CSE 379 Space Invader Lab 7 Documentation

University at Buffalo

Kevin Cheung (kcheung8)
Weijin Zhu (weijinzh)

May 5 , 2018

Contents

1 Description

1.1 Objective

Combine all the concepts we have learned throughout the course and the subroutines we have wrote from lab1 to lab6 together to make the game " Space Invader".

1.2 Division of Work

Both of us did the pre lab documentation together, board design and player movement.
Kevin Cheung finished all the function subroutines, mothership subroutines and monster subroutines.
Weijin Zhu finished player subroutines, random number subroutine and basic set up.
During lab open hours and recitations, both of us were there to code and debug.
For the documentation, Weijin Zhu did Description(part1) and Kevin Cheung finished the rest.

1.3 Outside Material

When making this game, we used subroutines that we wrote in lab1 through lab6 to do some of the functions that are required in the game. (ex: div-and-mod, read-character, output-character, output-string etc...). We used the ASCII Table to look up the value of some characters, (ex: the direction of the player, 'a' to move the player to the left, the value of 'a' on the ASCII Table is 97) and we used UM 10120-Volume 1:LPC213x User Manual (ex: address of IO0DIR, IO0SET, etc...)

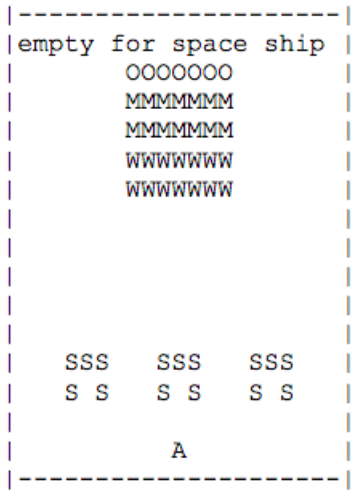
1.4 Debugging Steps

The first problem that we faced when we debugged our codes is after the player destroyed one column of the invaders, the column next to it should be the outmost column of the invaders and they should keep moving until they are next to the wall then move down. However, they just move down even they are still one column away from the wall. We fixed this problem by reassigning the address to this column, and the address that we use to indicate the outmost column.

The second problem that we faced is to create a random number generator. The random number generator has many more uses, such as to control the initial direction of invaders , the bullets that are fired by the invaders, and the random assigned points by destroying the mothership. We created the random number by adding more variables so we cannot predict the next values. We added variables such as the value that is in the timer and the user input ('a' or 'd' or 'space) ect...

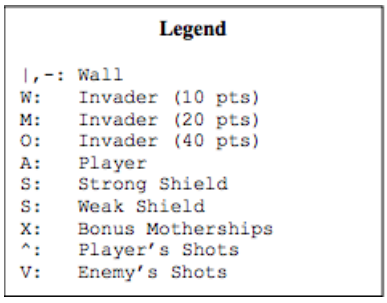
There are many other minor problems that we faced when we debugged our codes, such as when invaders move toward the wall, they will push the wall to the direction they are moving to, and also push the shields to the direction they are moving etc... However, at the end, we had solved all the problems, and the program would run smoothly.

1.5 Gameboard Layout



This version of Space Invaders that we created has the same rules as the original version that was created in 1978, but in a simpler version. The player is given 4 lives, when the player been shot by the invaders, he will lose 1 live. The game is over when he loses 4 lives. The main goal is to earn as more points as you can by shooting the invaders or the mothership and move on to the next level within the game duration of 2 minutes.

1.6 Score



To earn points:

* W (invader): 10 points

* M (invader): 20 points

* O (invader): 40 points

* X (mothership): 100 points - 300 points

* Reach the new level : 50 points

To lose points:

* Shot by the invaders' bullets: 100 points

* When invaders reach the line indicated on the board: 100 points

1.7 Rules

Game rules:

* The player begins with four lives and loses one live if shot by the invaders.

* The player is only allowed to move left or right.
board.

* One shot from the player will destroy one invader only.

* When the player's shot and the invaders' shot collide, they should go through.

* The game would be over if it reaches the duration of the game which is 2 minutes, or player loses all 4 lives.

* The player can only shot one bullet at a time, and second bullet can fire only when previous one disappeared.

* The player can attack the shields, and when attacks the shield, if it was a strong it would become a weak shield, and if it was a weak shield, the attack will destroy it. However, no score would be added if player attack the wall.

* When the game is paused, only press key "p" to resume the game.

1.8 Components

1-RGB: use to indicate the status of the game.Before the game starts, the RGB is white, during the game, the RGB should be green. When the game is paused, the RGB will turn blue. When the shot is fired, the RGB will be flash red, and the RGB will turn purple when the game is over.

2-Momentary Push Button: the main purpose of the momentary push button is to trigger an interrupt when pressed. It is used to start the game and pause the game.

3-LEDs: use to indicate the lives of the player. Start with four lives, and each time the player loses one live, one LED will turn off. When all the LEDs are off, the game is over.

4-Seven-Segment: use to display the score the user earned during each level. However since it only has four digit places which might result in overflow, the score is been set to zero when reach to a new level.

5-Timers- the main purpose is to control the movement of the game. It controls the speed of how the game goes, and the rate of refreshing the gameboard. All the invaders are moving in the same speed, and the speed of the mothership is faster than the invaders.

6-Serial Port: the purpose is to receive inputs from the user and display the contents.

1.9 Logic

First, we create a string called "board" to use to represent the game board, and the total of 35 invaders and wall will have its own address in the memory. We align 35 invaders (5 x 7) in the center of the game board by storing the their values that we found on the ASCII Table to the certain addresses. The first starting address is important because every other components are built by adding offsets to the starting address.To make all invaders move, we simply add offsets or subtract offsets from it depends on the direction they are moving, and write a "space" to its original spot. We then add timer to these instructions, the user would see them move in the constant speed. Same as the player movement, we use the key "a" to move the player to the left one spot, and use the key "d" to move the player to the right one spot. Every time the user hit either one of these two keys, it would triggers the UART interrupt, when compares the character that the user input to either one of these two keys, the address of the player that is currently in would add certain offsets or subtract certain offsets depends on which direction that the user wants the player to move.

Second, when the program is initialized, the user will be asked to input the current time, which will be used as initial value of random number. The initial value of random number determines the initial direction of invaders' movement. After that, instructions will be shown on the scree. The RGB LED will be used to indicate the status of the game. The four LEDs will be used to indicate how many lives that the player has left, and the momentary button will be used to start and to pause the game.

Third, at the beginning of the game, we load the address of the IO0DIR, and store the value of certain color into it, in this case which is going to be white to indicate the game hasn't started yet. We also store the value to the IO0DIR that corresponding to each LED to light up four LEDs. Once the user presses the momentary button, the RGB LED will turn green to show the game starts, and will change color if the status of the game changed, such as pause or end of the game.

Fourth, when the player starts to attack the invader, it would fire a bullet when the user hit the "space" key. We let the bullet move up at a certain speed by performing the same procedure as the invaders movement that was described above, and every moment we compare value of the bullet to the value of the space that's on the ASCII Table to detect if the bullet hits anything. Until it hits something that is not a space, we compare the value to see if it is an invader or a shield. If it is invader, we would write a space to that spot to show that the invader been destroyed. If it is a shield, we would replace "s" if it was "S", or "space" if it is "s".

Fifth, we store the points that the user has earned by shooting the invaders. We also implement the random number generator to the points given by the mothership. After all invaders have been destroyed, there will be a level up.

Sixth, when the timer reaches 2 minutes, the game would end automatically. The total break down components such as points, death counts, and number of motherships been shot, etc, will be shown on the screen. And after that, user will be asked if he wants to start a new game or he doesn't want to continue.

1.10 Usage

Step1: The user will use the PuTTy to run the game. However, before the user open the PuTTy, the user would first search the "Command" in "Start".

Step2: The user then type "mode" in the command and select the highest COM port.

Step3: The user then open the uVision, and set the frequency to be 14.7456M Hz, the User Memory Layout should be checked.

Step4: Build the project, and the program should be ready.

Step5: Open the PuTTy, select Serial and type in the COM port number that was found before and set the baud rate to be 1152000, then hit "OK".

Step6: A blank window would pop out, and the user then can click "load" to load the program. After loading the program, the game would first ask user to input the current time.

Step7: After the user input the current time, the user would see the instruction of how the game would flow.

Step8: The user then push the momentary button to start the game.

Step9: The user use the key "a" or "d" to to move the player to the left or to the right to avoid the invaders' attack. The user can shoot a bullet by pressing the "space" key.

Step10: The user can destroy all the invaders and reach the new level. After 2 minutes or the player lose 4 lives, the user will be abled to see a breakdown report of his performance.

Step11: Press 'c' on the keyboard to start a new game, and press 'q' to quit the game.

2 General Variables' Description

2.1 board

It is our game board's address.
Adding specific offset from the board's starting address, we can change corresponding element.

2.2 board2

It is the initial state of the board.
Used for restart the game and level up.

2.3 initialBlock

It always points to the starting address of the toppest row of monster.
It won't be changed when all_monster_right and all_monster_left are called.
It will be incremented by 32 only after either all_monster_down_on_right or all_monster_down_on_left is called.

2.4 ran_num

It stores random number. At the beginning of the game, user will be asked to input the current time, which will be the initial ran_num value.
Every UART0 input during the game and Timer0 interrupt, ran_num will be randomly changed by adding the Timer1 value.

2.5 score

It stores the current level's score.

2.6 EINT_flag

0 represents the game has not been started.
1 represents the game has been started.
2 represents the game has been paused by the External Interrupt Button (P0.14).
Its value will be changed everytime when user presses the External Interrupt Button (P0.14).

2.7 player_life

It stores the remaining lives of the player.
Its initial value is 4.
It will be reset to initial value only when the user restart new game.

2.8 total_monster

It stores the total number of remaining monsters.
Its initial value is 35.

2.9 level

It stores the current level.
Its initial value is 0.

2.10 level_score

It stores the total score in different level.

2.11 level_time

It stores the total time in different level.

2.12 level_death

It stores the total death count in different level.

2.13 level_mothership

It stores the total number of motherships being shot in different level.

2.14 level_total_mothership

It stores the total number of motherships occured in different level.

2.15 level_bonus

It stores total bonus point on each level.

2.16 loser_report

It is a report format in different level at the end of the game.

2.17 temp_time

It stores total time of that level.
After each level up, it will be reset to 0 and its original value will be stored in level_time.

2.18 totalScore

It is the sum of every level's score

2.19 seg_digit_flag

It ranges from 0 to 3 and represents which digit in 4 7-seg will be illuminated during strobling.

3 Player Variables' Description

3.1 player_current_location

It stores the address of the player ('A').
Its value would incremented by 1 aftre player_right is called.
Its value would decremented by 1 aftre player_left is called.

3.2 player_bullet_location

It stores the address of player's bullet.
Its value will be decremented by 32 every Timer0 Interrupt when player starts attacking till the bullet hits the wall.

3.3 player_attack_flag

0 represents that player is not attacking.
1 represents that player is attacking.

4 Monster Variables' Description

4.1 top_first_monster and top_last_monster

top_first_monster will point to the first monster on the top row.
top_last_monster will point to the last monster on the top row.
Both of them will be incremented by 1 after all_monster_right is called
Both of them will be decremented by 1 after all_monster_left are called.
Both of them will be incremented by 32 after either all_monster_down_on_right or all_monster_down_on_left is called.

4.2 monsterDirFlag

1 means monster is moving to right.
0 means the monster is moving the left.

4.3 monster_moving_flag and monster_moving_time_flag

monster_moving_flag will be incremeneted by 1 every time the Timer0 interrupt till it is equal to monster_moving_time_flag, then it will be reset to 0.
monster_moving_time_flag is initialized to 5 when the game begin. After level up, it will be decremented by 1 till it is equal to 1, then it will stop decrement.

Because Timer0 interrupts every 0.1 sec, therefore, by setting monster_moving_time_flag to specific number, monsters' moving speed can be controlled.

4.4 lowest_row_of_monster

It points the the starting address of the row of the last monster.
It will be decremented by 32 after either all_monster_down_on_right or all_monster_down_on_left is called.
After all monster in last row were killed, lowest_row_of_monster will be incremented by 32.

5 Mothership Variables' Description

5.1 mothership_location

It points the the address of the mothership.
It will be decremented by 1 after mothership_go_left.
It will be incremented by 1 after mothership_go_right.

5.2 mothership_attack_flag

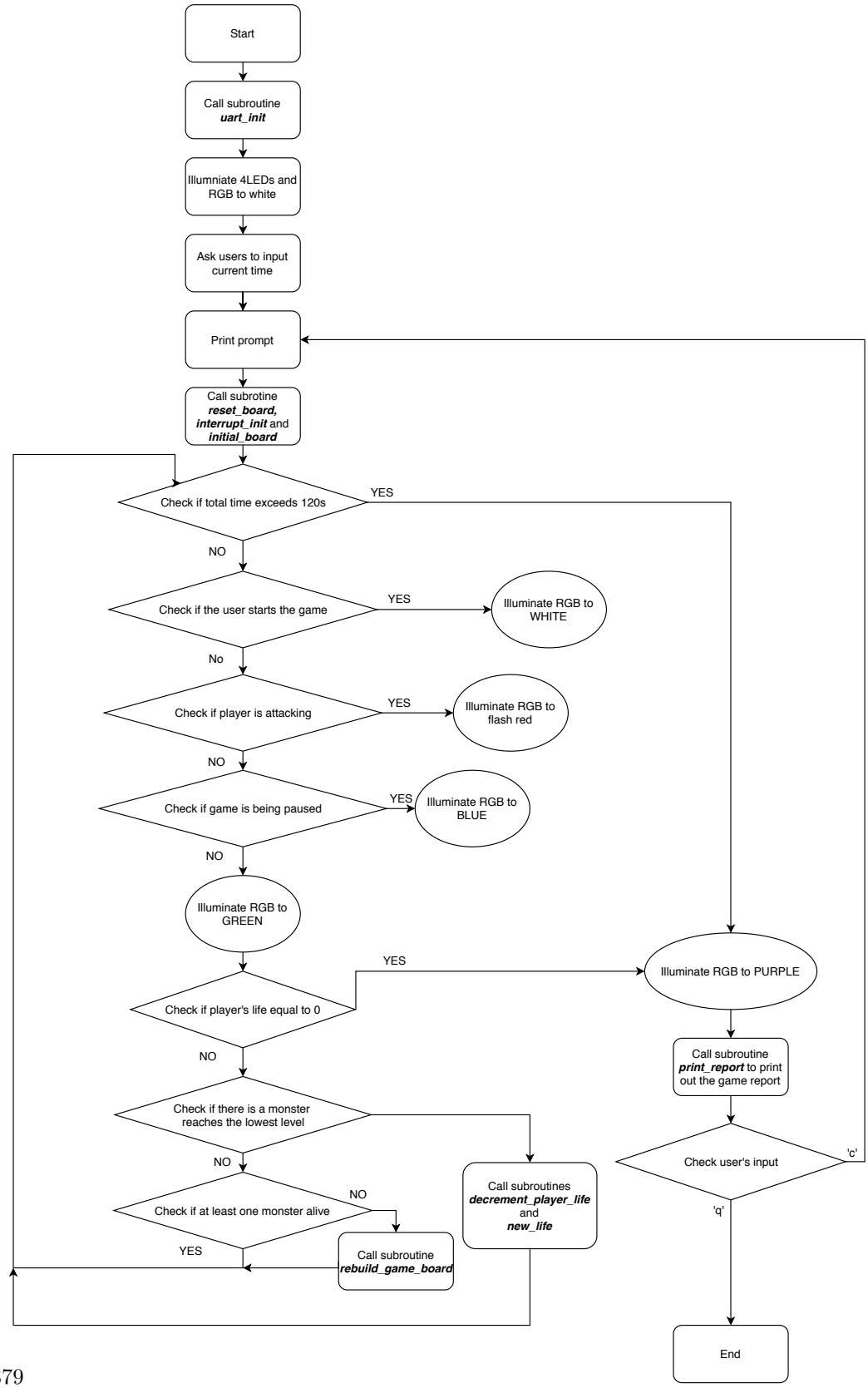
0 represents that there is no mothership.
1 represents that there is one mothership

5.3 mothership_dir_flag

0 represents that the mothership is going left.
1 represents that the mothership is going right.

6 Function Subroutines

6.1 lab5



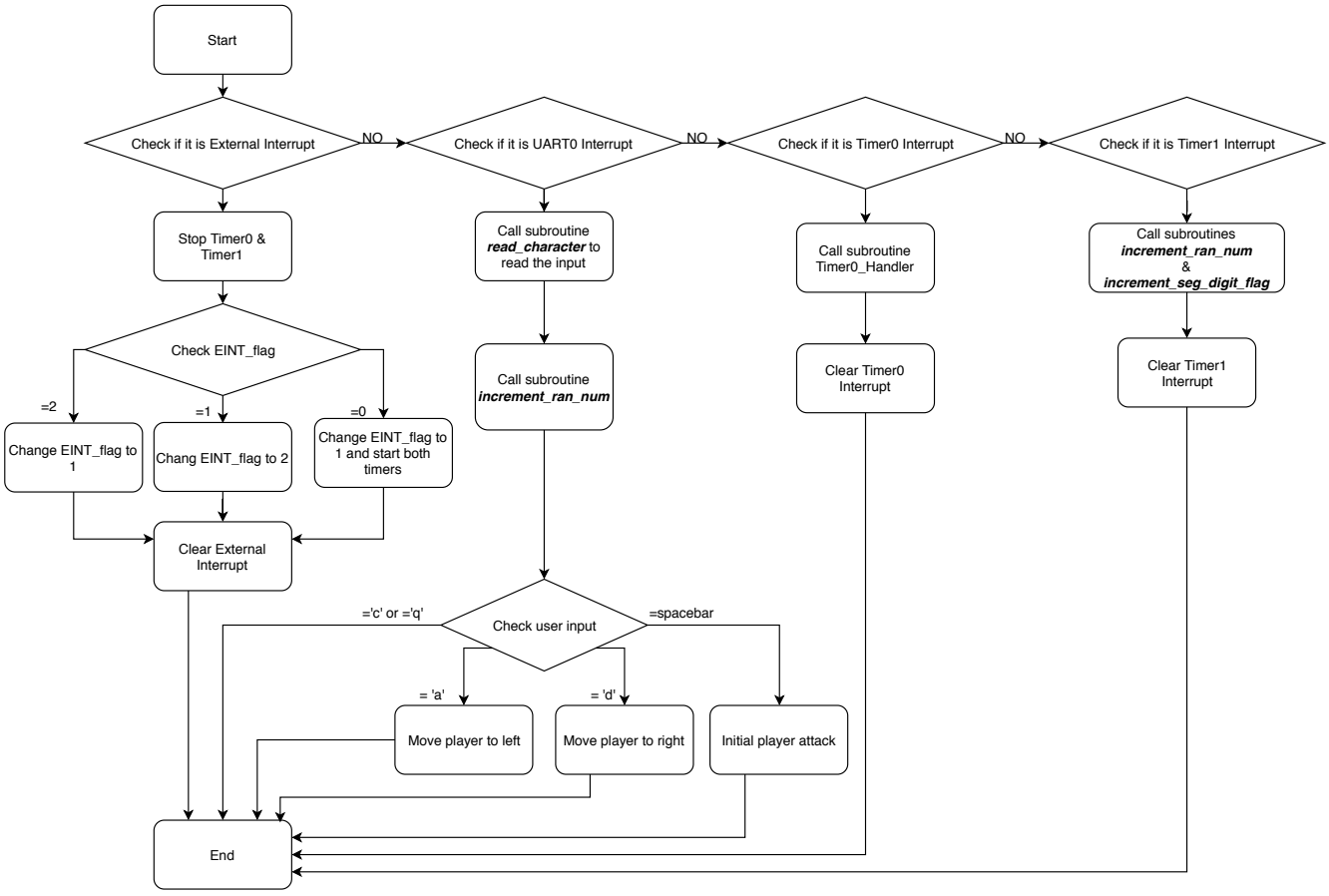
Explanation
lab5 first sets up UART, Interrupt, 7seg components in memory and 7segments display.
Secondly, user will be asked to input the current time as random number initialization.
After the user presses the EINT Interrupt Button, *lab5* will keep looping to check the game's status.

RGB LED will be changed depending on the game status.
- Before game starts, RGB will be illuminated to WHITE color.
- During the game, RGB will be illuminated to GREEN color.
- During the player's attack, RGB will be illuminated to flash RED.
- During the paused period, RGB will be illuminated to BLUE color.
- After the game, RGB will be illuminated to PURPLE color.

Apart from the change of the RGB color, lab5 loop also checks the time, player's life, total monster and level.

After the game is finished, user will be asked if he/she wants to continue to play to quit game.
It will check what user inputs. Except from 'c' and 'q', other key presses are considered invalid data and no next step will be executed until user enters a valid input.
If the user's input is 'c', a new game will be started.
If the user's input is 'q', 'GOODBYE' will be shown on the screen.

6.2 FIQ_Handler



Explanation
FIQ_Handler handlers the interrupts from External Push Button, UART0 Interrupt, Timer0 and Timer1.

1 External Push Botton Interrupt
If there is an interrupt from External Push Button, it will change the EINT_flag.

EINT_ flag = 0 = User has not started the game.
Therefore, Timer0 and Timer1 will be stopped and 7 segment display will be disabled.

EINT_ flag = 1 = User starts the game.
Therefore, Timer0 and Timer1 will be started and 7 segment display will be enabled.

EINT_ flag = 2 = User pauses the game.
Therefore, Timer0 will be stopped.
User can pressed 'p' to resume the game.

2 UART0 Interrupt

When it is UART0 Interrupt, *read_character* will be called first to clear the interrupt.
After that, it will determined whether the user's input is valid or not.

Valid inputs consist of 'a', 'd' and spacebar.
- 'a', *player_left* will be called.
- 'd', *player_right* will be called.
- spacebar, *player_attack_initial* will be called.

When the game is being paused, 'p' will be the only valid input so that user can resume the game.

Every input contains a value according to ASCII table. The value will be added into ran_num to ramdomise the ran_num.

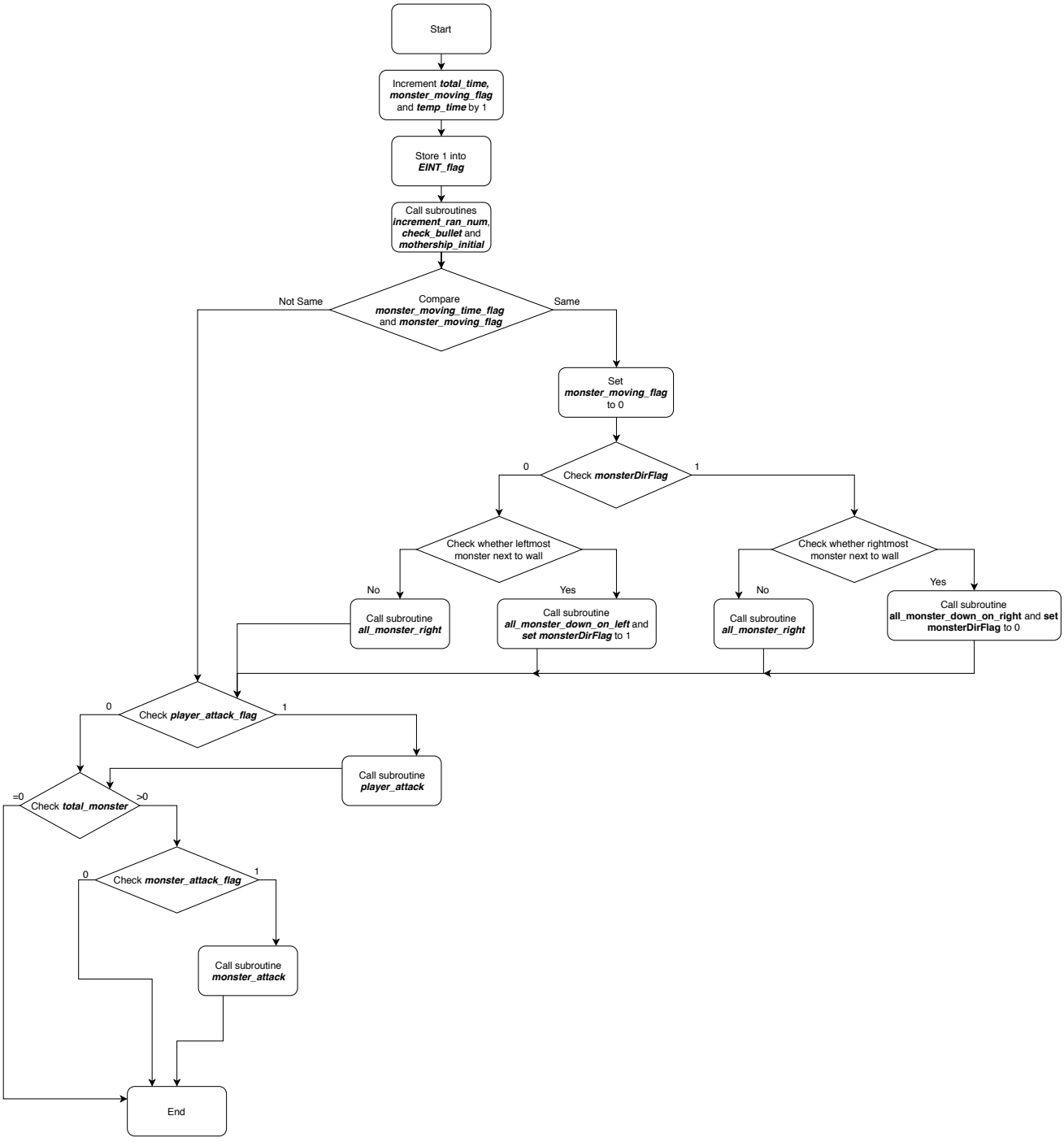
3 Timer0 Interrupt

Timer0_Handler and *increment_ran_num* will be called.

4 Timer1 Interrupt

increment_ran_num and *increment_seg_digit_flag* will be called.

6.3 Timer0_Handler



Explanation
Timer0_Handler services when Timer0 interrupts, including monsters' movement, monster's bullet, player's bullet and motherhsip attack.

1 Monsters' Movement
For the monsters' movement, it will first check whether the monster_moveing_time_flag equals to monster_moving_flag. If equal, it will determines the direction of the monster movement and call corresponding subroutines.

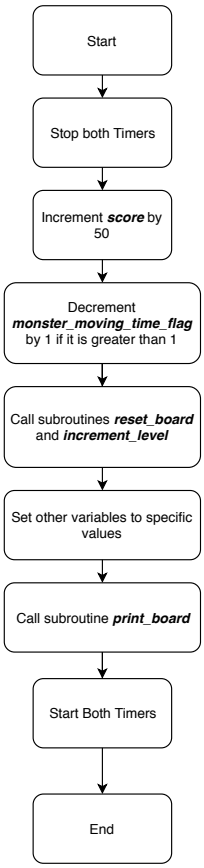
2 Monsters' Bullet
There are actually three stages for the monsters' bullet, including *monster_attack_initial, monster_attack and clean_monster_bullet.*
-*monster_attack_initial* initializes the monster bullet randomly.
-*clean_monster_bullet* will be called before monsters' movement and monster_attack. Detail of it will be explained in corresponding subroutine section.
-*monster_attack* will be called after monsters' movement. Detail of it will be explained corresponding subroutine section.

3 Player's Bullet
There are two stages for player's bullet, which are *clean_player_bullet* and *player_attack*
-*clean_player_bullet* will be called before any monsters' movement and *player_attack.*
-Detail of *clean_player_bullet* and *player_attack* will be explained in corresponding subroutine section.

4 Mothership's Attack
There are 5 subroutines regarding to motherhship's attack, including *mothership_initial*, *mothership_initial_from_right*, *mothership_initial_from_left*, *mothership_go_right* and *mothership_go_left.*

mothership_initial will be called every time.

6.4 rebuild_game_board



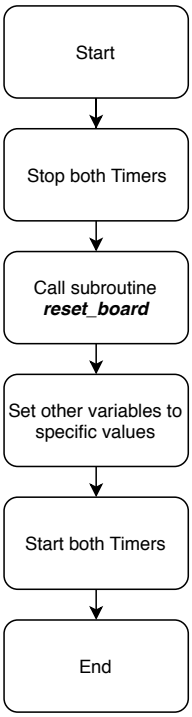
Explanation
rebuild_game_board is called after new level reached.

It first stops both timers.
Score will be incremented by 50 points as bonus and will be updated to level_score.
Then, it decrements the monster_moving_time_flag by 1 till it equals to 1.
reset_board and increment_level will be called.

Some variables will be set to specific values as list below.
- temp_time = 0
- level = level + 1
- initialBlock, top_first_monster, top_last_monster, lowest_row_of_monster to initial value
- monster_attack_flag = 0
- mothership_attack_flag = 0
- total_monster = 35
- player_attack_flag = 0
- monsterDirFlag = random integer between 0 and 1, inclusively.

After that, both timers will be started.

6.5 new_life

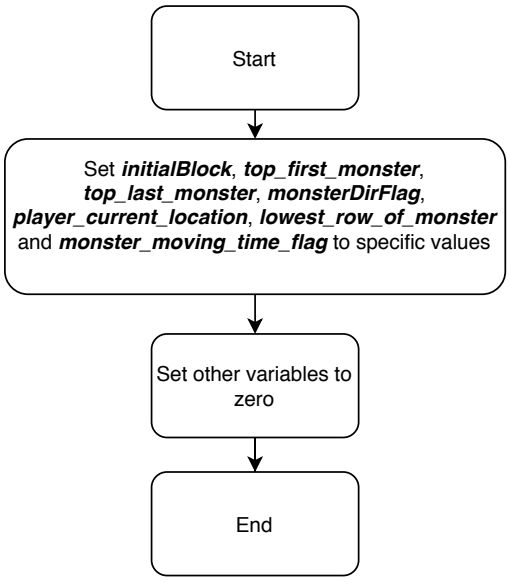


Explanation
new_life is called after monster arrives at deadline.(one row above the Shield, indicated by two arrows in the game board)

It first stops both timers.
reset_board will be called.
Some variables will be set to specific values as list below.
- player_life = player_life - 1
- initialBlock, top_first_monster, top_last_monster, lowest_row_of_monster to initial value
- monster_attack_flag = 0
- mothership_attack_flag = 0
- total_monster = 35
- player_attack_flag = 0
- monsterDirFlag = random integer between 0 and 1, inclusively.

After that, both timers will be started.

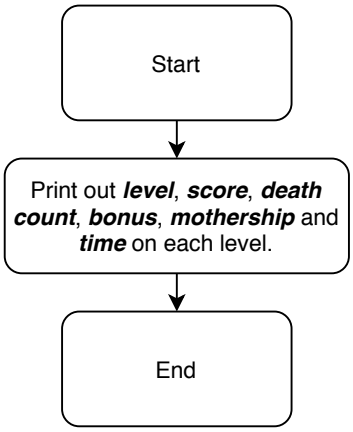
6.6 initial_board



Explanation
This subroutine sets up the basic variables for the game.
Below are all initial values of variables.
- level_bonus = 0
- level_total_mothership = 0
- total_time = 0
- totalScore = 0
- initialBlock = board + 68
- top_first_monster = board + 75
- top_last_monster = board + 89
- monsterDirFlag = 0 or 1
- player_current_location = board + 494
- player_attack_flag = 0
- monster_moving_flag = 0
- lowest_row_of_monster = initialBlock + 128
- monster_attack_flag = 0
- seg_digit_flag = 0
- mothership_attack_flag = 0
- EINT_flag = 0
- player_life = 4
- total_monster = 35
- monster_moving_time_flag = 5
- level = 0
- temp_time = 0
- level_death = 0
- level_mothership = 0

If variables does not on the list above, variables will be initialized after.

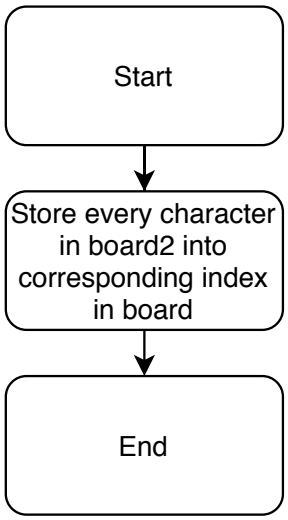
6.7 print_report



Explanation
Level (*): Score:(*)(*)(*)(*)—Timer:(*)(*)(*)s—DeathCount:(*)—Mothership:(*)—Bonus Point:(*)(*)(*)(*)—Total Mothership passed:(*)(*)
Above is the report format.

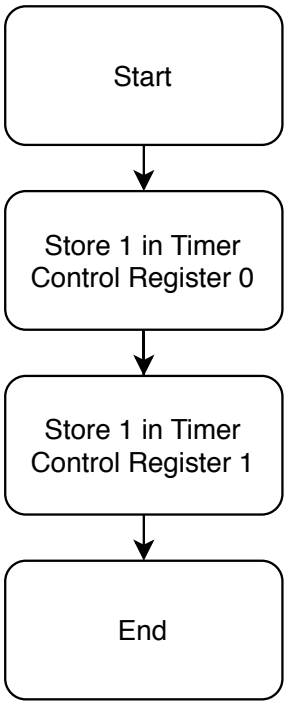
All (*) will be replaced by integer, ranging from 0 to 9.
For every (*), *div_and_mod* will be called repeatedly to get integer in corresponding places.
After printing out the breakdown of each level, it will print out the total score.

6.8 reset_board



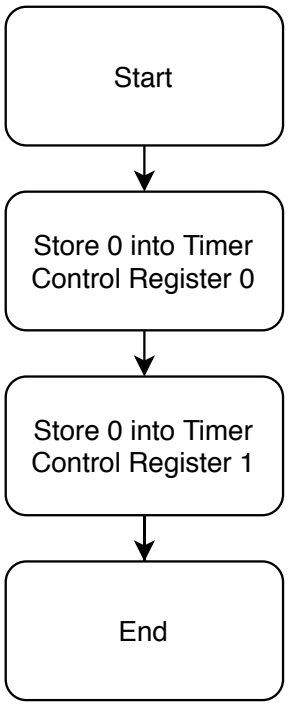
Explanation
It stores every character in board2 into corresponding spot in board.

6.9 start_timer



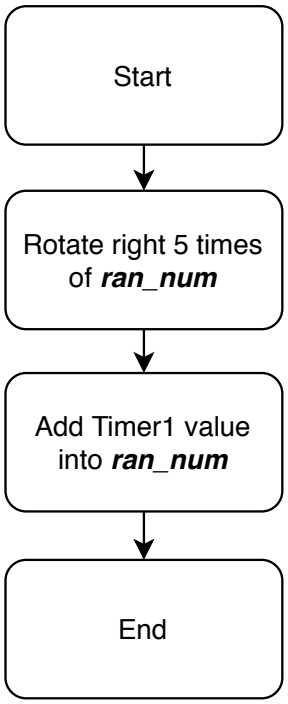
Explanation
It starts both timers.

6.10 stop_timer



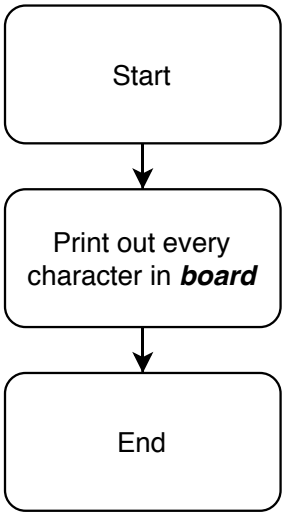
Explanation
It stops both timers.

6.11 increment_ran_num



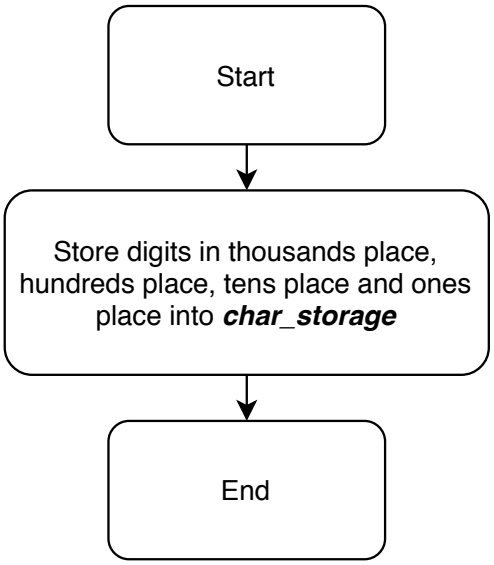
Explanation
It will be called at different time during the game.
Right rotate ran_num 5 times.
Then, it will get the Timer1's value and adds it to ran_num.

6.12 print_board



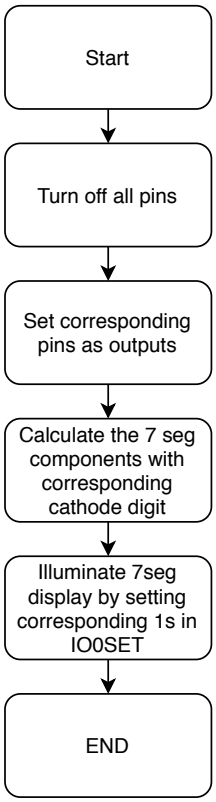
Explanation
It prints out every character in the board to PuTTy.

6.13 update_to_char_storage



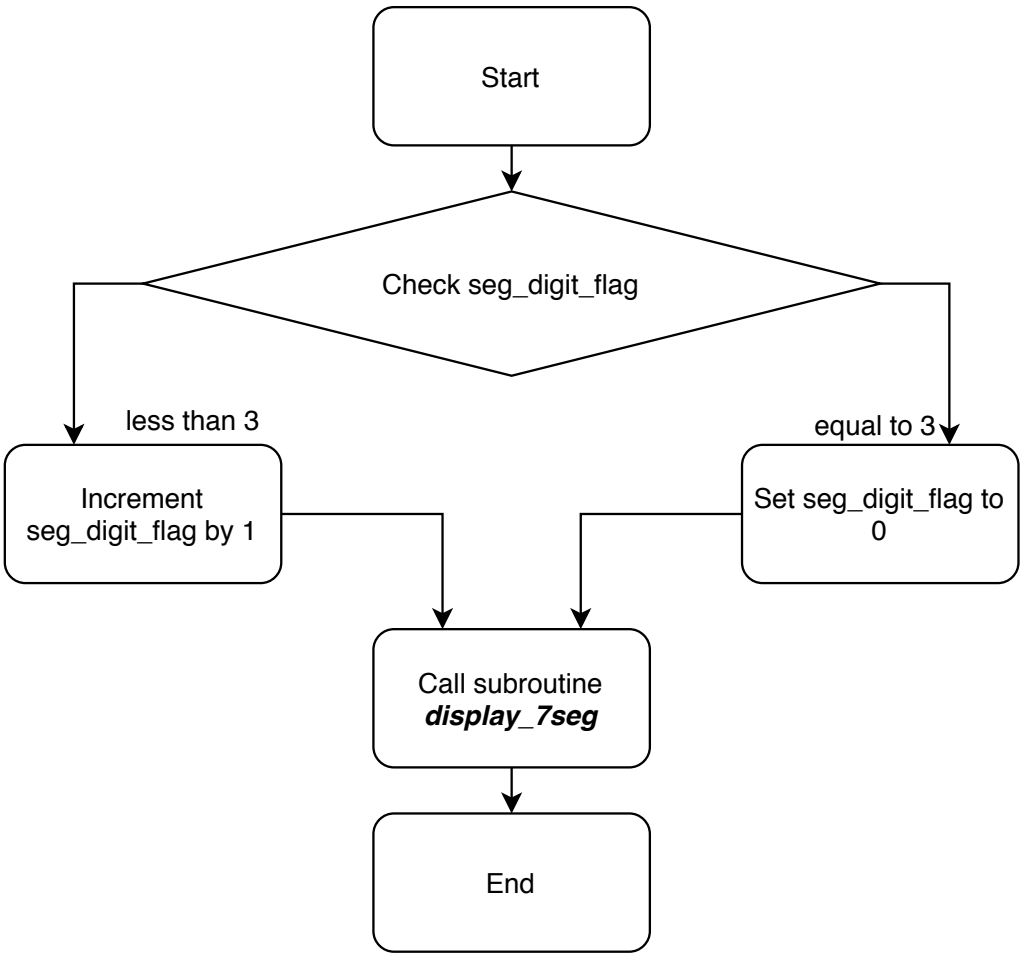
Explanation
By calling *div_and_mod*, it gets thousand, hundred, ten and one place of the score.
It updates four of them to char storage for later 7seg display.

6.14 display_7seg



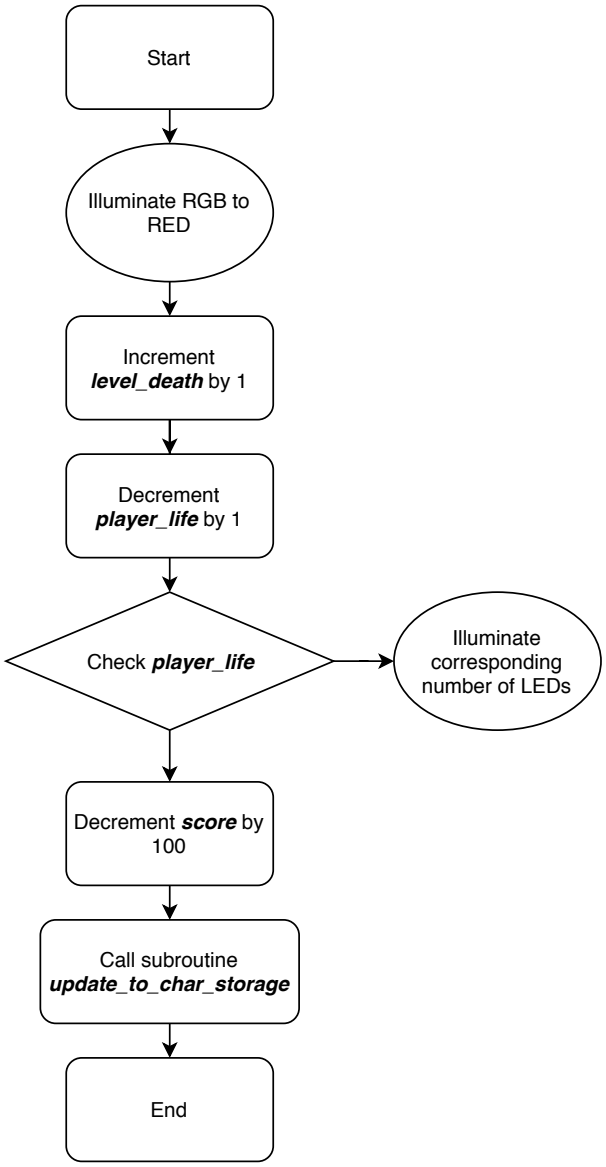
Explanation
It will be called everytime when there is a Timer1 Interrupt.
It will receives which digits of 7seg should be illuminated by accessing the seg_digit_flag.
After that, it accesses the char_storage to access what integer will be illuminated on 7seg.
By adding offset * 4 + the starting address (memFor7seg) to get the component.
Storing that component into IO0SET will illuminated our desired number on 7seg.

6.15 increment_seg_digit_flag



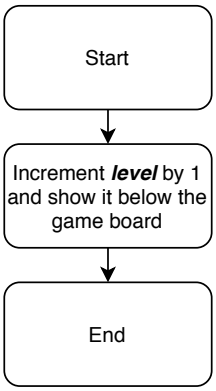
Explanation
It will be called everytime when there is Timer1 Interrupt.
It incremented the seg_digit_flag by 1 everytime till it reaches 4, then resets to 0 immediately so that *display_7seg* knows which digit of 7seg should be illuminated.

6.16 decrement_player_life



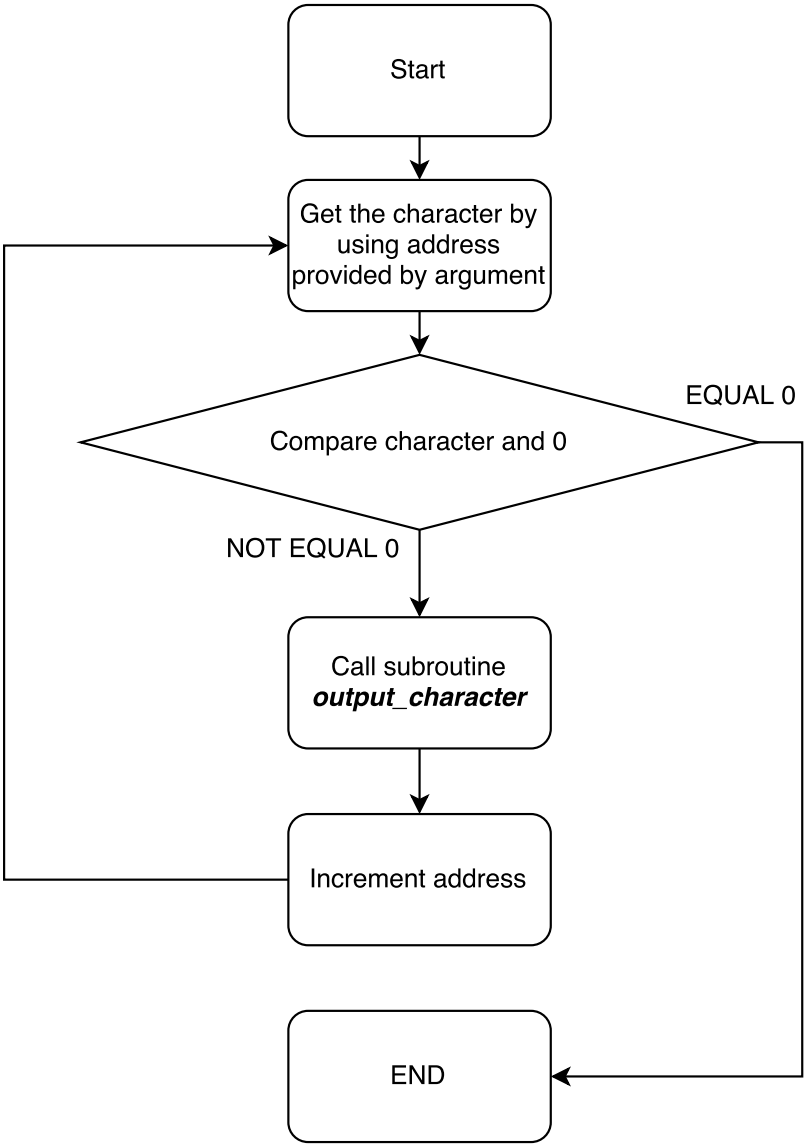
Explanation
It will be called when player loses life.

6.17 increment_level



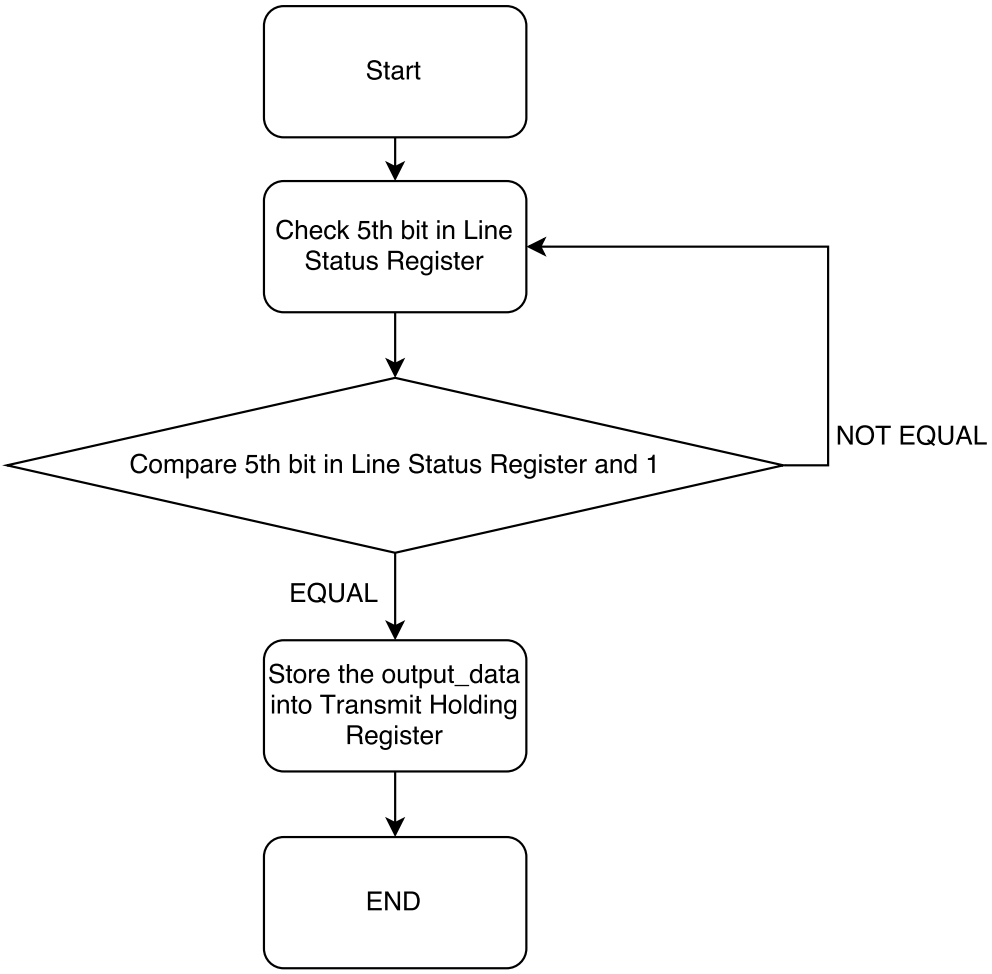
Explanation
It increments the level.

6.18 output_string



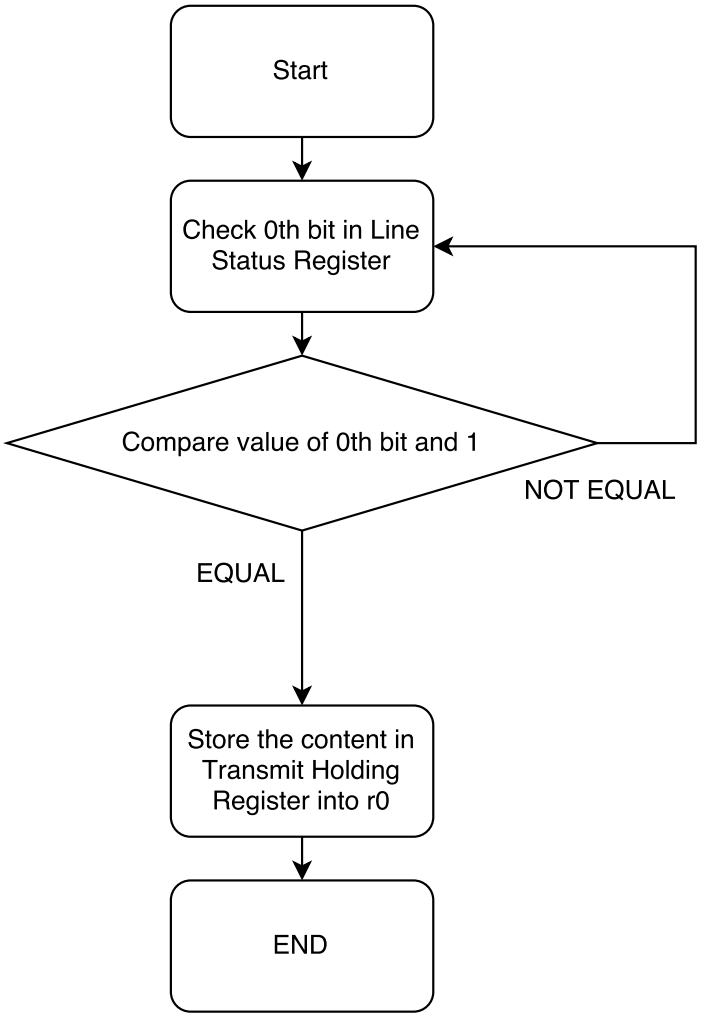
Explanation
It receives the starting address of the string.
It prints out every character of that string on PuTTy till it finds the null-terminator.

6.19 output_character



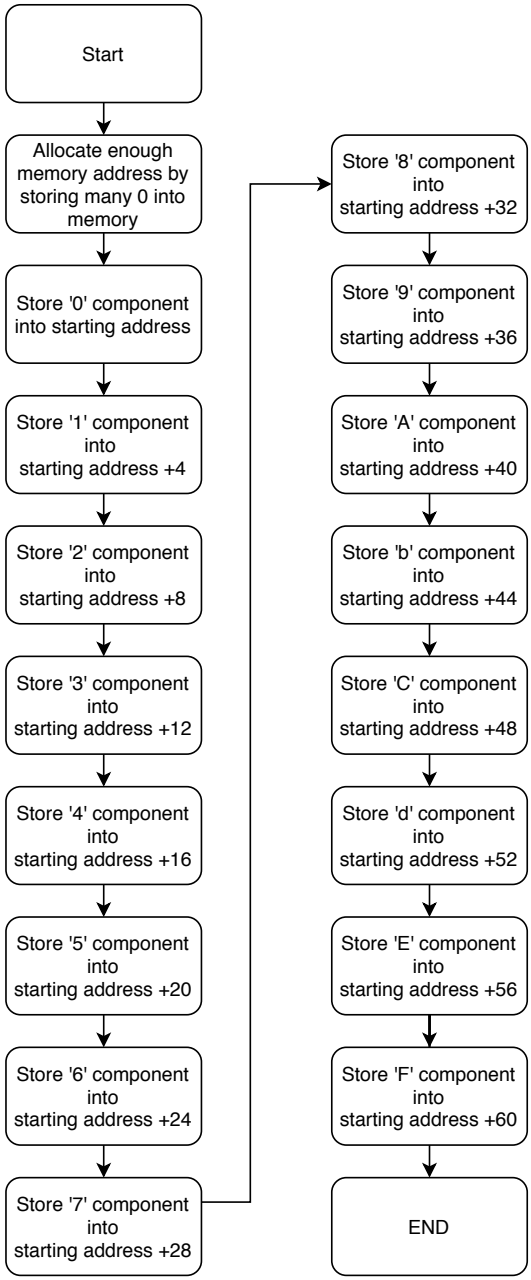
Explanation
It outputs a character to PuTTy.

6.20 read_character



Explanation
Read the input character from PuTTy and store it in r0.

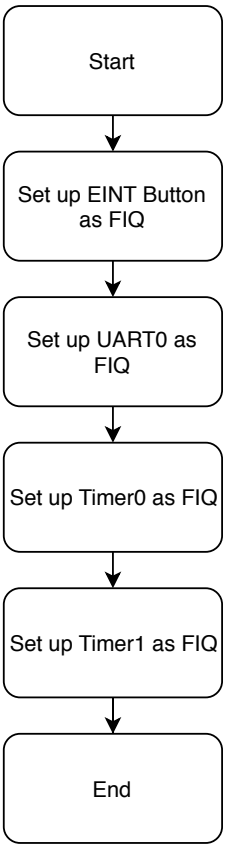
6.21 storeDataIntoMem



Explanation
The purpose of *storeDataIntoMem* is to store component value of hex into memory.

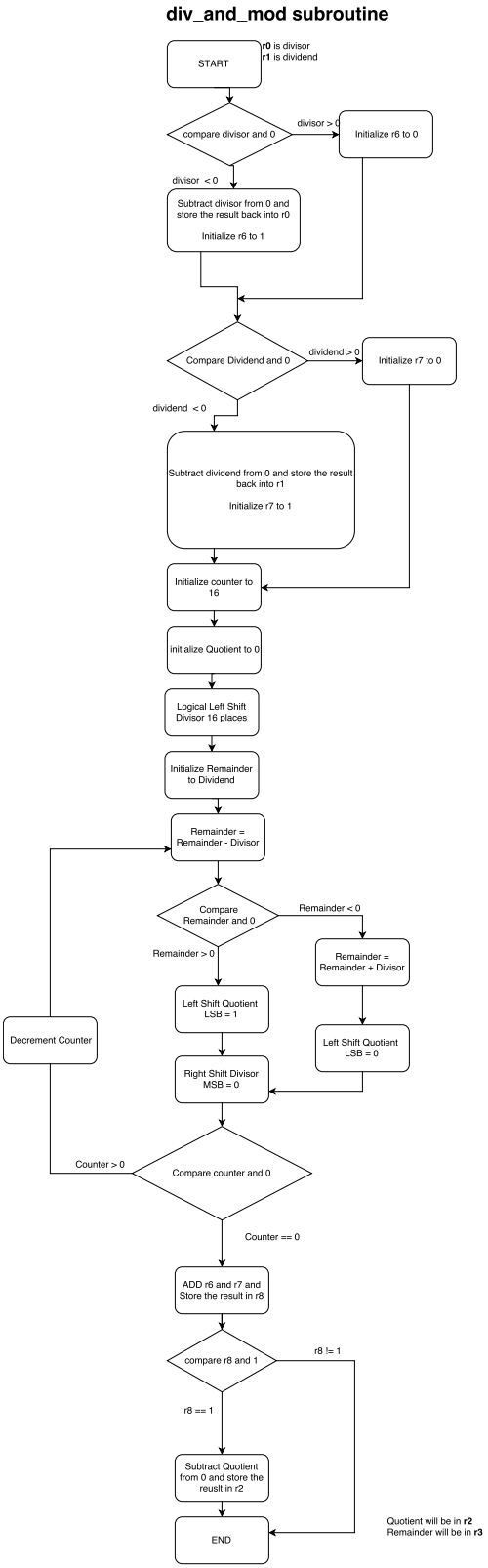
The *storeDataIntoMem* will store 17 strings, which consists of component value from 0-9 and A-F in order. After receiving the read_data, the effective address can be calculated by starting address + 4 x read_data. Component value will be stored into Set Reg to illuminate the correct hex value on 7-seg. It is better than 15 comparison in tern of efficiency.

6.22 interrupt_init



Explanation
Set up fast interrupts for External Interrupt, Timer0, Timer1 and UART0.

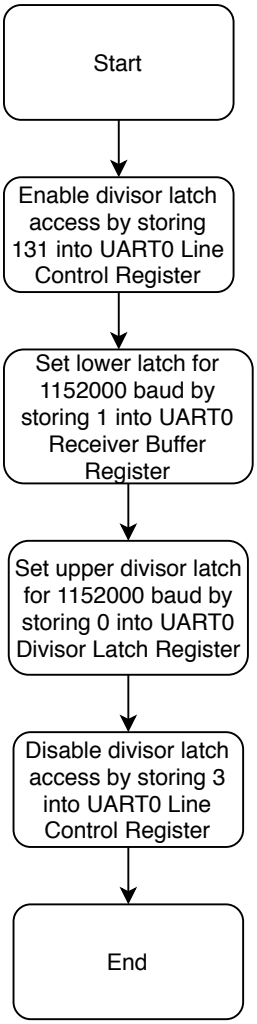
6.23 div_and_mod



Explanation
- r0 is divisor.
- r1 is dividend.
- r2 is quotient.
- r3 is remainder.

It can handle signed division.

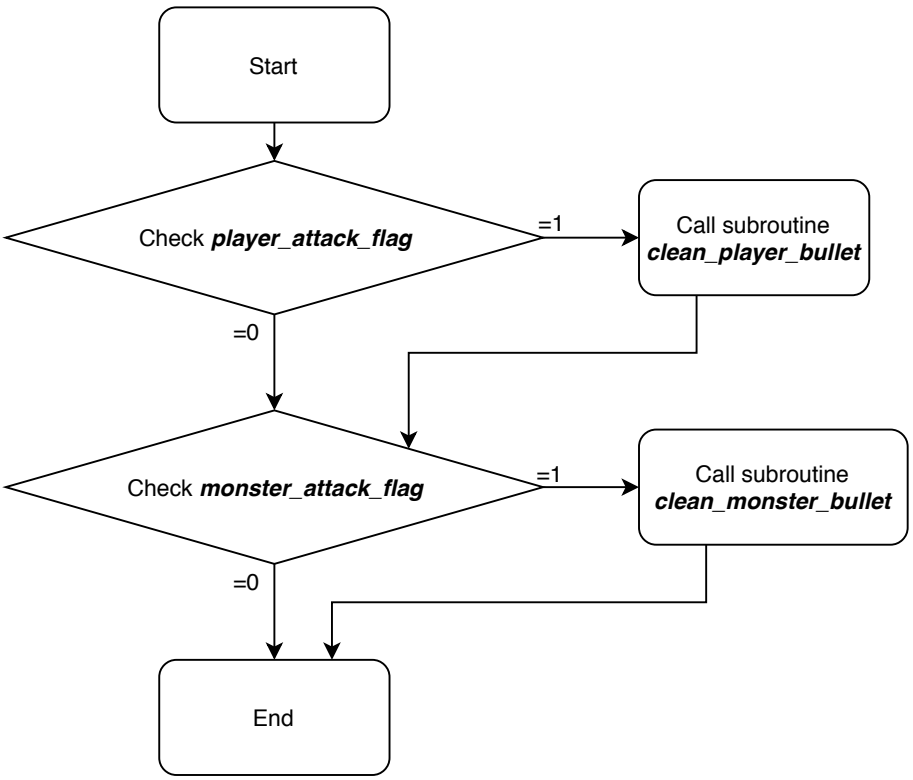
6.24 uart_init



Explanation
It sets the baud rate to 1152000.

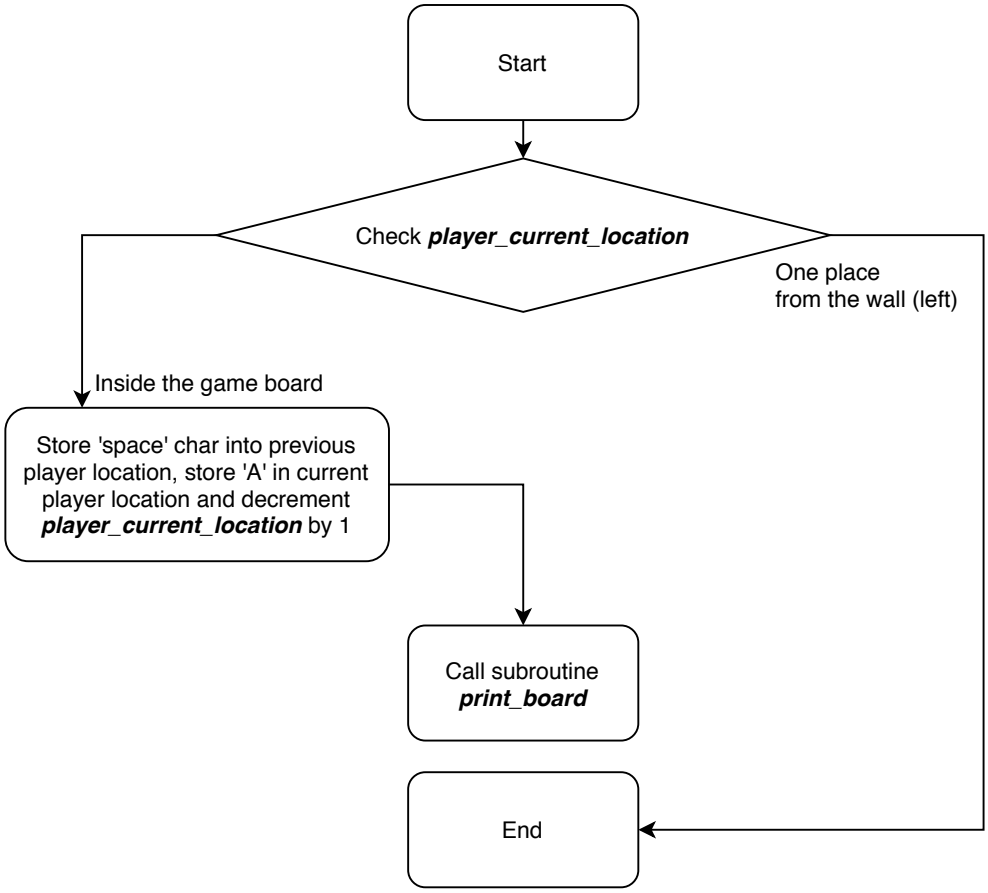
7 Player's Subroutines

7.1 check_bullet



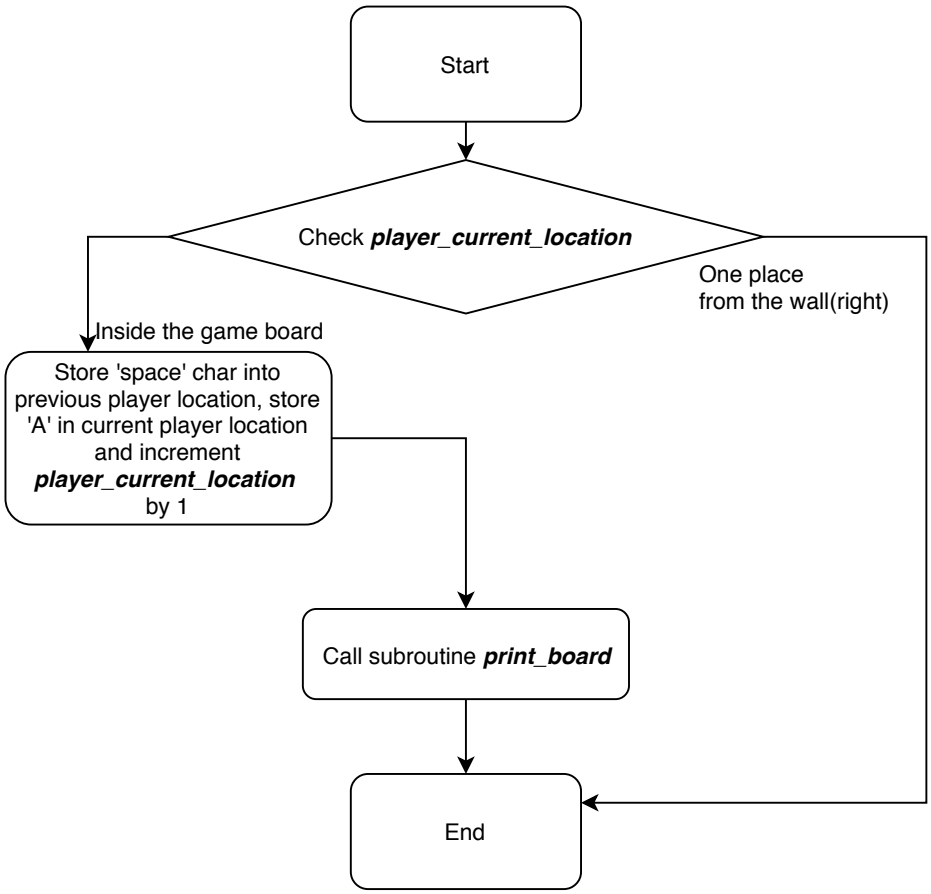
Explanation
Graphically, it will remove the char on bullet location. But because of other variables, for example player_bullet_location and monster_bullet_location, their locations can still be tracked down easily.
The reason that clean_bullet is not in the same subroutine as player_attack or monster_attack is that it will avoid bullets being moved to some strange location.

7.2 player_left



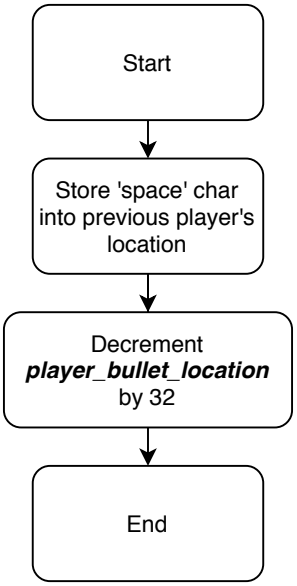
Explanation
Player goes to left by one spot.

7.3 player_right



Explanation
Player goes to right by one spot.

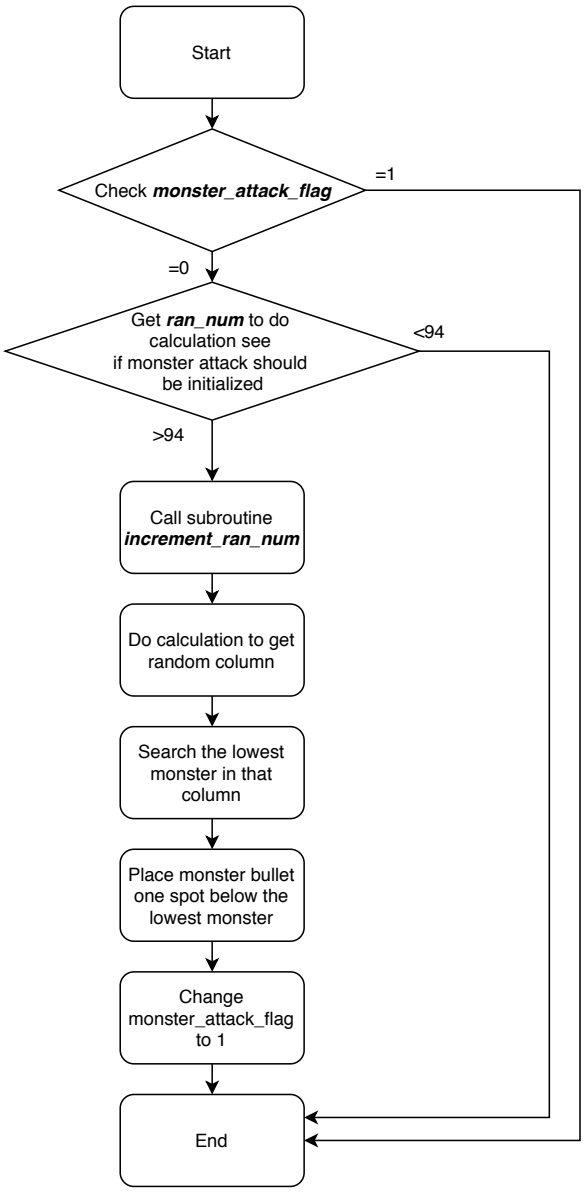
7.4 clean_player_bullet



Explanation
Place a 'spacebar' in player_bullet_location.
Decrement player_bullet_location by 32.

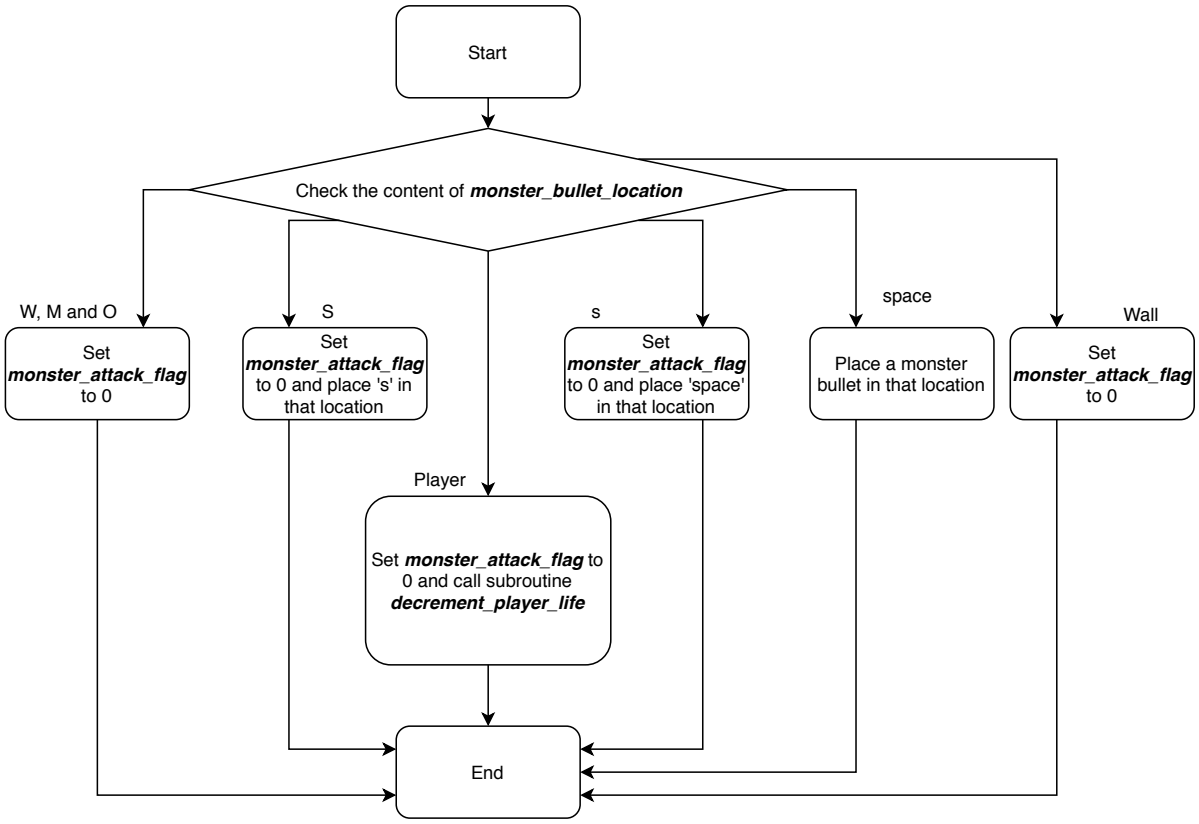
8 Monsters' Subroutines

8.1 monster_attack_initial



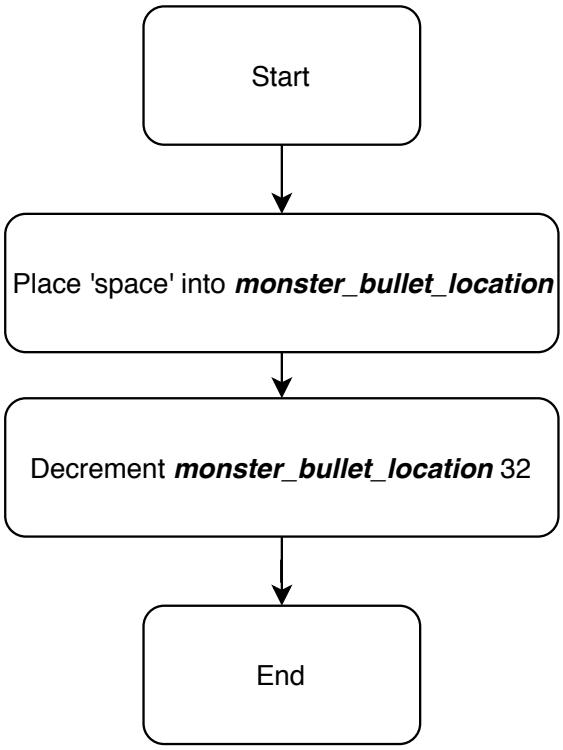
Explanation
monster_attack_initial will be called everytime when there is a Timer0 Interrupt.
If the remainder of ran_num mod 100 is greater than 94 with monster_attack_flag equals to 0, monster attack will be initialized.
After that, randomise the ran_num. Then, remainder of ran_num mod number of column equals to which column of monster will shot a bullet.
Afterward, find the lowest monster in that column, place the bullet under the lowest monster of that column and change monster_attack_flag to 1.

8.2 monster_attack



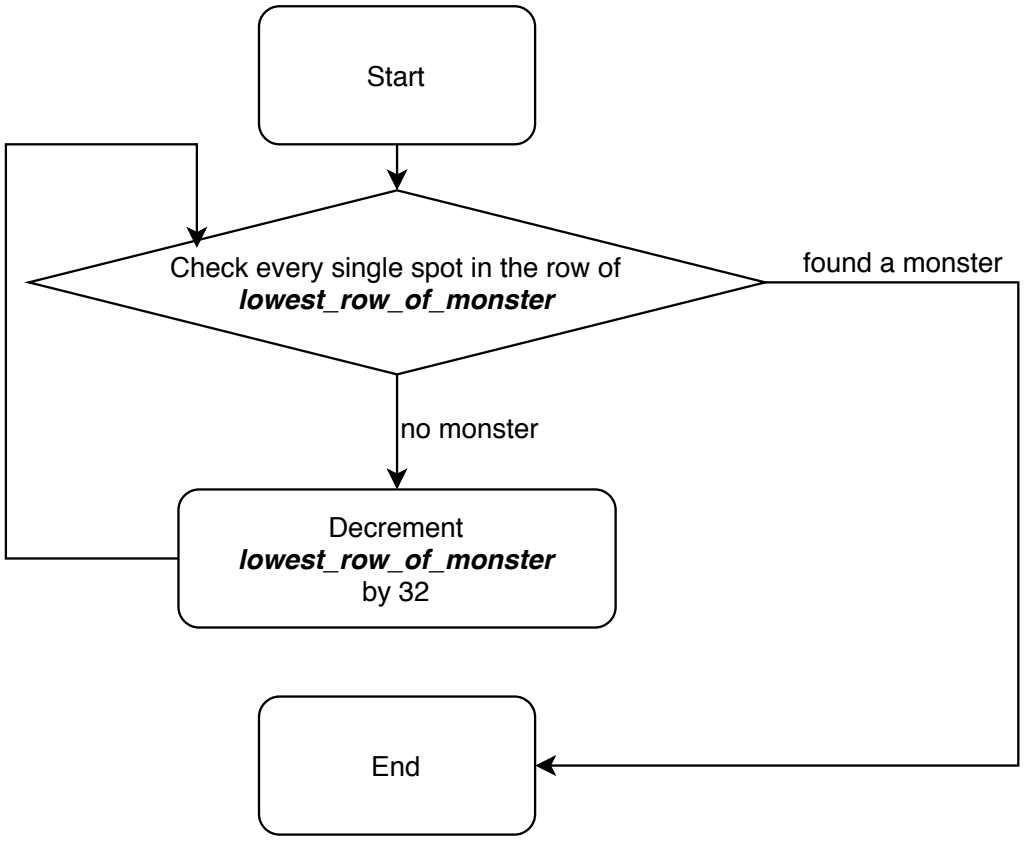
Explanation
Whether monster bullet continues or not depends on different situation.
Once the monster_bullet_location contains char different than spacebar, monster bullet attack will be terminated and will randomly generates next attack.
-W,M and O, if monster bullet hits a monster, it will stop the monster bullet attack and the monster being shot will remain intact.
- S, if monster bullet hits a strong shield, monster bullet attack will be terminated and the shield status will turn into weak shield, denoted by 's'
- s, if monster bullet hits a weak shield, monster bullet attack will be terminated and the shield will disappear and it will be replaced by a spacebar.
- spacebar, if monster bullet hits spacebar, everthing remains unchanged.
- wall, if monster bullet hits wall, monster bullet attack will be terminated.

8.3 clean_monster_bullet



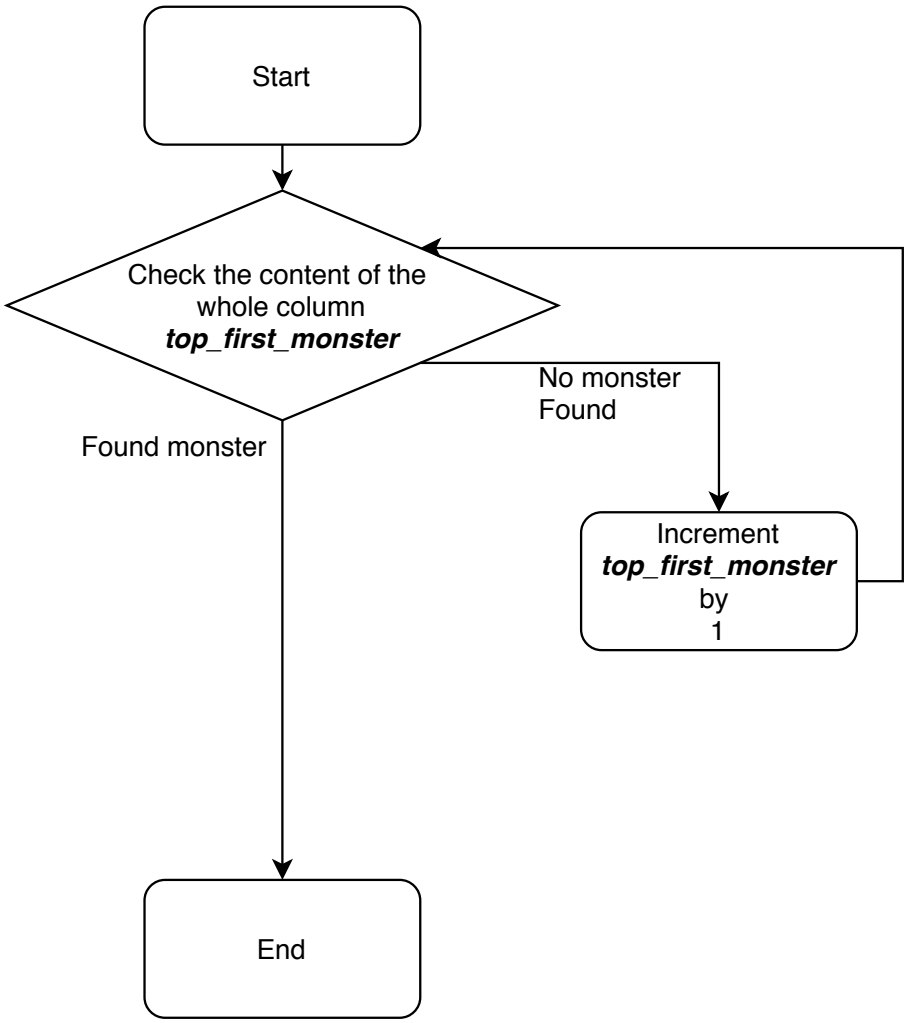
Explanation
It clears monster bullet to avoid monster bullet being moved to other spot during monsters' movement. Since there is monster_bullet_location, we can still track down on the monster bullet location easily.

8.4 check_lowest_monster



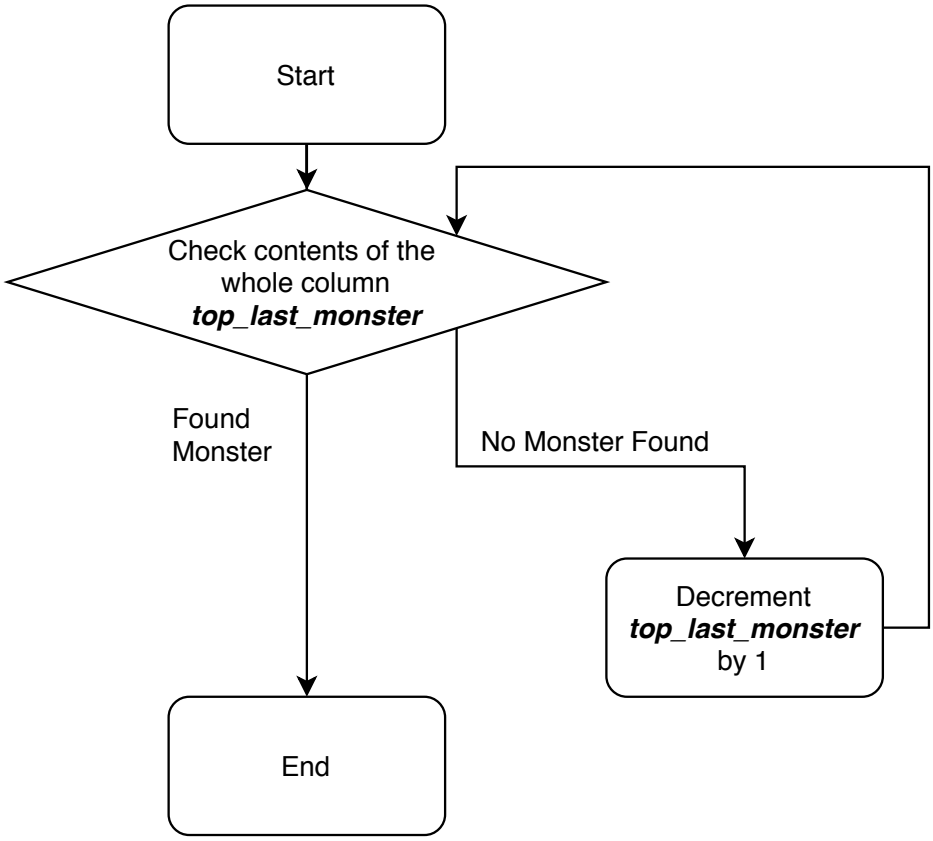
Explanation
With the lowest_row_of_monster, it is easy to tell which row might contain the lowest monster.
Because of unpredictable situation after monster being shot by player's bullet, therefore, it is necessary to check the lowest row of the monster in every Timer0 Interrupt.

8.5 check_first_column



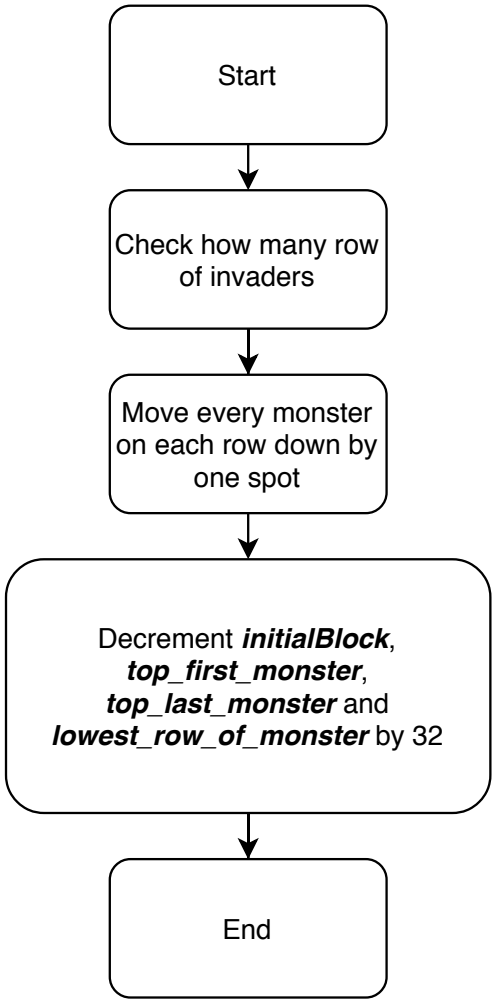
Explanation
Because of unpredictable situation after monster being shot by player's bullet, therefore, it is necessary to check the first column of the monster in every Timer0 Interrupt.
If all monster in first column were shot, monsters can actually move one more spot to left.
Hence, it is import to keep check of first column of the monster.

8.6 check_last_column



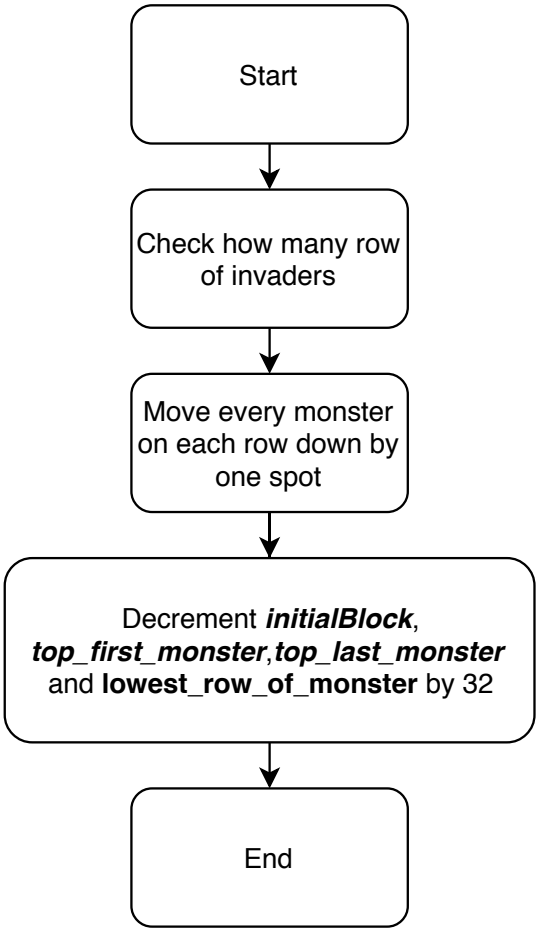
Explanation
Because of unpredictable situation after monster being shot by player's bullet, therefore, it is necessary to check the last column of the monster in every Timer0 Interrupt.
If all monster in last column were shot, monsters can actually move one more spot to right.
Hence, it is import to keep check of last column of the monster.

8.7 all_monster_down_on_left



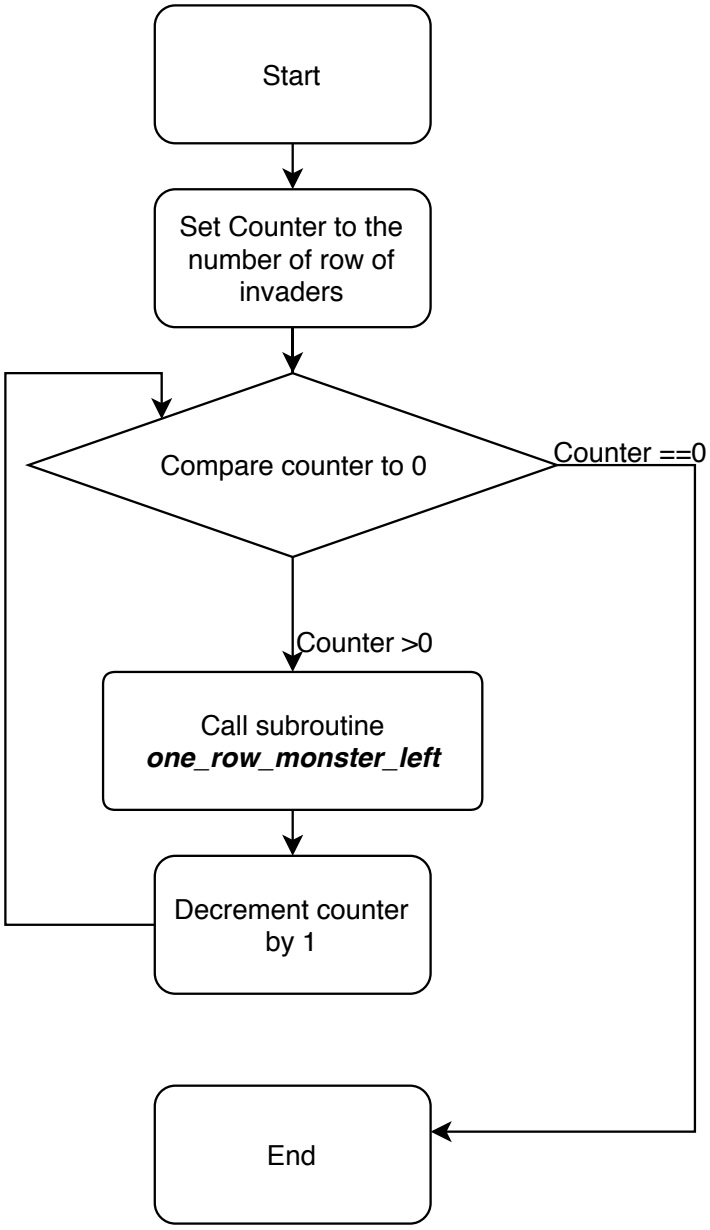
Explanation
When the first monster column is just next to the left wall, which means there is no more space for moving left, therefore, monsters have to go down on left.

8.8 all_monster_down_on_right



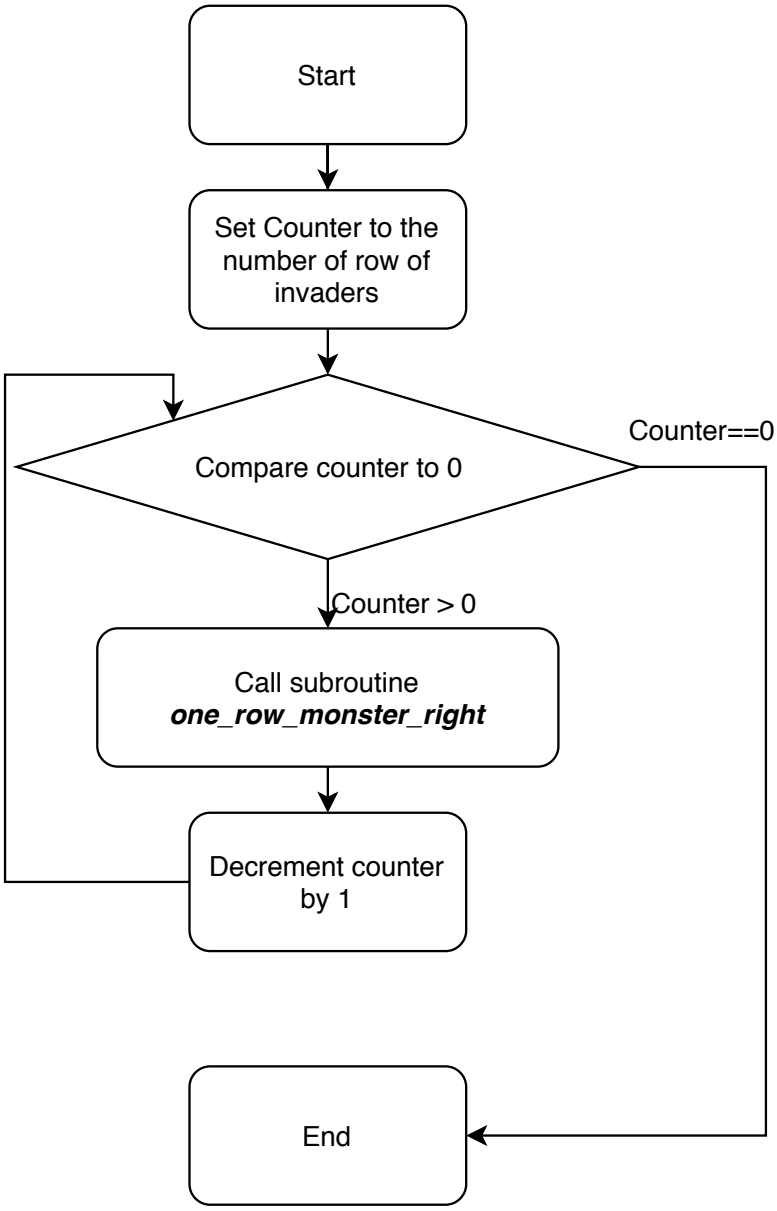
Explanation
When the last monster column is just next to the right wall, which means there is no more space for moving right, therefore, monsters have to go down on right.

8.9 all_monster_left



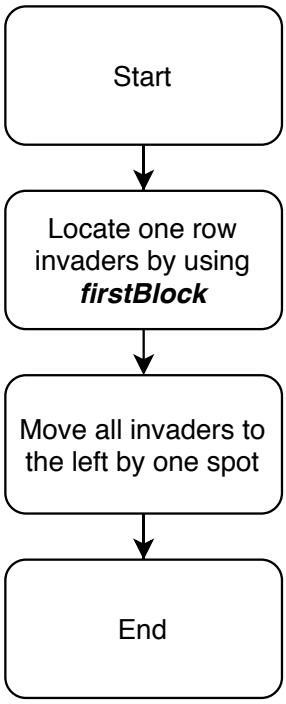
Explanation
It will keep calling one_row_monster_left until the last row is called.

8.10 all_monster_right



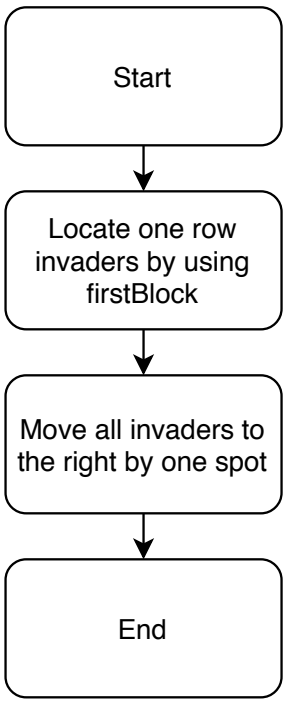
Explanation
It will keep calling one_row_monster_right until the last row is called.

8.11 one_row_monster_left



Explanation
The subroutine is provided the starting address of the each row. then it shifts every element in that row to left by one spot.

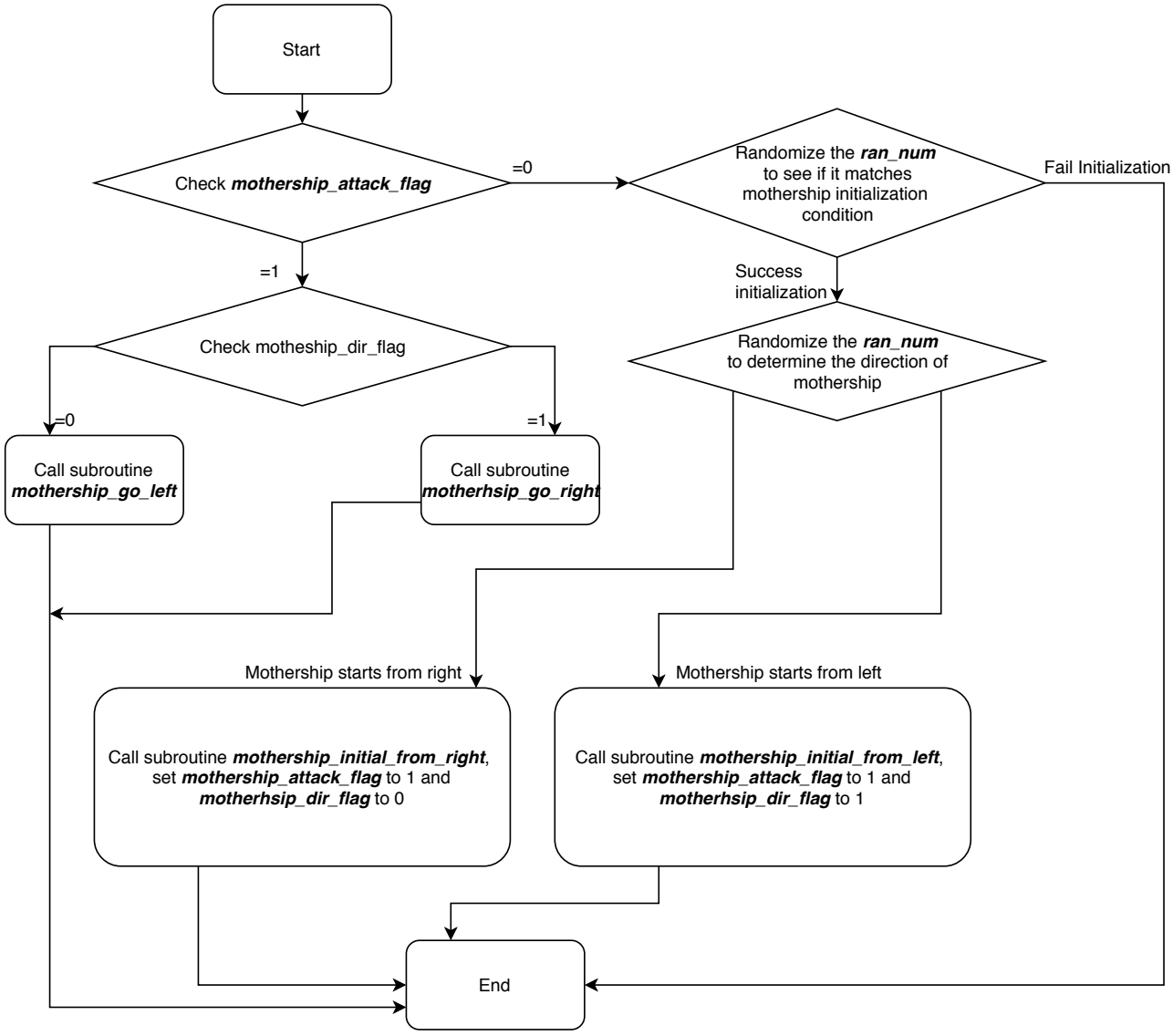
8.12 one_row_monster_right



Explanation
The subroutine is provided the starting address of the each row. then it shifts every element in that row to right by one spot.

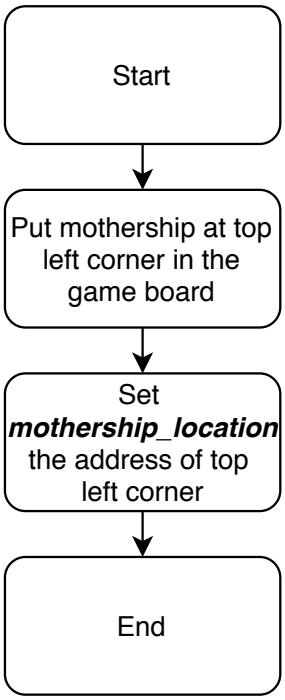
9 Mothership's Subroutines

9.1 mothership_initial



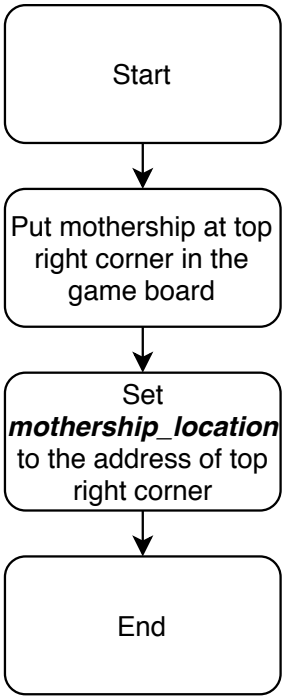
Explanation
It first checks the mothership_attack_flag. If there is no mothership attacking, it will generates an mothership attack initialization attempt by the ran_num. If success, it will generate direction of the mothership randomly and initialize mothership attack.

9.2 mothership_initial_from_left



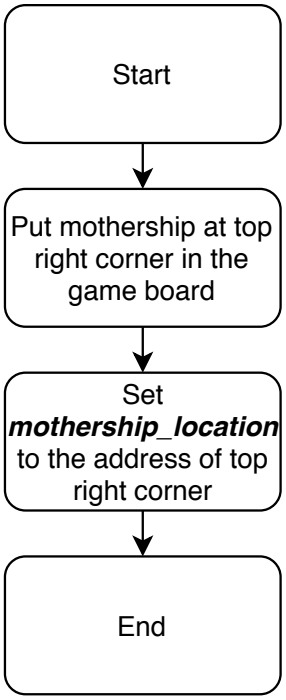
Explanation
Mothership will be generated from the left side of the wall.

9.3 mothership_initial_from_right



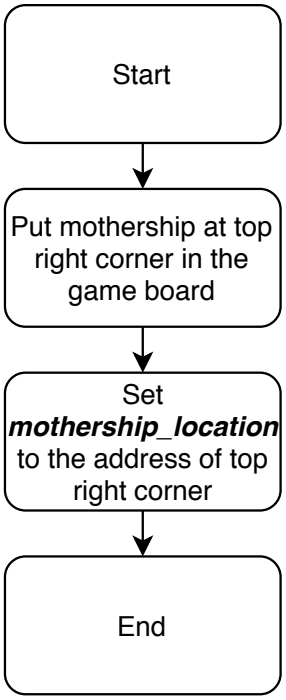
Explanation
Mothership will be generated from the right side of the wall.

9.4 mothership_go_left



Explanation
Mothership will be generated from the right side of the wall.

9.5 mothership_go_right



Explanation
Mothership will be generated from the right side of the wall.