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). На данный момент имеется много игр, интерпретирующих и расширяющих основные игровых правила оригинальной игры. 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.

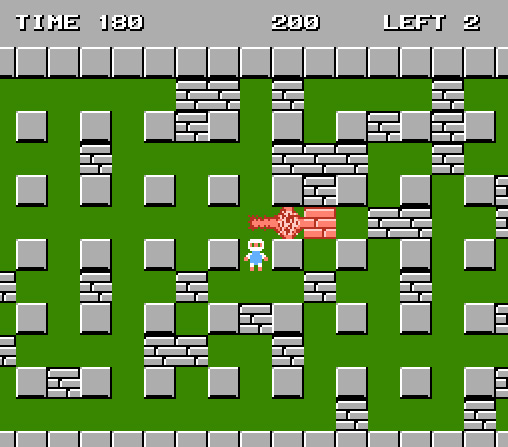


Рисунок 1.1 – Bomberman NES

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.

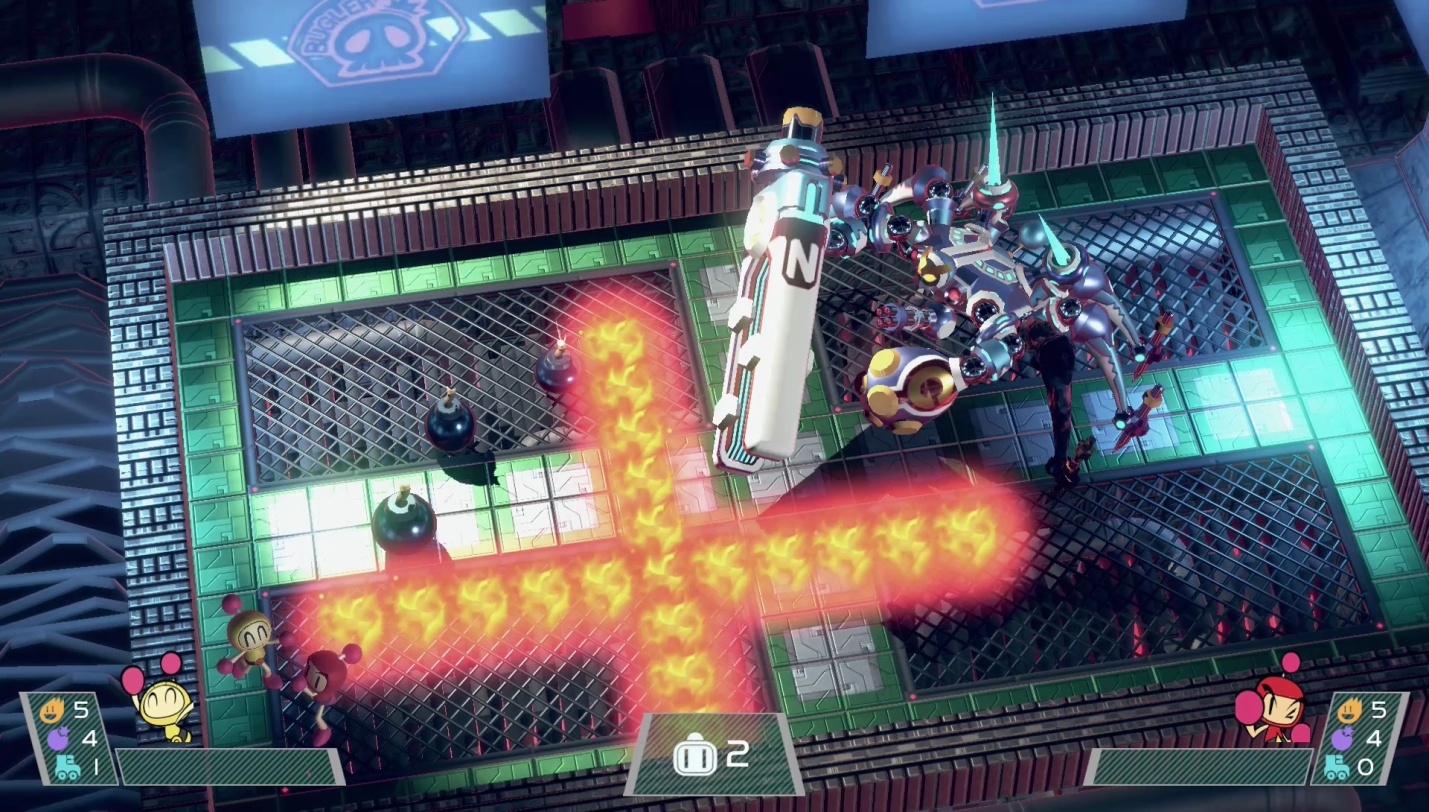


Рисунок 1.2 – Super Bomberman R

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, а также на основе тенденций последних лет популярности мультиплеерных игр было решено реализовать ремейк Bomberman NES для сетевой игры друг против друга.

# **ПОСТАНОВКА ЗАДАЧИ**

Разработать клиент-серверное игровое приложение “Bomberman”, реализующее основные механики одноименной игры, необходимые для игры друг против друга. Приложение будет состоять из клиента и сервера.

Сервер игрового приложения “Bomberman” должен реализовывать следующий функционал:

* основной игровой процесс;
* генерация карты;
* просчёт состояиня карты;
* прослушивание входящих подключений;
* идентификация клиента на сервере;
* обработка сообщений о действиях клиентов;
* рассылку текущего сострояния карты с целью дальнейшей отрисовки его клиентом.

Клиент игрового приложения “Bomberman” должен реализовывать следующий функционал:

* идентификация пользователя;
* подключение к серверу;
* получение состояния карты от сервера;
* отрисовка состояния карты;
* управление и отправка серверу сообщений о действиях пользователя.

# **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. Приложение будет разрабатываться в Windows Forms

# C#

**C#** — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Pascal, и, в особенности, Java — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

# 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.

# Windows Forms

Windows Forms — интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код — классы, реализующие API для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, С++, так и на VB.Net, J# и др.

Внутри .NET Framework, Windows Forms реализуется в рамках [пространства имён](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D1%81%D1%82%D0%B2%D0%BE_%D0%B8%D0%BC%D1%91%D0%BD_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) System.Windows.Forms.

## **Структура приложения**

Данное приложение будет иметь клиет-серверную архитектуру (рисунок 3.1). Сервер (рисунок 3.2) будет состоять непосредственно из серверной части, отвечающей за прослушивание входящих подключений, обработку сообщений о действиях клиентов, рассылку текущего состояния карты клиентам, и части, реализующий игровой процесс (просчёт физики, движения игроков, взырвов и т.д.).

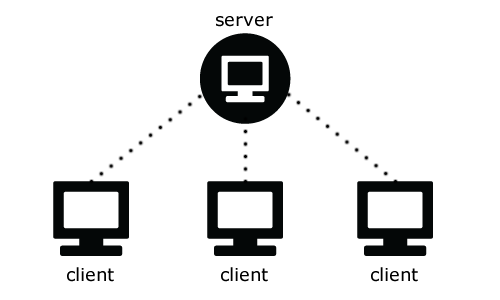


Рисунок 3.1 – Обобщённая схема клиент-серверной архитектуры

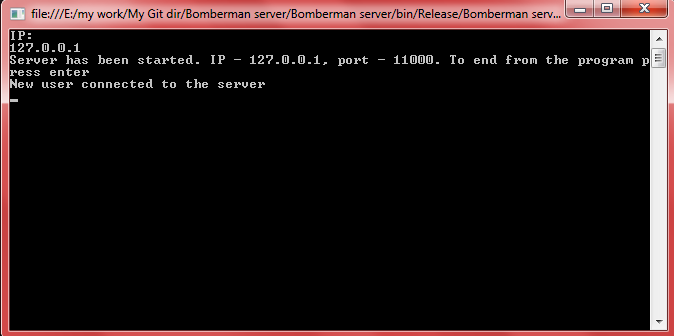


Рисунок 3.2 – Сервер

Клиент будет состоять непосредственно из клиентской части,

отвечающей за подключение к серверу, посылку сообщений о действиях пользователя, получение состояния карты, и части, которая отрисосвывает текущее состояние карты. Приложение клиента будет состоять из окна подключения пользователя к серверу (рисунок 3.3) и непосредственно основного окна приложения, где будет происходить отрисовка (рисунок 3.4).

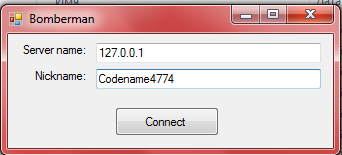
****

Рисунок 3.3 – Окно подключения

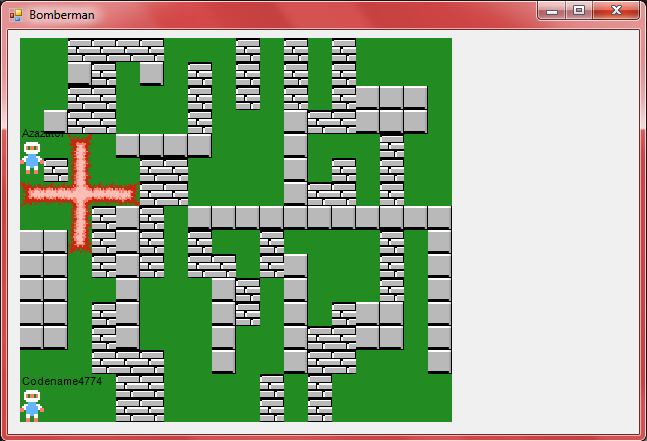


Рисунок 3.4 – Основное окно

## **Основной игровой процесс**

Для начала надо определиться с правилами игры.

Правила игры будут схожи с правилами оригинальной игры 1985 года к некоторыми отличиями: В игре будет присутствовать только мультиплеерный режим, в котором игроки соревнуются друг с другом. Задача каждого игрока убить как можно больше других игроков. В отличие от оригинальной игры на поле будут отсутвовать другие враги кроме остальных игроков. В остальном механики те же: игрок может расставлять бомбы, которые взрываются через определённое количество времени. На карте присутсутвуют разрушаемые и неразрушаемые стены. Убить другого игрока можно взрывом бомбы, но следует быть осторожным, т.к. взрывная волна может убить и самого игрока.

Весь игровой процесс будет просчитываться на сервере (состояние всех объектов, сценарии взрывов, физика движения игроков, физика взрыва стен и т.д.), после чего текущее состояние будет оправляться всем клиентам для дальнейшей отрисовки. Отрисовка будет происходить в окне, созданном средствами Windows Forms. Отрисовка будет происходить при помощи классов платформы .Net, находящихся в пространстве имён System.Drawing, System.Windows.Forms.

### Основное игровое ядро

Как приложение клиент, так и приложение сервер содержат в себе что-то наподобие игрового ядра, которе отвечает за реализацию самого игрового процесса. На стороне клиента за это отвечает класс GameCoreClient, на стороне сервера класс GameCoreServer.

Интерфейс класса GameCoreClient:

…

private Client client; //Содержит ссылку на объект типа Client. Используется для отпраки сообщений

public ObjectsLists objectsList; //Содержит в себе список всех игровых объектов

…

private void DrawPlayers(object state); //Отрисовка игроков

private void DrawStaticWalls(object state); //Отрисовка статических(неразрушаемых) стен

private void DrawDynamicWalls(object state); //Отрисовка динамических(разрушаемых) стен

private void DrawExplosions(object state);

//Отрисовка взрывов

private void DrawBombs(object state);

//Отрисовка бобм

private void DrawPlayerNames(object state); //Отрисовка ников игроков

public void CalcBuff(); //Отрисовка всех объектов. Содержит в себе вызовы выше перечисленных методов

public void ChangeBuffer(); //Смена буфера. В проекте реализована двойная буферизация с целью устранения мерцания изображения

public void RedrawFunc(); //Функция перерисовки окна. Содержит в себе рендер текущего состояния, очистка и смена текущего буфера

public void TimerEvent(object sender, EventArgs e);//Основное событие, происходящее по прерыванию таймера. Содержит в себе вызов метода RedrawFunc

public void KeyPressEvent(object sender, KeyPressEventArgs e); //Обработчик события нажатия клавишы пользователем.

void GetDirection(); //Просчёт направления движения игрока в замисимости от нажатых клавиш

public void KeyUpEvent(object sender, KeyEventArgs e); //Обработчик события отжатия клавишы пользователем

public void Dispose(); //Очистка ресурсов

private void LoadImages(string resDir); //Загрузка всех спрайтов

public void startCore(); //Запуск ядра

public GameCoreClient( int width, int height, Graphics graphicControl, Client client, int id, string dirResources); //Конструктор класса

Интерфейс класса GameCoreServer:

public delegate void SendFunc();

private SendFunc sendFunc;

// Содержит ссылка на метод отправки текущего состояния всех игровых объектов

public PhysicalMap map; //Содержит в себе текущую физическую модель карты

public ObjectsLists objectsList; //Содержит в себе список всех текущих игровых объектов

…

public void ChangePhysicalState(); //Просчитывание физического состояния карты и игроков

public void TimerEvent(object sender, EventArgs e);

); //Основное событие, происходящее по прерыванию таймера. Происходит просчёт движения игроков, просчёт физического состояния карты(происходит вызов метода ChangePhysicalState) и происходит отправка текущего состояния карты

public void OnDeathPlayer(PhysicalObject player); //Обработка события сметри игрока. Запускается сценарий смерти игрока

public void DeletePlayerFromField(object player); //Обработка конца сценария смерти игрока. Удаления игрока с поля

public void ExplosionBomb(PhysicalObject bomb); //Обработка события начала детонации бомбы. Запуск сценария детонации бомбы

public void DeleteBombFromField(object bomb); //Обработка конца сценария детонации бомбы. Запуск сценария взрыва бомбы

public void DeleteExplosionFromField(object explosion); //Обработка конца сценария взрыва

public void StartDestroingDynamicWall(DynamicWall wall); //Обработка события уничтожения бомбы. Запуск сценария уничтожения стены

public void DeleteDynamicWallFromField(object wall); //Обработка конца сценария уничтожения стены

private void GenerateWalls(); //Генерация карты

public void StartCore(); //Запуск ядра

public GameCoreServer(int width, int height, Size playerSize, Size playerOnDeathSize, Size bombSize, Size explosionSize, Size wallSize, string dirResources, SendFunc sendFunc); //Конструктор класса

### Классы игровых объектов

Всего в игре присутствует 4 вида игровых объектов: игрок (рисунок 3.5), бомба (рисунок 3.6), статическая стена (рисунок 3.7), динамическая стена (рисунок 3.8), взрыв (рисунок 3.9). Соответствующие классы: Player, Bomb, PhysicalObject, DynamicWall.

E:\my work\My Git dir\Bomberman client\bomberman.png

Рисунок 3.5 – Игрок

E:\my work\My Git dir\Bomberman client\Bomberman client\bin\Client\Resources\Bomb\bomb.png

Рисунок 3.6 – Бомба

E:\my work\My Git dir\Bomberman client\Bomberman client\bin\Client\Resources\Walls\StaticWall.png

Рисунок 3.7 – Статическая стена

E:\my work\My Git dir\Bomberman client\Bomberman client\bin\Client\Resources\Walls\DynamicWall.png

Рисунок 3.8 – Динамическая стена

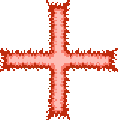


Рисунок 3.9 – Взрыв

Классовая иерархия изображена ниже (рисунок 3.10).

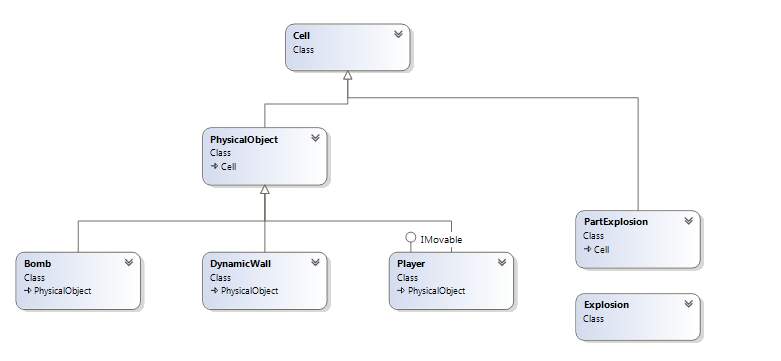


Рисунок 3.10 – Иерархия игровых объектов

Интерфейс класса Cell:

private Point location; //Позиция ячейки

public int X { set {} get {} }

public int Y { set {} get {} }

// Свойства, через которые осуществляется досуп к location

public Cell(Point location); //Коструктор класса

Интерфейс класса PhysicalObject:

public readonly Size size; //Размер объекта

public int currSpriteOffset; // Текущее смещение в спрайте

public virtual void ChangeMapMatrix(PhysicalMap PhysicalMap); //Нанесение объекта на физическую карту

public delegate void DeleteObjectFunc(PhysicalObject obj);

protected DeleteObjectFunc deleteObjectFunc; //Сссылка на функцию удаления объекта из списка

public PhysicalObject(Point location);

public PhysicalObject(Point location, Size size)

: base(location);

public PhysicalObject(Point location, Size spriteSize, DeleteObjectFunc deleteObjectFunc )

: base(location); //Кострукторы класса

Интерфейс класса Bomb:

private int areaOfExplosion; //Радиус взрыва

public Player owner; //Владелец бобмы

public bool isBlowedUp; //Бобма детонирует?

public void TimerEvent(object sender, EventArgs e); //Вызов метода удаления бомбы по окончанию детонации

public override void ChangeMapMatrix(PhysicalMap PhysicalMap); //Нанесение объекта на физическую карту

public Bomb(Point location, int areaOfExplosion, DeleteObjectFunc deleteBombFunc, Player owner)

: base(location); //Конструктор класса

Интерфейс класса DynamicWall:

public bool isBlowedUpNow; //Уничтожается ли сейчас стена?

public bool isWallBlowedUp(PhysicalMap map); //Проверка, взорвана ли стена

public DynamicWall(Point location, Size size);

: base(location, size) //Конструктор класса

Интерфейс класса Player:

public enum Direction { UP, DOWN, LEFT, RIGHT };

public Direction direction { get; set; } //Направление движения игрока

public Direction prevDirection { get; set; } //Предыдущее направление движения игрока

public enum AnimState { TURNUP, TURNUP1, TURNUP2, TURNDOWN, TURNDOWN1, TURNDOWN2, TURNLEFT, TURNLEFT1, TURNLEFT2, TURNRIGHT, TURNRIGHT1, TURNRIGHT2 };

public AnimState currAnimState;

//Текущее состояние анимации движения

public enum BombLevel { low = 2, medium = 3, high = 4 };

public BombLevel bombLevel; //Уровень бобм игрока (радиус бомб, бросаемых данным игроком)

public int prevLocation; //Предыдущее местоположение

private int step = 4; //Длина шага (в пикселях)

public readonly int maxCountBombs; //Максимальное количество бомб данного игрока, которое может одновременно находится на поле

private int currCountBombs;

public int CurrCountBombs { get { } set { } }

//Текущее количество бобм на поле

public delegate void SpawnPlayerFunc();

private bool isDead;

public bool IsDead { get { } set { } }

//Мёртв ли игрок?

private bool isDying;

public bool IsDying { get { } set { } }

//Умирает ли сейчас игрок? (Запущен ли сейча сценарий смерти игрока)

public BombFactory bombFactory; //Данный класс “Производит” бомбы, принадлежащие данному пользователю

public bool isMoved; //Двигается ли сейча игрок?

private Point newLocation = new Point(); //Используется в просчётах физики движения

public readonly int id; //ID пользователся на сервере

public bool isObjectOnWay(PhysicalMap map); //Имеется ли объект на пути движения игрока?

public override void ChangeMapMatrix(PhysicalMap PhysicalMap); //Нанесение объекта на физическую карту

public void OnMove(PhysicalMap map); //Просчёт новых координат

public bool isPlayerBlowedUp(PhysicalMap map); //Проверка, нашёл ли игрок на взрыв

private string thisName;

public string PlayerName { get { } set { } }

//Имя игрока

public Bitmap GetAnimState(Bitmap texture); //Взятие спрайта игрока в соответсвии с текущим состоянием движения

public Player(Point location, Size spriteSize, string name, DeleteObjectFunc deletePlayerFunc, Size bombSize, DeleteObjectFunc deleteBombFunc, int id)

: base(location, spriteSize, deletePlayerFunc); //Конструктор класса

Интерфейс класса PartExplosion:

private int countStates; //Количество состояний

private int currState; //текущее состояние

public int currSpriteOffset; //Текущее смещение в спрайте

public Size size; //Размер

private Explosion.OnEndAllExplosionFunc onEndFunc; //Метод, вызываемый по окончанию сценария части взрыва

public enum KindExplosionTexture { explosionTextureHorizontalMiddle, explosionTextureLeftEdge, explosionTextureRightEdge, explosionTextureVerticalMiddle, explosionTextureUpEdge, explosionTextureBottomEdge }

public Explosion.KindExplosionTexture kindExplosion; //Тип части взрыва

public void ChangeState(); //Изменение состояния части взрыва

public PartExplosion(Explosion.KindExplosionTexture kindExplosion, Size size, Point location, int countStates, Explosion.OnEndAllExplosionFunc onEndFunc) : base(location); //Коструктор класса

Интерфейс класса Explosion:

private int radius; //Радиус взрыва

public List<PartExplosion> partsExplosionLeft;

public List<PartExplosion> partsExplosionRight;

public List<PartExplosion> partsExplosionBottom;

public List<PartExplosion> partsExplosionUp;

public PartExplosion partExplosionCenter;

//Части взрыва

private Point explosionLocation; //Положение центра

private Point newLocation; //Используется в просчётах координат частей взрыва

private int countParts; //Количество частей

public delegate void OnEndAllExplosionFunc(object sender);

private OnEndAllExplosionFunc onEndAllExplosionFunc; //Метод, вызывемый по окочании взрыва

Size size; //размер части взрыва

public enum KindExplosionTexture { explosionTextureHorizontalMiddle, explosionTextureLeftEdge, explosionTextureRightEdge, explosionTextureVerticalMiddle, explosionTextureUpEdge, explosionTextureBottomEdge, explosionTextureCenter };

//Типы спрайтов частей взрыва

private enum ExplosionDirection { UP, DOWN, LEFT, RIGHT }; //Направление взрыва

public void ChangeState(); //Изменение состояния взрыва

public void onEndExplosionFunc(object sender); //Метод, вызываемые при окончании части взрыва

private bool canWePlaceExplosion(int X, int Y, PhysicalMap map, ExplosionDirection direction); //Провека, можно ли расположить в данной точке часть взрыва

private void InitExplosion(Size size, Point location, int radius,PhysicalMap map); //Инициализация взрыва

public void ChangePhysicalMap(PhysicalMap map); //Нанесение объекта на физическую карту

public void DrawPart(BufferedGraphics currBuffer, List<Bitmap> images, PartExplosion partExplosion); //Отрисовка части взрыва

public void DrawExplosion(BufferedGraphics currBuffer, List<Bitmap> images); //Отрисовка взрыва

public Explosion(Size size, Point location, int radius, PhysicalMap map, OnEndAllExplosionFunc funcEnd); //Конструктор класса

### Игровые сценарии

Всего в игре присутствует следующие игровые сценарии: сценарий детонации бобмы (рисунок 3.11), уничтожения динамической стены (рисунок 3.12), сценарий взрыва (рисунок 3.13), сценарий смерти игрока (рисунок 3.14). Сценарии в рамках данного проекта представляют собой запланированное изменение состояния игровых объектов. Количество состояний фиксированное, изменение происходит через определенные интервалы времени.

Сценарии детонации бобмы, уничтожения динамической стены, сценарий смерти игрока схожи по своей сути: в течении всего сценария объект изменяет своё отображеине на карте, по окончанию сценария объект удаляется с игровой сцены. Сценарий взрыва несколько сложнее, т.к. помимо выше перечисленного нужно просчитать положение его частей на карте и их количество.

E:\my work\My Git dir\Bomberman client\Bomberman client\bin\Client\Resources\Bomb\bombExplosion.png

Рисунок 3.11 – Состояния бомбы при детонации

E:\my work\My Git dir\Bomberman client\Bomberman client\bin\Client\Resources\Walls\DynamicWallDestroy.png

Рисунок 3.12 – Состояния динамической стены при уничтожении

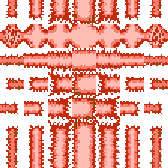


Рисунок 3.13 – Состояния частей взрыва

E:\my work\My Git dir\Bomberman client\bomberman_death.png

Рисунок 3.14 – Состояния игрока во время смерти

В проекте сценарии реализуют классы ScriptEngine, SimpleScript, ExplosionScript. Класс ScriptEngine является главным, который инициализирует все сценарии (скрипты).

Интерфейс класса ScriptEngine:

public delegate void OnEndFunc(object sender); //Сигнатура метода, вызываемого по окончанию сценария

public delegate void OnChangeFunc(); //Сигнатура метода, вызываемого по изсенению состояния

public delegate void TimerEvent(object sender, EventArgs e);//Сигнатура метода, вызываемого по страбатыванию таймера

public void StartSimpleScript(PhysicalObject obj, OnEndFunc onEndFunc, int delay, int countStates);

public void StartSimpleScript(PhysicalObject obj, Size onScriptSize, OnEndFunc onEndFunc, int delay, int countStates); //Запуск простого скритпа, т.е. всех, кроме скрипта взрыва

public void StartExplosion(Explosion explosion, OnEndFunc onEndFunc, int delay, int countStates); //Запуск скрипта взрыва

Класс SimpleScript отвечает за просчёт простых сценариев (детонация бомбы, уничтожение стены, смерть игрока).

Интерфейс класса SimpleScript:

private int countStates; //Количество состояний в сценарии

private int currState; //Текущее состояние

private dynamic obj; //Изменяемый объект

private ScriptEngine.OnEndFunc onEnd; //Метод, вызываемый по окончанию сценария

…

public void OnTimerEvent(object sender, EventArgs e); //Событие изменение состояния

private void SetOffset(); //Установка смещения в спрайте

public SimpleScript(PhysicalObject obj, ScriptEngine.OnEndFunc onEndFunc, int countStates, int delay);

public SimpleScript(PhysicalObject obj, Size onScriptSize, ScriptEngine.OnEndFunc onEndFunc, int countStates, int delay); //Конструкторы класса

Интерфейс класса Explosion схож с интерфейсов класса SimpleScript.

## **Серверная часть**

Схему работы сервера в данном проекте можно описать следующей схемой (рисунок 3.1). После инициализации и генерации карты, сервер начинает прослушивать входящие соединения. При подключении нового игрока сервер создаёт новое соединение с данным клиентом, генерирует уникальный id для данного пользователя в рамках данного сервера, после чего сервер начинает отсылать текущее состояние карты пользователю. Также сервер анализирует приходящие сообщения от клиентов и вносит изменения в состояние карты.

В рамках данного проекта будет использоваться стек протоколов TCP/IP, а также классы платформы .Net пространства имён System.Net, System.Net.Sockets.

1. Прослушивание входящих соединений. Создание соединения клиента с сервером. Отсылка состояния карты

За серверную часть отвечает класс ServerCore.

Интерфейс класса ServerCore:

public readonly IPAddress ipAdress; //Информация о IP адресе сервера

public readonly int portControl; //Порт входящих соединений

public readonly int maxLengthQueue; //Максимальная длина очереди входящих соединений

private int bufferSize;

public readonly IPEndPoint ipEndPointControl; //Конечная точка

public readonly Socket socketListener; //Сокет, прослушивающий входящие соединения

public readonly List<Socket> socketsList; //Список сокетов пользователей, на которые идёт отправка информации

public GameCoreServer gameCoreServer; //Ссылка на игровое ядро сервера

public BinaryFormatter serializer; //Отвечает за сериализацию состояния карты

public MessageAnalyzerServer messageAnalyzer; //Анализирует сообщения от пользователей

private int idCounter; //Счётчик id. Предназначен дял генерации уникального id для нового пользователя

public ServerCore(string host, int portControl, int maxLengthQueue, int sendFrequency); //Конструктор класса

public void SendData(); //Посылка текущего состояния карты

private void ReceiveCallback(object sender, SocketAsyncEventArgs e); //Вызывается при получении данных от пользователя. Происходит анализ сообщения

private void SendId(Socket client); //Посылка id пользователю

private void AddPlayerToList(string playerName); //Добавление игрока в список игроков

public void StartListen(object state); //Старт прослушки сходящих сообщений

Схема подключения была описана ранее (раздел 4.3). Следует отметить лишь то, что отсылка данных идёт по таймеру.

1. Анализ сообщений

За анализ вообщений отвечает класс MessageAnalyzer. Описание типов сообщений содержит класс KindMessages.

Интерфейс класса MessageAnalyzer:

GameCoreServer gameCoreServer; //Ссылка на игровое ядро сервера. Через неё происходит изменение состояния карты

public int GetNextValue(byte[] message, ref int i); //Взять следующее значение из сообщения

private Player FindPlayer(int id); //Найти игрока в списке, который отослал сообщение

public void AnalyzeMessage(byte[] message); //Анализ всего сообщения

public MessageAnalyzerServer(GameCoreServer gameCoreServer); //Конструктор класса

Интерфейс класса KindMessages:

public enum KindMessage { Player = 0 }; //Тип сообщения

public enum KindPlayerMessages { NewDirection = 0, Spawn = 1, Death = 2, PlaceBomb = 3, Connect = 4, Disconnect = 5, Location = 6, StopWalking = 7 }; //Информация сообщения игрока: игрок выбрал новое направление движения, игрок нажал клавишу возрождения, игрок умер, игрок нажал клавишу установки бобмы, игрок присоединился к серверу, игрок отсоединился от сервера, позиция игрока на карте, игрок остановился

public enum Direction { UP, DOWN, LEFT, RIGHT }; //Направление движения игрока

Разбор сообщения представляет собой последовательное чтение значение и их последовательный анализ. Разбор сообщения продолжается в зависимости от прочитанного значения.

## **Клиентская часть**

Клиентская часть данного проекта представлена классами Client и KindMessages.

Схема работы клиента: клиент, после подключения к серверу начинает получать состояние карты и отрисовывать его, а также он в зависимости от действий пользователя генерирует соответствующее сообщение и посылает его серверу. Интерфейс класса KindMessages описан в разделе 3.4.2.

Интерфейс класса Client:

…

private byte[] receivedData; //буфер для получаемых данных

public GameClasses.GameCoreClient gameCore; //Ссылка на игровое ядро клента. Через неё идёт передача данных для отрисовки

private BinaryFormatter serializer; //Десериализиует состояние карты

public readonly int id; //id клиента

public Client(string host, int portControl, string playerName); //Конструктор класса

private void GetBufferFromServer(object sender, SocketAsyncEventArgs e); //получение состояния карты от сервера

public void SendMessageToServer(params int[] data); //посылка сообщения клиенту

public void Dispose(); //очистка ресурсов

public void StartRecieving(object state); //Запуск получения данных от сервера

# **ПРОВЕРКА КОРРЕКТНОСТИ РАБОТЫ ПРОГРАММНОГО СРЕДСТВА**

Ввиду небольшого размера проекта и небольшого функционала и количетсва рабочих окон данного проекта, было решено описать проверку корректности работы данного средства.

# **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);

}