ELEC/COMP 317: Embedded Systems

2022 FALL

###### Project Members: Umutcan türkmen – volkan turgut

# Project Report

### Introduction

This is a basic snake game that uses both sides of the 128x64 GLCD screen and uses a joystick to navigate the snake.Our main purpose for making this project was to learn how to implement the concepts (such as interrupts and ADC converter) we learned throughout the course in C language.Two header files were created with the help of a GLCD font creator and revised to have the same widths for our purpose. A 4-letter name can be chosen before the game starts for saving the user’s score. Also at the end of the game there is a small animation that indicates ‘Game Over’ with letters interchanging places. EEProm is used to store the high score data and it is displayed when joystick’s button is pressed. Game Over screen and high score screen can also be navigated back and forth with the joystick press. Our program stores the highest 16 scores with given names and displays them in the screen. In this program GLCD screen is used with half of the x,y values meaning if snake has an x value of 5, it corresponds to 10 in the GLCD to increase visibility and make the gameplay easier.

### Program Overvıew

We have two main arrays that hold the x and y values of the snake, an integer value to hold the snake length and the direction which is an enumeration that has 4 possible values for each side. At the start of the program all values are initialized to the default values, then there are three main while loops in our game. First while loop waits until a click is inputted from the joystick and it saves the 4-letter to the playerName variable. Second while loop is the game loop and it loops until the snake collided with itself which is the losing condition. Last while loop has no boolean checks and loops until the program has been reset, it holds a boolean value to hold the variable for which screen to display (either game over screen or high score screen). High score values are held in EEPROM in an array of length 80 bytes, there are 16 name and score values inside it with a structure shown in **Figure 1**. These are the main functions of are program and their purposes:

GLCD\_Init(): It initalizes the GLCD with correct pins on the easy avr v7 board.

GLCD\_ClearAll(): Clears the GLCD screen with 0x00 values on each column in case there was a remaining data from previous operations since GLCD has its own storage.

InitializeADC(): Initialize the corresponding ADC converter pins.

InitializeTimer1andExtInt(): This is for initalizing the timer1 which is used for getting pseudorandom for food x,y.

InitalizeSnake(): Initialize snake values with the given length which defaults to 20.

moveSnake(snakex, snakey, snakelen, direction): Moves the snake in the given direction, parameters here are the array of x and y values of the snake.

checkOutOfSpace(snakex, snakey, snakelen): Checks and corrects the snake x,y values if they are out of space on either horizontally or vertically.

checkSnakeCollision(): Checks if the snake has collided with itself, if this value comes out true then the game loop stops and it is proceeded to the game over screen.

paint(snakex, snakey, snakelen): Paints the snake and the food on the GLCD board.

readAchievementsEEPROM(): Reads the 16 of the achievements from EEPROM to the local temporary array.

checkAchievement(): Checks if the current score is larger than the lowest score on the EEPROM and if so changes the value with the smallest value.

sortAchievements(): Sorts the achievements if there are any changes in it since value is inserted at the end of the array regardless of its value.

writeAchievementsEEPROM(): Writes the array in program memory to EEPROM with the changed values.

paintGameOver(): Displays game over animation.

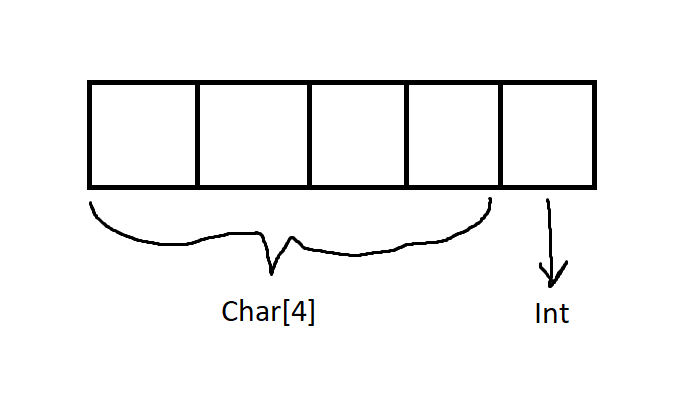
displayAchievements(): Displays achievements that is in the achievements array.

buzzEat(): Buzz sound when the food is eaten by the snake.

buzzGameOver(): Buzz sound with every frame of the game over animation.

ISR(ADC\_vect): Joystick interrupt for getting input from the user. It can take inputs as left-right, up-down, and press down.

ISR(INT0\_vect): For getting the random values for food x and y. When the snake eats the food, interrupt is called and the existing TCNT1H and TCNT1L is assigned to x and y values of the new food respectively. There is also an extra check that controls if the new food location collides with the snake and if it does it assigns new pseudorandom values until it does not collide anymore.



**Figure 1.**

### Maın problems

Our biggest problem was to figure out how the GLCD worked and it was also hard to get the first output from the GLCD using the correct pins. After that things went very well for us, second obstacle was how to get bitmaps of fonts for writing the high scores and the animating the game over sequence. Then we found out about an apache program written in Java that produces the desired fonts as bitmap outputs. We’ve used that program and corrected the corresponding letters that were distorted.

### Conclusion

I think that we’ve chosen the right project with the bigger complexity for us since our first idea of making a morse code converter was much simpler then this one. It was very fun when we were working on it, our first days was spent on working the GLCD correctly and when we got the hang of it, it was much easier to work on. We’ve learned to get random values from a deterministic machine that is a microcontroller using pseudorandomness that seems random from the user’s end. We’ve also learned how to code a microcontroller with C and even try in-lining assembly into the C language. Even though we don’t in-line assembly in our C code, we’ve learned how to do it out of curiosity. Since we built a strong foundation with the methods we’ve implemented early on to the project it helped us achieve our goals at the end much easier and we are satisfied with the level of abstraction in our code.

### SOURCE CODE

Program Memory Usage : 6950 bytes 21,2 % Full

Data Memory Usage : 502 bytes 24,5 % Full

/\*

\* snake.c

\*

\* Created: 23/12/2022 4:50:29 PM

\* Author : uturkmen16 & vturgut18

\*/

#include <avr/interrupt.h>

#include <avr/io.h>

#include <stdbool.h>

#include <stdio.h>

#include "new\_font.h"

#include "small\_font.h"

#define *F\_CPU* 8000000L

#include <util/delay.h>

#define Data\_Port PORTC

#define TotalPage 8

int foodx, foody;

*uint8\_t* gameOverCounter = 0;

int snakex[100], snakey[100];

int snakelen = 20;

int score = 0;

char playerName[4] = {'A','A','A','A'};

*uint8\_t* activePlayerNameChar = 0;

*uint8\_t* currentScoreIndex = 0;

bool snakeMoved = false;

bool gameStarted = false;

bool increaseActivePlayerNameChar = false;

bool decreaseActivePlayerNameChar = false;

bool increasePlayerNameChar = false;

bool decreasePlayerNameChar = false;

bool Continue = false;

enum Direction { Left = 0, Up = 1, Right = 2, Down = 3 };

enum Direction direction;

enum Direction nextDirection;

char achievements[80];

void GLCD\_Init();

void GLCD\_Command(char Command);

void GLCD\_ClearAll();

void GLCD\_Data(char Data);

void paint(int\* snakex, int\* snakey, int snakelen);

void moveLeft(int\* x, int\* y, int snakelen);

void moveRight(int\* x, int\* y, int snakelen);

void moveUp(int\* x, int\* y, int snakelen);

void moveDown(int\* x, int\* y, int snakelen);

void checkOutOfSpace(int\* x, int\* y, int snakelen);

bool checkSnakeCollision();

bool checkFoodCollision();

void paintLetter(char v, char letter, char startPage, char startColumn);

void paintLetterUpper(*uint8\_t* letter);

void paintLetterLower(*uint8\_t* letter);

void paintLetterSmall(*uint8\_t* v, char letter, *uint8\_t* startPage, char startColumn);

void paintNumberSmall(*uint8\_t* v, *uint8\_t* number, *uint8\_t* startPage, char startColumn);

void moveSnake(int\* x, int\* y, int snakelen, enum Direction dir);

void changeDirection();

void initalizeSnake();

void InitializeADC();

void InitializeTimer1andExtInt();

void paintGameOver();

void displayName();

void displayScore(char\* currentPlayerName, *uint8\_t* score, *uint8\_t* scoreIndex);

void displayAchievements();

void readAchievementsEEPROM();

void writeAchievementsEEPROM();

void sortAchievements();

void checkAchievement();

void buzzEat();

void buzzGameOver();

*uint8\_t* EEPROM\_read(*uint8\_t* address);

void EEPROM\_write(*uint8\_t* address, *uint8\_t* value);

int main(void) {

// Initialize uninitialized variables

direction = Right;

nextDirection = Right;

// Initialize GLCD

GLCD\_Init();

// Clear all GLCD display

GLCD\_ClearAll();

// Initialize ADC Converter

InitializeADC();

// Initialize Timer

InitializeTimer1andExtInt();

*\_delay\_ms*(1000);

// Initialize snake position

initalizeSnake();

PORTD |= (1 << PD2);

while(!Continue){

GLCD\_ClearAll();

displayName();

*\_delay\_ms*(100);

}

// Main game loop

while (!checkSnakeCollision()) { //until game finishes

changeDirection(); //change direction if necessary

moveSnake(snakex, snakey, snakelen, direction); //move the snake every loop

checkOutOfSpace(snakex, snakey, snakelen); //check if it is out of space and make it come at the opposite side of the screen

if (checkFoodCollision()) { //if food is eaten

buzzEat(); //ring the buzzer

PORTD |= (1 << PD2); //set the logic pin PD2 high so that the external interrupt occurs which takes the current TCNT1H and TCNT1L values to create the food. This was the best solution for creating the food appear at random places.

score++;

snakelen++;

}

paint(snakex, snakey, snakelen);

*\_delay\_ms*(50);

}

for(int i = 0; i < 4; i++) {

GLCD\_ClearAll();

*\_delay\_ms*(500);

paint(snakex, snakey, snakelen);

*\_delay\_ms*(500);

}

Continue = true;

// Read achievements from EEPROM

readAchievementsEEPROM();

checkAchievement();

sortAchievements();

writeAchievementsEEPROM();

while(1){

if(Continue) {

GLCD\_ClearAll();

paintGameOver();

*\_delay\_ms*(500);

}

else {

GLCD\_ClearAll();

displayAchievements();

*\_delay\_ms*(500);

}

}

}

void changeDirection() {

direction = nextDirection;

}

void initalizeSnake() {

for (int i = 0; i < snakelen; i++) {

snakex[i] = snakelen - 1 - i;

snakey[i] = 15;

}

}

void moveSnake(int\* x, int\* y, int snakelen, enum Direction dir) {

switch (dir) {

case Left:

moveLeft(x, y, snakelen);

return;

case Right:

moveRight(x, y, snakelen);

return;

case Up:

moveUp(x, y, snakelen);

return;

case Down:

moveDown(x, y, snakelen);

return;

}

}

void moveRight(int\* x, int\* y, int snakelen) {

for (int i = snakelen - 2; i >= 0; i--) {

x[i + 1] = x[i];

y[i + 1] = y[i];

}

x[0] = x[0] + 1;

}

void moveLeft(int\* x, int\* y, int snakelen) {

for (int i = snakelen - 2; i >= 0; i--) {

x[i + 1] = x[i];

y[i + 1] = y[i];

}

x[0] = x[0] - 1;

}

void moveUp(int\* x, int\* y, int snakelen) {

for (int i = snakelen - 2; i >= 0; i--) {

x[i + 1] = x[i];

y[i + 1] = y[i];

}

y[0] = y[0] - 1;

}

void moveDown(int\* x, int\* y, int snakelen) {

for (int i = snakelen - 2; i >= 0; i--) {

x[i + 1] = x[i];

y[i + 1] = y[i];

}

y[0] = y[0] + 1;

}

bool checkSnakeCollision() { //check head of the snake to all other parts, if it does collide end the game.

for (int i = 1; i < snakelen; i++) {

if (snakex[0] == snakex[i] && snakey[0] == snakey[i]) return true;

}

return false;

}

bool checkFoodCollision() { //check if snake ate the food or not

if (snakex[0] == foodx && snakey[0] == foody) return true;

return false;

}

void checkOutOfSpace(int\* x, int\* y, int snakelen) { //check if snake moved out of bounds, if it does make it appear from the other side of the screen

for (int i = 0; i < snakelen; i++) {

x[i] = x[i] % 64;

y[i] = y[i] % 32;

if (x[i] < 0) x[i] += 64;

if (y[i] < 0) y[i] += 32;

}

}

void GLCD\_Init() {

DDRA = 0xFF;

DDRD = 0xFF;

DDRB = 0xFF;

DDRC = 0xFF;

*\_delay\_ms*(20);

PORTA = 0x0C;

PORTD = 0x80;

PORTB = 0x02;

PORTD |= (1 << 7);

*\_delay\_ms*(20);

GLCD\_Command(0x3E); /\* Display OFF \*/

GLCD\_Command(0x42); /\* Set Y address (column=0) \*/

GLCD\_Command(0xB8); /\* Set x address (page=0) \*/

GLCD\_Command(0xC0); /\* Set z address (start line=0) \*/

GLCD\_Command(0x3F); // Display ON

}

void GLCD\_Change(bool screen) {

//false = left screen, true = right screen

PORTA = 0x0C;

PORTD = 0x80;

if(screen) PORTB = 0x01;

else PORTB = 0x02;

PORTD |= (1 << 7);

GLCD\_Command(0x3E); /\* Display OFF \*/

GLCD\_Command(0x42); /\* Set Y address (column=0) \*/

GLCD\_Command(0xB8); /\* Set x address (page=0) \*/

GLCD\_Command(0xC0); /\* Set z address (start line=0) \*/

GLCD\_Command(0x3F); // Display ON

}

void GLCD\_Command(char Command) {

Data\_Port = Command;

PORTA &= ~(1 << 2); // Make RS LOW

PORTA &= ~(1 << 3); // Make R/W LOW

PORTD |= (1 << 6); // HIGH-LOW Enable

*\_delay\_us*(5);

PORTD &= ~(1 << 6);

*\_delay\_us*(5);

}

void GLCD\_ClearAll() /\* GLCD all display clear function \*/

{

for (int v = 0; v < 2; v++){

if(v == 0) GLCD\_Change(false);

else if(v == 1) GLCD\_Change(true);

for (int i = 0; i < TotalPage; i++) {

GLCD\_Command((0xB8) + i); /\* Increment page \*/

for (int j = 0; j < 64; j++) {

GLCD\_Data(0x00);

}

}

GLCD\_Command(0x40); /\* Set Y address (column=0) \*/

GLCD\_Command(0xB8); /\* Set x address (page=0) \*/

}

}

void readAchievementsEEPROM(){

for(int i = 0; i < 80; i++) {

//achievements[i] = eeprom\_read\_byte((uint8\_t\*)(i+10));

achievements[i]=EEPROM\_read(i+10);

}

}

void writeAchievementsEEPROM(){

for(int i = 0; i < 80; i++) {

//eeprom\_write\_byte((uint8\_t\*)(i+10),achievements[i]);

EEPROM\_write((i+10), achievements[i]);

}

}

void displayAchievements() {

for(int i = 0; i < 16; i++) {

displayScore(&achievements[i\*5],achievements[i\*5 + 4],i);

}

}

void paint(int\* snakex, int\* snakey, int snakelen)

{

for (int v = 0; v < 2; v++){

if(v == 0) GLCD\_Change(false);

else if(v == 1) GLCD\_Change(true);

GLCD\_Command(0x40); /\* Set Y address (column=0) \*/

GLCD\_Command(0xB8); /\* Set x address (page=0) \*/

for (int i = 0; i < TotalPage; i++) {

//Set page number between 0-7

GLCD\_Command(0xB8 + i);

for (int j = 0; j < 32; j++) {

int data = 0x00;

// Paint snake

for (int t = 0; t < snakelen; t++) {

//For left screen

if (v == 0) {

if (snakex[t] / 32 == 0 && snakex[t] == j && snakey[t] / 4 == i) {

switch (snakey[t] % 4) {

case 0:

data |= 0x03;

break;

case 1:

data |= 0x0C;

break;

case 2:

data |= 0x30;

break;

case 3:

data |= 0xC0;

break;

}

}

}

//For right screen

else if (v == 1) {

if (snakex[t] / 32 == 1 && snakex[t] % 32 == j && snakey[t] / 4 == i) {

switch (snakey[t] % 4) {

case 0:

data |= 0x03;

break;

case 1:

data |= 0x0C;

break;

case 2:

data |= 0x30;

break;

case 3:

data |= 0xC0;

break;

}

}

}

}

// Paint food left

if (v == 0 && foodx / 32 == 0 && foodx == j && foody / 4 == i) {

switch (foody % 4) {

case 0:

data |= 0x03;

break;

case 1:

data |= 0x0C;

break;

case 2:

data |= 0x30;

break;

case 3:

data |= 0xC0;

break;

}

}

// Paint food right

if (v == 1 && foodx / 32 == 1 && foodx % 32 == j && foody / 4 == i) {

switch (foody % 4) {

case 0:

data |= 0x03;

break;

case 1:

data |= 0x0C;

break;

case 2:

data |= 0x30;

break;

case 3:

data |= 0xC0;

break;

}

}

GLCD\_Data(data);

GLCD\_Data(data);

}

}

GLCD\_Command(0x40); /\* Set Y address (column=0) \*/

GLCD\_Command(0xB8); /\* Set x address (page=0) \*/

}

}

void GLCD\_Data(char Data) /\* GLCD data function \*/

{

Data\_Port = Data; /\* Copy data on data pin \*/

PORTA |= (1 << 2); // RS to HIGH

PORTA &= ~(1 << 3); // RW to LOW

PORTD |= (1 << 6); // Enable

*\_delay\_us*(5);

PORTD &= ~(1 << 6); // Disable

*\_delay\_us*(5);

}

ISR(ADC\_vect) {

int ADCOut = ADCL | (ADCH << 8); //read ADC value is saved, the result gives a result between 0 and 1024 because ADCH is 2 bits and ADCL is 8 bits which makes a 10 bit resolution.

switch (ADMUX) { //since in ADC initialize first channel selected was 5, we will stick to it in switch and make sure we put it on the top to not miss the first reading

case 0xC5: //check if it is channel 5, the 5th channel is reading the analog value of x axis of joystick

if (ADCOut > 674) { //if the reading is higher then a threshold of 674, which was put to cancel out the noise because if we put 512 which is right between 1024 and 0 snake sometimes moves arbitrary.

//We also didn't want player to regret midway his choice of direction so we left a little error margin for the player in case they want to change to another direction before it is too late.

// RIGHT

if (direction != Left && gameStarted) { //checks if game started and direction is not left since it is prohibited to turn 180 degree in this game.

nextDirection = Right; //we used a flag because we found a bug which is while the next square is not displayed in the game, the code could easily enter two interrupts of ADC where let's assume the

//direction is left and we first choose up, then right. The new direction becomes right and then game executes it in while loop and snake overlaps with itself and the game does not finish.

//Therefore we are changing the flag and we are changing the direction in the while loop. This is something we both learned from COMP 302 course software programming where we

//programmed a game as well. We all used game view controller pattern for necessity. That pattern was about asynchronously reading the input from the player and synchronously reflecting it to the game.

//If an input is read, a flag is activated or deactivated to later execute it at the right time and clearing the flag for further uses.

}

else if(!gameStarted && activePlayerNameChar < 3) { //if game has started go to one next char at the right in the first section of the game

increaseActivePlayerNameChar = true;

decreaseActivePlayerNameChar = false;

*\_delay\_ms*(10);

}

*\_delay\_us*(5);

} else if (ADCOut < 350) { //check if ADCout is lower than 350 which indicates that the joystick movement is left.

// LEFT

if (direction != Right && gameStarted) {

nextDirection = Left;

}

else if(!gameStarted && activePlayerNameChar > 0) {

increaseActivePlayerNameChar = false;

decreaseActivePlayerNameChar = true;

*\_delay\_ms*(10);

}

*\_delay\_us*(5);

}

ADMUX = 0xC4; //connect the 4th channel to ADC. This is very important because we want to read 3 different ADC conversions at the same time. But we can only choose 1 channel from the ADMUX register and we chose it

//to be the 4th channel in ADC Initialization. Now we change the ADMUX to the desired channel and continue to check for the case of this channel. Select the 4th channel

break;

case 0xC4: //Check if it is the 4th channel. Since will be always selected in the previous case, all of the cases will be repeatedly executed. Therefore we will be able to read all the ADC conversions at one interrupt.

if (ADCOut > 700) {

// UP

if (direction != Down && gameStarted) { //if game started and direction is not down change the direction up

nextDirection = Up;

}

else if(!gameStarted) { //if game not started choose the left character to be changed in the first section for the name

increasePlayerNameChar = false;

decreasePlayerNameChar = true;

*\_delay\_ms*(1);

}

*\_delay\_us*(5);

} else if (ADCOut < 350) {

// DOWN

if (direction != Up && gameStarted) {

nextDirection = Down;

}

else if(!gameStarted) {

increasePlayerNameChar = true;

decreasePlayerNameChar = false;

*\_delay\_ms*(1);

}

*\_delay\_us*(5);

}

ADMUX = 0xC7; //connect the ADC to channel 7.

break;

case 0xC7: //check if ADMUX is the 7th channel.

if(ADCOut < 50) {

//For the x y values at the center, 512 is assigned to them which corresponds to 2.5V which is middle of 0 and 5V. So the center is 2.5V, down or left is 0V, up or right is 5V.

//For push button, the value is 5V at initial position and when it is pressed it becomes zero. It works like a switch. I didn't have to use ADC converter for this purpose but I did anyways.

Continue = !Continue;

gameStarted = true;

}

ADMUX = 0xC5; //return back to the first channel.

break;

default:

// Default code

break;

}

ADCSRA |= 1 << ADSC; //start ADC conversion again so that it repeatedly enters the interrupt.

}

void InitializeADC() {

DDRA = 0x00;

DDRA |= 0b01001111;

ADCSRA |= 1 << ADPS2 | 1<< ADPS1 | 1<< ADPS0;

//division for xtal frequency and clock frequency. This prescalar specifies the speed and accuracy for ADC conversion.

//It is recommended that ADC conversion frequency needs to be between 50 kHz and 200kHz according to our research. With lower prescalar the frequency will be higher. The higher frequency means fast

//conversion but less accurate compared to low frequency conversion which is low speed high accuracy. 8.000.000/128=62.5 kHz. Since our program doesn't rely on very accurate results,

//since the only important thing is to distinguish between up down left right, we didn't need a high ADC frequency.

ADMUX |= 1 << REFS0 | 1 << REFS1 | 1 << 0 | 1 << 2; //adjusting ADMUX so that internal reference voltage is used and ADC channel 5 is connected.

ADCSRA |= 1 << ADIE; //ADC control and status register: enabling interrupt register and then enabling ADC

ADCSRA |= 1 << ADEN;

sei(); //enabling interrupts

ADCSRA |= 1 << ADSC; //starting ADC conversion

}

void InitializeTimer1andExtInt(){

DDRD=0b11111111;

GICR = 1<<INT0; // Enable INT0

MCUCR = 1<<ISC01 | 1<<ISC00; // Trigger INT0 on rising edge D2

TCNT1=0; //adjusting timer1 counter value

TCCR1B= (1<<0); //adjusting the timer controller b register

}

void paintGameOver() { //for dipslaying the game over animation with the motion of going up and down

buzzGameOver();

gameOverCounter %= 4;

if(gameOverCounter == 0) {

paintLetter(0,'G',3,7);

paintLetter(0,'A',3,20);

paintLetter(0,'M',3,33);

paintLetter(0,'E',3,46);

paintLetter(1,'O',3,7);

paintLetter(1,'V',3,20);

paintLetter(1,'E',3,33);

paintLetter(1,'R',3,46);

}

else if(gameOverCounter == 1) {

paintLetter(0,'G',2,7);

paintLetter(0,'A',4,20);

paintLetter(0,'M',2,33);

paintLetter(0,'E',4,46);

paintLetter(1,'O',2,7);

paintLetter(1,'V',4,20);

paintLetter(1,'E',2,33);

paintLetter(1,'R',4,46);

}

else if(gameOverCounter == 2) {

paintLetter(0,'G',3,7);

paintLetter(0,'A',3,20);

paintLetter(0,'M',3,33);

paintLetter(0,'E',3,46);

paintLetter(1,'O',3,7);

paintLetter(1,'V',3,20);

paintLetter(1,'E',3,33);

paintLetter(1,'R',3,46);

}

else if(gameOverCounter == 3) {

paintLetter(0,'G',4,7);

paintLetter(0,'A',2,20);

paintLetter(0,'M',4,33);

paintLetter(0,'E',2,46);

paintLetter(1,'O',4,7);

paintLetter(1,'V',2,20);

paintLetter(1,'E',4,33);

paintLetter(1,'R',2,46);

}

gameOverCounter++;

}

void displayName() {

//gameOverCounter is used again since displayName() and gameOver() won't be used simultaneously

if(decreaseActivePlayerNameChar) {

activePlayerNameChar--;

decreaseActivePlayerNameChar = false;

}

else if(increaseActivePlayerNameChar) {

activePlayerNameChar++;

increaseActivePlayerNameChar = false;

}

if(increasePlayerNameChar) {

playerName[activePlayerNameChar]++;

if(playerName[activePlayerNameChar] > 'Z') {

playerName[activePlayerNameChar] = 'A';

}

increasePlayerNameChar = false;

}

else if(decreasePlayerNameChar) {

playerName[activePlayerNameChar]--;

if(playerName[activePlayerNameChar] < 'A') {

playerName[activePlayerNameChar] = 'Z';

}

decreasePlayerNameChar = false;

}

gameOverCounter %= 2;

if(gameOverCounter == 0) {

paintLetter(0,playerName[0],3,17);

paintLetter(0,playerName[1],3,45);

paintLetter(1,playerName[2],3,8);

paintLetter(1,playerName[3],3,35);

}

else if(gameOverCounter == 1) {

if(activePlayerNameChar != 0) paintLetter(0,playerName[0],3,17);

if(activePlayerNameChar != 1) paintLetter(0,playerName[1],3,45);

if(activePlayerNameChar != 2) paintLetter(1,playerName[2],3,8);

if(activePlayerNameChar != 3) paintLetter(1,playerName[3],3,35);

}

gameOverCounter++;

}

void displayScore(char\* currentPlayerName, *uint8\_t* score, *uint8\_t* scoreIndex) { //displaying the score at last which consists of multiple GLCD functions as you can see below.

*uint8\_t* v = scoreIndex / 8;

*uint8\_t* page = scoreIndex % 8;

paintLetterSmall(v,currentPlayerName[0],page,0);

paintLetterSmall(v,currentPlayerName[1],page,7);

paintLetterSmall(v,currentPlayerName[2],page,14);

paintLetterSmall(v,currentPlayerName[3],page,21);

*uint8\_t* tmp = score;

paintNumberSmall(v,(tmp % 10),page,49);

tmp /= 10;

paintNumberSmall(v,(tmp % 10),page,42);

tmp /= 10;

paintNumberSmall(v,(tmp % 10),page,35);

}

void paintLetterSmall(*uint8\_t* v, char letter, *uint8\_t* startPage, char startColumn) { //painting small letter for GLCD

if(v == 0) GLCD\_Change(false);

else if(v == 1) GLCD\_Change(true);

GLCD\_Command(0x40 + startColumn);

GLCD\_Command(0xB8 + startPage);

*uint8\_t* letterindex = letter - 65;

for(int i = 0; i < small\_font\_letter\_size; i++) {

GLCD\_Data(pgm\_read\_byte(&(small\_Font[small\_font\_letter\_start\_index + letterindex\*small\_font\_letter\_size + i])));

}

}

void paintNumberSmall(*uint8\_t* v, *uint8\_t* number, *uint8\_t* startPage, char startColumn) { //painting small number for GLCD

if(v == 0) GLCD\_Change(false);

else if(v == 1) GLCD\_Change(true);

GLCD\_Command(0x40 + startColumn);

GLCD\_Command(0xB8 + startPage);

for(int i = 0; i < small\_font\_number\_size; i++) {

GLCD\_Data(pgm\_read\_byte(&(small\_Font[small\_font\_number\_start\_index + number\*small\_font\_number\_size + i])));

}

}

void paintLetter(char v, char letter, char startPage, char startColumn){ //painting letter for GLCD

if(v == 0) GLCD\_Change(false);

else if(v == 1) GLCD\_Change(true);

GLCD\_Command(0x40 + startColumn);

GLCD\_Command(0xB8 + startPage);

*uint8\_t* letterindex = letter - 65;

paintLetterUpper(letterindex);

GLCD\_Command(0x40 + startColumn);

GLCD\_Command(0xB8 + startPage + 1);

paintLetterLower(letterindex);

}

void deleteLetter(char v, char startPage, char startColumn){ //delete letter for GLCD

if(v == 0) GLCD\_Change(false);

else if(v == 1) GLCD\_Change(true);

GLCD\_Command(0x40 + startColumn);

GLCD\_Command(0xB8 + startPage);

for (int i = 0; i < new\_font\_letter\_size / 2; i++) GLCD\_Data(0x00);

}

void paintLetterUpper(*uint8\_t* letter) { //print uppercase letter for GLCD

for(int i = 0; i < new\_font\_letter\_size / 2; i++) {

GLCD\_Data(pgm\_read\_byte(&(new\_Font[new\_font\_letter\_start\_index + letter\*new\_font\_letter\_size + i])));

}

}

void paintLetterLower(*uint8\_t* letter) { //print lowercase letter for GLCD

for(int i = new\_font\_letter\_size / 2; i < new\_font\_letter\_size; i++) {

GLCD\_Data(pgm\_read\_byte(&(new\_Font[new\_font\_letter\_start\_index + letter\*new\_font\_letter\_size + i])));

}

}

void sortAchievements() { //after checking the achievements and if the current player needs to be in the scoreboard, putting him in the bottommost rank, this function sorts the whole scoreboard.

for (int i = 0; i < 16; i++) {

// iterates the array elements from index 1

for (int j = i + 1; j < 16; j++) {

if (achievements[5\*i + 4] < achievements[5\*j + 4]) { //sort all of the achievemnts in an decreasing order.

for(int k = 0; k < 5; k++) {

*uint8\_t* temp = achievements[5\*i + k];

achievements[5\*i + k] = achievements[5\*j + k];

achievements[5\*j + k] = temp;

}

}

}

}

}

void checkAchievement(){

if(score >= achievements[79]) { //check if the current score surpasses the least score on scoreboard

for(int i = 0; i < 4; i++){

achievements[i + 75] = playerName[i]; //if it does write the new name and new score to the least score in the scoreboard. This function is for to only saving the score to the scoreboard. It will be sorted correctly later.

}

achievements[79] = score;

}

}

void buzzEat(){

DDRD = 0xff; // Configure PORTC as output

for(int i=0; i<40; i++){

PORTD = 0xff; // Turn ON the Buzzer connected to PORTC

*\_delay\_ms*(1); // Wait for some time

PORTD = 0x00; // Turn OFF the Buzzer connected to PORTC

*\_delay\_ms*(1); // Wait for some time

}

PORTD=0;

}

void buzzGameOver(){

DDRD = 0xff; // Configure PORTC as output

for(int i=0; i<10; i++){

PORTD = 0xff; // Turn ON the Buzzer connected to PORTC

*\_delay\_ms*(3); // Wait for some time

PORTD = 0x00; // Turn OFF the Buzzer connected to PORTC

*\_delay\_ms*(3); // Wait for some time

}

PORTD=0;

}

ISR(INT0\_vect)

{

foodx=TCNT1H%32;

foody=TCNT1L%32;

int i=-1;

while(i<snakelen){

i++;

if(snakex[i]==foodx && snakey[i]==foody){ //check if food overlaps with any of snake's locations

foody++; //if it does increment y and execute the whole while loop from the beginning

i=-1;

}

}

PORTD &= ~(1 << PD2); //clearing the PD2 in order to deactivate INT0 interrupt.

*\_delay\_us*(5); // Software debouncing control delay

}

*uint8\_t* EEPROM\_read(*uint8\_t* address){ //function for reading from EEPROM

*uint8\_t* result;

while(EECR&(1<<EEWE)); //wait for last writing operation to be finished

EEAR=address; //writing the address value into the EEPROM address register

EECR|=(1<<EERE); //writing 1 to EEPROM reading enable bit

result=EEDR;

return result; //returning the value that is written into EEPROM Data Register

}

void EEPROM\_write(*uint8\_t* address, *uint8\_t* value){ //function for writing to EEPROM

while(EECR&(1<<EEWE)); //wait till last writing operation to be finished just in case

EEAR=address; //write the address value into the EEPROM address register

EEDR=value; //write the data value into EEPROM data register

EECR|=(1<<EEMWE); //enabling by writing 1 to EEPROM Master Writing Enable bit

EECR|=(1<<EEWE); //enabling by writing 1 to EEPROM Writing Enable bit

}