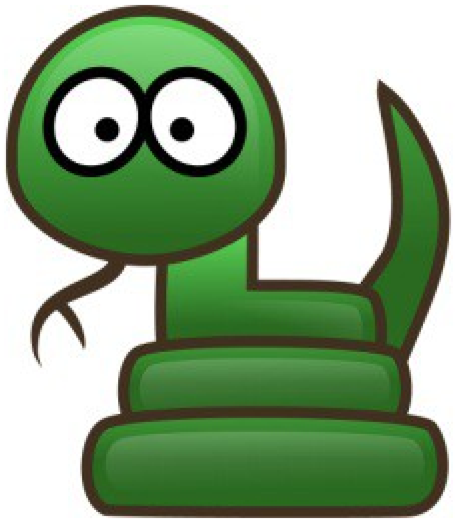
MICROCONTROLLER PROJECT:
SNAKE GAME



PROFESSOR: Lutz Leutelt

Participants: Caus Danu, Taimoor Ahmed, Richard Jansohn

Our snake game has the following features:

- **3 states: start state, play mode/state, game over mode/state**
- **snake grows when eating the fruit**
- **game over when there is a wall collision**
- **game over when there is a self collision**
- **game over when you move left and you change direction to right (or vice-versa), or when you move up and you change direction to down (or vice-versa)(because it is treated like the head overwrites the next body block)**
- **Fruit is generated randomly on the screen(using a time seed for the random function generator)**
- **Fruit is not generated in the snake body**

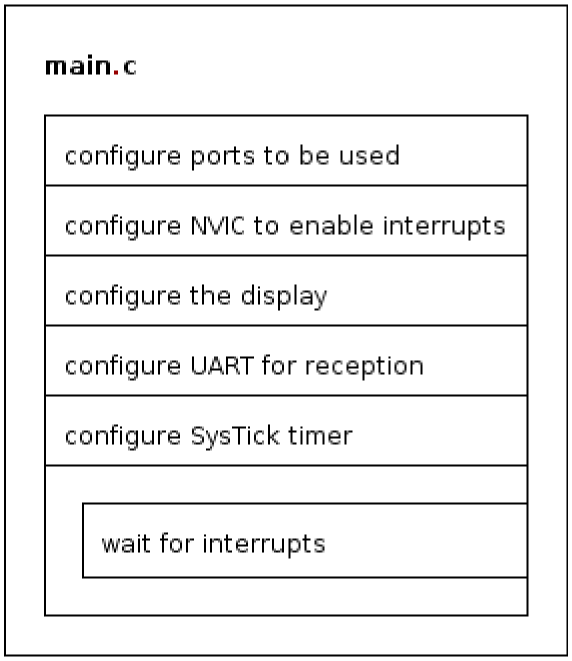
UART and Interrupts:

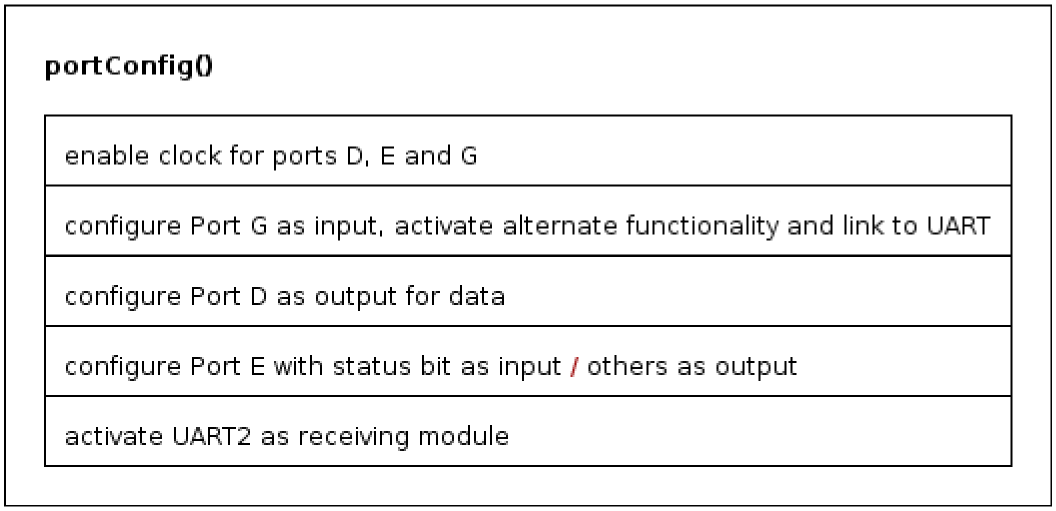
The UART configuration at first seemed straight-forward, but later it was hard to debug :). It needed to receive information from the keyboard and the program would decide based on the key pressed, which direction the snake moves next. The four keys we decided to use to provide the required direction input were 'a', 's', 'd' and 'w', as used in most PC-games on the market to provide a comfortable user experience. After configuring the UART2 port, the need arose to set up the interrupt registers of the Stellaris Microcontroller.

The Interrupts required were for the SysTick Timer and for the UART. The Timer is configured in the NVICConfig() method, setting the SysTick Reload Value in register NVIC_ST_RELOAD, as well as enabling it using the interrupt masks NVIC_ST_CTRL_ENABLE and NVIC_ST_CTRL_INTEN. Then the UART interrupt is enabled, setting interrupt number 33 in the Interrupt Vector Table. This UART interrupt is linked to PortG(0) in the portConfig() method from “sysconfig.c”.
At Last, both of these interrupts also need to be inserted in the corresponding line in the “startup_ccs.c” file .

The header file to make these interrupts take effect on the program: “InterruptHandlers.h” contains the global variable ”newChar” to store the keyboard character received by the UART. Once this value is registered and a new key is pressed, the old character is overwritten. Also needed are the interrupt methods contained in “InterruptHandlers.c”, namely timerIntHandler() and UARTIntHandler(). InterruptHandlers.h file is included in the snake source file to be able to use movesnake() in the Timer interrupt handler or to be able to read the keyboard characters stored in “newChar” when changing direction.

NSDs for main and portConfig() :


main.c
configure ports to be used
configure NVIC to enable interrupts
configure the display
configure UART for reception
configure SysTick timer
wait for interrupts


portConfig()
enable clock for ports D, E and G
configure Port G as input, activate alternate functionality and link to UART
configure Port D as output for data
configure Port E with status bit as input / others as output
activate UART2 as receiving module

Graphics:

All the global variables:

```
int SNAKE_SIZE = 5; //Global variable that stores the size of the snake (initial size being 5 by default)
unsigned char screenmatrix[128][30] = {0x00}; // screen matrix to be printed at each timer interrupt

//Point data structure is declared in "snake.h"
Point snake[100]={{0,0},{1,0},{2,0},{3,0},{4,0}}; // array storing the coordinates of the snake blocks
Point Fruit; // global variable to store the coordinates of the fruit
```

Point data structure (defined in "snake.h"):

```
typedef struct Point {
        int x;  // horizontal coordinate (to account for the horizontal movement)
        int y;  // vertical coordinate (to account for the vertical movement)
} Point;
```

For the game graphics we created a global matrix called "screenmatrix" with width 30 and height 128. In this matrix we insert 8x8 blocks using a special function called insertBlock(int posx, int posy,unsigned char p[128][30]), where p is the screenmatrix in all of our cases when we use this function.

To keep track of the snake blocks we created a global array of Point structures. A Point structure is nothing else than a pair of x and y coordinates. These coordinates represent the top left corner of the block which will be inserted in the screenmatrix.

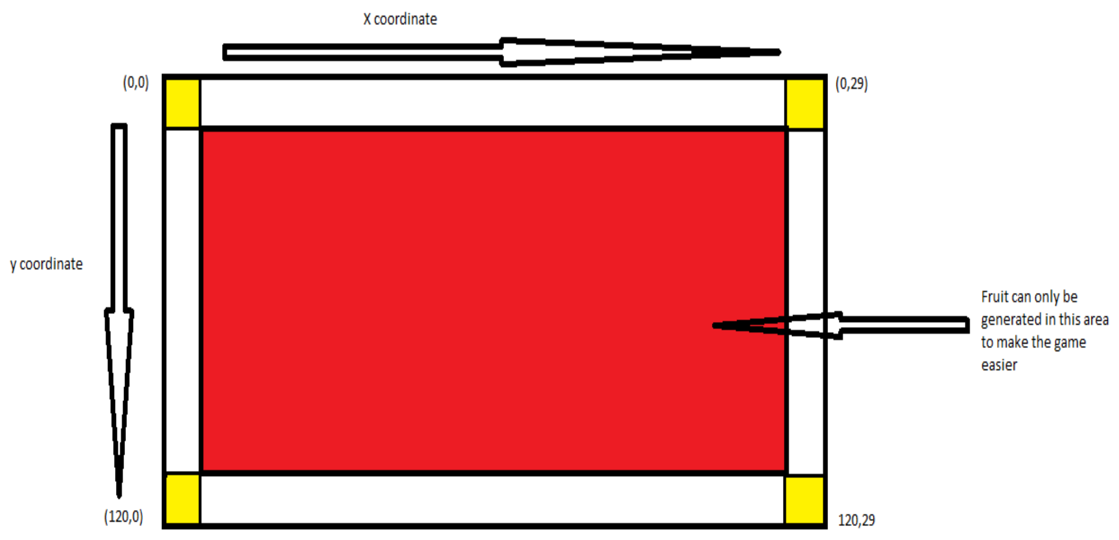
Fruit is also a global Point structure with an x and y coordinate.

There is a special function used to insert all the snake blocks and the fruit into the global screenmatrix called "insertSnakeAndFruit()".

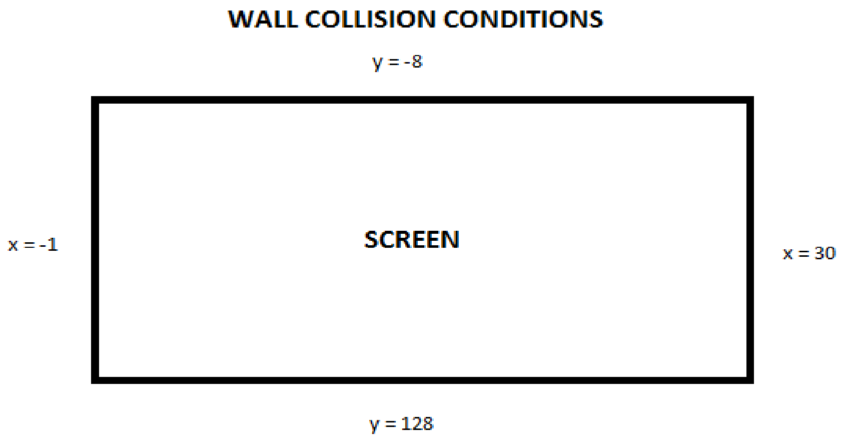
There is also a function "printGO()" that inserts blocks in such a way so that we see "GO" written on the screen.

"printOUCH()" inserts blocks in such a way so that we see "OUCH!" on the screen

"generateFruit()" generates the random coordinates of the Fruit. These random coordinates are within a certain range so that we do not generate the fruit right on the screen margin. This was a matter of personal preference.

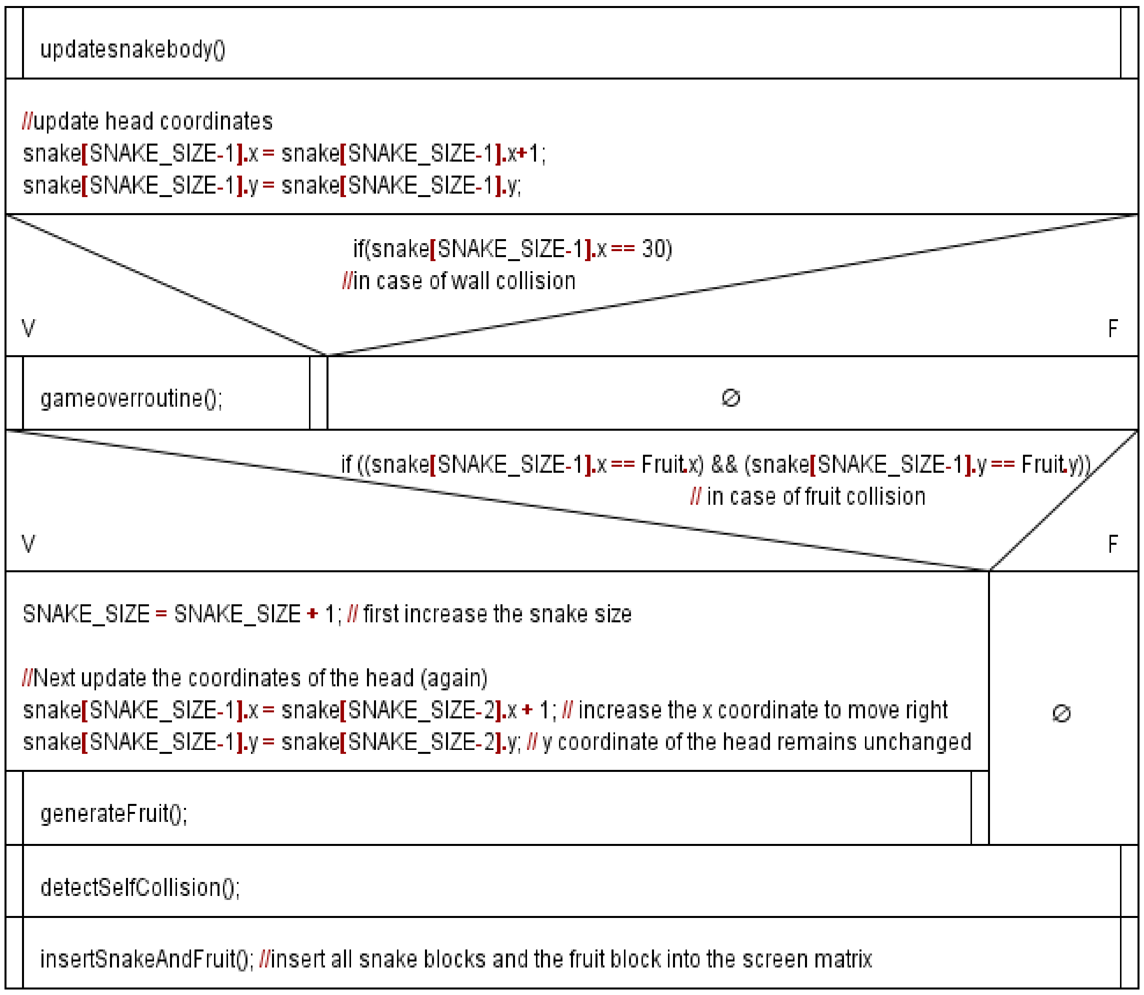

X coordinate
(0,0)
(0,29)
y coordinate
Fruit can only be generated in this area to make the game easier
(120,0)
120,29

In the moveright(), moveleft(), moveup() and movedown() functions we check for wall collisions. Below are presented the conditions to be checked in each of the functions respectively.



NSD of the moveright() function. The other functions have similar logic, the only difference being the head coordinate values and wall collision conditions

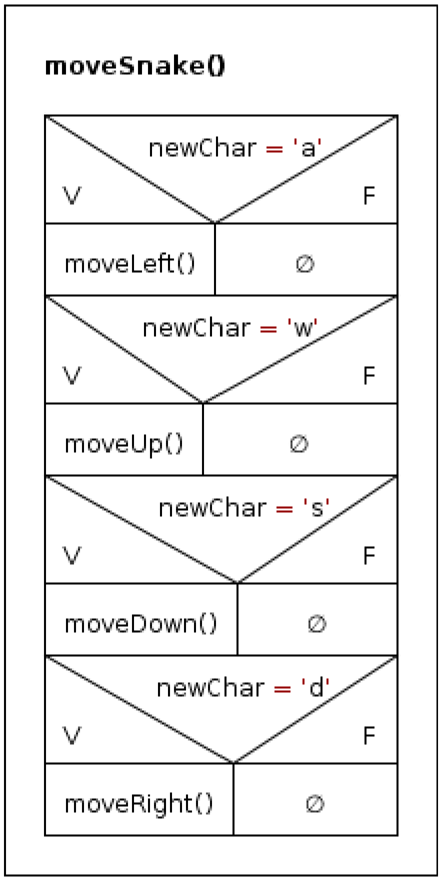
moveright()



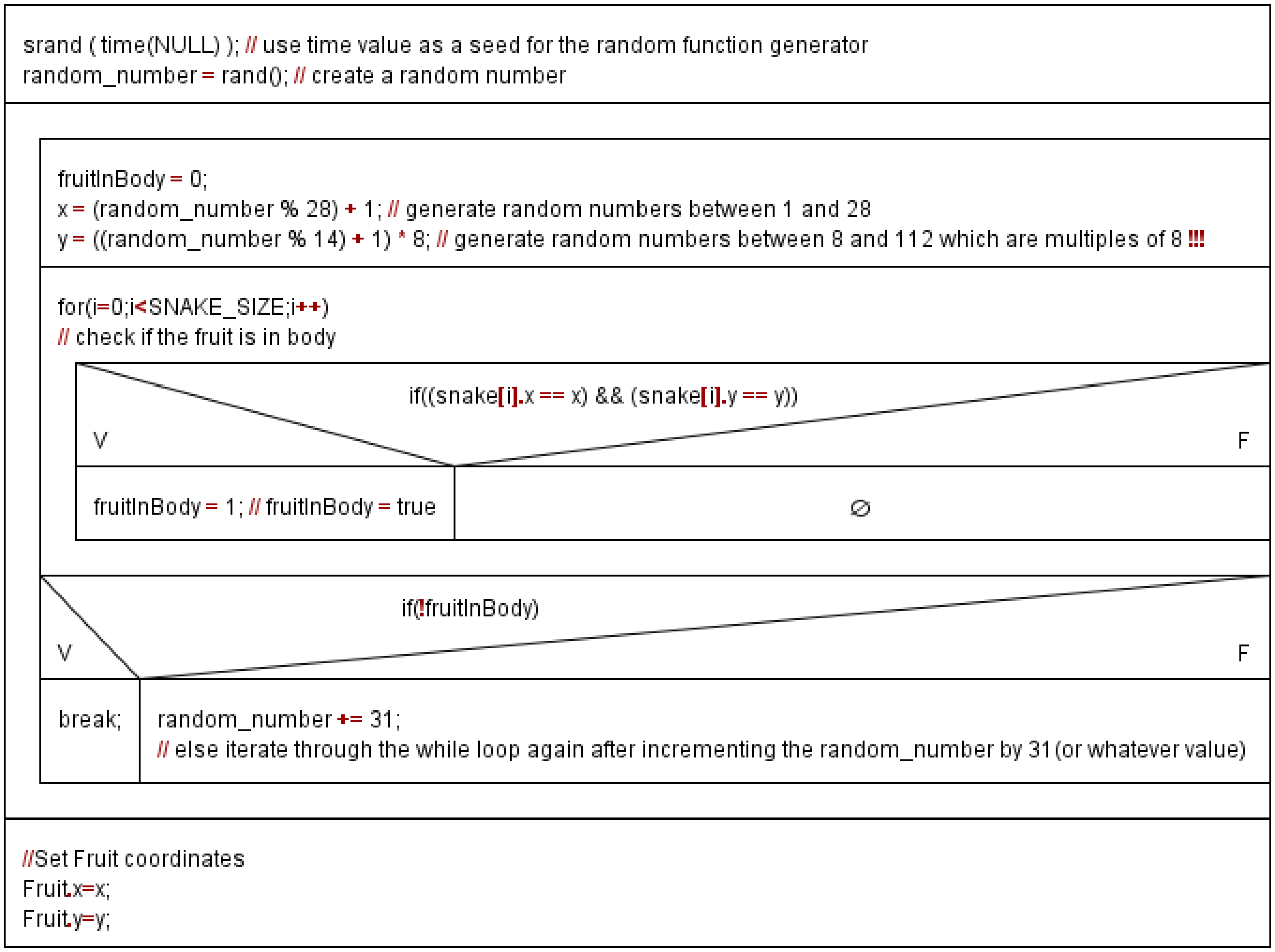
```
updatesnakebody()
//update head coordinates
snake[SNAKE_SIZE-1].x = snake[SNAKE_SIZE-1].x+1;
snake[SNAKE_SIZE-1].y = snake[SNAKE_SIZE-1].y;
if(snake[SNAKE_SIZE-1].x == 30)
//in case of wall collision
    V:
        gameoverroutine();
    F:
        ∅
if ((snake[SNAKE_SIZE-1].x == Fruit.x) && (snake[SNAKE_SIZE-1].y == Fruit.y))
// in case of fruit collision
    V:
        SNAKE_SIZE = SNAKE_SIZE + 1; // first increase the snake size

        //Next update the coordinates of the head (again)
        snake[SNAKE_SIZE-1].x = snake[SNAKE_SIZE-2].x + 1; // increase the x coordinate to move right
        snake[SNAKE_SIZE-1].y = snake[SNAKE_SIZE-2].y; // y coordinate of the head remains unchanged
        generateFruit();
    F:
        ∅
detectSelfCollision();
insertSnakeAndFruit(); //insert all snake blocks and the fruit block into the screen matrix
```

NSD of the moveSnake() function, which is made public in "snake.h" file. “newChar“ is a global variable that stores the pressed key. It is defined in the InterruptHandlers.c file since it is closely related to the UART.



generateFruit()



```
srand ( time(NULL) ); // use time value as a seed for the random function generator
random_number = rand(); // create a random number

    fruitInBody = 0;
    x = (random_number % 28) + 1; // generate random numbers between 1 and 28
    y = ((random_number % 14) + 1) * 8; // generate random numbers between 8 and 112 which are multiples of 8 !!!

    for(i=0;i<SNAKE_SIZE;i++)
    // check if the fruit is in body
        if((snake[i].x == x) && (snake[i].y == y))
        V
            fruitInBody = 1; // fruitInBody = true
        F
            ∅

    if(!fruitInBody)
    V
        break;
    F
        random_number += 31;
        // else iterate through the while loop again after incrementing the random_number by 31(or whatever value)

//Set Fruit coordinates
Fruit.x=x;
Fruit.y=y;
```

Project Structure and Function Explanation Summary:

Function	Explanation
Sysconfig.c	
portConfig()	Configures all ports used
NVICConfig()	Sets reload value for SysTick timer and enables timer and UART interrupts
configDisplay()	Configures the display as shown in the preparation sheets
configUART()	Switch on clock for UART, wait for activation, set Baud Rate Divisor and serial format. Activate UART
lcd_datcmd(unsigned char addr_code, unsigned char value)	Prints data to the LCD as provided in the preparation sheets
printMatrix(unsigned char p[128][30])	Prints the global screenmatrix on the display
disp_onoff()	Turns the display on and off according to the preparation sheets
Timer.c	
wait()	A small software counter for delay purposes
configTimer()	Switch on clock for Timer0, enable 2x16 bit and periodic modes, set prescale value
timerWait(unsigned int usec)	Pauses for a specified amount of time in microseconds (hardware wait function)
InterruptHandlers.c	
TimerIntHandler()	Clears the screen, moves the snake in the new or previous direction and prints it on screen
UARTIntHandler()	Stores the pressed key value in the global variable "newChar"
Snake.c	
ClearMatrix()	Clears the screen
FillMatrix()	Turns screen black (not used)
InsertBlock()	Inserts an 8x8 pixel block in screenmatrix
PrintGO()	Prints GO on screen (start-up display)
PrintOUCH()	Prints OUCH on screen (game over display)
ResetSnake()	Sets snake coordinates such that it will appear in the top left corner of the screen and sets 'right' direction as the default moving direction
GenerateFruit()	Generates a "Fruit" - an 8x8 pixel block at a random position on the screen. Randomness is achieved using a time seed.
gameoverroutine()	Enter an infinite loop and break only when "q" key is pressed. Print GO after the "q" key is pressed
insertSnakeAndFruit()	Insert the 8x8 blocks in the screenmatrix that create the snake and the fruit

detectSelfCollision()	For loop to detect if the snake head coordinates are equal to the coordinates of one of the body blocks
updatesnakebody()	Go from tail to head in the snake array and update the coordinates of the snake blocks such that each body block gets the coordinates from the next snake block in the array
moveright()	Update head coordinates such that the snake moves right. Update snake body, detect self- collision,fruit collision, wall collision and handle these situations. Insert the snake and fruit blocks in the screenmatrix so that later it will be printed in the Timer Handler.
moveleft()	Update head coordinates such that the snake moves left. Update snake body, detect self- collision,fruit collision, wall collision and handle these situations. Insert the snake and fruit blocks in the screenmatrix so that later it will be printed in the Timer Handler.
movedown()	Update head coordinates such that the snake moves down. Update snake body, detect self- collision,fruit collision, wall collision and handle these situations. Insert the snake and fruit blocks in the screenmatrix so that later it will be printed in the Timer Handler.
moveup()	Update head coordinates such that the snake moves up. Update snake body, detect self- collision,fruit collision, wall collision and handle these situations. Insert the snake and fruit blocks in the screenmatrix so that later it will be printed in the Timer Handler.
movesnake()	Moves the snake right,left,up or down, according to what key was pressed. It is supposed to be made public in the snake.h header file. The moveleft(),moveright(),movedown() and moveup() functions are not supposed to be made public in the header file. They are treated like privet methods. Therefore, movesnake() is like an abstraction of the details of moving the snake