Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования(ЯП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

ИГРОВОЕ ПРИЛОЖЕНИЕ “SNAKE”

Студент: гр. 551006 Жизневский В.С.

Руководитель: Марина И. М.

Минск 2016

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc469528394)

[1 СРАВНЕНИЕ АНАЛОГОВ 8](#_Toc469528395)

[1.1 Slither.io 8](#_Toc469528396)

[1.2 Snake Rewind 9](#_Toc469528397)

[2 ПОСТАНОВКА ЗАДАЧИ 11](#_Toc469528398)

[3 ПРОЕКТИРОВАНИЕ. РАЗРАБОТКА 12](#_Toc469528399)

[3.1 Описание ресурсов 12](#_Toc469528400)

[3.1.1 C++ 12](#_Toc469528401)

[3.1.2 Microsoft Visual Studio 12](#_Toc469528402)

[3.1.3 Bass.dll 13](#_Toc469528403)

[3.1.4 openGL. Freeglut 13](#_Toc469528404)

[3.2 Структура приложения 14](#_Toc469528405)

[3.3 Основной игровой процесс 15](#_Toc469528406)

[3.3.1 Класс mainGame 16](#_Toc469528407)

[3.3.2 Класс TSnake 19](#_Toc469528408)

[3.3.3 Класс TFood 21](#_Toc469528409)

[3.4 Музыкальное и звуковое сопровождение игрового процесса 23](#_Toc469528410)

[3.4.1 Класс TSound 23](#_Toc469528411)

[3.5 Уровни сложности 25](#_Toc469528412)

[3.6 Генерация случайных препятствий на игровой сцене 26](#_Toc469528413)

[3.6.1 Корректирование стен 27](#_Toc469528414)

[3.6.2 Класс TWall 28](#_Toc469528415)

[3.7 Таблица рекордов 30](#_Toc469528416)

[3.7.1 Сортировка 31](#_Toc469528417)

[3.7.2 Обновление рекорда 32](#_Toc469528418)

[3.8 Управление игровым процессом 32](#_Toc469528419)

[3.9 Идентификация игрока 33](#_Toc469528420)

[4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 34](#_Toc469528421)

[4.1 Системные требования 34](#_Toc469528422)

[4.2 Запуск программы. Начало игры 34](#_Toc469528423)

[4.3 Правила игры 34](#_Toc469528424)

[4.4 Управление 34](#_Toc469528425)

[ЗАКЛЮЧЕНИЕ 35](#_Toc469528426)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 36](#_Toc469528427)

[ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ 37](#_Toc469528428)

# **ВВЕДЕНИЕ**

Индустрия развлечений оказалась одной из первых, занявших свою крепкую позицию в период быстрого развития компьютерных технологий наряду с военной и медицинской. Люди поняли, что намного удобнее хранить и воспроизводить какую-либо игру (а может, даже несколько) на одном устройстве, не выходя из дома. Так и появились компьютерные игры.

Ни для кого не секрет, что компьютерные игры прочно заняли свою позицию в современной индустрии развлечений. Существуют попытки выделить компьютерные игры как отдельную область искусства, наряду с театром, кино и т.п. Разработка игр может оказаться не только увлекательным, но и прибыльным делом, примеров этому предостаточно в истории.

Первые игры отличались простотой интерфейса и логики, но со временем они становились все сложнее и сложнее, над их созданием уже работал не один человек, а целая команда разработчиков. Первые примитивные компьютерные и видео игры были разработаны в 1950-х и 1960-х годах. Они работали на таких платформах, как осциллографы, университетские мейнфреймы и компьютеры EDSAC. Самой первой компьютерной игрой стал симулятор ракеты, созданный в 1942 году Томасом Голдсмитом Младшим (англ. Thomas T. Goldsmith Jr.) и Истл Рей Менном (англ. Estle Ray Mann). Позже, в 1952 году, появилась программа "OXO", имитирующая игру "крестики-нолики", созданная А.С. Дугласом как часть его докторской диссертации в Кембриджском Университете. Игра работала на большом университетском компьютере, известном как EDSAC (Electronic Delay Storage Automatic Calculator). В настоящее время, разработка игры - это многомиллионный процесс, в котором задействована целая команда разработчиков, сложные современные технологии и даже маркетинговые ходы.

Первая игра серии Bomberman была выпущена в 1983 году для ряда домашних компьютеров и игровой консоли NES. В Европе игра распространялась под названием Eric and the Floaters. Игра получила огромную популярность и стала началом одной из крупнейших серий видеоигр, в настоящее время включающей более 60. Игры серии разрабатывались и издавались как самой Hudson Soft, так и рядом других компаний и выходили на большинстве существующих игровых систем.

Большинство игр серии выполнены в жанре лабиринтной аркады. Игрок управляет персонажем, находящимся в лабиринте, состоящем из разрушаемых и неразрушаемых стен. Он может оставлять бомбу, взрывающуюся через небольшое время и разрушающую стены рядом с ней. Специальные бонусы могут увеличить количество одновременно оставляемых бомб, дальность их взрыва, скорость перемещения героя, дать возможность взрыва бомб по нажатию кнопки, невосприимчивость от взрыва бомб, прохождение сквозь разрушаемые стены. На уровне присутствуют противники. В некоторых играх серии целью игры является нахождение скрытой за одной из разрушаемых стен двери, ведущей в следующий уровень с предварительным уничтожением врагов. Другие игры рассчитаны на многопользовательскую игру на одном экране, целью в них является победа над всеми противниками.

В связи с популярностью и легендарностью данной игры целью данной курсовой работы было выбрано создание многопользовательского ремейка оригинальной версии игры “Bomberman” для NES 1985 года выпуска.

# **СРАВНЕНИЕ АНАЛОГОВ**

Как уже было упомянуто ранее, серия игр Bomberman насчитывает около 60 игр. Bomberman побывал на большом количестве платформ (Gameboy, Sega, Sony PS 1, 2, 3, мобильные телефоны и т.д) и, следовательно, на данный момент имеется много аналогов данному программному средству. Далее будут рассмотрены несколько из них.

## **Bomberman для NES**

Bomberman NES (рисунок 1.1) — первая из игр серии, выпущенная для консоли NES. Разработчиком является Hudson Soft. В Европе известна по названием Eric and the Floaters.

Главный персонаж игры — Бомбермен (представленный как робот, которому нужно сбежать с завода по производству бомб). Задача героя — расставлять по уровню бомбы, взрывать стены, мешающие проходу и врагов, которые убивают прикосновением.

Главная задача Бомбермена — за отведенное на уровне время уничтожить врагов и найти среди стен дверь, ведущую на другой уровень. Если время заканчивается, это не карается смертью, как в других играх, однако игрок не получает бонусные очки за завершение уровня, и в дополнение ко всему, неожиданно появляется множество врагов-«копеек», избежать столкновения с которыми практически невозможно.

Через каждые 5 уровней игрок попадает в пустой бонус-уровень, где бесконечно появляются враги, задача героя быстро их всех уничтожить, зарабатывая на этом очки.

Всего в игре 50 уровней.

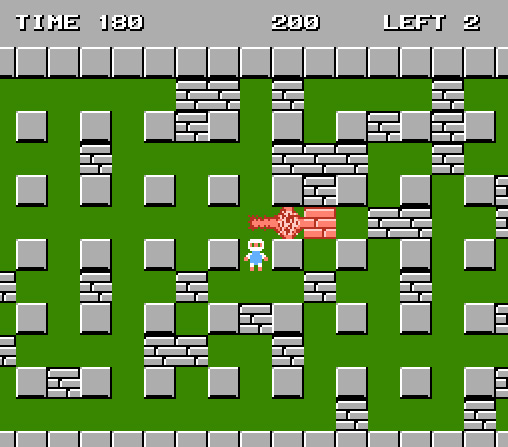


Рисунок 1.1 – Bomberman NES

Несмотря на то, что игра уже вышла давно и графически устарела, она всё равно остается ещё в какой-то мере популярной и востребованной.

## **Super Bomberman R**

Super Bomberman R (рисунок 1.2) была выпущена 3 марта 2017 года компанией Konami для консоли Nintendo Switch.

В **Super Bomberman R** простое правило игры остается тем же самым: управлять главным героем (Bomberman), метать бомбы и сражаться с врагами и соперниками.

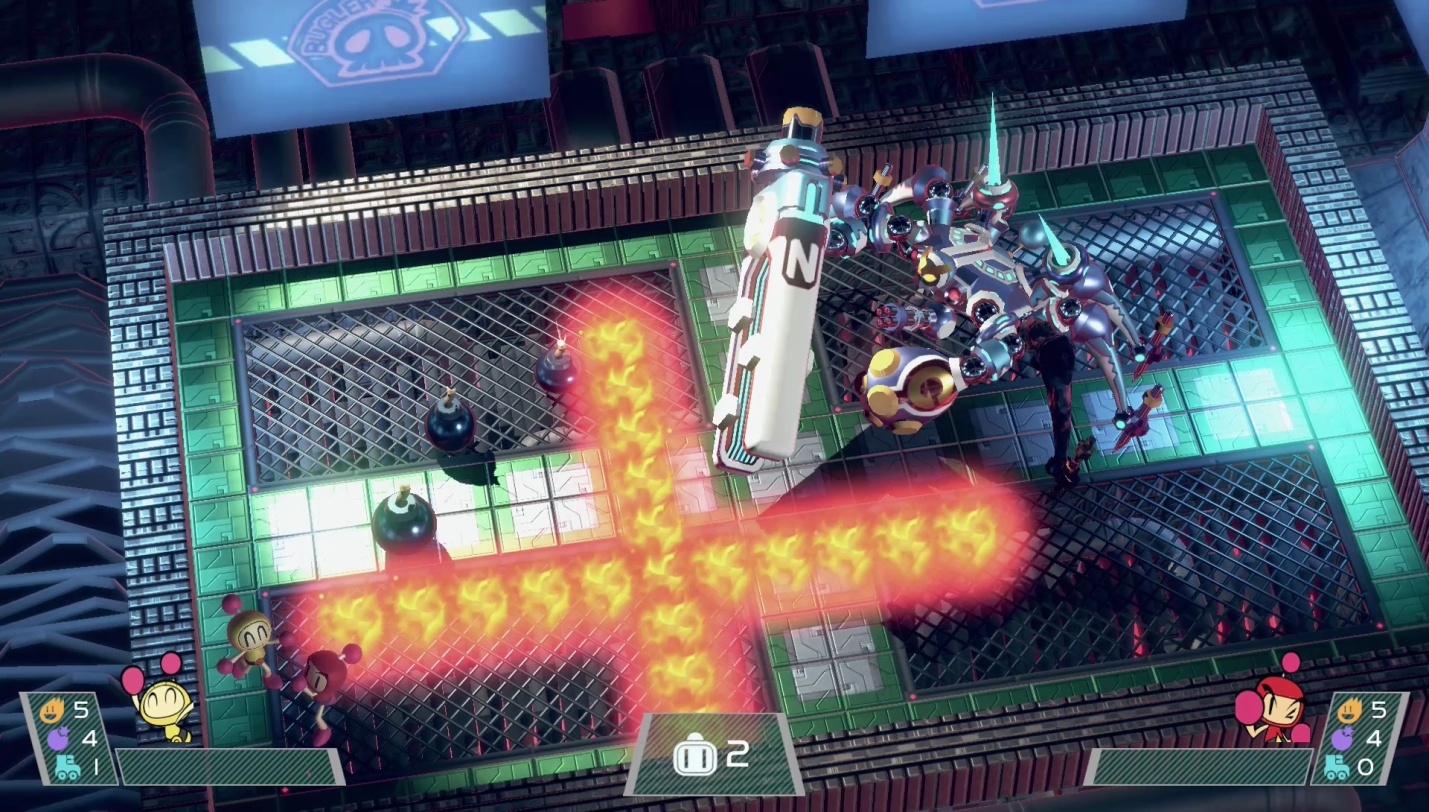


Рисунок 1.2 – Super Bomberman R

Присутствует соревновательный режим битвы “Battle” (рисунок 1.3) а сюжетную кампанию в режиме одиночной игры “Story” также возможно будет проходить совместно: каждый из 50 уровней можно зачищать один за другим вдвоём. Кроме того, появилось новое правило, когда условие прохождения уровня заключается не только в уничтожении всех врагов.



Рисунок 1.3 – Super Bomberman R режим “Battle”

Особенности игры:

* bomberman возвращается с той же игровой механикой и управлением, но со свежим идеями, обогатившими классический выпуск;
* фотореалистичная графика;
* братья и сёстры Bomberman и хорошо известные враги возвращаются с яркими красками индивидуальности;
* трёхмерные карты с динамическими разрушаемыми объектами окружения;
* режим битвы для максимум 8 игроков, локальные сражения и сетевые баталии;
* богатая сюжетная история со звёздным составом актёров озвучки.

Следует отметить, что на данный момент это последняя игра серии, выпущенная компанией Konami, которая на данный момент владеет правами на данную франшизу.

На основе анализа аналогов можно сделать вывод о том, какие уникальные идеи для разнообразия геймплея можно реализовать в данном курсовом проекте.

# **ПОСТАНОВКА ЗАДАЧИ**

Разработать игровое приложение “Snake”, реализующее основные принципы одноименной игры.

Игровое приложение “Snake” должно реализовывать следующий функционал:

* основной игровой процесс;
* музыкальное и звуковое сопровождение игрового процесса;
* наличие разных уровней сложности;
* генерация для некоторых уровней сложности случайных препятствий на игровой сцене;
* подсчёт набранных очков. Система подсчёта очков;
* таблица рекордов и связанный с ней функционал (сортировка элементов таблицы, механизм обновления рекорда и т.д.);
* управление пользователем игровым процессом (регулировка уровня громкости, остановка/продолжение игрового процесса, вызов справки);
* идентификация игрока с целью последующего внесения его результата в таблицу рекордов.

# **ПРОЕКТИРОВАНИЕ. РАЗРАБОТКА**

## **Описание ресурсов**

Разработка данного приложения будет происходить в IDE Microsoft Visual Studio 2013 на языке C++ при использовании стандартных библиотек языка C++, встроенных в Visual Studio, а также библиотеки для работы со звуком bass.dll и библиотеки freeglut для работы с OpenGL.

# C++

C**++** — компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков.

C**++** широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие.

Язык появился в 1983 году. Автором является Бьёрн Страуструп.

# Microsoft Visual Studio

Microsoft Visual Studio — продукт компании Microsoft, включающий интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данный продукт позволяет разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

# Bass.dll

BASS (BASS audio library) — звуковая библиотека, предназначена для использования в операционных системах Windows.

Библиотека BASS разработана un4seen developments. и содержит набор функций для воспроизведения звуковых файлов различных форматов, а также для записи звука.

Библиотека BASS достаточно проста в использовании, имеет небольшой размер и поддерживает следующие возможности:

* воспроизведение звуковых образцов (сэмплов) в формате WAV/AIFF/MP3/MP2/MP1/OGG;
* потоковое воспроизведение звуковых файлов в формате MP3/MP2/MP1/OGG/WAV/AIFF;
* потоковое воспроизведение звука из интернета (по протоколам http и ftp, с поддержкой серверов Shoutcast, Icecast & Icecast2);
* и многое другое.

Непосредственное воспроизведение звука происходит через так называемые "каналы", поэтому все виды звуковых данных (sample, stream и mod music) связываются с каналами воспроизведения звука.

# openGL. Freeglut

OpenGL (Open Graphics Library) — спецификация, определяющая платформонезависимый (независимый от языка программирования) программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

Включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов. Используется при создании компьютерных игр, САПР, виртуальной реальности, визуализации в научных исследованиях. На платформе Windows конкурирует с Direct3D.

freeglut — открытая альтернатива OpenGL Utility Toolkit (GLUT). GLUT (и, следовательно, freeglut) позволяет пользователю создавать окна, предоставляющие контекст OpenGL на широком спектре платформ, и управлять ими, а также взаимодействовать с мышью, клавиатурой и джойстиком. freeglut предназначена для полной замены GLUT, и имеет очень немного отличий от неё.

С того времени, как оригинальный GLUT прекратил развитие, freeglut начал развиваться с целью улучшения предоставляемого инструментария. Он выпущен под лицензией X Consortium.

## **Структура приложения**

Данное приложение будет состоять из начального окна (рисунок 3.1), где будет происходить идентификация пользователя и выбор сложности, окна с игровой сценой (рисунок 3.2), и конечного окна с таблицей рекордов (рисунок 3.3). Также будут иметься вспомогательные окна, отображающие справку по игре.

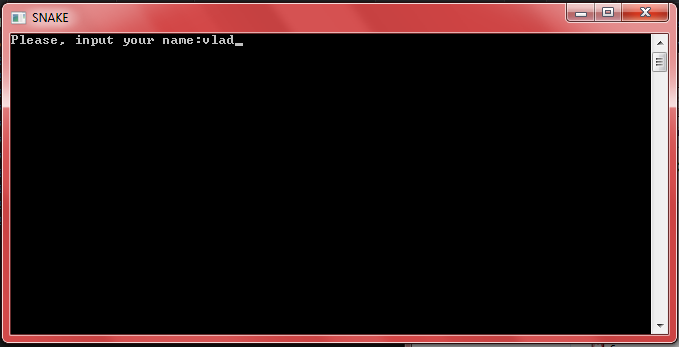


Рисунок 3.1 – Начальное окно

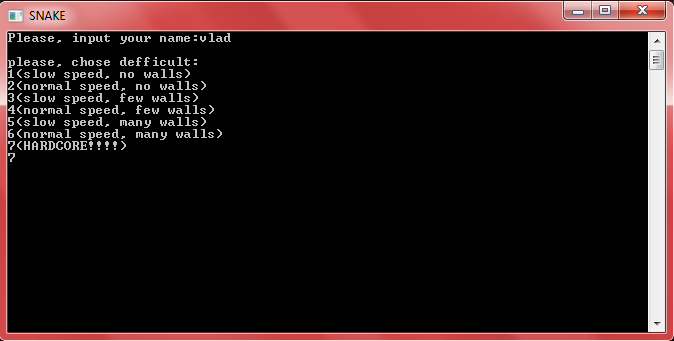


Рисунок 3.2 – Окно выбора сложности

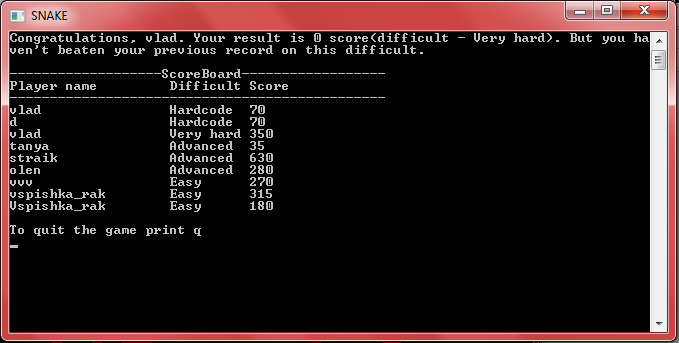
****

Рисунок 3.3 – Таблица рекордов

## **Основной игровой процесс**

Для начала надо определиться с правилами игры.

В классическом представлении правила таковы: Игрок управляет змеёй, которая передвигается по плоскости (как правило, ограниченной стенками), собирая еду (или другие предметы), избегая столкновения с собственным хвостом и краями игрового поля. В некоторых вариантах на поле присутствуют дополнительные препятствия. Каждый раз, когда змея съедает кусок пищи, она становится длиннее, что постепенно усложняет игру. Игрок управляет направлением движения головы змеи (обычно 4 направления: вверх, вниз, влево, вправо), а хвост змеи движется следом. Игрок не может остановить движение змеи. В некоторых версиях при попытке выйти за границу поля змея перемещается в противоположный край поля, однако в нашем случае эта попытка будет означать конец игры.

Весь игровой процесс будет происходить на игровой сцене, которая будет отображаться на форме (рисунок 3.2), созданной при помощи средств библиотеки freeglut. Отрисовка игрового процесса будет происходить при помощи средств OpenGL.

После инициализации игрового окна и функций для отрисовки (ф-я drawCurrState) и функции-обработчика события, генерируемого таймером (ф-я mainEvent) запускается основной игровой цикл, который организовывает основной игровой процесс.

Код функции drawCurrState:

void drawCurrState()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_QUADS);

settings.drawBlock(food.x, food.y, GREEN); //отрисовка блока еды

wall.drawWalls(); //Отрисовка стен

snake.drawBody(); //Отрисовка змеи

glEnd();

glutSwapBuffers();

}

Код функции mainEvent:

void mainEvent(int value)

{

if (!settings.isGameStopped)

{

snake.makeAction(); //просчёт одного хода змеи

glutPostRedisplay(); //отрисовка текущего состояния игрового процесса

if (settings.timeToShowRecords)

{

settings.showRecords();

}

else

if (settings.isGameOver())

{

settings.leaveGame();

}

}

glutTimerFunc(settings.delay, mainEvent, 0);

}

### Класс mainGame

Класс mainGame содержит основные настройки приложения (например, ширина и высота игрового окна, задержка таймера, координатная сетка), общие данные (имя игрока, количество набранных очков и т.д.), а также основные методы для отрисовки объектов (методы drawBlock и т.д.), для генерации препятствий (методы fillSpace, fillField и т.д.), проверки игрового состояния (методы isGameOver, isSnakeBumpintoItsSelf, isSnakeBumpintoWalls и т.д.) и т.д.

Интерфейс класса mainGame:

class mainGame

{

public:

int delay;

int width;

int height;

int startPosWinX;

int startPosWinY;

float scale;

float scaleWin;

int startPosSnakeX;

int startPosSnakeY;

int startSizeSnake;

int countOfWalls;

TTypeOfCell \*\*field;

char playerName[20];

TDifficult difficult;

long int score;

int windowIndex;

bool timeToShowRecords;

bool isGameStopped;

HWND descrCon;

mainGame(int delay, int width, int height, int startPosWinX, int startPosWinY, float scale, float scaleWin, int startPosSnakeX, int startPosSnakeY, int startSizeSnake, int countOfWalls, bool isGameStopped) : delay(delay), width(width), height(height), startPosWinX(startPosWinX), startPosWinY(startPosWinY), scale(scale), scaleWin(scaleWin), startPosSnakeX(startPosSnakeX), startPosSnakeY(startPosSnakeY),

startSizeSnake(startSizeSnake), countOfWalls(countOfWalls), score(0), timeToShowRecords(false), isGameStopped(isGameStopped)

{

field = (TTypeOfCell\*\*)malloc(sizeof(TTypeOfCell\*) \* height);

for (int j = 0; j < height; j++)

{

field[j] = (TTypeOfCell\*)malloc(

sizeof(TTypeOfCell) \* width);

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

field[j][i] = NONE;

}

}

mainGame();

void drawBlock(int x, int y);

void drawBlock(int x, int y, float R, float G, float B);

void fillField();

void fillSpace();

bool isGameOver();

bool isSnakeBumpintoItsSelf();

bool isSnakeBumpintoWalls();

char \*getTitle();

void leaveGame();

void stop\_continueGame();

void stopGame();

void continueGame();

void showRecords();

#define HINTMSG 0

#define ENDOFGAMEMSG 1

#define CONTROLMSG 2

void mainGame::showMessage(HWND handle, int messageIndex, bool isFirstTime);

private:

void fillSpaceBorders();

void fillSpaceCorners();

void fillSpaceIntoField();

};

В данном пункте будут рассмотрены методы isSnakeBumpintItsSelf, isSnakeBumpintoWalls и isGameOver.

Метод isSnakeBumpintItsSelf проверяет одно из условий конца игры, а именно не врезалась ли змейка в свой хвост. Код метода:

bool mainGame::isSnakeBumpintoItsSelf()

{

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

if ((snake.snakeBody[0].x == snake.snakeBody[i].x) && (snake.snakeBody[0].y == snake.snakeBody[i].y))

return true;

return false;

}

А метод isSnakeBumpintoWalls проверяет не врезалась ли змейка в стену. Код метода:

bool mainGame::isSnakeBumpintoWalls()

{

for (int i = 0; i < wall.walls.size(); i++)

if ((snake.snakeBody[0].x == wall.walls[i].x) && (snake.snakeBody[0].y == wall.walls[i].y))

return true;

return false;

}

Метод isGameOver проверяет или не врезалась ли она в границу игрового поля. Код метода:

bool mainGame::isGameOver()

{

if (((snake.snakeBody[0].x < 0) || (snake.snakeBody[0].y < 0)) || isSnakeBumpintoItsSelf() || ((snake.snakeBody[0].x >= settings.width) || (snake.snakeBody[0].y >= settings.height)) || isSnakeBumpintoWalls())

{

return true;

}

return false;

}

### Класс TSnake

Класс TSnake содержит координаты тела змейки (её головы и хвоста), её размер, направление и т.д., а также метод makeAction, при вызове которого просчитывается один ход змейки в зависимости от направления, метод для отрисовки змейки drawBody и т.д.

Интерфейс класса TSnake:

class Tsnake

{

public:

int direction;

bool isSnakeTurn;

int sizeSnake;

vector<TPieceBody> snakeBody;

TSnake(int direction, int sizeSnake) : direction(direction), sizeSnake(sizeSnake), isSnakeTurn(false)

{

snakeBody.push\_back({ settings.startPosSnakeX, settings.startPosSnakeY });

for (int i = 1; i < sizeSnake; i++)

{

snakeBody.push\_back({-1, -1});

}

};

void drawBody();

void makeAction();

private:

void makeMove();

bool eat();

};

В данном пункте будут рассмотрена разработка метода makeAction.

Как было сказано ранее, метод makeAction вызывает методы makeMove (просчитывает один ход змейки в зависимости от направления и изменяет её положение в игровом пространстве) и eat (проверяет, съела ли змейка еду за этот ход. Если да, то увеличение змейки и добавление очков). Код метода makeAction:

void TSnake::makeAction()

{

snake.isSnakeTurn = true;

snake.makeMove();

snake.eat();

}

Код метода makeMove:

void TSnake::makeMove()

{

for (int i = snakeBody.size() - 1; i > 0; i--) //пересчёт координат хвоста

{

snakeBody[i].x = snakeBody[i - 1].x;

snakeBody[i].y = snakeBody[i - 1].y;

}

switch (direction) //просчёт координаты головы в зависимоти от направления

{

case GLUT\_KEY\_UP:

snakeBody[0].y++;

break;

case GLUT\_KEY\_DOWN:

snakeBody[0].y--;

break;

case GLUT\_KEY\_LEFT:

snakeBody[0].x--;

break;

case GLUT\_KEY\_RIGHT:

snakeBody[0].x++;

break;

}

}

Код метода eat:

bool TSnake::eat()

{

if ((food.x == snakeBody[0].x) && (food.y == snakeBody[0].y)) //если змейка съела блок еды

{

snakeBody.push\_back({ -1, -1 });

sizeSnake++;

settings.field[food.y][food.x] = NONE;

food.drop(); //генерирование новых координат для еды

settings.score = settings.score + 10 \* settings.difficult.scoreMultiplier; //увеличение очков

char temp[50];

temp[0] = '\0';

strcat(temp,settings.getTitle());

glutSetWindowTitle(temp); //обновление отображения очков

sound.playSound(SOUNDEATINDEX);

return true;

}

return false;

}

### Класс TFood

Класс TFood содержит координаты блока еды, а также метод drop, который генерирует новые случайные координаты для блока еды (они не должны совпадать с координатами стен и с координатами змейки).

Интерфейс класса TFood:

class Tfood

{

public:

int x;

int y;

TFood() :x(-1), y(-1) {};

void drop();

}

Код метода drop:

void TFood::drop()

{

bool isOver = false;

while (!isOver) //выполнение до тех пор, пока координаты не будут корректными

{

isOver = true;

x = (rand() % settings.width); //генерирование координаты x

y = (rand() % settings.height); //генерирование координаты y

for (int i = 0; i < snake.sizeSnake; i++) //проверка координат на корректность (не совпали ли они с координатами змейки)

{

if ((x == snake.snakeBody[i].x) && (y == snake.snakeBody[i].y))

{

isOver = false;

break;

}

}

if (isOver) //проверка координат на корректность (не совпали ли они к координатами стен)

{

for (int i = 0; i < wall.walls.size(); i++)

{

if ((x == wall.walls[i].x) && (y == wall.walls[i].y))

{

isOver = false;

break;

}

}

}

if (isOver)

{

settings.field[y][x] = FOOD;

}

}

}

## **Музыкальное и звуковое сопровождение игрового процесса**

Для создания более интересного геймплея было решено внедрить музыкальное и звуковое сопровождение игрового процесса. В качестве музыки было решено взять музыку в т.н. стиле “8 бит”. Музыкальный ассортимент будет представлять собой плейлист, состоящий из 6 песен (в рамках данного курсового проекта, в дальнейшем планируется расширение), и песню, которая будет играть во время показа таблицы рекордов. Выбор 1-й и следующей композиции происходит случайно из плейлиста.

Также были подобраны 8-битные звуки для создания атмосферы игр в стиле “8-бит”.

1. Класс TSound

Класс TSound отвечает за звуковую и музыкальную составляющую. Он содержит в себе плейлист, а также звуки, текущий уровень громкости звуков, индекс текущей песни и методы для добавления музыки, звуков (методы addToPlaylist, getDescrFromFileStr, getDescrFromFileSmp), методы для увеличения или уменьшения уровня громкости как музыки, так и звуков (методы changeMusicVol(int add), changeSoundVol(int add)), а также метод для остановки или продолжения воспроизведения (методы startPlayMusic, stopPlayMusic) и т.д.

Интерфейс класса TSound:

typedef char\* TFileName;

typedef HSTREAM TFileDescr;

typedef HSAMPLE TFileDescrSample;

typedef struct \_TPlaylistItem

{

QWORD songLength;

TFileDescr fileDescr;

} TPlaylistItem;

typedef vector <TPlaylistItem> TPlayList;

typedef vector <TFileDescrSample> TSoundList;

#define SOUNDEATINDEX 0

#define SOUNDMOVEINDEX 1

#define SOUNDENDOFTHEGAMEINDEX 2

class TSound{

public:

void startPlayMusic();

void stopPlayMusic();

void changeSoundVol(int add);

void changeMusicVol(int add);

void playSound(int soundIndex);

void addToPlaylist(TFileName musicName);

TSound::TSound(TFileName soundOfEat, TFileName soundOfMove, TFileName soundOfEnd, TFileName endSongName);

void setNextMusicIndex();

TPlayList playList;

int currMusicIndex;

HCHANNEL soundChannel;

void playEndSong();

private:

float currSoundVol;

TSoundList soundList;

HSTREAM endSong;

HSTREAM getDescrFromFileStr(TFileName fileName);

HSAMPLE getDesrcFromFileSmp(TFileName fileName);

};

В данном пункте будут рассмотрены методы addToPlaylist(TFileName musicName), setNextMusicIndex.

В качестве параметра метода addToPlaylist выступает имя аудиофайла, который нужно добавить в плейлист. Код метода:

void TSound::addToPlaylist(TFileName musicName)

{

TPlaylistItem currSong;

currSong.fileDescr = getDescrFromFileStr(musicName); //считывание из файла

currSong.songLength = BASS\_ChannelGetLength(currSong.fileDescr, BASS\_POS\_BYTE);

BASS\_ChannelSetSync(currSong.fileDescr, BASS\_SYNC\_END, 0, playMusic, this);

//установка обработки события конца песни (обработчик переключает песню на другую)

playList.push\_back(currSong);

}

## **Уровни сложности**

Для того чтобы разнообразить игровой процесс, было решено ввести разные уровни сложности. Всего будет 7 уровней: “Very easy”, “Easy”, “Normal”, “Advanced”, “Hard”, “Very hard”, “Hardcore”. Параметры уровня сложности: скорость игры (чем выше сложность, тем больше скорость), количество стен (чем выше сложность, тем больше стен будет помещено на игровую сцену), множитель очков (чем больше сложность, тем больше множитель очков). В начале игры пользователю будет предложен на выбор один из уровней сложности (рисунок 3.2). Все уровни сложности описаны в файле commonTypes.cpp.

Код файла commonTypes.cpp:

//описание параметров уровня сложности

typedef struct \_TDifficult{

char name[10];

int index;

int speed;

int countOfWalls;

char info[400];

float scoreMultiplier;

}TDifficult;

//описание уровней сложности

#define VERYEASY "Very easy" , 0, 250, 0, "slow speed, no walls", 1

#define EASY "Easy", 1, 100, 0, "normal speed, no walls", 1.5

#define NORMAL "Normal", 2, 250, 50, "slow speed, few walls", 2.5

#define ADVANCED "Advanced", 3, 150, 50, "normal speed, few walls", 3.5

#define HARD "Hard", 4, 250, 100, "slow speed, many walls", 4

#define VERYHARD "Very hard", 5, 150, 100, "normal speed, many walls", 5

#define HARDCORE "Hardcode", 6, 100, 100, "HARDCORE!!!!", 7

#define COUNTOFDIFFICULTS 7

const TDifficult DIFFICULTS[COUNTOFDIFFICULTS] = { { VERYEASY }, { EASY }, { NORMAL }, { ADVANCED }, { HARD }, { VERYHARD }, { HARDCORE } };

После выбора пользователем индекса уровня сложности будет выбран соответствующий уровень сложности из массива DIFFICULT.

## **Генерация случайных препятствий на игровой сцене**

Вся генерация случайных препятствий основывается на 3-х примитивах: линия длиной 3 блока (рисунок 3.4), квадрат 2х2 (рисунок 3.5), угол (рисунок 3.6). В зависимости от параметра countOfWalls выбранного уровня сложности нужно разместить countOfWalls примитивов на игровой сцене. Они могут пресекаться, тем самым создавая препятствия уникальной формы. На каждом шаге генерации случайным образом выбирается примитив, и генерируются его координаты, после этого идёт добавление блоков этого примитива на игровую сцену.

E:\my work\Pictures\line.png

Рисунок 3.4 – Линия



Рисунок 3.5 – Квадрат



Рисунок 3.6 – Угол

Методы генерации примитивов описаны в классе TWall.

Основной цикл генерации препятствий реализован в методе spawnWalls (класс mainGame):

void TWall::spawnWalls(int count)

{

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

switch (rand() % COUNTOFTYPEOFWALL) //случайный выбор следующего примитива

{

case line3:

placeLine();

break;

case corner:

placeCorner();

break;

case quad:

placeQuad();

break;

}

}

}

1. Корректирование стен

Далее нужно откорректировать сгенерированные препятствия, чтобы минимизировать недопустимые препятствия (при которых из некоторых мест не существует выхода). Примеры таких ситуаций (рисунок 3.7). Устранение их идёт по методу заполнения таких недопустимых мест стенами.

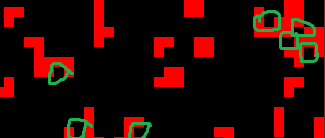


Рисунок 3.7 – Недопустимые места

Данные корректировки реализованы в методe fillSpace (класс mainGame):

void mainGame::fillSpace()

{

fillSpaceIntoField(); //устранение недопустимых мест внутри поля

fillSpaceBorders(); //устранение недопустимых мест на границе

fillSpaceCorners(); //устранение недопустимых в углах

fillSpaceBorders();

fillSpaceIntoField();

}

Место является недопустимым, если к нему нету подхода с 3-х направлений. В случае, когда проверяются граница и поле, проход нужно сделать в обе стороны, т.к. во время устранения недопустимых мест могут образоваться новые.

Рассмотрим принцип устранения недопустимых мест на примере метода fillSpaceCorners. Код метода:

void mainGame::fillSpaceCorners()

{

if ((field[0][1] == WALL) || (field[1][0] == WALL)) //проверка нижнего левого угла

{

field[0][0] = WALL;

wall.walls.push\_back({ 0, 0 });

}

if ((field[0][width - 2] == WALL) || (field[1][width - 1] == WALL))

{

field[0][width - 1] = WALL;

wall.walls.push\_back({ width - 1, 0 });

}

if ((field[height - 1][1] == WALL) || (field[height - 2][0] == WALL))

{

field[height - 1][0] = WALL;

wall.walls.push\_back({ 0, height - 1 });

}

if ((field[height - 1][width - 2] == WALL) || (field[height - 2][width - 1] == WALL))

{

field[height - 1][width - 1] = WALL;

wall.walls.push\_back({ width - 1, height - 1});

}

}

Принцип работы методов fillSpaceIntoField, fillSpaceBorders аналогичен.

1. Класс TWall

Класс TWall содержит координаты всех блоков стен, а также методы добавления примитивов (placeLine, placeQuad, placeCorner). Интерфейс класса TWall:

class TWall

{

public:

vector<TWallBlock> walls;

void spawnWalls(int count);

bool placeBlock();

bool placeLine();

bool placeCorner();

bool placeQuad();

void drawWalls();

}

Рассмотрим метод placeLine:

#define LINEPOS 2

#define HOR 0

#define VERT 1

bool TWall::placeLine()

{

walls.push\_back({ rand() % settings.width, rand() % settings.height });

bool isPlaced = false;

while (!isPlaced)

{

if ((walls[walls.size() - 1].x > (snake.snakeBody[0].x - settings.startSizeSnake)) && (walls[walls.size() - 1].x < (snake.snakeBody[0].x + settings.startSizeSnake)) && (walls[walls.size() - 1].y >(snake.snakeBody[0].y - settings.startSizeSnake)) && (walls[walls.size() - 1].y < (snake.snakeBody[0].y + settings.startSizeSnake))) // проверка корректности сгенерированных координат

{

walls[walls.size() - 1].x = rand() % settings.width;

walls[walls.size() - 1].y = rand() % settings.height;

//генерирование новых координат

}

else

{

switch (rand() % LINEPOS){

case HOR:

{

walls.push\_back({ walls[walls.size() - 1].x + 1, walls[walls.size() - 1].y });

walls.push\_back({ walls[walls.size() - 1].x + 1, walls[walls.size() - 1].y });

}

break;

//горизонтальное расположение линии

case VERT:

{

walls.push\_back({ walls[walls.size() - 1].x, walls[walls.size() - 1].y + 1 });

walls.push\_back({ walls[walls.size() - 1].x, walls[walls.size() - 1].y + 1 });

}

break;

//вертикальное расположение линии

}

isPlaced = true;

}

}

return isPlaced;

}

Принцип работы методов placeQuad, placeCorner аналогичен.

## **Таблица рекордов**

Для ведения статистики сыгранных игр и возможности сравнения результатов пользователя с результатом других пользователей было решено разработать таблицу рекордов. Она должна представлять собой таблицу с полями: “Имя игрока”, ”количество очков”, ”сложность”.

Для хранения данных был использован шаблон vector (заголовочный файл <vector>) Для работы с этим шаблоном были выбраны функции из заголовочного файла <algorythm>. Эти заголовочные файлы принадлежат стандартной библиотеке STL.

Сама таблица рекордов и методы работы с ней содержаться в классе TScoreBoard. Интерфейс класса:

typedef struct \_TPlayerInfo{

char playerName[20];

long int countOfScore;

char difficult[10];

int diffIndex;

} TPlayerInfo;

//Структура элемента таблицы

#define ENDOFVECTORINFO {"", -1, "", -1}

typedef vector <TPlayerInfo> TArrInfo;

typedef vector<TPlayerInfo>::iterator TRange;

typedef vector<TRange> TRanges;

typedef TRange TPlayerInfoIter;

class TInfo{

public:

void printData();

void addDataToList(TPlayerInfo data);

TInfo();

private:

char \*fileName;

TPlayerInfo player;

TArrInfo arrInfo;

TRanges ranges;

int prevCountScore;

TArrInfo getDataFromFile();

void writeDataToFile();

TRanges getRanges();

void sortData();

void partSort();

void printPlayerResults();

};

1. Сортировка

Сортировка таблицы происходит в 2 этапа:

* сортировка по уровням сложности;
* сортировка по количеству очков внутри каждого уровня сложности.

За сортировку таблицы отвечает метод sortData. Код метода:

void TInfo::sortData()

{

stable\_sort(arrInfo.begin(), arrInfo.end(), &compareFuncDiff); //сортировка по уровням сложности

ranges = getRanges(); //взятие границ по уровням сложности

partSort(); //сортировка по очкам внутри уровней сложности

}

Сортировка производится при обновлении и изменении таблицы.

1. Обновление рекорда

Если пользователь уже играл ранее на данном уровне сложности, то идёт не добавление, а обновление соответствующего поля таблицы в зависимости от того сколько набрал пользователь (сохраняется лучший результат).

## **Управление игровым процессом**

Взаимодействие пользователя с программой происходит при помощи следующих клавиш:

* стрелка вниз, вверх, влево, вправо (управление змейкой);
* пробел (остановка игры);
* 1, 2 (увеличение/уменьшение громкости музыки);
* 3, 4 (увеличение/уменьшение громкости звука);
* F1, F2 (вызов справки по управлению (рисунок 3.8)/игре (рисунок 3.9)).

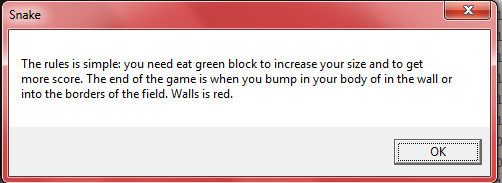


Рисунок 3.8 – Окно подсказки

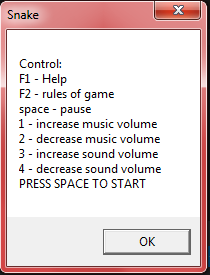


Рисунок 3.9 - Управление

Функции для обработки прерываний от клавиатуры (ф-и keyboardInterrControl и keyboardInterr), которые привязываются при помощи функций glutKeyboardFunc, glutSpecialFunc библиотеки freeglut.

## **Идентификация игрока**

Для ведения статистики была внедрена идентификация игрока перед началом игры (рисунок 3.1), для того чтобы в последующем занести результаты в таблицу.

# **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

1. **Системные требования**

Для успешного запуска игры нужно наличие поддержки видеокартой OpenGL и DX9. Системные требования OpenGL:

* процессор с частотой 1,2 ГГц или выше;
* 192 МБ оперативной памяти.

1. **Запуск программы. Начало игры**

Для запуска программы необходимо открыть исполняемый файл Snake.exe. Откроется начальное окно программы (рисунок 3.1). Для того чтобы начать играть нужно ввести своё имя и выбрать сложность, после чего начнётся игра (рисунок 3.2).

1. **Правила игры**

Цель игры: набрать как можно больше очков. Для того чтобы набирать очки, нужно поедать блоки еды. При этом змейка увеличивается в размерах.

Управлять можно только головой. Конец игры наступает, когда змейка врезается либо в свой хвост, либо в стену, либо пересекает край экрана.

1. **Управление**

Для управления программой во время игрового процесса предоставлены следующие клавиши:

* стрелка вниз, вверх, влево, вправо (управление змейкой);
* пробел (остановка/продолжение игры);
* 1, 2 (увеличение/уменьшение громкости музыки);
* 3, 4 (увеличение/уменьшение громкости звука);
* F1, F2 (вызов справки по управлению (рисунок 3.8)/игре (рисунок 3.9)).

Для закрытия таблицы рекордов нужно ввести ‘q’в поле ввода, после чего приложение закроется.

**ЗАКЛЮЧЕНИЕ**

Безусловно, игра “Змейка” является классикой в области компьютерных игр. Несмотря на возраст данной игры, на сегодняшний день находятся современные интерпретации данной идеи и появляются новые популярные проекты на базе “Змейки” (например, slither.io), что говорит об актуальности данной игры и в наше время.

В рамках данного курсового проекта было разработано игровое приложение “Snake”. В данной программе реализован основной игровой процесс, а также его дополнения и расширения:

* генерация случайных препятствий на игровой сцене;
* музыкальное и звуковое сопровождение игрового процесса;
* таблица рекордов;
* уровни сложности.

При разработке программы в качестве основы использовалась библиотека freeglut, которая взаимодействует с OpenGL, а также Bass.

Поставленная цель выполнена. Программа реализует все поставленные задачи.

В будущем планируется программу усовершенствовать, а именно:

* добавить больше параметров уровней сложности (размеры игрового поля, постепенное ускорение игры и т.д.);
* усовершенствовать графическую составляющую;
* расширить плейлист и звуковой ассортимент;
* добавить новые уровни сложности.

Список использованных источников

[1]  [Шилдт Г.: Самоучитель С++: Пер. с англ. – 3-е изд. – СПб.: БХВ- Питербург, 2007. – 688 с.](http://www.piter.com/collection/yazyk-programmirovaniya-ss-s/product/clr-via-c-programmirovanie-na-platforme-microsoft-net-framework-45-na-yazyke-c-4-e-izd-2)

[2] Каталог API и справочных материалов [Электронный ресурс]. –https://msdn.microsoft.com/library

[3] Дейтел, Х. М. Как программировать на С / Х. М Дейтел, П. Дж. Дейтел. – М.: ЗАО «Издательство БИНОМ», 2000. – 1008 с.: ил.

[4] Электронный учебно-методический комплекс по дисциплине «Языки программирования. Часть 1» [Электронный ресурс]. – Электрон. текстовые дан и прог.(6,1 Мб). – Минск: Бахтизин В.В., 2013.

[5] Конструирование программ и языки программирования: Учено-методическое пособие для студ. спец. 40 01 01 “Программное обеспечение информационных технологий” дневной формы обуч. В 2 ч./ В.В.Бахтизин [и др.] – Минск: БГУИР 2006.

[6] Шупляк В.И.: С++. Практический курс: учеб. пособие / Шупляк В.И – Минск: Новое знание, 2008. – 576 с.

**ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma once

#include "targetver.h"

#include <stdio.h>

#include <tchar.h>

#include <GL\freeglut.h>

#include <time.h>

#include <stdlib.h>

#include <iostream>

#include <vector>

#include <algorithm>

#include <string>

#include <windows.h>

#include "bass.h"

#include "stdafx.h"

#include "commonTypes.h"

#include "scoreBoard.h"

#include "TSound.h"

using namespace std;

typedef float\* TColor;

#define RED 1, 0, 0

#define GREEN 0, 1, 0

#define BLUE 0, 0, 1

#define WHITE 1, 1, 1

enum TTypeOfCell {NONE, SNAKE, FOOD, WALL};

TInfo scoreBoard;

TSound sound("eat.wav", "move.wav", "", "endSong.mp3");

class mainGame{

public:

int delay;

int width;

int height;

int startPosWinX;

int startPosWinY;

float scale;

float scaleWin;

int startPosSnakeX;

int startPosSnakeY;

int startSizeSnake;

int countOfWalls;

TTypeOfCell \*\*field;

char playerName[20];

TDifficult difficult;

long int score;

int windowIndex;

bool timeToShowRecords;

bool isGameStopped;

HWND descrCon;

mainGame(int delay, int width, int height, int startPosWinX, int startPosWinY, float scale, float scaleWin,

int startPosSnakeX, int startPosSnakeY, int startSizeSnake, int countOfWalls, bool isGameStopped) :

delay(delay), width(width), height(height), startPosWinX(startPosWinX), startPosWinY(startPosWinY),

scale(scale), scaleWin(scaleWin), startPosSnakeX(startPosSnakeX), startPosSnakeY(startPosSnakeY),

startSizeSnake(startSizeSnake), countOfWalls(countOfWalls), score(0), timeToShowRecords(false), isGameStopped(isGameStopped)

{

field = (TTypeOfCell\*\*)malloc(sizeof(TTypeOfCell\*) \* height);

for (int j = 0; j < height; j++)

{

field[j] = (TTypeOfCell\*)malloc(sizeof(TTypeOfCell) \* width);

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

field[j][i] = NONE;

}

}

mainGame();

void drawBlock(int x, int y);

void drawBlock(int x, int y, float R, float G, float B);

void fillField();

void fillSpace();

bool isGameOver();

bool isSnakeBumpintoItsSelf();

bool isSnakeBumpintoWalls();

char \*getTitle();

void leaveGame();

void stop\_continueGame();

void stopGame();

void continueGame();

void showRecords();

#define HINTMSG 0

#define ENDOFGAMEMSG 1

#define CONTROLMSG 2

void mainGame::showMessage(HWND handle, int messageIndex, bool isFirstTime);

private:

void fillSpaceBorders();

void fillSpaceCorners();

void fillSpaceIntoField();

}settings;

typedef struct \_TPieceBody{

int x;

int y;

} TPieceBody;

class TFood{

public:

int x;

int y;

TFood() :x(-1), y(-1) {};

void drop();

} food;

class TSnake{

public:

int direction;

bool isSnakeTurn;

int sizeSnake;

vector<TPieceBody> snakeBody;

TSnake(int direction, int sizeSnake) : direction(direction), sizeSnake(sizeSnake), isSnakeTurn(false)

{

snakeBody.push\_back({ settings.startPosSnakeX, settings.startPosSnakeY });

for (int i = 1; i < sizeSnake; i++)

{

snakeBody.push\_back({-1, -1});

}

};

void drawBody();

void makeAction();

private:

void makeMove();

bool eat();

} snake(GLUT\_KEY\_UP, 3);

typedef enum \_TTypeWall {line3, corner, quad} TTypeWall;

#define COUNTOFTYPEOFWALL 3

typedef struct \_TWallBlock

{

int x;

int y;

} TWallBlock;

class TWall

{

public:

vector<TWallBlock> walls;

void spawnWalls(int count);

bool placeBlock();

bool placeLine();

bool placeCorner();

bool placeQuad();

void drawWalls();

} wall;

mainGame::mainGame()

{

delay = 200;

width = 60;

height = 35;

startPosWinX = 200;

startPosWinY = 200;

scale = 1;

scaleWin = 5;

startPosSnakeX = width / 2;

startPosSnakeY = height / 2;

startSizeSnake = 3;

countOfWalls = 100;

score = 0;

timeToShowRecords = false;

isGameStopped = true;

field = (TTypeOfCell\*\*)malloc(sizeof(TTypeOfCell\*)\* height);

for (int j = 0; j < height; j++)

{

field[j] = (TTypeOfCell\*)malloc(sizeof(TTypeOfCell)\*width);

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

field[j][i] = NONE;

}

}

void mainGame::drawBlock(int x, int y)

{

glColor3f(1, 1, 1);

glVertex2f(x \* settings.scale, y \* settings.scale);

glVertex2f(x \* settings.scale + settings.scale, y \* settings.scale);

glVertex2f(x \* settings.scale + settings.scale, y \* settings.scale + settings.scale);

glVertex2f(x \* settings.scale, y \* settings.scale + settings.scale);

}

void mainGame::drawBlock(int x, int y, float R, float G, float B)

{

glColor3f(R, G, B);

glVertex2f(x \* settings.scale, y \* settings.scale);

glVertex2f(x \* settings.scale + settings.scale, y \* settings.scale);

glVertex2f(x \* settings.scale + settings.scale, y \* settings.scale + settings.scale);

glVertex2f(x \* settings.scale, y \* settings.scale + settings.scale);

}

bool mainGame::isSnakeBumpintoItsSelf()

{

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

if ((snake.snakeBody[0].x == snake.snakeBody[i].x) && (snake.snakeBody[0].y == snake.snakeBody[i].y))

return true;

return false;

}

bool mainGame::isSnakeBumpintoWalls()

{

for (int i = 0; i < wall.walls.size(); i++)

if ((snake.snakeBody[0].x == wall.walls[i].x) && (snake.snakeBody[0].y == wall.walls[i].y))

return true;

return false;

}

bool mainGame::isGameOver()

{

if (((snake.snakeBody[0].x < 0) || (snake.snakeBody[0].y < 0)) || isSnakeBumpintoItsSelf()

|| ((snake.snakeBody[0].x >= settings.width) || (snake.snakeBody[0].y >= settings.height)) || isSnakeBumpintoWalls())

{

return true;

}

return false;

}

void mainGame::fillField()

{

for (int i = 0; i < wall.walls.size(); i++)

{

if (((wall.walls[i].x < width) && (wall.walls[i].y < height)) && ((wall.walls[i].x >= 0) && (wall.walls[i].y >= 0)))

field[wall.walls[i].y][wall.walls[i].x] = WALL;

}

}

void mainGame::fillSpaceCorners()

{

if ((field[0][1] == WALL) || (field[1][0] == WALL))

{

field[0][0] = WALL;

wall.walls.push\_back({ 0, 0 });

}

if ((field[0][width - 2] == WALL) || (field[1][width - 1] == WALL))

{

field[0][width - 1] = WALL;

wall.walls.push\_back({ width - 1, 0 });

}

if ((field[height - 1][1] == WALL) || (field[height - 2][0] == WALL))

{

field[height - 1][0] = WALL;

wall.walls.push\_back({ 0, height - 1 });

}

if ((field[height - 1][width - 2] == WALL) || (field[height - 2][width - 1] == WALL))

{

field[height - 1][width - 1] = WALL;

wall.walls.push\_back({ width - 1, height - 1 });

}

}

void mainGame::fillSpaceBorders()

{

for (int i = 1; i < width - 1; i++)

{

int count = 0;

if (field[0][i - 1] == WALL)

count++;

if (field[0][i + 1] == WALL)

count++;

if (field[1][i] == WALL)

count++;

if (count > 1)

{

field[0][i] = WALL;

wall.walls.push\_back({ i, 0 });

}

}

for (int i = width - 2; i > 0; i--)

{

int count = 0;

if (field[0][i - 1] == WALL)

count++;

if (field[0][i + 1] == WALL)

count++;

if (field[1][i] == WALL)

count++;

if (count > 1)

{

field[0][i] = WALL;

wall.walls.push\_back({ i, 0 });

}

}

for (int i = 1; i < width - 1; i++)

{

int count = 0;

if (field[height - 1][i - 1] == WALL)

count++;

if (field[height - 1][i + 1] == WALL)

count++;

if (field[height - 2][i] == WALL)

count++;

if (count > 1)

{

field[height - 1][i] = WALL;

wall.walls.push\_back({ i, height - 1 });

}

}

for (int i = width - 2; i > 0; i--)

{

int count = 0;

if (field[height - 1][i - 1] == WALL)

count++;

if (field[height - 1][i + 1] == WALL)

count++;

if (field[height - 2][i] == WALL)

count++;

if (count > 1)

{

field[height - 1][i] = WALL;

wall.walls.push\_back({ i, height - 1 });

}

}

for (int j = 1; j < height - 1; j++)

{

int count = 0;

if (field[j - 1][0] == WALL)

count++;

if (field[j + 1][0] == WALL)

count++;

if (field[j][1] == WALL)

count++;

if (count > 1)

{

field[j][0] = WALL;

wall.walls.push\_back({ 0, j });

}

}

for (int j = height - 2; j > 0; j--)

{

int count = 0;

if (field[j - 1][0] == WALL)

count++;

if (field[j + 1][0] == WALL)

count++;

if (field[j][1] == WALL)

count++;

if (count > 1)

{

field[j][0] = WALL;

wall.walls.push\_back({ 0, j });

}

}

for (int j = 1; j < height - 1; j++)

{

int count = 0;

if (field[j - 1][width - 1] == WALL)

count++;

if (field[j + 1][width - 1] == WALL)

count++;

if (field[j][width - 2] == WALL)

count++;

if (count > 1)

{

field[j][width - 1] = WALL;

wall.walls.push\_back({ width - 1, j });

}

}

for (int j = height - 2; j > 1; j--)

{

int count = 0;

if (field[j - 1][width - 1] == WALL)

count++;

if (field[j + 1][width - 1] == WALL)

count++;

if (field[j][width - 2] == WALL)

count++;

if (count > 1)

{

field[j][width - 1] = WALL;

wall.walls.push\_back({ width - 1, j });

}

}

}

void mainGame::fillSpaceIntoField()

{

for (int j = 1; j < height - 1; j++)

{

for (int i = 1; i < width - 1; i++)

{

if (field[j][i] == NONE)

{

int count = 0;

if ((field[j][i + 1] == WALL))

count++;

if ((field[j][i - 1] == WALL))

count++;

if ((field[j - 1][i] == WALL))

count++;

if ((field[j + 1][i] == WALL))

count++;

if (count > 2)

{

field[j][i] = WALL;

wall.walls.push\_back({ i, j });

}

}

}

}

for (int j = 1; j < height - 1; j++)

{

for (int i = width - 2; i > 0; i--)

{

if (field[j][i] == NONE)

{

int count = 0;

if ((field[j][i + 1] == WALL))

count++;

if ((field[j][i - 1] == WALL))

count++;

if ((field[j - 1][i] == WALL))

count++;

if ((field[j + 1][i] == WALL))

count++;

if (count > 2)

{

field[j][i] = WALL;

wall.walls.push\_back({ i, j });

}

}

}

}

for (int j = height - 2; j > 0; j--)

{

for (int i = 1; i < width - 1; i++)

{

if (field[j][i] == NONE)

{

int count = 0;

if ((field[j][i + 1] == WALL))

count++;

if ((field[j][i - 1] == WALL))

count++;

if ((field[j - 1][i] == WALL))

count++;

if ((field[j + 1][i] == WALL))

count++;

if (count > 2)

{

field[j][i] = WALL;

wall.walls.push\_back({ i, j });

}

}

}

}

for (int j = height - 2; j > 0; j--)

{

for (int i = width - 2; i > 0; i--)

{

if (field[j][i] == NONE)

{

int count = 0;

if ((field[j][i + 1] == WALL))

count++;

if ((field[j][i - 1] == WALL))

count++;

if ((field[j - 1][i] == WALL))

count++;

if ((field[j + 1][i] == WALL))

count++;

if (count > 2)

{

field[j][i] = WALL;

wall.walls.push\_back({ i, j });

}

}

}

}

}

void mainGame::fillSpace()

{

fillSpaceIntoField();

fillSpaceBorders();

fillSpaceCorners();

fillSpaceBorders();

fillSpaceIntoField();

}

char \*mainGame::getTitle()

{

char temp[50] = "Snake. Player name: ";

strcat(temp, playerName);

strcat(temp, ". Score: ");

char buffScore[40];

char \*tempScore = \_itoa(score, buffScore, 10);

strcat(temp, tempScore);

return temp;

}

void mainGame::stop\_continueGame()

{

if (isGameStopped)

{

isGameStopped = false;

}

else

{

isGameStopped = true;

}

}

void mainGame::stopGame()

{

isGameStopped = true;

}

void mainGame::continueGame()

{

isGameStopped = false;

}

void mainGame::leaveGame()

{

TPlayerInfo temp;

temp.playerName[0] = '\0';

strcat(temp.playerName, playerName);

temp.countOfScore = score;

temp.difficult[0] = '\0';

temp.diffIndex = difficult.index;

strcat(temp.difficult, difficult.name);

scoreBoard.addDataToList(temp);

showMessage(FindWindowA(NULL, settings.getTitle()), ENDOFGAMEMSG, false);

glutHideWindow();

sound.stopPlayMusic();

sound.playEndSong();

scoreBoard.printData();

timeToShowRecords = true;

}

void mainGame::showRecords()

{

LPCSTR title = "SNAKE";

glutDestroyWindow(windowIndex);

glutLeaveMainLoop();

GetConsoleWindow();

ShowWindow(settings.descrCon, SW\_SHOW);

char answ;

getchar();

cin >> answ;

while (answ != 'q')

cin >> answ;

}

#define CONTROLTEXT "Control:\n\rF1 - Help\n\rF2 - rules of game\n\rspace - pause\n\r1 - increase music volume\n\r2 - decrease music volume\n\r3 - increase sound volume\n\r4 - decrease sound volume\n\rPRESS SPACE TO CONTINUE"

#define HINTTEXT "The rules is simple: you need eat green block to increase your size and to get more score. The end of the game is when you bump in your body of in the wall or into the borders of the field. Walls is red.\n\rPRESS SPACE TO CONTINUE"

#define ENDOFTHEGAMEMSG "Game Over"

#define CONTROLTEXTCON "Control:\n\rF1 - Help\n\rF2 - rules of game\n\rspace - pause\n\r1 - increase music volume\n\r2 - decrease music volume\n\r3 - increase sound volume\n\r4 - decrease sound volume\n\rPRESS SPACE TO START"

#define HINTTEXTCON "The rules is simple: you need eat green block to increase your size and to get more score. The end of the game is when you bump in your body of in the wall or into the borders of the field. Walls is red."

#define FIRSTTIME true

#define NOTFIRSTTIME false

void mainGame::showMessage(HWND handle, int messageIndex, bool isFirstTime)

{

switch (messageIndex)

{

case HINTMSG:

if (isFirstTime)

MessageBoxA(handle, HINTTEXTCON, "Snake", MB\_OK);

else

MessageBoxA(handle, HINTTEXT, "Snake", MB\_OK);

break;

case CONTROLMSG:

if (isFirstTime)

MessageBoxA(handle, CONTROLTEXTCON, "Snake", MB\_OK);

else

MessageBoxA(handle, CONTROLTEXT, "Snake", MB\_OK);

break;

case ENDOFGAMEMSG:

MessageBoxA(handle, ENDOFTHEGAMEMSG, "Snake", MB\_OK);

break;

default:

break;

}

}

void TFood::drop()

{

bool isOver = false;

while (!isOver)

{

isOver = true;

x = (rand() % settings.width);

y = (rand() % settings.height);

for (int i = 0; i < snake.sizeSnake; i++)

{

if ((x == snake.snakeBody[i].x) && (y == snake.snakeBody[i].y))

{

isOver = false;

break;

}

}

if (isOver)

{

for (int i = 0; i < wall.walls.size(); i++)

{

if ((x == wall.walls[i].x) && (y == wall.walls[i].y))

{

isOver = false;

break;

}

}

}

if (isOver)

{

settings.field[y][x] = FOOD;

}

}

}

bool TWall::placeBlock()

{

walls.push\_back({ rand() % settings.width, rand() % settings.height });

bool isPlaced = false;

while (!isPlaced)

{

if ((walls[walls.size() - 1].x == snake.snakeBody[0].x) && (walls[walls.size() - 1].y == snake.snakeBody[0].y))

{

walls[walls.size() - 1].x = rand() % settings.width;

walls[walls.size() - 1].y = rand() % settings.height;

}

else

{

settings.field[walls[walls.size() - 1].y][walls[walls.size() - 1].x] = WALL;

isPlaced = true;

}

}

return isPlaced;

}

#define LINEPOS 2

#define HOR 0

#define VERT 1

bool TWall::placeLine()

{

walls.push\_back({ rand() % settings.width, rand() % settings.height });

bool isPlaced = false;

while (!isPlaced)

{

if ((walls[walls.size() - 1].x > (snake.snakeBody[0].x - settings.startSizeSnake)) && (walls[walls.size() - 1].x < (snake.snakeBody[0].x + settings.startSizeSnake))

&& (walls[walls.size() - 1].y >(snake.snakeBody[0].y - settings.startSizeSnake)) && (walls[walls.size() - 1].y < (snake.snakeBody[0].y + settings.startSizeSnake)))

{

walls[walls.size() - 1].x = rand() % settings.width;

walls[walls.size() - 1].y = rand() % settings.height;

}

else

{

switch (rand() % LINEPOS){

case HOR:

{

walls.push\_back({ walls[walls.size() - 1].x + 1, walls[walls.size() - 1].y });

walls.push\_back({ walls[walls.size() - 1].x + 1, walls[walls.size() - 1].y });

}

break;

case VERT:

{

walls.push\_back({ walls[walls.size() - 1].x, walls[walls.size() - 1].y + 1 });

walls.push\_back({ walls[walls.size() - 1].x, walls[walls.size() - 1].y + 1 });

}

break;

}

isPlaced = true;

}

}

return isPlaced;

}

#define CORNERPOS 4

#define UPLEFT 0

#define DOWNLEFT 1

#define UPRIGTH 2

#define DOWNRIGTH 3

bool TWall::placeCorner()

{

walls.push\_back({ rand() % settings.width, rand() % settings.height });

bool isPlaced = false;

while (!isPlaced)

{

if ((walls[walls.size() - 1].x > (snake.snakeBody[0].x - settings.startSizeSnake)) && (walls[walls.size() - 1].x < (snake.snakeBody[0].x + settings.startSizeSnake))

&& (walls[walls.size() - 1].y >(snake.snakeBody[0].y - settings.startSizeSnake)) && (walls[walls.size() - 1].y < (snake.snakeBody[0].y + settings.startSizeSnake)))

{

walls[walls.size() - 1].x = rand() % settings.width;

walls[walls.size() - 1].y = rand() % settings.height;

}

else

{

switch (rand() % CORNERPOS){

case UPLEFT:

{

walls.push\_back({ walls[walls.size() - 1].x, walls[walls.size() - 1].y - 1 });

walls.push\_back({ walls[walls.size() - 1].x - 1, walls[walls.size() - 1].y });

}

break;

case DOWNLEFT:

{

walls.push\_back({ walls[walls.size() - 1].x, walls[walls.size() - 1].y + 1 });

walls.push\_back({ walls[walls.size() - 1].x - 1, walls[walls.size() - 1].y });

}

break;

case UPRIGTH:

{

walls.push\_back({ walls[walls.size() - 1].x, walls[walls.size() - 1].y - 1 });

walls.push\_back({ walls[walls.size() - 1].x + 1, walls[walls.size() - 1].y });

}

break;

case DOWNRIGTH:

{

walls.push\_back({ walls[walls.size() - 1].x, walls[walls.size() - 1].y + 1 });

walls.push\_back({ walls[walls.size() - 1].x + 1, walls[walls.size() - 1].y });

}

break;

}

isPlaced = true;

}

}

return isPlaced;

}

bool TWall::placeQuad()

{

walls.push\_back({ rand() % settings.width, rand() % settings.height });

bool isPlaced = false;

while (!isPlaced)

{

if ((walls[walls.size() - 1].x > (snake.snakeBody[0].x - settings.startSizeSnake)) && (walls[walls.size() - 1].x < (snake.snakeBody[0].x + settings.startSizeSnake))

&& (walls[walls.size() - 1].y >(snake.snakeBody[0].y - settings.startSizeSnake)) && (walls[walls.size() - 1].y < (snake.snakeBody[0].y + settings.startSizeSnake)))

{

walls[walls.size() - 1].x = rand() % settings.width;

walls[walls.size() - 1].y = rand() % settings.height;

}

else

{

walls.push\_back({ walls[walls.size() - 1].x, walls[walls.size() - 1].y + 1 });

walls.push\_back({ walls[walls.size() - 1].x + 1, walls[walls.size() - 1].y });

walls.push\_back({ walls[walls.size() - 1].x, walls[walls.size() - 1].y - 1 });

isPlaced = true;

}

}

return isPlaced;

}

void TWall::spawnWalls(int count)

{

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

switch (rand() % COUNTOFTYPEOFWALL)

{

case line3:

placeLine();

break;

case corner:

placeCorner();

break;

case quad:

placeQuad();

break;

}

}

}

void TWall::drawWalls()

{

for (int i = 0; i < walls.size(); i++)

settings.drawBlock(walls[i].x, walls[i].y, RED);

}

void TSnake::drawBody()

{

settings.drawBlock(snake.snakeBody[0].x, snakeBody[0].y, BLUE);

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

{

settings.drawBlock(snake.snakeBody[i].x, snakeBody[i].y, WHITE);

}

}

void TSnake::makeMove()

{

for (int i = snakeBody.size() - 1; i > 0; i--){

snakeBody[i].x = snakeBody[i - 1].x;

snakeBody[i].y = snakeBody[i - 1].y;

}

switch (direction)

{

case GLUT\_KEY\_UP:

snakeBody[0].y++;

break;

case GLUT\_KEY\_DOWN:

snakeBody[0].y--;

break;

case GLUT\_KEY\_LEFT:

snakeBody[0].x--;

break;

case GLUT\_KEY\_RIGHT:

snakeBody[0].x++;

break;

}

}

bool TSnake::eat()

{

if ((food.x == snakeBody[0].x) && (food.y == snakeBody[0].y))

{

snakeBody.push\_back({ -1, -1 });

sizeSnake++;

settings.field[food.y][food.x] = NONE;

food.drop();

settings.score = settings.score + 10 \* settings.difficult.scoreMultiplier;

char temp[50];

temp[0] = '\0';

strcat(temp,settings.getTitle());

glutSetWindowTitle(temp);

sound.playSound(SOUNDEATINDEX);

return true;

}

return false;

}

void TSnake::makeAction()

{

snake.isSnakeTurn = true;

snake.makeMove();

snake.eat();

}

void mainEvent(int value)

{

if (!settings.isGameStopped)

{

snake.makeAction();

glutPostRedisplay();

if (settings.timeToShowRecords)

{

settings.showRecords();

}

else

if (settings.isGameOver())

{

settings.leaveGame();

}

}

glutTimerFunc(settings.delay, mainEvent, 0);

}

void drawCurrState()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glBegin(GL\_QUADS);

settings.drawBlock(food.x, food.y, GREEN);

wall.drawWalls();

snake.drawBody();

glEnd();

glutSwapBuffers();

}

void keyboardInterrControl(int key, int x, int y){

if (!settings.isGameStopped)

{

switch (key)

{

case GLUT\_KEY\_UP:

if ((snake.direction != GLUT\_KEY\_DOWN) && (snake.isSnakeTurn))

{

if (snake.direction != GLUT\_KEY\_UP)

sound.playSound(SOUNDMOVEINDEX);

snake.direction = GLUT\_KEY\_UP;

snake.isSnakeTurn = false;

}

break;

case GLUT\_KEY\_DOWN:

if ((snake.direction != GLUT\_KEY\_UP) && (snake.isSnakeTurn))

{

if (snake.direction != GLUT\_KEY\_DOWN)

sound.playSound(SOUNDMOVEINDEX);

snake.direction = GLUT\_KEY\_DOWN;

snake.isSnakeTurn = false;

}

break;

case GLUT\_KEY\_LEFT:

if ((snake.direction != GLUT\_KEY\_RIGHT) && (snake.isSnakeTurn))

{

if (snake.direction != GLUT\_KEY\_LEFT)

sound.playSound(SOUNDMOVEINDEX);

snake.direction = GLUT\_KEY\_LEFT;

snake.isSnakeTurn = false;

}

break;

case GLUT\_KEY\_RIGHT:

if ((snake.direction != GLUT\_KEY\_LEFT) && (snake.isSnakeTurn))

{

if (snake.direction != GLUT\_KEY\_RIGHT)

sound.playSound(SOUNDMOVEINDEX);

snake.direction = GLUT\_KEY\_RIGHT;

snake.isSnakeTurn = false;

}

break;

}

}

switch (key)

{

case GLUT\_KEY\_F1:

settings.stopGame();

settings.showMessage(FindWindowA(NULL, settings.getTitle()), CONTROLMSG, NOTFIRSTTIME);

break;

case GLUT\_KEY\_F2:

settings.stopGame();

settings.showMessage(FindWindowA(NULL, settings.getTitle()), HINTMSG, NOTFIRSTTIME);

break;

default:

break;

}

}

#define SPACE 32

#define KEY\_INCVOLMUSIC '1'

#define KEY\_DECVOLMUSIC '2'

#define KEY\_INCVOLSOUND '3'

#define KEY\_DECVOLSOUND '4'

void keyboardInterr(unsigned char key, int x, int y)

{

switch (key)

{

case SPACE: settings.stop\_continueGame();

break;

case KEY\_INCVOLMUSIC: sound.changeMusicVol(3);

break;

case KEY\_DECVOLMUSIC: sound.changeMusicVol(-3);

break;

case KEY\_INCVOLSOUND: sound.changeSoundVol(5);

break;

case KEY\_DECVOLSOUND: sound.changeSoundVol(-5);

default:

break;

}

}

void initGame()

{

LPCSTR title = "SNAKE";

SetConsoleTitleA(title);

char answ;

cout << "Please, input your name:";

cin >> settings.playerName;

if (settings.playerName == "")

strcat\_s(settings.playerName, "player");

cout << endl << "please, chose defficult:" << endl;

for (int i = 1; i <= COUNTOFDIFFICULTS; i++)

cout << i << '(' << DIFFICULTS[i - 1].info << ')' << endl;

bool isCorrectAnsw = false;

while (!isCorrectAnsw)

{

cin >> answ;

if ((answ > '7') || (answ < '1'))

{

cout << "Incorrect answer. Please, input data:" << endl;

isCorrectAnsw = false;

}

else

isCorrectAnsw = true;

}

switch (answ)

{

case '1': settings.difficult = { VERYEASY };

break;

case '2': settings.difficult = { EASY };

break;

case '3': settings.difficult = { NORMAL };

break;

case '4': settings.difficult = { ADVANCED };

break;

case '5': settings.difficult = { HARD };

break;

case '6': settings.difficult = { VERYHARD };

break;

case '7': settings.difficult = { HARDCORE };

break;

default:

break;

}

system("cls");

settings.descrCon = FindWindowA(NULL, title);

ShowWindow(settings.descrCon, SW\_HIDE);

}

void exitFromGame(int value)

{

glutLeaveMainLoop();

}

void initSound()

{

sound.addToPlaylist("1.mp3");

sound.addToPlaylist("2.mp3");

sound.addToPlaylist("3.mp3");

sound.addToPlaylist("4.mp3");

sound.addToPlaylist("5.mp3");

sound.addToPlaylist("6.mp3");

//sound.addToPlaylist("7.mp3");

}

void startGame(int argc, \_TCHAR argv[], TDifficult difficult)

{

srand(time(NULL));

initSound();

wall.spawnWalls(difficult.countOfWalls);

settings.delay = difficult.speed;

settings.fillField();

settings.fillSpace();

food.drop();

glutInit(&argc, (char\*\*)argv);

glutInitDisplayMode(GLUT\_RGB);

glutInitWindowSize(settings.width\*settings.scaleWin \* 2, settings.height\*settings.scaleWin \* 2);

glutInitWindowPosition(settings.startPosWinX, settings.startPosWinY);

char temp[50];

temp[0] = '\0';

strcat(temp, settings.getTitle());

settings.showMessage(NULL, HINTMSG, FIRSTTIME);

settings.showMessage(NULL, CONTROLMSG, FIRSTTIME);

settings.windowIndex = glutCreateWindow(temp);

HWND handle = FindWindowA(NULL, settings.getTitle());

glutDisplayFunc(drawCurrState);

glutTimerFunc(settings.delay, mainEvent, 0);

glutSpecialFunc(keyboardInterrControl);

glutKeyboardFunc(keyboardInterr);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(0, settings.width, 0, settings.height, -1, 1);

sound.changeMusicVol(-80);

sound.startPlayMusic();

glutMainLoop();

SetFocus(handle);

}

int \_tmain(int argc, \_TCHAR argv[])

{

initGame();

startGame(argc, argv, settings.difficult);

return(0);

}

typedef struct \_TDifficult{

char name[10];

int index;

int speed;

int countOfWalls;

char info[400];

float scoreMultiplier;

}TDifficult;

#define VERYEASY "Very easy" , 0, 250, 0, "slow speed, no walls", 1

#define EASY "Easy", 1, 100, 0, "normal speed, no walls", 1.5

#define NORMAL "Normal", 2, 250, 50, "slow speed, few walls", 2.5

#define ADVANCED "Advanced", 3, 150, 50, "normal speed, few walls", 3.5

#define HARD "Hard", 4, 250, 100, "slow speed, many walls", 4

#define VERYHARD "Very hard", 5, 150, 100, "normal speed, many walls", 5

#define HARDCORE "Hardcode", 6, 100, 100, "HARDCORE!!!!", 7

#define COUNTOFDIFFICULTS 7

const TDifficult DIFFICULTS[COUNTOFDIFFICULTS] = { { VERYEASY }, { EASY }, { NORMAL }, { ADVANCED }, { HARD }, { VERYHARD }, { HARDCORE } };

using namespace std;

typedef struct \_TPlayerInfo{

char playerName[20];

long int countOfScore;

char difficult[10];

int diffIndex;

} TPlayerInfo;

#define ENDOFVECTORINFO {"", -1, "", -1}

typedef vector <TPlayerInfo> TArrInfo;

typedef vector<TPlayerInfo>::iterator TRange;

typedef vector<TRange> TRanges;

typedef TRange TPlayerInfoIter;

class TInfo{

public:

void printData();

void addDataToList(TPlayerInfo data);

TInfo();

private:

char \*fileName;

TPlayerInfo player;

TArrInfo arrInfo;

TRanges ranges;

int prevCountScore;

TArrInfo getDataFromFile();

void writeDataToFile();

TRanges getRanges();

void sortData();

void partSort();

void printPlayerResults();

};

#include "stdafx.h"

#include "commonTypes.h"

#include "scoreBoard.h"

using namespace std;

#define NEWRECORFIRSTTIME -1

#define NORECORD -2

void TInfo::printPlayerResults()

{

switch (prevCountScore)

{

case NEWRECORFIRSTTIME:

printf("Congratulations, %s. Your result is %d score(difficult - %s).", player.playerName, player.countOfScore, player.difficult);

break;

case NORECORD:

printf("Congratulations, %s. Your result is %d score(difficult - %s). But you haven't beaten your previous record on this difficult.", player.playerName, player.countOfScore, player.difficult);

break;

default:

printf("Congratulations, %s. You have beaten your previous record on this difficult. Your result is %d score(difficult - %s). Your previous result is %d score.", player.playerName, player.countOfScore, player.difficult, prevCountScore);

break;

}

}

void TInfo::printData()

{

printPlayerResults();

cout << endl << endl <<"-------------------ScoreBoard------------------";

printf("\n%-20s%-10s%-8s", "Player name", "Difficult", "Score");

printf("\n-----------------------------------------------");

for (int i = 0; i < (arrInfo.size()) - 1; i++)

{

printf("\n%-20s%-10s%-8d", arrInfo[i].playerName, arrInfo[i].difficult, arrInfo[i].countOfScore);

}

printf("\n\nTo quit the game print q\n");

}

class TFindFunc{

public:

TFindFunc(TPlayerInfo currPlayer) : currPlayer(currPlayer) {};

bool operator()(TPlayerInfo player) const {

if ((strcmp(player.playerName, currPlayer.playerName) == 0) && (player.diffIndex == currPlayer.diffIndex))

return true;

else

return false;

};

private:

TPlayerInfo currPlayer;

};

void TInfo::addDataToList(TPlayerInfo data)

{

player = data;

if (data.countOfScore != 0)

{

TPlayerInfoIter founded = find\_if(arrInfo.begin(), arrInfo.end(), TFindFunc(player));

if (founded != arrInfo.end())

{

if (founded->countOfScore < player.countOfScore)

{

prevCountScore = founded->countOfScore;

founded->countOfScore = player.countOfScore;

}

else

prevCountScore = NORECORD;

}

else

{

arrInfo.push\_back(player);

prevCountScore = NEWRECORFIRSTTIME;

}

sortData();

writeDataToFile();

}

}

bool compareFuncDiff(TPlayerInfo first, TPlayerInfo second)

{

return (first.diffIndex > second.diffIndex);

}

bool compareFuncScore(TPlayerInfo first, TPlayerInfo second)

{

return (first.countOfScore > second.countOfScore);

}

TRanges TInfo::getRanges()

{

TRanges res;

int currIndex = COUNTOFDIFFICULTS - 1;

TRange currIter = arrInfo.begin();

res.push\_back(arrInfo.begin());

for (int i = 0; i < arrInfo.size(); i++)

{

if (arrInfo[i].diffIndex != currIndex)

{

res.push\_back(currIter);

currIndex--;

}

currIter++;

}

res.push\_back(arrInfo.end());

return (res);

}

void TInfo::partSort()

{

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

{

sort(ranges[i - 1], ranges[i], &compareFuncScore);

}

}

void TInfo::sortData()

{

stable\_sort(arrInfo.begin(), arrInfo.end(), &compareFuncDiff);

ranges = getRanges();

partSort();

}

long long filesize(FILE \*stream)

{

long long pos, length;

pos = ftell(stream);

fseek(stream, 0L, SEEK\_END);

length = ftell(stream);

fseek(stream, pos, SEEK\_SET);

return length;

}

void TInfo::writeDataToFile()

{

FILE \*file;

file = fopen(fileName, "wb");

TPlayerInfo \*buff = new TPlayerInfo;

for (int i = 0; i < arrInfo.size(); i++)

{

\*buff = arrInfo[i];

fwrite(buff, sizeof(TPlayerInfo), 1, file);

}

fclose(file);

}

TArrInfo TInfo::getDataFromFile()

{

FILE \*file;

vector <TPlayerInfo> res;

TPlayerInfo \*buff = new TPlayerInfo;

file = fopen(fileName, "rb");

if (file == NULL)

{

file = fopen(fileName, "wb");

res.push\_back(ENDOFVECTORINFO);

}

else

{

int count = filesize(file) / sizeof(TPlayerInfo);

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

{

fread(buff, sizeof(TPlayerInfo), 1, file);

res.push\_back(\*buff);

}

}

fclose(file);

return res;

}

TInfo::TInfo()

{

fileName = "scoreBoard.dat";

arrInfo = getDataFromFile();

prevCountScore = NORECORD;

}

using namespace std;

typedef char\* TFileName;

typedef HSTREAM TFileDescr;

typedef HSAMPLE TFileDescrSample;

typedef struct \_TPlaylistItem

{

QWORD songLength;

TFileDescr fileDescr;

} TPlaylistItem;

typedef vector <TPlaylistItem> TPlayList;

typedef vector <TFileDescrSample> TSoundList;

#define SOUNDEATINDEX 0

#define SOUNDMOVEINDEX 1

#define SOUNDENDOFTHEGAMEINDEX 2

class TSound{

public:

void startPlayMusic();

void stopPlayMusic();

void changeSoundVol(int add);

void changeMusicVol(int add);

void playSound(int soundIndex);

void addToPlaylist(TFileName musicName);

TSound::TSound(TFileName soundOfEat, TFileName soundOfMove, TFileName soundOfEnd, TFileName endSongName);

void setNextMusicIndex();

TPlayList playList;

int currMusicIndex;

HCHANNEL soundChannel;

void playEndSong();

private:

float currSoundVol;

TSoundList soundList;

HSTREAM endSong;

HSTREAM getDescrFromFileStr(TFileName fileName);

HSAMPLE getDesrcFromFileSmp(TFileName fileName);

};

#include "stdafx.h"

#include "TSound.h"

HSTREAM TSound::getDescrFromFileStr(TFileName fileName)

{

HSTREAM res;

res = BASS\_StreamCreateFile(false, fileName, 0, 0, 0);

return res;

}

HSAMPLE TSound::getDesrcFromFileSmp(TFileName fileName)

{

HSAMPLE res;

res = BASS\_SampleLoad(false, fileName, 0, 0, 3, 0);

return res;

}

TSound::TSound(TFileName soundOfEat, TFileName soundOfMove, TFileName soundOfEnd, TFileName endSongName)

{

BASS\_Init(-1, 44100, BASS\_DEVICE\_STEREO, 0, NULL);

soundList.push\_back(getDesrcFromFileSmp(soundOfEat));

soundList.push\_back(getDesrcFromFileSmp(soundOfMove));

endSong = getDescrFromFileStr(endSongName);

currSoundVol = 1.0f;

}

void CALLBACK playMusic(HSYNC handle, DWORD channel, DWORD data, void \*user)

{

((TSound\*)user)->setNextMusicIndex();

BASS\_ChannelPlay(((TSound\*)user)->playList[((TSound\*)user)->currMusicIndex].fileDescr, true);

}

void TSound::startPlayMusic()

{

setNextMusicIndex();

BASS\_ChannelPlay(playList[currMusicIndex].fileDescr, false);

}

void TSound::addToPlaylist(TFileName musicName)

{

TPlaylistItem currSong;

currSong.fileDescr = getDescrFromFileStr(musicName);

currSong.songLength = BASS\_ChannelGetLength(currSong.fileDescr, BASS\_POS\_BYTE);

BASS\_ChannelSetSync(currSong.fileDescr, BASS\_SYNC\_END, 0, playMusic, this);

playList.push\_back(currSong);

}

void TSound::setNextMusicIndex()

{

int prevIndex = currMusicIndex;

currMusicIndex = rand() % playList.size();

if (playList.size() != 1)

while (currMusicIndex == prevIndex)

{

currMusicIndex = rand() % playList.size();

}

}

void TSound::playSound(int soundIndex)

{

soundChannel = BASS\_SampleGetChannel(soundList[soundIndex], false);

BASS\_ChannelSetAttribute(soundChannel, BASS\_ATTRIB\_VOL, currSoundVol);

BASS\_ChannelPlay(soundChannel, true);

}

void TSound::stopPlayMusic()

{

BASS\_ChannelPause(playList[currMusicIndex].fileDescr);

}

void TSound::changeMusicVol(int add)

{

float currVol;

BASS\_ChannelGetAttribute(playList[currMusicIndex].fileDescr, BASS\_ATTRIB\_VOL, &currVol);

currVol += (add / 100.0f);

if (currVol < 0.0f)

currVol = 0.0f;

if (currVol > 1.0f)

currVol = 1.0f;

for (int i = 0; i < playList.size(); i++)

{

BASS\_ChannelSetAttribute(playList[i].fileDescr, BASS\_ATTRIB\_VOL, currVol);

}

BASS\_ChannelSetAttribute(endSong, BASS\_ATTRIB\_VOL, currVol);

}

void TSound::changeSoundVol(int add)

{

float addedVol = (add / 100.0f);

currSoundVol += addedVol;

if (currSoundVol < 0.0f)

currSoundVol = 0.0f;

if (currSoundVol > 1.0f)

currSoundVol = 1.0f;

}

void TSound::playEndSong()

{

BASS\_ChannelPlay(endSong, false);

}