**Министерство образования РФ**

**Нижегородский государственный университет имени Н. И. Лобачевского**

**Институт ИТММ**

**Отчет по лабораторным работам**

Выполнил: студент группы 381703-2

Капустин Г. К.

Проверил: доцент кафедры МОСТ

к.т.н. Сысоев А. В.

Н. Новгород, 2018

Оглавление

[Введение 4](#_Toc515564105)

[Блок 1 5](#_Toc515564106)

[Постановка задачи 5](#_Toc515564107)

[Руководство пользователя 6](#_Toc515564108)

[Руководство программиста 7](#_Toc515564109)

[Блок 2 8](#_Toc515564110)

[Постановка задачи 8](#_Toc515564111)

[Руководство пользователя 9](#_Toc515564112)

[Руководство программиста 11](#_Toc515564113)

[Блок 3 12](#_Toc515564114)

[Постановка задачи 12](#_Toc515564115)

[Руководство пользователя 13](#_Toc515564116)

[Руководство программиста 15](#_Toc515564117)

[Блок 4 16](#_Toc515564118)

[Постановка задачи 16](#_Toc515564119)

[Руководство пользователя 17](#_Toc515564120)

[Руководство программиста 19](#_Toc515564121)

[Блок 5 20](#_Toc515564122)

[Постановка задачи 20](#_Toc515564123)

[Руководство пользователя 21](#_Toc515564124)

[Руководство программиста 22](#_Toc515564125)

[Блок 6 23](#_Toc515564126)

[Постановка задачи 23](#_Toc515564127)

[Руководство пользователя 24](#_Toc515564128)

[Руководство программиста 26](#_Toc515564129)

[Заключение 27](#_Toc515564130)

[Литература 28](#_Toc515564131)

[Приложение 29](#_Toc515564132)

[Код работы №6: 29](#_Toc515564133)

[drawer.h 29](#_Toc515564134)

[mainGame.h 29](#_Toc515564135)

[snakeObjects.h 30](#_Toc515564136)

[drawer.cpp 30](#_Toc515564137)

[mainGame.cpp 34](#_Toc515564138)

[snakeObjects.cpp 36](#_Toc515564139)

[main.cpp 38](#_Toc515564140)

# Введение

Цель лабораторной работы – освоение основных концепций и парадигм объектно-ориентированного программирования, овладение практическими навыками в разработке и проектировании программ.

# Блок 1

## Постановка задачи

Разработать класс **Время**.

Класс должен хранить время в формате: часы, минуты, секунды (в диапазоне от 0 часов 0 минут 0 секунд до 23 часа 59 минут 59 секунд).

Класс должен предоставлять операции: 1) установить время, 2) узнать время, 3) вычислить разницу между заданным временем и установленным (в часах, минутах и секундах), 4) сдвинуть время на заданное смещение (в часах, минутах и секундах) в меньшую и в большую сторону.

Класс должен содержать все необходимые конструкторы, оператор присваивания, а также «уметь» выводить себя на консоль.

## Руководство пользователя

При запуске программы пользователь может установить системное время или ввести его с клавиатуры.



Рис. 1

После установки времени необходимо выбрать интересующую пользователя функцию программы.

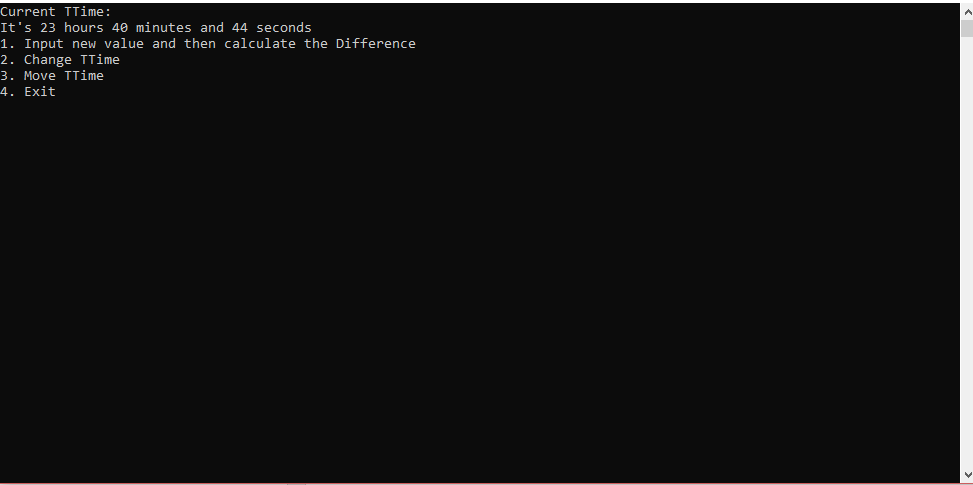


Рис. 2

## Руководство программиста

Класс содержит три поля: hour, min, sec, назначение которых интуитивно понятно, пять методов: SetTime, ChangeTime, SetSysTime, ShowTime, Difference (установить время, поменять время на какой-то временной отрезок, установить системное время, показать время в консоли и посчитать разницу между введенным временем соответственно).

# Блок 2

## Постановка задачи

Разработать класс **Матрица**.

Класс должен хранить квадратную матрицу целых чисел заданного размера (от 2 до 8 строк).

Класс должен предоставлять следующие операции: 1) задать размер матрицы, 2) узнать размер матрицы, 3) задать элемент матрицы по его индексам, 4) узнать элемент матрицы по его индексам, 5) проверить, обладает ли матрица диагональным преобладанием, 6) вычислить сумму двух матриц одного размера.

## Руководство пользователя

При запуске программы появляется консольное окно, в которое нужно ввести необходимый размер матрицы.



Рис. 3

Случайным образом генерируется 2 матрицы заданного размера, считается их сумма, разность, умножение первой матрицы на число, размер матриц, выводится элемент номер (2,2), проверяется диагональное преобладание первой матрицы.

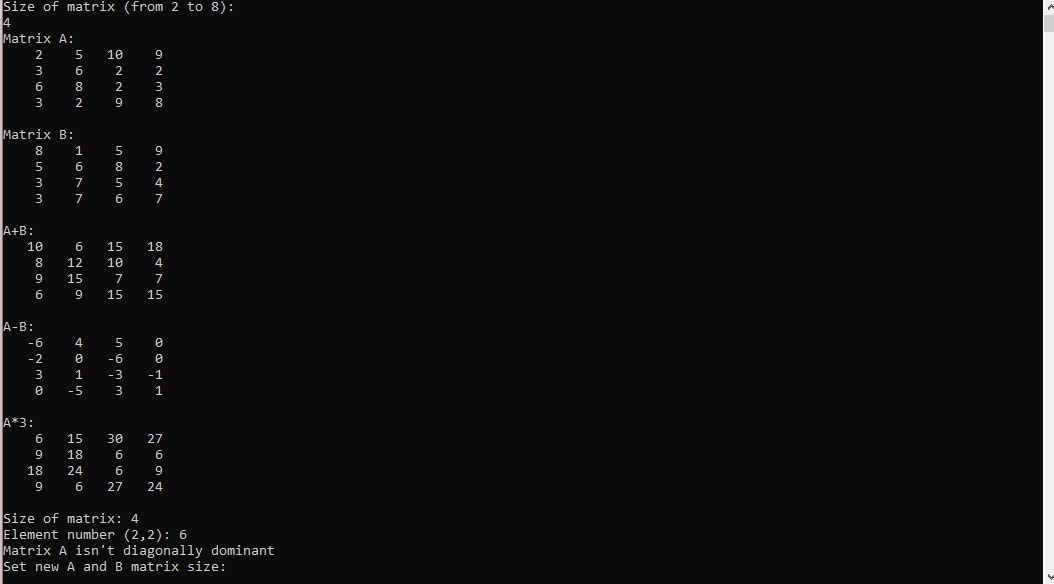
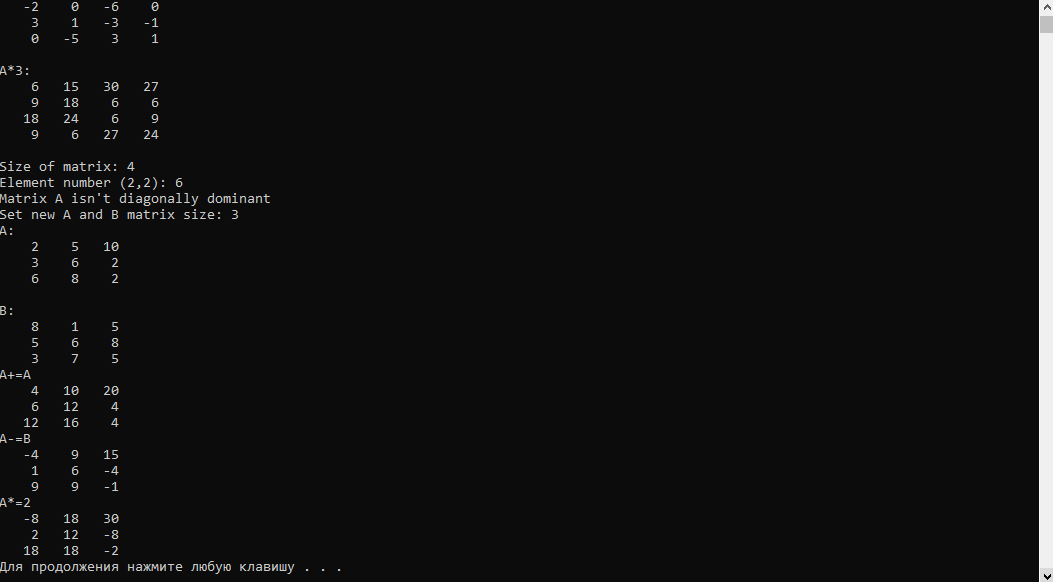


Рис. 4

Пользователю предлагается ввести новый размер матрицы, после чего происходит изменение размерности с сохранением старых значений.

Рис. 5

## Руководство программиста

Элементы матрицы в динамическом двумерном массиве указателей типа int, так же есть поле, хранящее размерность матрицы – size. Конструктор объекта matrix перегружен, в качестве аргумента может быть передан либо размер будущей матрицы, либо константная ссылка на объект этого же типа. Происходит инициализация переменной size, динамически выделяется память. В классе содержится набор методов get-set для получения и установки размерности, элементов матрицы. Метод isDiagDominate возращает 1 в случае, если матрица является диагонально преобладающей и 0, если нет. Метод print выводит на экран консоли все элементы матрицы. Реализован ряд операторов, таких как сложение с матрицей, умножение на число, составное присваивание в различных вариациях. В деструкторе выполняется освобождение памяти. Функция main имеет линейный вид, в ходе ее выполнения демонстрируются возможности класса.

# Блок 3

## Постановка задачи

Разработать класс **Однострочный текстовый редактор**

Класс должен предоставлять возможность разместить в выбранной позиции окна консоли поле заданной длины для ввода с клавиатуры последовательности символов. В минимальном варианте длина последовательности не должна превышать длину поля ввода.

Класс должен предоставлять следующие операции: 1) задать длину поля ввода, 2) узнать длину поля ввода, 3) задать позицию поля ввода в окне консоли, 4) узнать позицию поля ввода в окне консоли, 5) разместить однострочный текстовый редактор в окне консоли, 6) обеспечить ввод пользователем строки с длиной не больше длины поля ввода, 7) выдать введенную пользователем строку.

## Руководство пользователя

При запуске программы выводится консольное меню. Сверху расположен переключатель позиции редактора. Для выбора нужного функционала, необходимо ввести число, написанное напротив пункта.

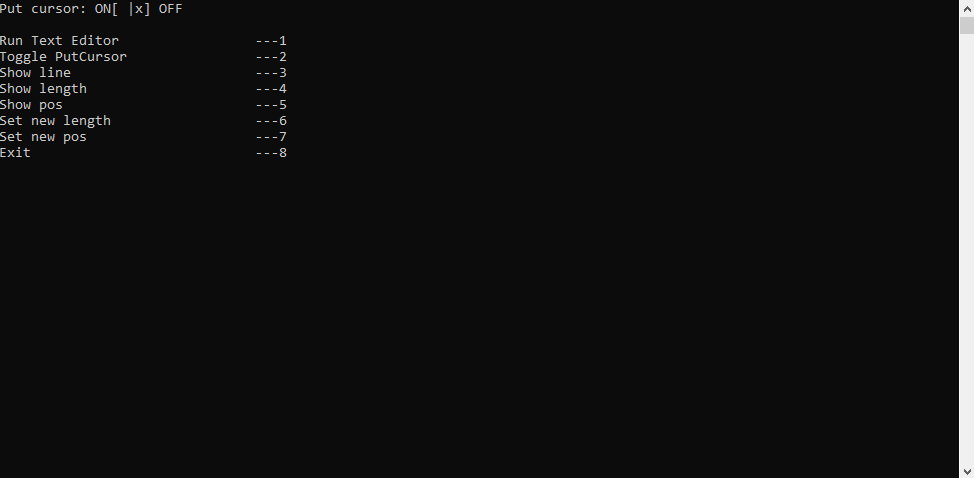


Рис. 6

Функционал пунктов меню интуитивно понятен и соответствует названию. При выборе “Run Text Editor” каретка курсора перемещается в заданные координаты в консоли, пользователь может набирать текст, нажать esc для выхода, backspace, чтобы стереть текст и enter, чтобы запомнить введенные символы.

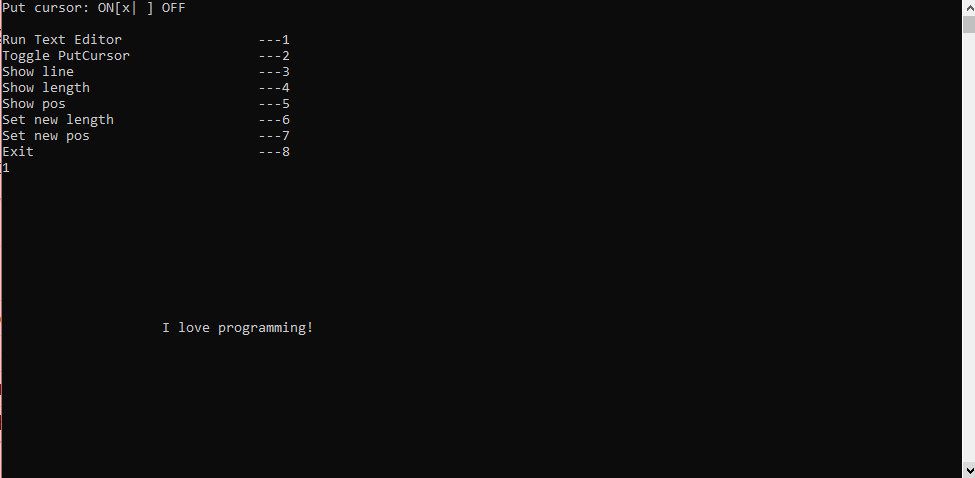


Рис. 7

Длина вводимого текста ограничена, в случае если она равна нулю будет выведено соответствующее сообщение.

## Руководство программиста

Текстовый редактор содержит три поля: string line для хранения последовательности введенных символов, COORD position для запоминания координат ввода и size\_t size максимальную длину ввода. Реализован набор set, get методов, присутствует метод putCursor, передвигающий каретку на выбранные координаты. Основной функционал заключен в методе main. При запуске метода происходит ввод и моментальный вывод последовательности символов в цикле.

# Блок 4

## Постановка задачи

Разработать класс **Контакты**.

Класс должен хранить информацию о контактах владельца. Каждый контакт содержит следующие данные: фамилия; имя; отчество; телефон; день рождения (день, месяц, год); признак, относится ли контакт к избранным. Контакты хранятся упорядоченно по фамилии, имени, отчеству. Фамилия, имя, отчество (ФИО) являются обязательными полями. Данные вводятся на русском языке.

Класс должен предоставлять следующие операции: 1) создать новый контакт, 2) изменить выбранный контакт, 3) найти контакт по ФИО, 4) найти контакт по телефону, 5) выдать все контакты на заданную букву, 6) узнать текущее число контактов, 7) внести контакт в список избранных, 8) удалить контакт из списка избранных, 9) выдать все избранные контакты, 10) удалить контакт, 11) сохранить контакты в файл, 12) считать контакты из файла.

## Руководство пользователя

При запуске программы пользователь попадает в основное меню, навигация в котором осуществляется с помощью стрелок (вверх, вниз) и нажатий клавиш enter и esc. При наведении на какой-либо контакт показывается краткая информация о нем.

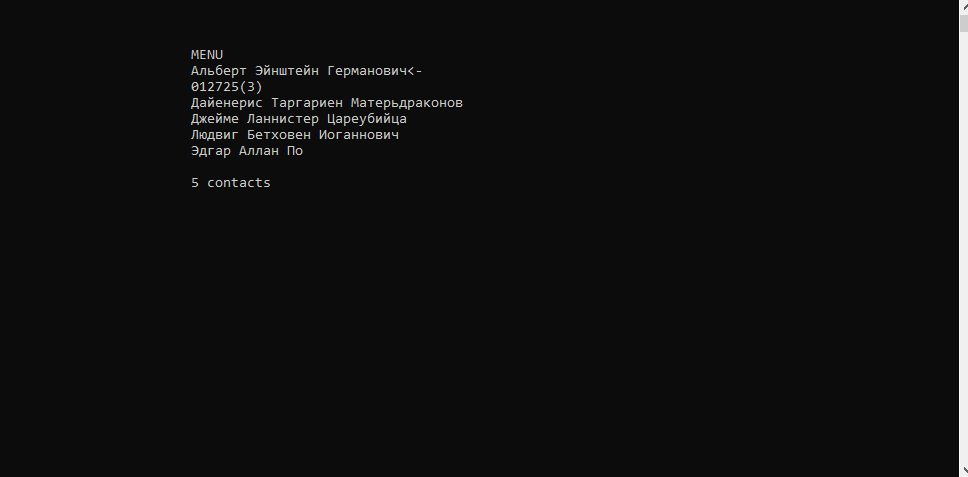


Рис. 7

При выборе нажатии выборе контакта показывается подробная информация о нем, предоставляется ряд возможных функций.

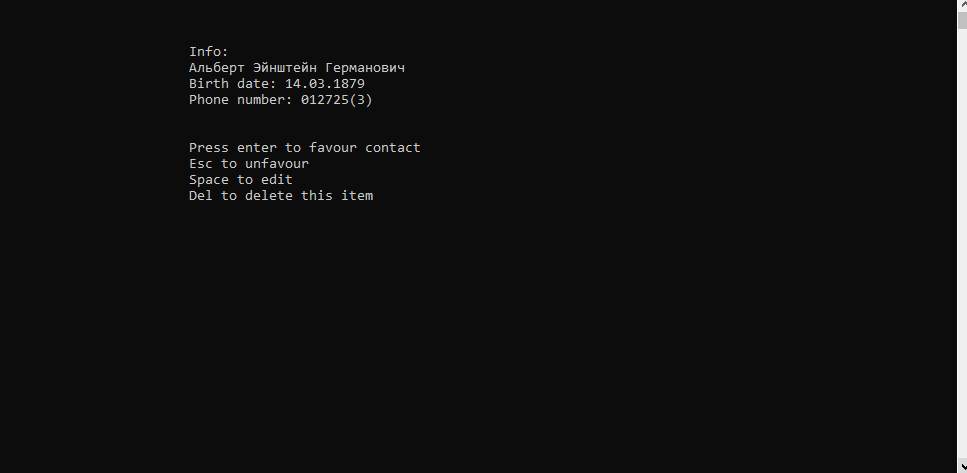


Рис. 8

После входа в первый пункт «MENU», пользователю предлагается выбрать интересующее его действие.

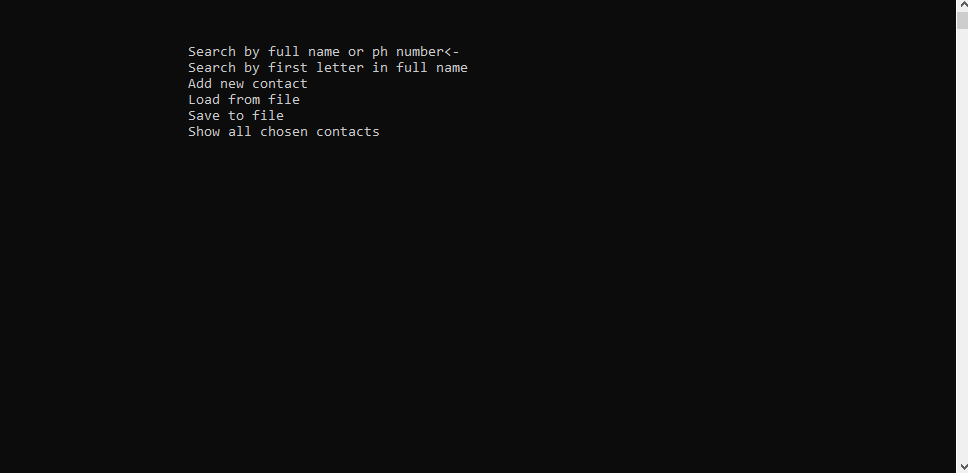


Рис. 9

## Руководство программиста

Два основных класса (info и contacts) связаны отношением агрегации. Первый объект представляет собой набор полей, хранящих информацию о человеке: ФИО, дата рождения, телефонный номер. Реализованы все необходимые методы для взаимодействия с этим классом: get, set и пр. Так же присутствуют ряд операторов: оператор присваивания, поток ввода-вывода. Класс contacts хранит пару векторов info и признаки «избрал ли контакт», «пуста ли контактная база». Представлены методы добавления новых контактов, удаления, изменения информации, поиска по ним и прочие в соответствии с заданием. Каждый метод, который может потенциально нарушить работу программы содержит в себе ряд проверок, для предотвращения ошибок.

# Блок 5

## Постановка задачи

Разработать классы **Кредит** и **Процессинговый центр**.

Класс **Кредит** должен имитировать работу интернет-банка в части управления кредитом для зарплатных клиентов банка (тех, чья зарплата перечисляется работодателем на счета работников, открытые в данном банке). Кредит может выдаваться клиенту на выбранный им срок. Возможные варианты: 1 год, 2 года, 3 года, 5 лет, 15 лет. Считать, что процентные ставки по кредитам зависят только от срока и суммы кредита. Диапазоны сумм: до 100 тыс. рублей, от 100 до 500 тыс. рублей, от 500 тыс. до 1 млн рублей, от 1 до 3 млн рублей. Процентные ставки указываются за 1 год (например, 12% годовых по кредиту на 3 года на сумму от 500 тыс. до 1 млн рублей или 15% годовых по кредиту на 1 год на сумму от 100 до 500 тыс. рублей). Выплаты по кредиту начисляются клиенту раз в месяц. Клиент может выплатить сумму больше или равную начисленной. Клиент может досрочно погасить кредит. Считать, что кредит одобряется банком клиенту, если текущая сумма на его зарплатном счету достаточна для шести ежемесячных выплат по кредиту. После одобрения банком сумма кредита перечисляется на зарплатный счет клиента.

База клиентов хранится в классе **Процессинговый центр**. Считать, что информация о клиенте состоит из номера зарплатного счета (для упрощения – номер счета от «0001» до «9999»), ФИО владельца, суммы на зарплатном счету, (для упрощения – без копеек), информации о кредите, пароля (произвольная строка, выбранная пользователем, с длиной больше 3 символов). Считать, что авторизация клиента в интернет-банке происходит по номеру зарплатного счета и паролю.

Класс **Кредит** должен предоставлять следующие операции: 1) авторизовать клиента, 2) показать информацию о доступных клиенту кредитах 3) проверить наличие взятого в банка кредита, 4) проверить возможность получения выбранного кредита, 5) получить выбранный кредит, 6) показать текущее состояние кредита, 7) погасить начисления по кредиту, 8) досрочно погасить кредит.

Все операции должны сопровождаться необходимыми проверками на указанные выше ограничения.

Класс **Процессинговый центр** должен использоваться для поддержки работы класса **Кредит** и может быть разработан в минимально-необходимом варианте.

## Руководство пользователя

При запуске появится окно, предлагающее зарегистрироваться в системе.



Рис. 10

После чего пользователь будет зарегистрирован и автоматически авторизован. Появляется основное меню программы со всем функционалом. Для выбора необходимо ввести интересующий пользователя пункт.

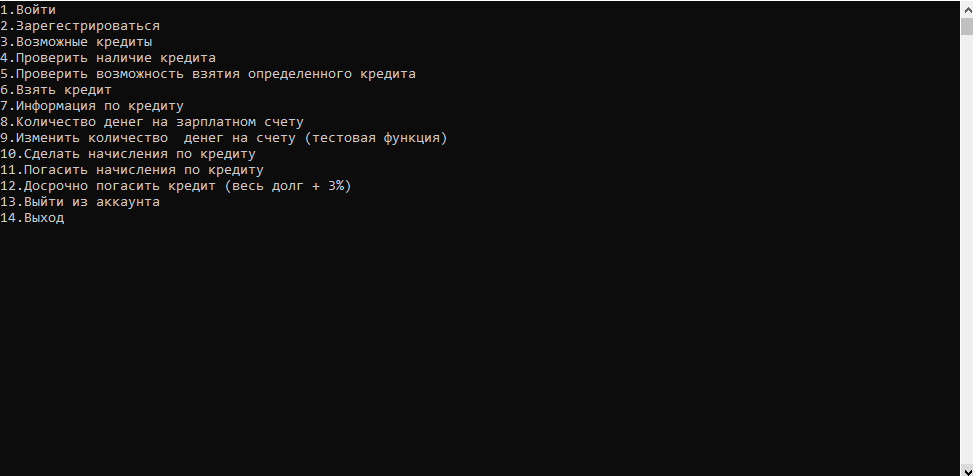


Рис. 11

## Руководство программиста

В программе представлена система объектов, состоящая из 2х структур и 2х классов, связанных отношением агрегирования. Структура creditInfo используется для получения информации о текущем кредите, структура client содержит ФИО клиента, пароль, количество денег на счету, задолженность, начисления по кредиту, оставшееся время кредита (в месяцах) и процентную ставку по кредиту. Класс processingCenter содержит вектор client и необходим для корректной работы с данными клиентов. Имеется ряд get, set, а также нескольких bool “является ли” методов. Класс credit содержит processingCenter поле, id клиента с которым в данный момент ведется работа, и bool признак isSessionActive. Предоставлены все необходимые методы: добавить нового клиента, авторизоваться, взять кредит, заплатить кредит и проч. Каждый метод, который может потенциально нарушить работу программы содержит в себе ряд проверок, для предотвращения ошибок.

# Блок 6

## Постановка задачи

Разработать систему классов и реализовать с ее помощью игру **Змейка**.

Играет один игрок (человек), управляющий «змейкой».Игра идет на прямоугольном поле N x M клеток. Поле ограничено «стенами» так что вместе со стенами размер поля – (N + 2) x (M + 2) клеток.

При старте игры змейка имеет длину 5 клеток, форму в виде горизонтального отрезка и располагается в произвольном месте поля, не пересекая и не касаясь стен. При старте игры «голова» змейки располагается слева, «хвост» справа. Голова змейки окрашена в цвет, отличный от цвета остальных клеток ее тела. При старте игры в произвольной клетке поля (не совпадающей с клетками, занятыми змейкой) возникает «пища». При запуске игрового процесса (по специальной команде или автоматически при старте игры) змейка начинает автоматическое движение влево с некоторой заданной скоростью.

Движение заключается в том, что за каждый такт голова змейки перемещается на одну клетку в текущем направлении движения, а клетка, в которой располагался хвост, становится пустой. Игрок может сменить направление движения змейки с помощью клавиш-«стрелок» (вверх, вниз, влево, вправо).

Если на текущем такте движения голова змейки должна будет занять клетку стены или клетку, которая уже занята любой из клеток ее тела, игра прекращается и считается проигранной. Задача игрока вырастить змейку до заданного при старте игрового процесса размера. Змейка вырастает в длину на одну клетку (с хвоста) при каждом поглощении пищи, т.е. в тот момент, когда ее голова на очередном такте движения занимает клетку, в которой расположена пища. На этом же такте в произвольном месте игрового поля (не совпадающей с клетками, занятыми змейкой) снова появляется пища.

Если змейка выросла до заданной при старте игрового процесса длины, игра считается выигранной.

## Руководство пользователя

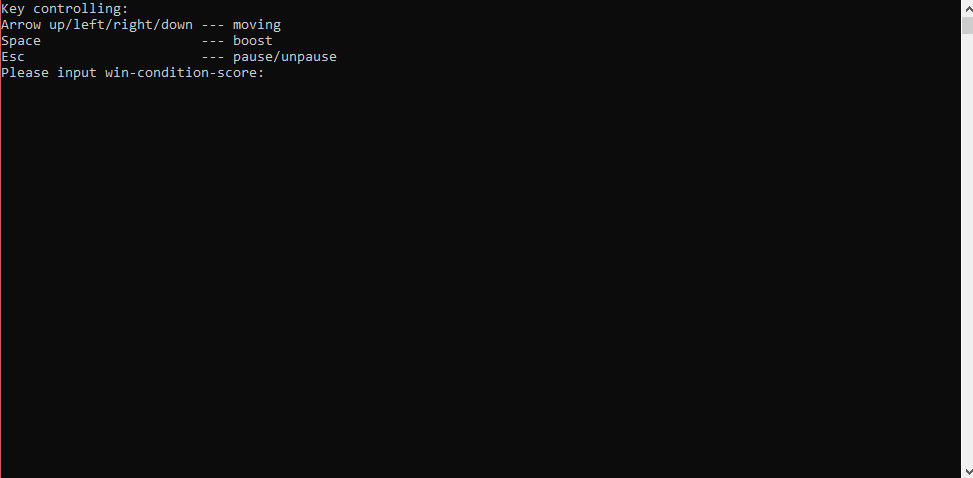
При запуске программы появляется консольное окно с инструкцией к программе. Пользователь вводит условие победы, размер поля.

Рис. 12

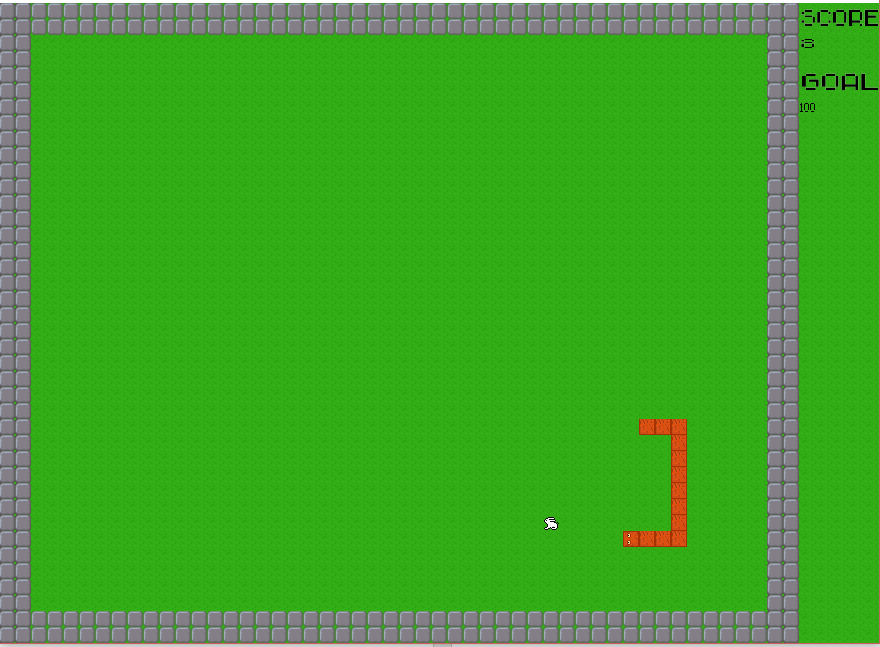
После, консольное окно закрывается, открывается новое, в котором происходит игровой процесс.

Рис. 13

В правой части экрана написан текущий счет и счет, необходимый для победы.

## Руководство программиста

Программа реализована с помощью системы классов, связанной отношением агрегирования.

Класс drawer позволяет работать с библиотекой SDL 2.0, выполняя следующие задачи: загрузка текстур, загрузка шрифтов, создание окна, рендер. Присутствуют советующие служебные поля и методы, такие как указатели на текстуры, указатель на окно приложения и проч. Более подробно об устройстве SDL можно прочитать в документации к данной библиотеке.

Класс snakeObjects содержит информацию об направлении движения головы змейки, положение на игровом пространстве тела, расположение еды, размер поля. Метод move отвечает за один ход игры: координаты туловища смещаются на 1 клетку в заданном направлении, происходят проверки на признак «съедено ли яблоко», если да – выполняются необходимые действия: длина змейки увеличивается, генерируется новая еда.

Класс mainGame имеет следующие поля: ссылки на объект drawer и snakeObjects, текущий и необходимый для победы счета, bool переменная gameResult. Игровой процесс происходит в цикле в соответствующем методе gameLoop.

# Заключение

В ходе работы были спроектированы и реализованы разнообразные программы с использованием концепции объектно-ориентированного программирования, тем самым были приобретены важные навыки в данной области.

# Литература

1. Design Patterns: Elements of Reusable Object-Oriented Software («Приёмы объектно-ориентированного проектирования. Паттерны проектирования») Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес, ISBN 0-201-63361-2, 5-469-01136-4
2. Code Complete: Second Edition («Совершенный код. Мастер-класс»), Стив Макконнелл, ISBN 978-5-7502-0064-1
3. Документация к библиотеке SDL [wiki.libsdl.org](file:///D:\download\chrome\wiki.libsdl.org)

# Приложение

## Код работы №6:

### drawer.h

#pragma once

#include <SDL.h>

#include <SDL\_image.h>

#include <string>

#include <SDL\_ttf.h>

class drawer

{

SDL\_Window \*gWindow = NULL;

SDL\_Renderer \*gRenderer = NULL;

SDL\_Texture \* mTexture = NULL;

SDL\_Texture \*textTexture = NULL;

TTF\_Font \*gFont = NULL;

SDL\_Rect gSpriteClips[10];

SDL\_Rect \*getSpriteClips();

void freeT();

void free();

public:

int SCREENH;

int SCREENW;

void renderPresent();

bool init();

bool loadMedia(); //loads snake sprite into rectangles

bool loadMediaText();

void render(int \_x, int \_y, int i);

void renderText(int \_x, int \_y, int sizex, int sizey, std::string t);

bool loadFromFile(std::string path); //loads any texture into mTexture

bool loadFromRenderedText(std::string textureText, SDL\_Color textColor);

void close();

drawer();

~drawer();

};

### mainGame.h

#pragma once

#include "drawer.h"

#include "snakeObjects.h"

#include <iostream>

#include <sstream>

class mainGame

{

snakeObjects &snake;

drawer &dwr;

short unsigned int scoreCurrent = 0;

short unsigned int scoreGoal;

bool gameResult = 0;

int step(short int dir); // returns: 0 when head hits the walls, 1 when snake moved on field w/out any events, 2 when snake ate an apple

void drawScore();

void initDraw();

public:

void setFieldSize(short int x, short int y);

void gameLoop(short int score);

mainGame(snakeObjects &\_snake, drawer &\_dwr);

~mainGame();

};

### snakeObjects.h

#pragma once

#include <vector>

#include <ctime>

#define STARTBLOCKS 5

struct point

{

short int x;

short int y;

point& operator =(const point& s);

bool operator == (const point& s) const;

};

class snakeObjects

{

point fieldSize;

std::vector <point> snake;

point apple;

short int lastDirection = 3; // 1 - right, 2 - up, 3 - left, 4 - down

void createApple();

public:

snakeObjects();

void setFieldSize(short int fieldSizeX, short int fieldSizeY);

void init();

point getHead() const;

point getTail() const;

std::vector <point>& getSnake();

int move(short int direction); // returns: 0 when head hits the walls, 1 when snake moved on field w/out any events, 2 when snake ate an apple

point getApple() const;

bool headAteTale() const;

short int size() const;

};

### drawer.cpp

#include "drawer.h"

drawer::drawer()

{ }

void drawer::renderPresent()

{

SDL\_RenderPresent(gRenderer);

}

bool drawer::init()

{

bool success = true;

//Initialize SDL

if (SDL\_Init(SDL\_INIT\_VIDEO) < 0)

{

printf("SDL could not initialize! SDL Error: %s\n", SDL\_GetError());

success = false;

}

else

{

//Set texture filtering to linear

if (!SDL\_SetHint(SDL\_HINT\_RENDER\_SCALE\_QUALITY, "1"))

{

printf("Warning: Linear texture filtering not enabled!");

}

gWindow = SDL\_CreateWindow("Snake game v 1.0", SDL\_WINDOWPOS\_UNDEFINED, SDL\_WINDOWPOS\_UNDEFINED, SCREENW \* 16, SCREENH \* 16, SDL\_WINDOW\_SHOWN);

if (gWindow == NULL)

{

printf("Window could not be created! SDL Error: %s\n", SDL\_GetError());

success = false;

}

else

{

//Create renderer for window

gRenderer = SDL\_CreateRenderer(gWindow, -1, SDL\_RENDERER\_ACCELERATED);

if (gRenderer == NULL)

{

printf("Renderer could not be created! SDL Error: %s\n", SDL\_GetError());

success = false;

}

else

{

//Initialize renderer color

SDL\_SetRenderDrawColor(gRenderer, 0xFF, 0xFF, 0xFF, 0xFF);

//Initialize PNG loading

int imgFlags = IMG\_INIT\_PNG;

if (!(IMG\_Init(imgFlags) & imgFlags))

{

printf("SDL\_image could not initialize! SDL\_image Error: %s\n", IMG\_GetError());

success = false;

}

if (TTF\_Init() == -1)

{

printf("SDL\_ttf could not initialize! SDL\_ttf Error: %s\n", TTF\_GetError());

success = false;

}

}

}

}

return success;

}

void drawer::free()

{

if (mTexture != NULL)

{

SDL\_DestroyTexture(mTexture);

mTexture = NULL;

}

}

void drawer::freeT()

{

if (textTexture != NULL)

{

SDL\_DestroyTexture(textTexture);

textTexture = NULL;

}

}

void drawer::renderText(int \_x, int \_y, int sizex, int sizey, std::string t)

{

SDL\_Rect renderQuad = { \_x \* 16, \_y \* 16, sizex, sizey };

SDL\_Color textColor = { 0, 0, 0 };

if (!loadFromRenderedText(t, textColor))

{

printf("Failed to render text texture!\n");

}

//Render to screen

SDL\_RenderCopy(gRenderer, textTexture, NULL, &renderQuad);

}

bool drawer::loadFromRenderedText(std::string textureText, SDL\_Color textColor)

{

//Get rid of preexisting texture

freeT();

//Render text surface

SDL\_Surface\* textSurface = TTF\_RenderText\_Solid(gFont, textureText.c\_str(), textColor);

if (textSurface == NULL)

{

printf("Unable to render text surface! SDL\_ttf Error: %s\n", TTF\_GetError());

}

else

{

//Create texture from surface pixels

textTexture = SDL\_CreateTextureFromSurface(gRenderer, textSurface);

if (textTexture == NULL)

{

printf("Unable to create texture from rendered text! SDL Error: %s\n", SDL\_GetError());

}

//Get rid of old surface

SDL\_FreeSurface(textSurface);

}

//Return success

return textTexture != NULL;

}

bool drawer::loadMediaText()

{

//Loading success flag

bool success = true;

//Open the font

gFont = TTF\_OpenFont("11719.ttf", 28);

if (gFont == NULL)

{

printf("Failed to load lazy font! SDL\_ttf Error: %s\n", TTF\_GetError());

success = false;

}

else

{

//Render text

SDL\_Color textColor = { 0, 0, 0 };

if (!loadFromRenderedText("Score:", textColor))

{

printf("Failed to render text texture!\n");

success = false;

}

}

return success;

}

void drawer::render(int \_x, int \_y, int i)

{

//Set rendering space and render to screen

SDL\_Rect renderQuad = { \_x \* 16, \_y \* 16, 16, 16 };

//Render to screen

SDL\_RenderCopy(gRenderer, mTexture, &gSpriteClips[i], &renderQuad);

}

bool drawer::loadFromFile(std::string path)

{

bool success = 1;

//Get rid of preexisting texture

free();

//The final texture

SDL\_Texture\* newTexture = NULL;

//Load image at specified path

SDL\_Surface\* loadedSurface = IMG\_Load(path.c\_str());

if (loadedSurface == NULL)

success = 0;

else

{

//Color key image

SDL\_SetColorKey(loadedSurface, SDL\_TRUE, SDL\_MapRGB(loadedSurface->format, 0, 0xFF, 0xFF));

//Create texture from surface pixels

newTexture = SDL\_CreateTextureFromSurface(gRenderer, loadedSurface);

if (newTexture == NULL)

success = 0;

else

SDL\_FreeSurface(loadedSurface);

}

mTexture = newTexture;

return success;

}

bool drawer::loadMedia()

{

bool success = true;

//Load sprite sheet texture

if (!loadFromFile("snaketexture2.png"))

success = false;

else

for (int i = 0, k = 0; i < 2; i++)

for (int j = 0; j < 4; j++, k++)

{

gSpriteClips[k].x = j \* 16;

gSpriteClips[k].y = i \* 16;

gSpriteClips[k].w = 16;

gSpriteClips[k].h = 16;

}

return success;

}

SDL\_Rect \* drawer::getSpriteClips()

{

return gSpriteClips;

}

void drawer::close()

{

free();

freeT();

SDL\_DestroyRenderer(gRenderer);

SDL\_DestroyWindow(gWindow);

gWindow = NULL;

gRenderer = NULL;

TTF\_CloseFont(gFont);

gFont = NULL;

IMG\_Quit();

SDL\_Quit();

}

drawer::~drawer()

{

close();

}

### mainGame.cpp

#include "mainGame.h"

mainGame::mainGame(snakeObjects &\_snake, drawer &\_dwr) :

snake(\_snake), dwr(\_dwr) { }

int mainGame::step(short int dir) // returns: 0 when head hits the walls or eats itself, 1 when snake moved on field w/out any events, 2 when snake ate an apple

{

point \_tale = snake.getTail();

point \_head = snake.getHead();

point apple = snake.getApple();

int k = snake.move(dir);

if (k == 0)

return 0;

if (snake.headAteTale())

return 0;

point tale\_ = snake.getTail();

point head\_ = snake.getHead();

dwr.render(apple.x, apple.y, 5);

switch (dir)

{

case 1:dwr.render(head\_.x, head\_.y, 1); break;

case 2:dwr.render(head\_.x, head\_.y, 0); break;

case 3:dwr.render(head\_.x, head\_.y, 3); break;

case 4:dwr.render(head\_.x, head\_.y, 2); break;

}

dwr.render(\_head.x, \_head.y, 4);

if (!(\_tale == head\_))

dwr.render(\_tale.x, \_tale.y, 6);

return k;

}

void mainGame::setFieldSize(short int x, short int y)

{

snake.setFieldSize(x, y);

dwr.SCREENW = x + 5;

dwr.SCREENH = y;

}

void mainGame::initDraw()

{

for (int i = 0; i < dwr.SCREENW; i++)

{

for (int j = 0; j < dwr.SCREENH; j++)

{

dwr.render(i, j, 6);

}

}

std::vector <point> ssnake = snake.getSnake();

dwr.render(ssnake[0].x, ssnake[0].y, 3);

for (int i = 0; i < dwr.SCREENW - 5; i++)

{

for (int j = 0; j < dwr.SCREENH; j++)

{

dwr.render(0, j, 7);

dwr.render(1, j, 7);

dwr.render(dwr.SCREENW - 6, j, 7);

dwr.render(dwr.SCREENW - 7, j, 7);

}

dwr.render(i, 0, 7);

dwr.render(i, 1, 7);

dwr.render(i, dwr.SCREENH - 1, 7);

dwr.render(i, dwr.SCREENH - 2, 7);

}

for (int i = 1; i < ssnake.size(); i++)

{

dwr.render(ssnake[i].x, ssnake[i].y, 4);

}

dwr.renderText(dwr.SCREENW - 5, 0, 80, 28, "Score");

drawScore();

dwr.renderText(dwr.SCREENW - 5, 4, 80, 28, "Goal");

std::ostringstream ost;

ost << scoreGoal;

std::string s\_num = ost.str();

dwr.render(dwr.SCREENW - 5, 2, 6);

dwr.renderText(dwr.SCREENW - 5, 6, 16, 16, s\_num);

dwr.renderPresent();

}

void mainGame::drawScore()

{

std::ostringstream ost;

ost << scoreCurrent;

std::string s\_num = ost.str();

dwr.render(dwr.SCREENW - 5, 2, 6);

dwr.renderText(dwr.SCREENW - 5, 2, 16, 16, s\_num);

}

void mainGame::gameLoop(short int score)

{

if (dwr.init())

{

if (dwr.loadMedia() && dwr.loadMediaText())

{

scoreGoal = score;

snake.init();

bool quit = false;

SDL\_Event e;

short direct = 3;

int sleepDelay;

bool pause = 1;

initDraw();

while (!quit)

{

sleepDelay = 100;

while (SDL\_PollEvent(&e) != 0)

{

switch (e.type)

{

case SDL\_QUIT:

quit = true;

break;

case SDL\_KEYDOWN:

switch (e.key.keysym.sym)

{

case SDLK\_UP:

direct = 2;

break;

case SDLK\_DOWN:

direct = 4;

break;

case SDLK\_LEFT:

direct = 3;

break;

case SDLK\_RIGHT:

direct = 1;

break;

case SDLK\_SPACE:

sleepDelay = 40;

break;

case SDLK\_ESCAPE:

pause = !pause;

break;

}

break;

default:

break;

}

}

if (!pause) //gamebody

{

switch (step(direct))

{

case 0:

quit = 1;

break;

case 1:

break;

case 2:

scoreCurrent++;

drawScore();

break;

}

if (scoreCurrent == scoreGoal) //player won

{

gameResult = 1;

quit = 1;

}

dwr.renderPresent();

SDL\_Delay(sleepDelay);

}

else

SDL\_Delay(250);

}

if (gameResult)

dwr.renderText(dwr.SCREENW / 2 - 170 / 16, dwr.SCREENH / 2 - 100 / 16, 350, 100, "GAME IS WON!");

else

dwr.renderText(dwr.SCREENW / 2 - 170 / 16, dwr.SCREENH / 2 - 100 / 16, 350, 100, "GAME IS OVER!");

dwr.renderPresent();

SDL\_Delay(1500);

}

}

}

mainGame::~mainGame()

{

}

### snakeObjects.cpp

#include "snakeObjects.h"

point& point::operator =(const point& s)

{

x = s.x;

y = s.y;

return \*this;

}

bool point::operator == (const point& s) const

{

return x == s.x && y == s.y;

}

void snakeObjects::createApple()

{

bool i = 1;

srand(time(0));

while (i)

{

apple.x = 2 + rand() % (fieldSize.x - 4);

apple.y = 2 + rand() % (fieldSize.y - 4);

bool flag = 1;

for (int j = 0; j < snake.size() && flag; j++)

{

if (snake[j] == apple)

flag = 0;

}

if (flag)

i = 0;

}

}

snakeObjects::snakeObjects()

{ }

void snakeObjects::setFieldSize(short int fieldSizeX, short int fieldSizeY)

{

fieldSize.x = fieldSizeX;

fieldSize.y = fieldSizeY;

}

void snakeObjects::init()

{

point part;

srand(time(0));

part.x = 2 + rand() % (fieldSize.x - 8);

part.y = 2 + rand() % (fieldSize.y - 4);

snake.push\_back(part);

for (int i = 0; i < STARTBLOCKS - 1; i++)

{

part.x++;

snake.push\_back(part);

}

createApple();

}

point snakeObjects::getHead() const

{

return snake[0];

}

point snakeObjects::getTail() const

{

return snake[snake.size() - 1];

}

std::vector <point>& snakeObjects::getSnake()

{

return snake;

}

int snakeObjects::move(short int direction) // returns: 0 when head hits the walls, 1 when snake moved on field w/out any events, 2 when snake ate an apple

{

int k = 1;

if (lastDirection + 2 == direction || lastDirection - 2 == direction)

direction = lastDirection;

point tmp1 = snake[0];

switch (direction)

{

case 1:

snake[0].x++;

break;

case 2:

snake[0].y--;

break;

case 3:

snake[0].x--;

break;

case 4:

snake[0].y++;

break;

}

if (snake[0].x == 1 || snake[0].x == fieldSize.x - 2 || snake[0].y == 1 || snake[0].y == fieldSize.y - 2)

return 0;

point tmp2 = tmp1;

for (int i = 1; i < snake.size(); i++)

{

tmp2 = snake[i];

snake[i] = tmp1;

tmp1 = tmp2;

}

if (snake[0] == apple)

{

k = 2;

snake.push\_back(tmp1);

createApple();

}

lastDirection = direction;

return k;

}

point snakeObjects::getApple() const

{

return apple;

}

bool snakeObjects::headAteTale() const

{

for (int i = 1; i < snake.size(); i++)

{

if (snake[0] == snake[i])

return 1;

}

return 0;

}

short int snakeObjects::size() const

{

return snake.size();

}

### main.cpp

#include "mainGame.h"

#include <Windows.h>

#undef main

int main(int argc, char\* args[])

{

snakeObjects snake; //this class contains information about snake, food and also int contains size of the game field

drawer Drawer; // this class provides ability to work with sdl: load textures, render them, create windows etc

mainGame gameObj(snake, Drawer); //this one is the main class it draws image, controls keyboard

short int score;

short int w;

short int h;

do

{

if (::IsWindowVisible(::GetConsoleWindow()) != FALSE)

::ShowWindow(::GetConsoleWindow(), SW\_SHOW);

system("cls");

std::cout << "Key controlling: \nArrow up/left/right/down --- moving\nSpace\t\t\t --- boost\nEsc\t\t\t --- pause/unpause\nPlease input win-condition-score: ";

std::cin >> score;

std::cout << std::endl << "Input width and height of the field (in blocks, 1 block = 16 px, > 17) ";

std::cout << "\*\*50-40 will be just fine\*\*" << std::endl;

std::cin >> w >> h;

} while (score < 1 || score>300 || w <= 17 || h <= 17);

ShowWindow(::GetConsoleWindow(), SW\_HIDE);

gameObj.setFieldSize(w, h);

gameObj.gameLoop(score);

return 0;

}