МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: «Консольная игра змейка»

Студент гр. 0304 _____ Мажуга Д.Р.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент: Мажуга Д.Р.

Группа 0304

Тема работы: «Консольная игра змейка»

Исходные данные:

Реализовать консольную игру «Змейка».

Содержание пояснительной записки:

«Содержание», «Введение», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 17 страниц.

Дата выдачи задания: 05.04.2021

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Мажуга Д.Р.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

Была поставлена задача разработки консольной игры «Змейка».

Правила игры: Игрок управляет «Змейкой»(Существо состоящие из одинаковых символов идущих друг за другом, внешних похожих на змею), которая ползает по полю, собирая яблоки и избегая столкновения со стенками и с собственным хвостом. Каждый раз когда змейка съедает яблоко, она становится больше чем усложняет игру.

Управления осуществляется при помощу клавишь «w», «a», «s», «d», кнопка выхода из игры «b».

Демонстрация работы программы.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Была поставлена задача разработки консольной игры «Змейка».

Правила игры: Игрок управляет «Змейкой»(Существо состоящие из одинаковых символов идущих друг за другом, внешних похожих на змею), которая ползает по полю, собирая яблоки и избегая столкновения со стенками и с собственным хвостом. Каждый раз когда змейка съедает яблоко, она становится больше чем усложняет игру.

Целью данной работы является создание программы, реализующие консольную игру на основе правил игры, найденных в интренете.

1. РЕАЛИЗАЦИЯ ВСПОМОГАТЕЛЬНЫХ ФУНКЦИЙ

1.1. Обработка терминала.

Поскольку используемый «по умолчанию» в Linux режим ввода с клавиатуры (так называемый канонический режим) нам не подходит, так как для того чтобы ввести какие-то данные с клавиатуры нам не обходимо нажать клавишу «Ввода» (Enter), а для того чтобы змейка комфортно перемещалась нам необходимо перейти в так называемый не какнонический режим в этом нам помогут библиотеки <termios.h> и <unistd.h>, перейдем к реализации функци с помощью этой библиотеке:

Для начала мы объявляем структуру termios_saved_settings, в которую в последствии сохраним все начальные настройки. Далее мы создаем функцию set_terminal_settings(), в которой мы объявляем новую структуру в которую мы и будем сохранять все новые настройки, также перед тем как изменить настройки терминала мы сохраняем их в структуру termios_saved_settings, с помощью функции tcgetattr(), после чего мы отключаем канонический режим и echo режим (режим при котором после нажатия любой клавиши она выводится на экран) с помощью побитовых флагов, которые лежат в библеотеке <termios.h> . Далее устанавливаю VMIN и VTIME в массиве параметров c_cc в нулевые значения. VMIN отвечает за минимальное количество символов, которого будет дожидаться терминал в неканоническом режиме. Если там будет что-то кроме нуля, любая попытка читать с терминала заблокирует программу до получения VMIN символов. VTIME отвечает за то, сколько (при ненулевом VMIN) терминал будет ожидать символы. Также в set_terminal settings() используется esc-последовательность “\x1b[?25l“ отвечающая за отключения курсора. Функция atexit(), отвечает за то, чтобы переданая ей функция выполнилаь при коректном завершении функции set_terminal_settings().

Далее мы создаём функцию reset_terminal_settings(), котрая отвечает за возращение терминала в кононический режим, с помощью функции tcsetattr(), в

которую мы предаём сохраненые нами настройки, также очищаем экран с помощю функции systen(“clear“) и восстанавливаем курсор с помощью esc-посследовательности “\x1b[?25h“.

1.2. Обработка сигналов.

Поскольку вся наша программа работает в н каноническом режиме, нам нужно бескпокоится о корректном выходе из программы, для того чтобы это обеспечить мы с помощью библиотеки <signal.h>, написали функцию обработки сигналов sighandler(), которая на вход принимает сигнал который мы хотим обработать. Заранее объявив глобальную пременую run, в самой функции если нам приходит сигнал мы зануляем переменную run и в функцию signal(), передаем нашу функцию sighandler(), для того чтобы она обработала переданами нам сигнал.

2. РЕАЛИЗАЦИЯ ИГРЫ ЗМЕЙКА

2.1. Инициализация и отрисовка поля.

Глобально содаем двойной массив, который в последствии заполняем пробелами, с помощью функции init_field(). После чего в функции create_field(), мы отрисовываем границе нашего поля и выводим количество набраных нами очков на экран. В начале мы вызываем esc-последовательность “\x1b[H“ передвигающую курсор в левый верхний угол , далее с помощью циклов выводим на экран границы.

2.2. Реализация функции ввода.

В функции input(), мы считываем данные с помощью функции read(), в переменную input, инициализированную в функции, в данном случае мы используем read(), поскольку библиотечные функции буферизованного ввода работает не так как они работает в каноническом режимиме. Далее с помощью switch(), мы изменяем текущее направление змейки.

2.3. Функция изменения положения головы на поле.

В функции change_head(), с помощью switch(), в зависимости от направления движения, мы изменяем координаты головы змеи.

2.4. Логика игры.

В функции logic(), мы проверяем текущую координату головы, если же она вышла за пределы поля игра завершается, также мы проверяем не врезалась ли змейка в себя, проверяя текущюю координату головы и если следующие место в плоскости не пробел или не яблоко, то игра завершается.

2.5. Передвижение змейки.

В функции snake_move(), мы с помощью функции memmove(), передвигаем текущие положение головы назад, после чего мы проверяем мы изменяем координату головы с помощью функции change_head(), после чего вызываем функцию logic(), далее мы прописываем условие роста змейки, т.е если мы съедаем яболоко, то змейка растёт, генереируем новую позицию

яблока и увеличиваем количество очков. После всех этих процедур мы ставим в первый элемент змейки(т.е в голову) элемент змеи. Далее мы прописываем условие для того чтобы при старте задать змее определенный размер заранее известный в объявлённой нами переменной snake_buff, в этом условии мы проверяем если размер змейки меньше этой пременной, то мы увелечиваем размер змейки, иначе мы понимаем что это конец змейки отрисовываем в координате размера змейки(т.е в хвосте) пробел.

2.6. Функция получения случайной координаты.

В функции get_random_pos(), выдаёт случайные координаты по строчкам и по столбцам с помощи функции rand(), если эти случайные координаты пустые, то она возвращает структуру с случайными координатами.

2.7. Функция генерации змейки.

В функции spawn_snake(), мы задайм случайную координату с помощью функции get_random_pos(), и отрисовываем элемент змей на месте этой координаты.

2.8. Функция генерации еды

В функции gen_food(), мы задайм случайную координату с помощью функции get_random_pos(), и отрисовываем яблоко на месте этой координаты.

3. ФУНКЦИЯ MAIN

В фунции main(), мы преводим терминал в не канонический режим функцией set_terminal_settings(), устанавливаем обработку сигналов функцией signal(), в которую пердаём наш обработчик сигналов, далее ииницализируем поле, спавним змейку, отрисовываем поле и генерируем еду с помощью выше описанных функций.

Далее создаём цикл который должен работать пока наша атомарная переменная run не нулевая(нулевая она только втом случае, если мы передли сигнал). В самом цикле мы проверяем напрваление если змейка не стоит, то она продалжает двигатся и мы отрисовываем поле снова, спомощью функций snake_move() и create_field(), после чнго происходит ввод с помощью функции input(), после него вызывается функция uslepp(), функция uslepp() - приостанавливает работу программу на указанное количество времени в аргументе в микросекундах.

ЗАКЛЮЧЕНИЕ

В результате была разработана консольная игра змейка, соответсвующая поставленым условиям задачи. Программа успешно компилируется и выполняется на платформе ОС Linux.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Онлайн справочник по C/C++: cppreference.com

2. Linux manual page

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <signal.h>
#include <unistd.h>
#include <time.h>
#include <string.h>

#define MAX_LENGTH 500
#define HEIGHT 20
#define WIDTH 85
#define BILLION 1000000000

enum {STOP = 0, UP, DOWN, RIGHT, LEFT};

const char wall = '#';
const char snake_el = '*';
const char apple = '@';

const int height = HEIGHT - 3;
const int width = WIDTH - 2;

unsigned char play_field[HEIGHT][WIDTH];
sig_atomic_t run = 1;
int apl_cnt = 0;
int dir = STOP;
int new_dir;

typedef struct Coord{
    int col;
    int row;
}Coord;

Coord snake[MAX_LENGTH];
int snake_size = 1;
int snake_buff = 4;
int score = 0;

void sighandler(int sig){
    run = 0;
    signal(sig, sighandler);
}

struct termios saved_settings;

void reset_terminal_settings(){ //Перевод терминала в канонический режим

    printf("\x1b[?25h");
    tcsetattr(STDIN_FILENO, TCSANOW, &saved_settings);
    system("clear");
}


void set_terminal_settings(){ //Перевод работы терминала в не канонический режим
    struct termios new;
```

```
    //Сохранием настройки терминала
    tcgetattr(STDIN_FILENO, &saved_settings);

    tcgetattr(STDIN_FILENO, &new);
    new.c_lflag &= ~(ICANON | ECHO);
    new.c_cc[VMIN] = 0;
    new.c_cc[VTIME] = 0;
    tcsetattr(STDIN_FILENO, TCSAFLUSH, &new);
    printf("\x1b[?25l");
    system("clear");
    atexit(reset_terminal_settings);
}


void init_field(){

    for(int i = 0; i < height; i++)
        for(int j = 0; j < width; play_field[i][j++] = ' ');

}

void create_field(){
    printf("\x1b[H"); //Передвигает курсор на (0;0)

    for(int i = width + 2; i; i--)
        putchar(wall);
    putchar('\n');

    for(int i = 0; i < height; i++){
        putchar(wall);
        for(int j = 0; j < width; j++)
            putchar(play_field[i][j]);

        putchar(wall);
        putchar('\n');
    }

    for(int i = 0; i < width + 2; i++)
        putchar(wall);
    putchar('\n');

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

Coord get_random_pos(){
    Coord result;

    while(1){
        result.row = rand() % height;
        result.col = rand() % width;
        if(play_field[result.row][result.col] == ' ')
            break;
    }

    return result;
}

void input(){
    int input;

    if(read(STDIN_FILENO, &input, 3)){
        switch (input) {
```

```
            case 'w': case 'W': dir = UP; break;
            case 's': case 'S': dir = DOWN; break;
            case 'a': case 'A': dir = LEFT; break;
            case 'd': case 'D': dir = RIGHT; break;
            case 'b': case 'B': reset_terminal_settings(); system("clear"); exit(0); break;
        }
    }
}

void change_head(){
    switch(dir){
        case UP:
            snake[0].row--;
            break;
        case DOWN:
            snake[0].row++;
            break;
        case LEFT:
            snake[0].col--;
            break;
        case RIGHT:
            snake[0].col++;
    }
}

void logic(){
    if(snake[0].row < 0 || snake[0].col < 0
            || snake[0].row > height - 1 || snake[0].col > width - 1) {
        reset_terminal_settings();
        system("clear");
        printf("Game over! You hit the wall!\n");
        printf("Pleas press any button to leave\n");
        getchar();
        exit(0);
    }

    if(play_field[snake[0].row][snake[0].col] != ' '
        && play_field[snake[0].row][snake[0].col] != apple){
        reset_terminal_settings();
        system("clear");
        printf("Game over! You eat your self!\n");
        printf("Pleas press any button to leave\n");
        getchar();
        exit(0);
    }
}

void gen_food()
{
    Coord random_pos = get_random_pos();
    play_field[random_pos.row][random_pos.col] = apple;
}

void snake_move(){
    memmove(&snake[1], &snake[0], sizeof(Coord) * snake_size);
    change_head();
    logic();

    if(play_field[snake[0].row][snake[0].col] == apple){
        snake_size++;
        gen_food();
        score++;
```

```
    }

    play_field[snake[0].row][snake[0].col] = snake_el;
    if(snake_size < snake_buff)
        snake_size++;
    else
        putchar(play_field[snake[snake_size].row][snake[snake_size].col] = ' ');
}

void spawn_snake(){
    snake[0] = get_random_pos();
    play_field[snake[0].row][snake[0].col] = snake_el;
}

int main(){
    set_terminal_settings();
    signal(SIGINT, sighandler);

    init_field();
    spawn_snake();
    create_field();
    gen_food();

    while(run){
        if(dir != STOP){
            snake_move();
            create_field();
        }

        input();
        usleep(150000);
        input();
    }
    return 0;
```