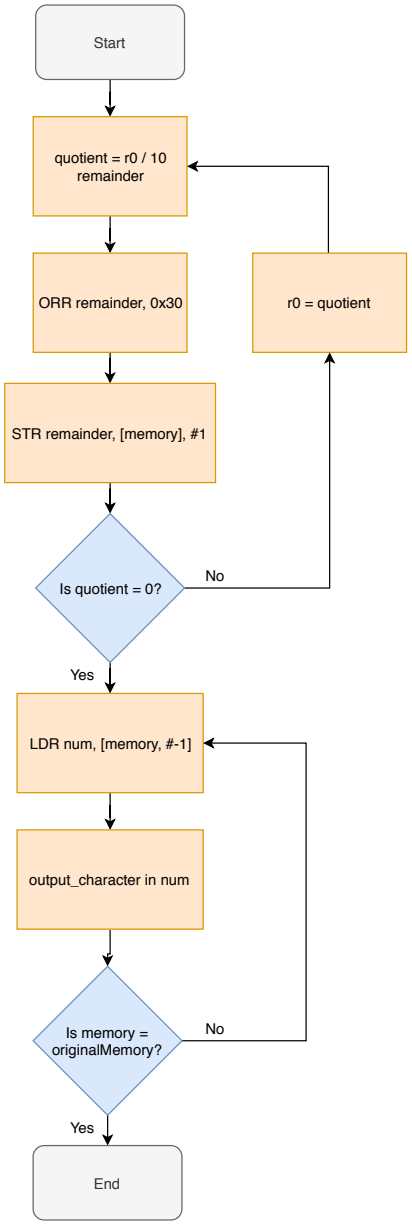
## 2.10 output_decimal

output_decimal takes a unsigned number from r0 and outputs a decimal number to the screen. Pass free memory address to r4.

Division is done using the div_and_mod subroutine.

**Usage**
To use pass a unsigned number in r0 and an address to freeMemory in r4. It will output the decimal value to the screen.

### 2.10.1 output_decimal Flow Chart

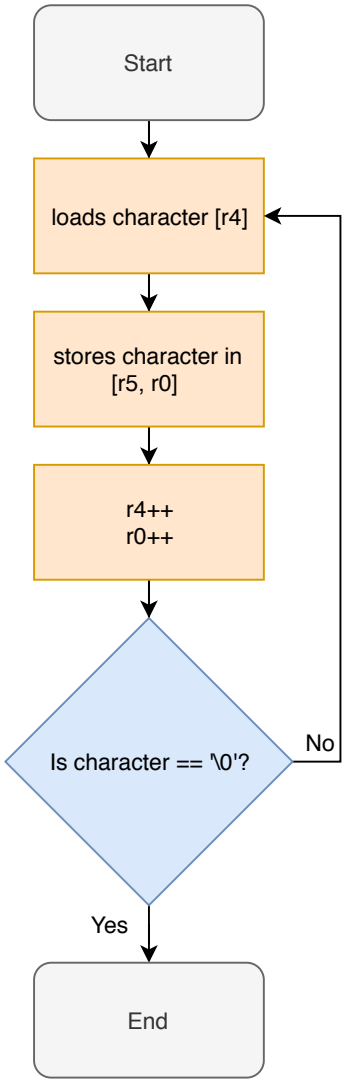


## 2.11 store_string

store_string stores a string from one memory location to another. Parameters: r5 is the address to ram. r4 is the address to the string. r0 is the offset to store the string in ram. Returns: r0 is the new Offset.

**Usage**
To use this subroutine pass the address to string in r4, address to ram in r5 and offset to free memory in r0. Then it will store the string from r4 to r5+r0.

### 2.11.1 store_string Flow Chart



## 2.12 div_and_mod

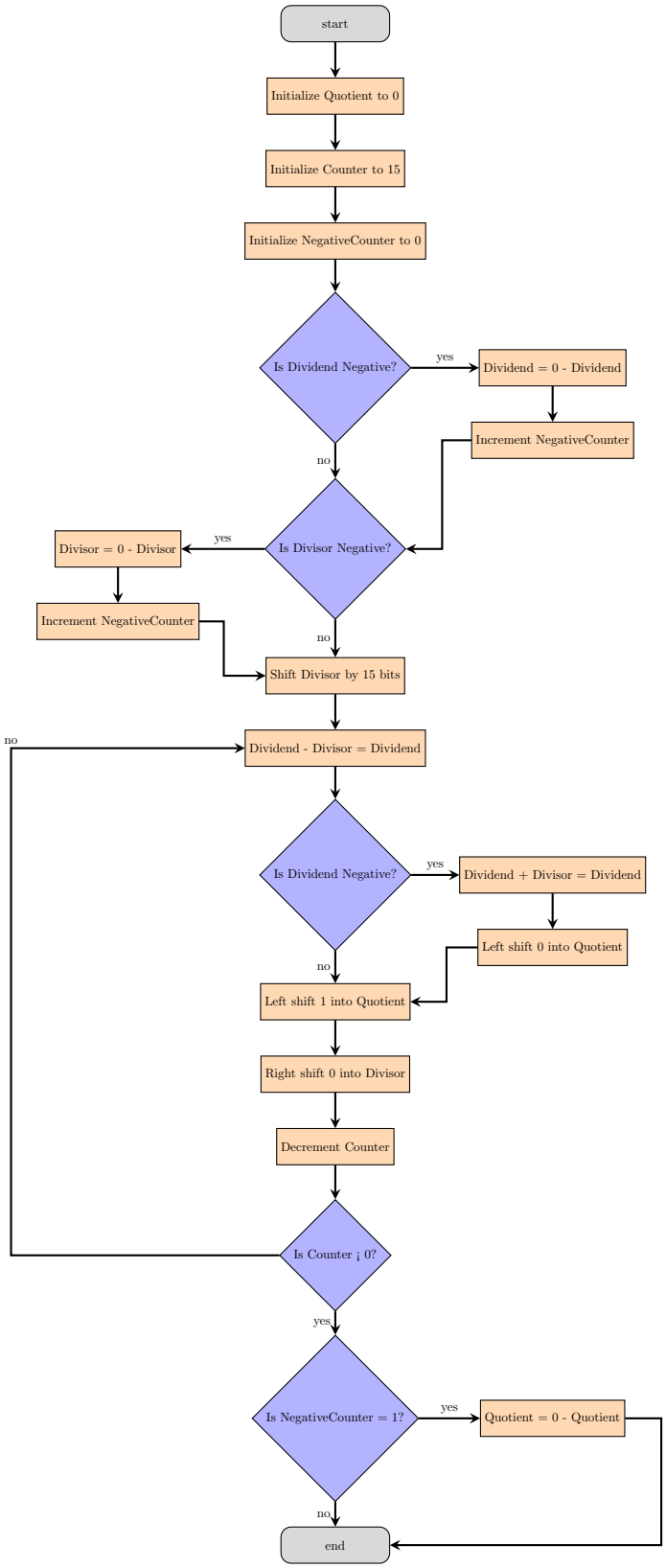
div_and_mod takes a signed divisor and dividend and returns a signed quotient and unsigned remainder. The divisor is passed through r0. The dividend is passed through r1. The quotient is returned through r0. The remainder is returned through r1.

**Usage**
To use this subroutine you need to pass a 15bit or less divisor and dividend.

**Exceptions**

1. It can only handle 15bit divisors and dividends.

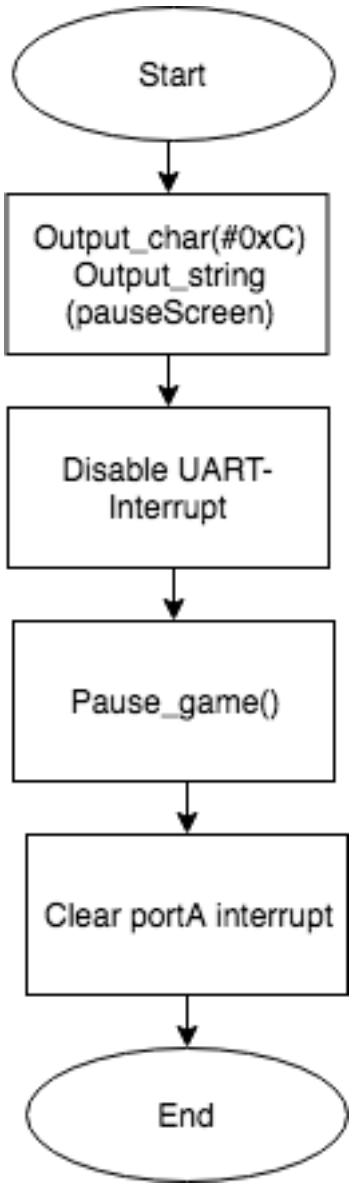
### 2.12.1 div_and_mod Flow Chart



## 2.13 PortAHandler

This handler pauses the game prints the pauseScreen and XORs the disable interrupt bit for uart.

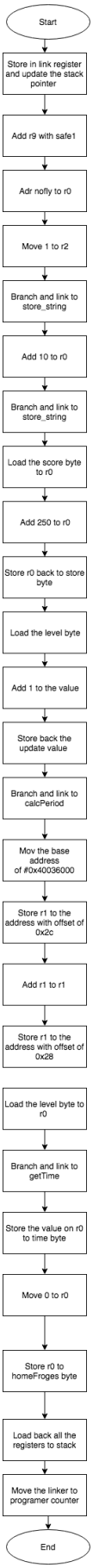
### 2.13.1 PortAHandler Flow Chart



### 2.14 nextLevel

This increments the level resets the home row and changes the time.

### 2.14.1 nextLevel Flow Chart



## 2.15 pause_game

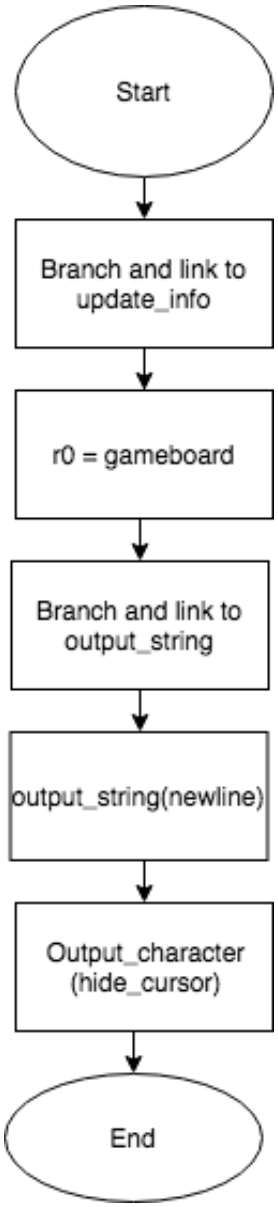
This subroutine is called when keypad interrupt occurs. it pauses the game, and prints the pause menu.

### 2.15.1 pause_game Flow Chart

## 2.16 print_board

This subroutine prints the board for the frogger game.

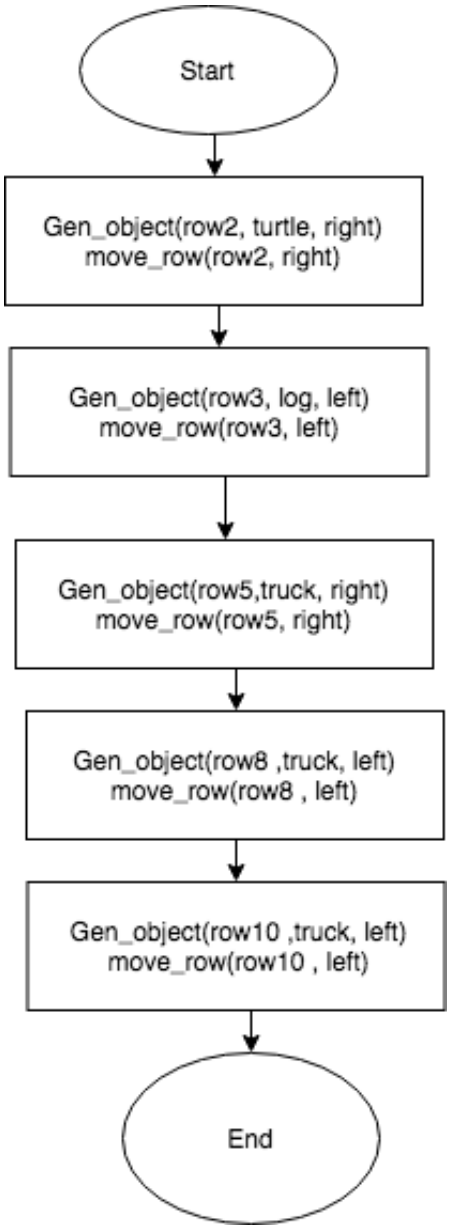
### 2.16.1 print_board Flow Chart



## 2.17 update_boardSlow

This subroutine updates the slow elements on the board.

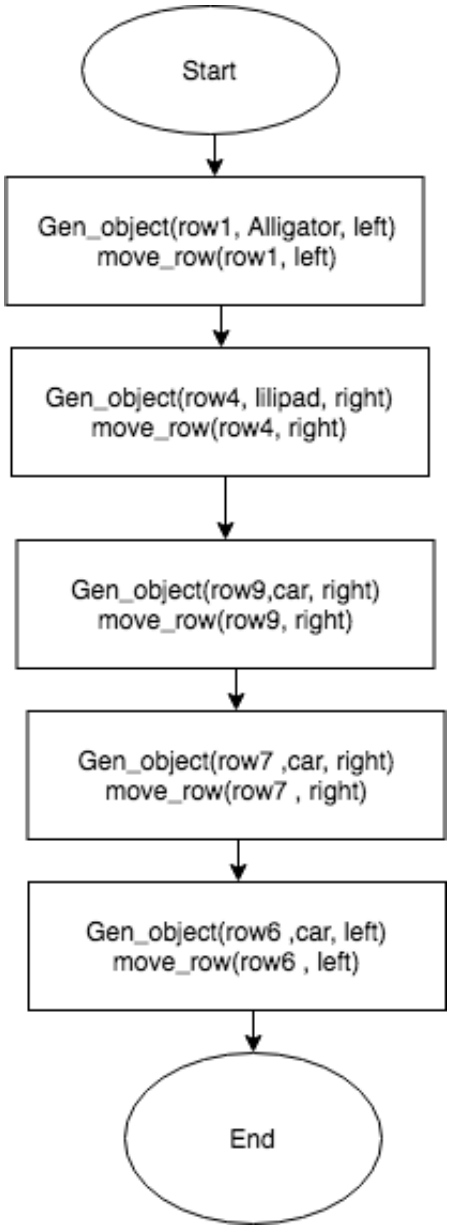
### 2.17.1 update_boardSlow Flow Chart



## 2.18 update_boardFast

This subroutine updates the fast elements on the board.

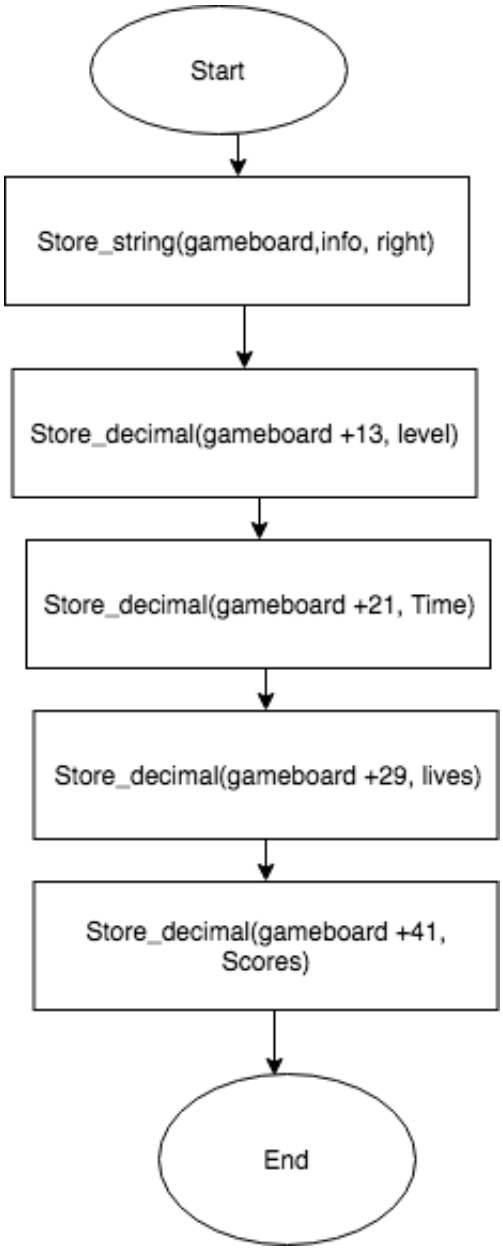
### 2.18.1 update_boardFast Flow Chart



## 2.19 update_info

This update the visual info on the top row of the board.

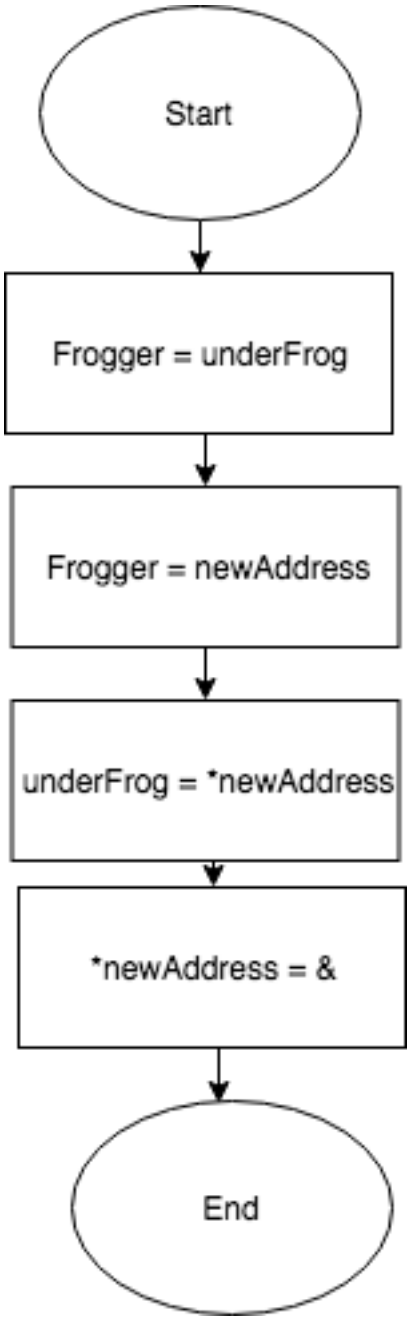
### 2.19.1 update_info Flow Chart



## 2.20 move_frog

This subroutine moves the frog to the give new address without damaging the boards integrity.

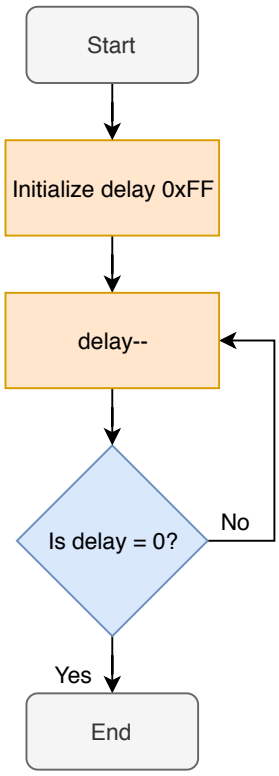
### 2.20.1 move_frog Flow Chart



## 2.21 delay

This subroutine delays the program when called.

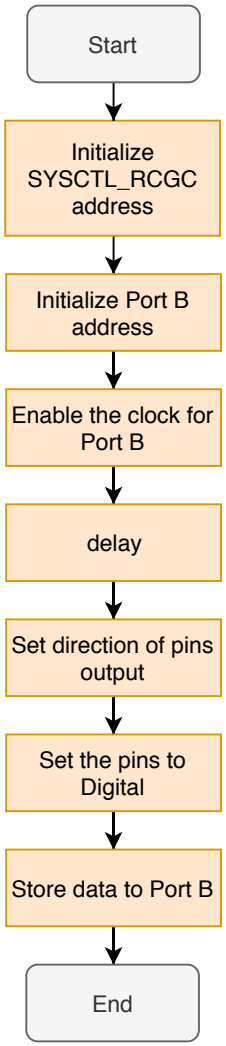
### 2.21.1 delay Flow Chart



## 2.22 illuminate_LEDs

This subroutine initializes the leds with all 4 lights activated.

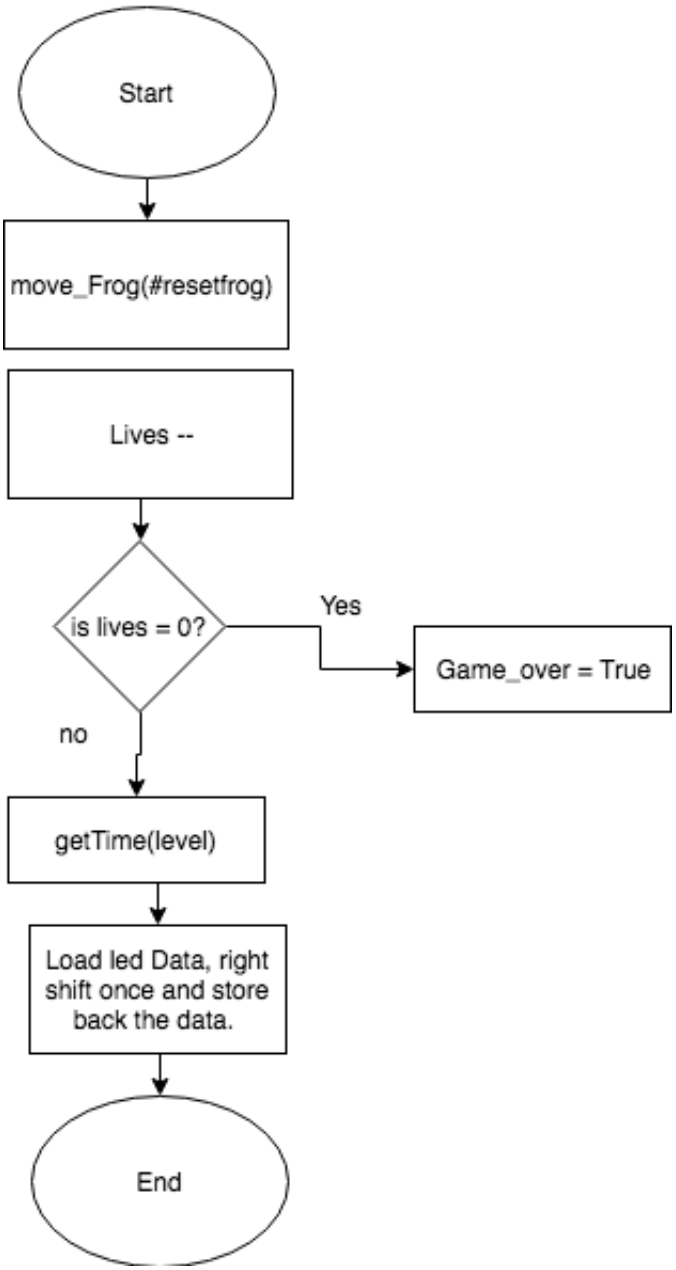
### 2.22.1 illuminate_LEDs Flow Chart



## 2.23 reset_frog

This subroutine move the frog back to the starting location and decrements the life counter and appropriately changes the lights on the board.

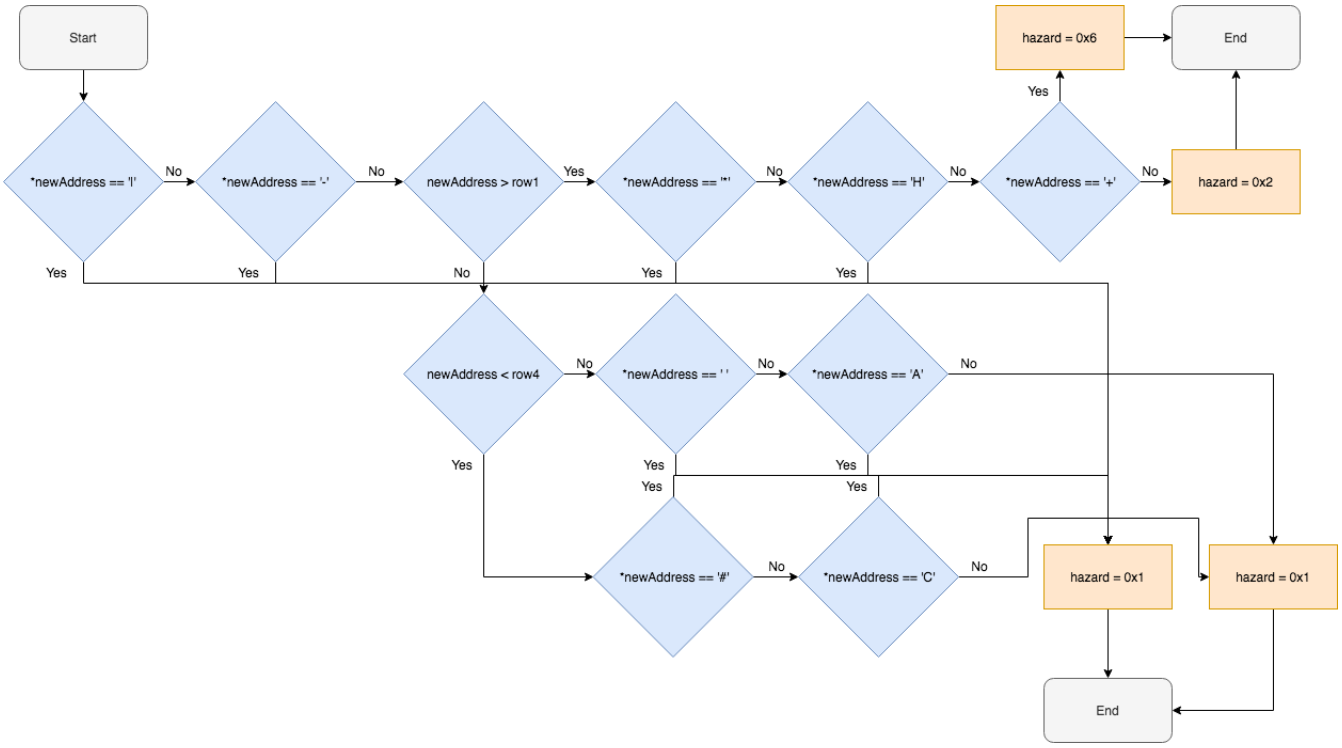
### 2.23.1 reset_frog Flow Chart



## 2.24 check_hazards

This program check the hazards at the address it is give thourgh r0 and returns a number in r1 indicating what is there.
;Fly found : 110 =0x06
;home : 010 = 0x02
;hazard : 001 =0x01
;safe : 000 = 0x00

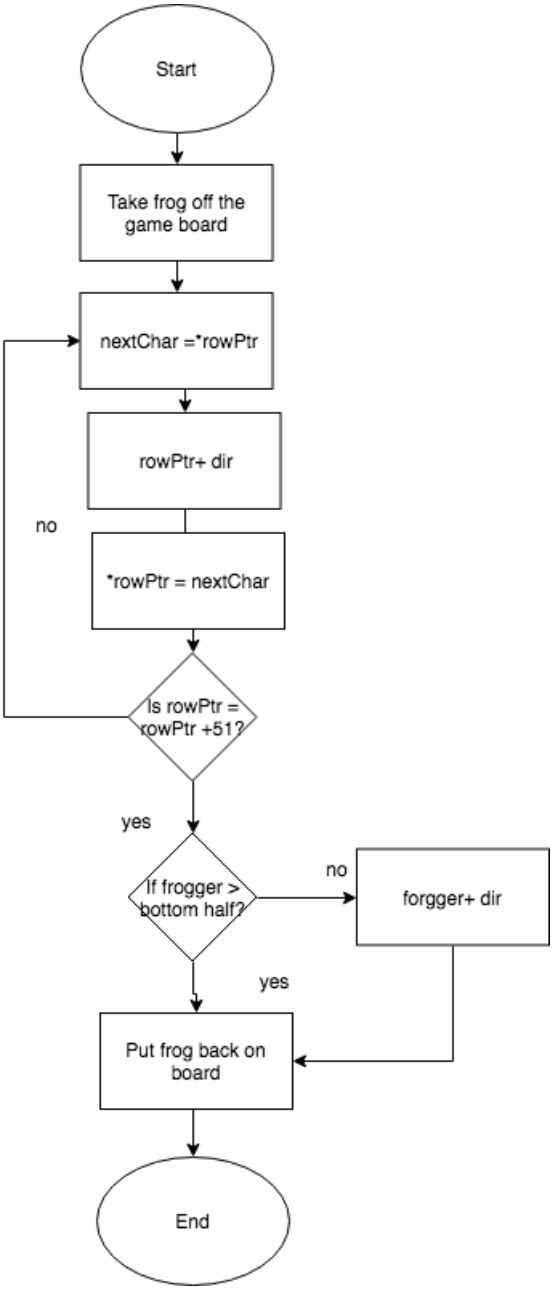
### 2.24.1 check_hazards Flow Chart



## 2.25 move_row

This subroutine shifts the row in the direction given.
;Parameters:
;r0: rowPointer
;r2: direction

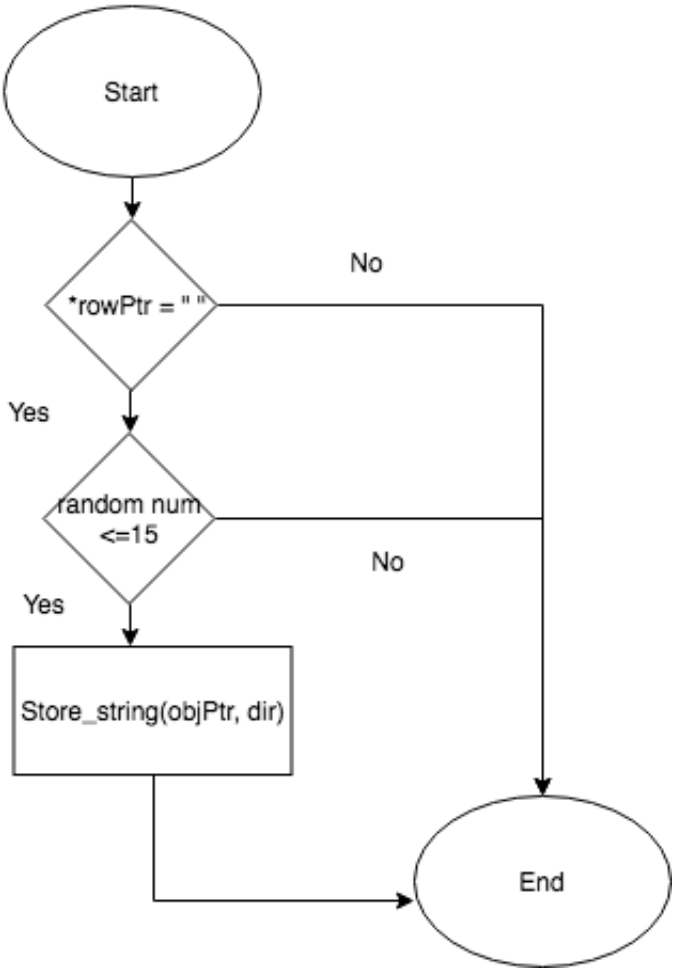
### 2.25.1 move_row Flow Chart



## 2.26 gen_obj

This subroutine generates objects in the given row.
;Parameters:
;r0: rowPointer
;r1: objectPointer
;r2: direction

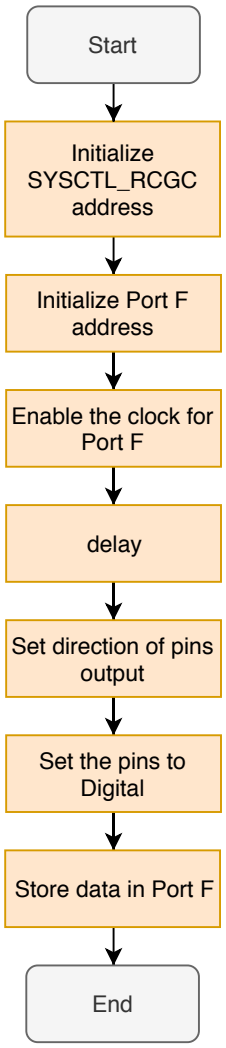
### 2.26.1 gen_obj Flow Chart



## 2.27 illuminate_RGB_LED

This subroutine initilizes the rgb led and turns it white.

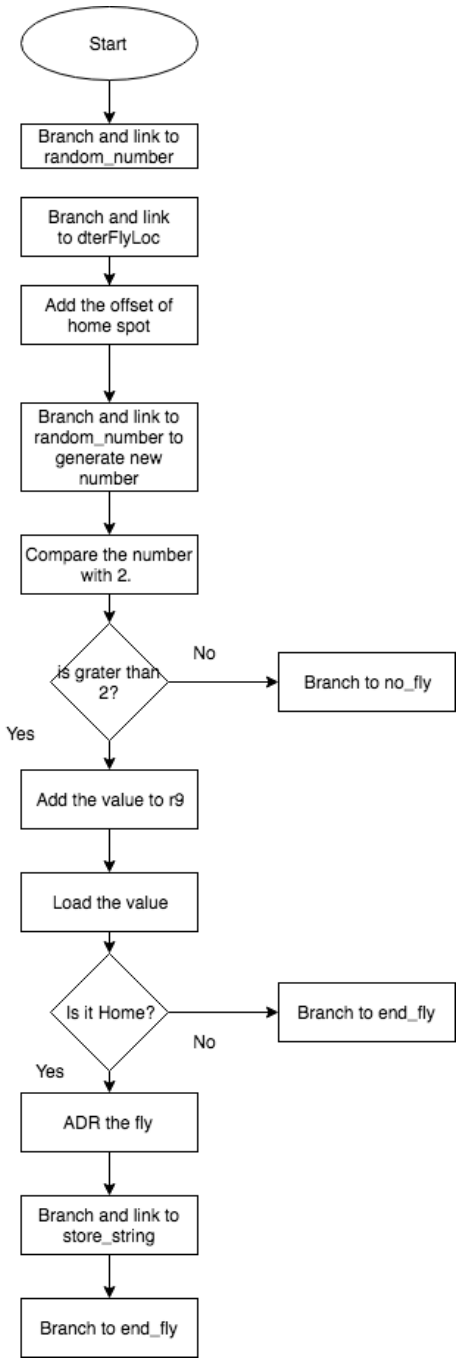
### 2.27.1 illuminate_RGB_LED Flow Chart



## 2.28 gen_fly

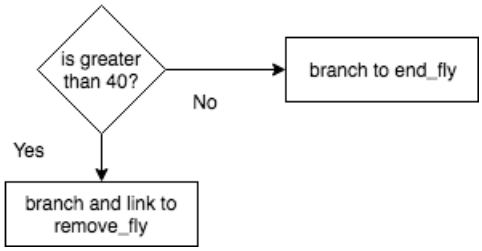
This subroutine is called to generated the fly in the random home spot. It added the fly on the top row based on the random number.

### 2.28.1 gen_fly Flow Chart



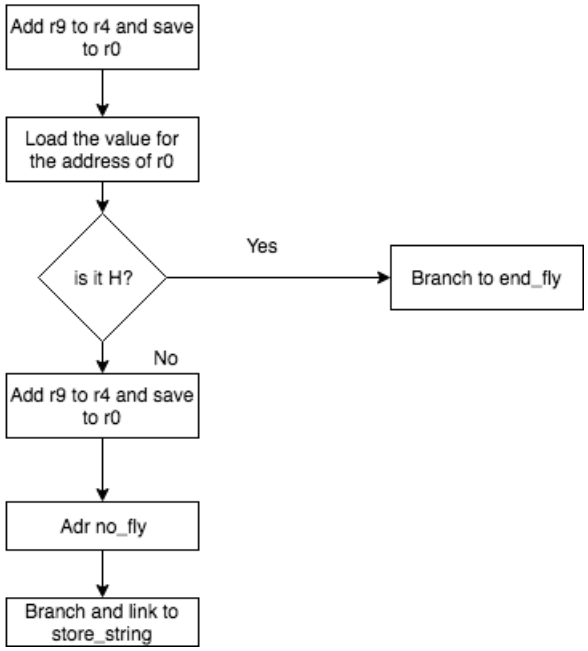
## no_fly

This subroutine is called to figure out whether  or not to generator fly. .



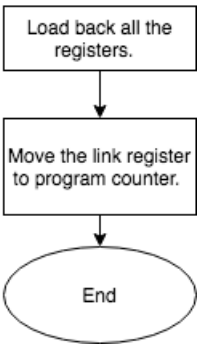

## Remove_fly

This subroutine is called to remove the fly when the timer runs out.



## End_fly

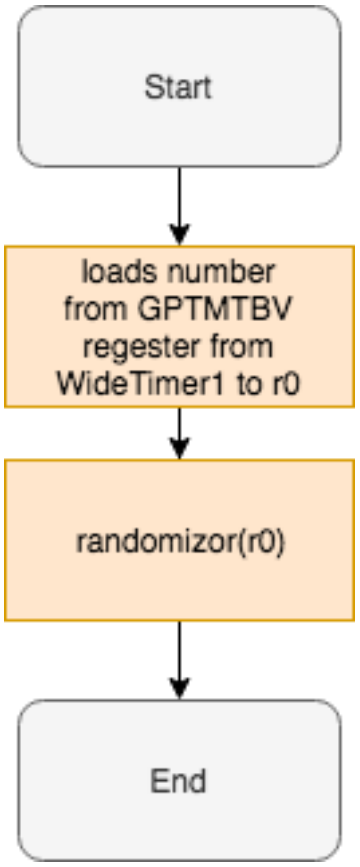
This subroutine is called at the end of the process of generating fly or anywhere we need to terminate the process of generating fly.



## 2.29 random_number

this subroutine returns a random number after randomizing a number from the timer1B counter.

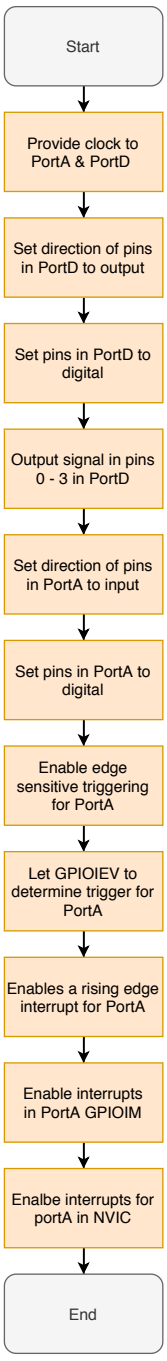
### 2.29.1 random_number Flow Chart



## 2.30 keypad_init

This subroutine initializes the keypad for interrupts.

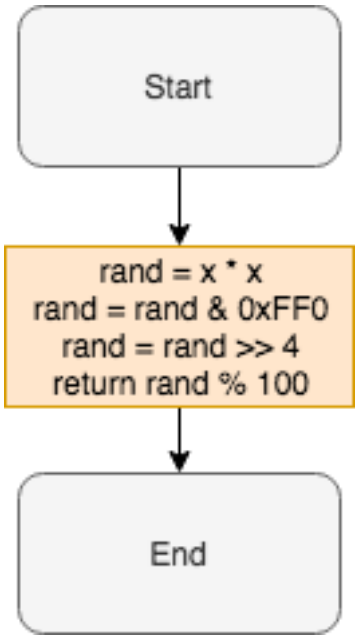
### 2.30.1 keypad_init Flow Chart



## 2.31 randomizor

This subroutine randomizes the number given to it.

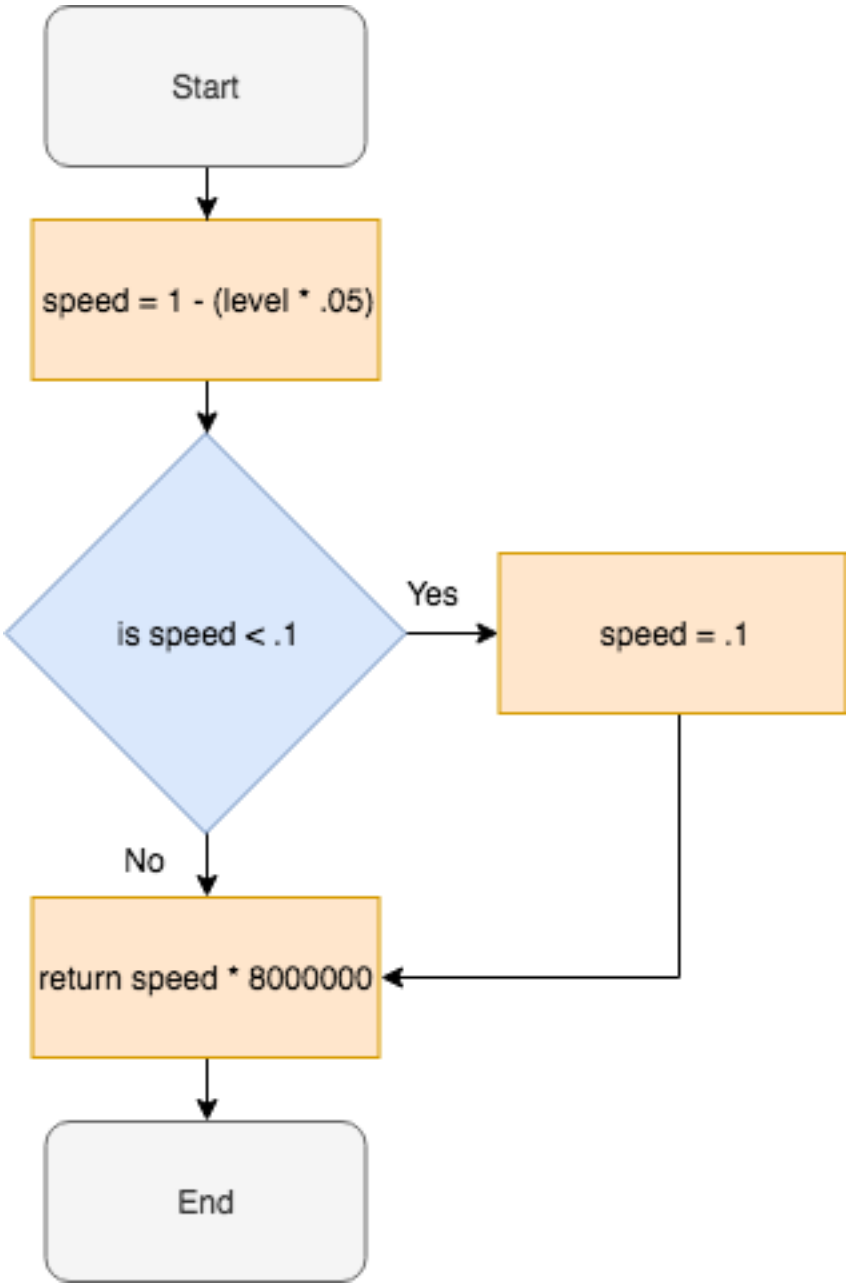
### 2.31.1 randomizor Flow Chart



## 2.32 calcPeriod

Calculates the period for the timers to interrupt at.

### 2.32.1 calcPeriod Flow Chart



## 2.33 dterFlyLoc

This determines the location to generate a random fly from the number given to it.

### 2.33.1 dterFlyLoc Flow Chart