Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по курсу «Программирование»

на тему «Игра «Змейка»»

Выполнил:

студент группы

Принял:

к.т.н., доцент Федюнин Р.Н.

Пенза, 2019

**Постановка задачи**

Необходимо разработать программу для игры «Змейка». Программа должна иметь вывод в терминал, управление с клавиатуры. Необходимо сделать игру интуитивно понятной для пользователя.

Многомодульность программы. Программа должна быть поделена на логические модули. Это упростит поиск ошибок при отладке и тестировании программы, а также позволит легко расширять функционал программы.

Использование сложных типов данных – массивов, структур, списков. Это необходимо для более простой и интуитивной обработки данных в коде программы.

Устройство ввода – клавиатура. Необходимо различать и идентифицировать действия, произведенные с их помощью, это сделает использование программы удобным.

**Выбор решения**

При запуске игры открывается окно с игрой. Под игровым полем находятся счетчики для вывода текущего счета и текущей скорости. Все взаимодействие с игрой состоит из:

1. Выбор стороны движения. При нажатии любой из клавиш “WASD” происходит проверка на столкновение и последующая корректировка движения.
2. При столкновении игра завершается, показывая на экране текущий счет.

**Описание разработки программы**

Разработанная программа состоит из следующих модулей:

1. game.c – Основной файл игры. Отвечает за инициализацию, очистку памяти и отрисовку.
2. field.c – Работа с игровым полем. Создание, инициализация, очистка ресурсов поля, генерация игрового поля, отрисовка игрового поля.
3. snake\_body.c – Работа с самой змейкой. Инициализация, очистка ресурсов, добавление «хвоста».

В качестве среды разработки была выбрана программа QtCreator. Данная IDE обладает всеми необходимыми средствами при разработке и отладке программы. Для отладки использовались: точка останова, трассировка, анализ содержимого переменных.

**Описание программы**

Модуль game.c является основным модулем программы. При его запуске происходит первичная отрисовка игрового поля, инициализация всех структур данных. Все управление игрой сведено к выбору стороны движения, т.е. все управление сведено к клавишам “WASD”.

На рисунке приведена схема взаимодействия программы:

Field

MAIN

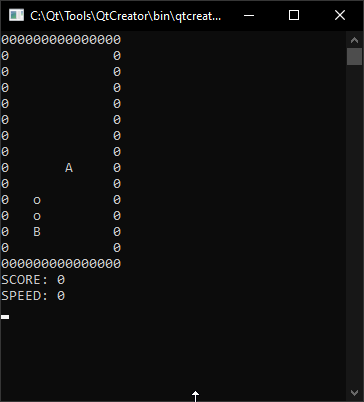
Movement.h

Types.h

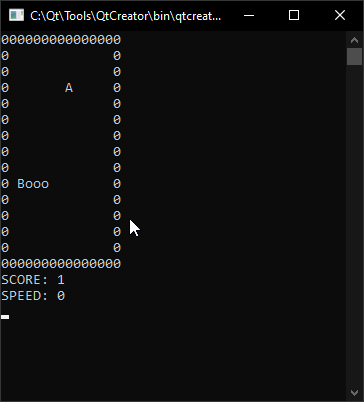
Snake\_body

**Руковоство пользователя**

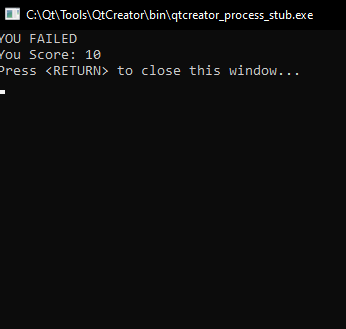
Программа Snake.exe предназначена для развлечения и легкой и нетребовательной к ресурсам компьютера и времени игрой.

Окно терминала при запуске игры:

Изменение направления движения:



Экран проигрыша:



**Заключение**

При выполнении данной курсовой работы были получены навыки разработки многомодульных программ. Была освоена работа с памятью, консолью и векторными инструкциями процессора. Так же были получены основные навыки отладки и тестирования программ IDE QtCreator на языках С и Assembler/

В рамках курсовой работы была написана игра «Змейка» для игры в терминале Windows.

В дальнейшем программу можно улучшить, добавив гарфический интерфейс, генерацию уровней и сами уровни.

**Список используемых источников**

1. Керниган Б. Ритчи Д. Язык программирования С. 1985 г.

2. MSDN.

3. Прата С. Язык программирования С++. 2019 г.

4. А.А. Тюгашев. Языки программирования. Учебное пособие. 2018 г.

5. Руслан Аблязов. Программирование на ассемблере на платформе x86-64

**Листинг программы**

Game.c

#include <string.h>

#include <stddef.h>

#include <stdlib.h>

#include <conio.h>

#include <stdio.h>

#include <windows.h>

#include "snake\_body.h"

#include "field.h"

#include "movement.h"

#include "types.h"

#define SET\_POS(a, RC) !((snake\_body->a - a) == field->RC - 1 || (snake\_body->a - a) == 0) ? snake\_body->a - a : field->RC - 1 - snake\_body->a;

game\_field \*field;

Snake \*snake\_body;

int score = 0;

int speed = 0;

void gen\_fruit()

{

int x;

int y;

do

{

x = rand() % (field->ROW - 2) + 1;

y = rand() % (field->COL - 2) + 1;

}

while(field->area[x][y] == 'B' || field->area[x][y] == 'o');

field->area[x][y] = 'A';

}

void game\_over()

{

system("cls");

printf("YOU FAILED\nYou Score: %d\n", score);

game\_field\_cleanup(&field);

cleanup\_snake\_body(&snake\_body);

exit(EXIT\_SUCCESS);

}

bool revert = false;

void move\_snake(int x, int y)

{

int tmp\_X = snake\_body->x;

int tmp\_Y = snake\_body->y;

snake\_body->x = SET\_POS(x, ROW);

snake\_body->y = SET\_POS(y, COL);

if((field->area[snake\_body->x][snake\_body->y] == 'o'))

{

if( snake\_body->next->x == snake\_body->x || snake\_body->next->y == snake\_body->y ) {

snake\_body->x = tmp\_X;

snake\_body->y = tmp\_Y;

revert = true;

return; //to move;

}

else {

game\_over();

}

}

if(field->area[snake\_body->x][snake\_body->y] == 'A')

{

//++score;

\_\_asm

(

"mov eax, score \n\

inc eax \n\

mov score, eax \n"

);

add\_tail(&snake\_body);

gen\_fruit();

}

field->area[snake\_body->x][snake\_body->y] = 'B';

Snake \*tmp = snake\_body->next;

while(tmp != NULL)

{

int tx = tmp->x;

int ty = tmp->y;

tmp->x = tmp\_X;

tmp->y = tmp\_Y;

field->area[tmp->x][tmp->y] = 'o';

tmp\_X = tx;

tmp\_Y = ty;

tmp = tmp->next;

}

field->area[tmp\_X][tmp\_Y] = ' ';

}

void Draw()

{

field->field\_ops\_t->draw\_game\_field(&field);

printf("\nSCORE: %d\n", score);

printf("SPEED: %d\n", speed / 15);

}

void speed\_change()

{

static bool is\_changed = false;

if(score % 10 == 0 && !is\_changed) {

speed += 15;

is\_changed = true;

}

else if (score % 10 != 0) {

is\_changed = true;

}

}

int main()

{

int xW = 15, yW = 15;

int x = -1, y = 0;

field = init\_field();

field->field\_ops\_t->init\_game\_field\_size(&field, xW, yW);

field->field\_ops\_t->generate\_game\_area(&field);

snake\_body = init\_snake\_body();

while(1){

//Draw();

\_\_asm

(

"call Draw"

);

Sleep(100 - speed);

int input = -1;

if(\_kbhit())

input = \_getch();

if(input == 119)

{

MOVE\_UP

}

if(input == 115)

{

MOVE\_DOWN

}

if(input == 97)

{

MOVE\_LEFT

}

if(input == 100)

{

MOVE\_RIGHT

}

move\_snake(x, y);

if(revert) {

x \*= -1;

y \*= -1;

revert = false;

}

}

game\_field\_cleanup(&field);

cleanup\_snake\_body(&snake\_body);

\_getch();

return 0;

}

Snake\_body.h

#ifndef \_\_SNAKE\_BODY\_H\_\_

#define \_\_SNAKE\_BODY\_H\_\_

#include <stdlib.h>

#include <stddef.h>

struct snake

{

int x;

int y;

struct snake \*next;

};

typedef struct snake Snake;

Snake \* init\_snake\_body();

void cleanup\_snake\_body(Snake\*\* snake\_body);

void add\_tail(Snake \*\*snake\_body);

#endif

Snake\_body.c

#include "snake\_body.h"

void add\_tail(Snake \*\*snake\_body)

{

if((\*snake\_body) == NULL) {

(\*snake\_body) = (Snake \*)malloc(sizeof(Snake));

(\*snake\_body)->x = 4;

(\*snake\_body)->y = 4;

(\*snake\_body)->next = NULL;

return;

}

static int i = 0;

Snake \*tmp = (\*snake\_body);

while(tmp->next != NULL){

tmp = tmp->next;

}

Snake \*ttmp = (Snake \*)malloc(sizeof(Snake));

ttmp->x = tmp->x;

ttmp->y = tmp->y;

ttmp->next = NULL;

tmp->next = ttmp;

}

Snake \*init\_snake\_body()

{

Snake\* snake\_body = NULL;

for(int i = 0; i < 3; ++i)

add\_tail(&snake\_body);

return snake\_body;

}

void cleanup\_snake\_body(Snake \*\*snake\_body)

{

while(snake\_body)

{

Snake \*tmp = (\*snake\_body)->next;

free(snake\_body);

snake\_body = tmp;

}

}

Field.h

#ifndef \_\_FIELD\_H\_\_

#define \_\_FIELD\_H\_\_

#include <stdlib.h>

typedef struct field\_ops game\_field\_ops;

typedef struct field\_t game\_field;

struct field\_t

{

char \*\* area;

char \*\* old\_area;

int ROW;

int COL;

game\_field\_ops \*field\_ops\_t;

};

struct field\_ops

{

void (\*init\_game\_field\_size)(game\_field \*\*,int, int);

void (\*generate\_game\_area)(game\_field \*\*);

void (\*draw\_game\_field)(game\_field \*\*);

};

void generate\_game\_area\_t(game\_field \*\*self);

void init\_game\_field\_size\_t(game\_field \*\*self, int row, int col);

void draw\_game\_field\_t(game\_field \*\*self);

game\_field \* init\_field();

void game\_field\_cleanup(game\_field \*\*self);

#endif

Field.c

#include <string.h>

#include <stdlib.h>

#include <stdio.h>

#include <windows.h>

#include <emmintrin.h>

#include "field.h"

void generate\_game\_area\_t(game\_field \*\*self)

{

(\*self)->area = (char \*\*)malloc((\*self)->ROW \* sizeof(char \*));

(\*self)->old\_area = (char \*\*)malloc((\*self)->ROW \* sizeof(char \*));

for(int i = 0; i < (\*self)->ROW; ++i) {

(\*self)->area[i] = (char \*)malloc((\*self)->COL \* sizeof(char));

(\*self)->old\_area[i] = (char \*)malloc((\*self)->COL \* sizeof(char));

}

for(int i = 0; i < (\*self)->ROW; ++i) {

for(int j = 0; j < (\*self)->COL; ++j) {

if(i == 0 || j == 0 || i == (\*self)->ROW - 1 || j == (\*self)->COL - 1) {

(\*self)->area[i][j] = '0';

(\*self)->old\_area[i][j] = '0';

}

else {

(\*self)->area[i][j] = ' ';

(\*self)->old\_area[i][j] = ' ';

}

}

}

(\*self)->area[4][4] = 'B';

(\*self)->old\_area[4][4] = 'B';

(\*self)->area[(\*self)->ROW / 2 + 1][(\*self)->COL / 2 + 1] = 'A';

(\*self)->old\_area[(\*self)->ROW / 2 + 1][(\*self)->COL / 2 + 1] = 'A';

for(int i = 0; i < (\*self)->ROW; ++i) {

fwrite((\*self)->area[i], sizeof(char), (\*self)->COL, stdout);

fwrite("\n", sizeof(char), 1, stdout );

}

}

void init\_game\_field\_size\_t(game\_field \*\*self, int x, int y)

{

(\*self)->ROW = y;

(\*self)->COL = x;

}

game\_field \* init\_field()

{

game\_field \* field = (game\_field \*)calloc(1, sizeof(game\_field));

field->field\_ops\_t = (game\_field\_ops \*)calloc(1, sizeof(game\_field\_ops));

field->field\_ops\_t->init\_game\_field\_size = &init\_game\_field\_size\_t;

field->field\_ops\_t->generate\_game\_area = &generate\_game\_area\_t;

field->field\_ops\_t->draw\_game\_field = &draw\_game\_field\_t;

return field;

}

void draw\_game\_field\_t(game\_field \*\*self)

{

COORD pos;

HANDLE h\_stdout = GetStdHandle(STD\_OUTPUT\_HANDLE);

pos.X = 0;

pos.Y = 0;

for(int i = 0; i < (\*self)->ROW; ++i) {

if((0xFFFFFFFF & ~\_mm\_movemask\_epi8(\_mm\_cmpeq\_epi8(

\_mm\_load\_si128((\_\_m128i const \*)(\*self)->area[i]),

\_mm\_load\_si128((\_\_m128i const \*)(\*self)->old\_area[i])))))

{

pos.Y = i;

SetConsoleCursorPosition(h\_stdout, pos);

fwrite((void \*)(\*self)->area[i], sizeof(char), (\*self)->COL, stdout);

\_mm\_store\_si128((\_\_m128i \*)(\*self)->old\_area[i],

\_mm\_load\_si128((\_\_m128i const \*)(\*self)->old\_area[i]));

}

}

}

void game\_field\_cleanup(game\_field \*\*self)

{

free((\*self)->field\_ops\_t);

free((\*self));

}

Movement.h

#ifndef \_\_MOVEMENT\_H\_\_

#define \_\_MOVEMENT\_H\_\_

#define MOVE\_UP x = 1; y = 0;

#define MOVE\_DOWN x = -1; y = 0;

#define MOVE\_LEFT x = 0; y = 1;

#define MOVE\_RIGHT x = 0; y = -1;

#endif

Types.h

#ifndef \_\_TYPES\_H\_\_

#define \_\_TYPES\_H\_\_

typedef unsigned char bool;

#define true 1

#define false 0

#endif // TYPES\_H