Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Операционные системы и системное программирование (ОСиСП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

по курсовому проекту
на тему

ИГРОВОЕ ПРИЛОЖЕНИЕ «АРКАНОИД»

БГУИР КП 1-40 01 01 616 ПЗ

Выполнил
студент: гр. 851006 Матюшонок М.С.

Проверил: Жиденко А.Л.

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ПОИТ
_____
(подпись)
Лапицкая Н.В. 2020г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Матюшонку Максиму Сергеевичу*

1. Тема работы *Игровое приложение «Арканоид»*

2. Срок сдачи законченной работы *02.12.2020г.*

3. Исходные данные к работе *Среда разработки JetBrains CLion 2020.2.1 x64. Язык программирования C++. Интерфейс программирования Windows API. Реализовать следующий функционал: загрузку игрового уровня из файла. Платформу, управляемую пользователем с использованием клавиатуры и мыши. Создать мячик, взаимодействующий с платформой и блоками. Предусмотреть масштабирование окна и корректное отображение игровой сессии. Реализовать систему генерации, взаимодействия игровых бонусов с пользователем и игровой сессией. Создать систему попыток на прохождение игры. В случае, если закончились попытки или уровни – необходимо дать пользователю возможность сохранить своё имя в таблице лидеров. Реализовать возможность посмотреть список лидеров*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение
1 Анализ предметной области
2 Постановка задачи
3 Разработка программного средства
4 Руководство по установке и использованию программного средства
Заключение
Список использованных источников
Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)
Схема алгоритма в формате А1
6. Консультант по курсовой работе *Жиденко А.Л.*

7. Дата выдачи задания *26.10.2020г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):
Раздел 1. Введение к 31.10.2020г. – 10 % готовности работы
Раздел 2 к 5.11.2020г.– 30% готовности работы;
Раздел 3 к 10.11.2020г. – 60% готовности работы;
Раздел 4 к 15.11.2020г. – 80% готовности работы;
Раздел 5.Заключение. Приложения к 20.11.2020г. – 90% готовности работы; оформление пояснительной записки и графического материала к 25.11.2020г. – 100% готовности работы.
Защита курсового проекта с 02.12.2020г.

РУКОВОДИТЕЛЬ _____ *Жиденко А.Л.*
(подпись)

Задание принял к исполнению _____ *Матюшонок М.С. 26.10.2020г.*
(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день наиболее популярной и универсальной техникой для дома и офиса является персональный компьютер. Он предоставляет огромные возможности для работы и отдыха. С помощью современного компьютера без труда можно обрабатывать фото и видео, работать с электронными таблицами и документами, играть в видеоигры и смотреть фильмы.

История компьютерных игр началась задолго до появления персональных компьютеров в привычном нам понимании. До этого были популярны аркадные автоматы и игровые приставки. Однако время шло, а компьютеры становились быстрее, меньше и самое главное дешевле. Компьютеры стали доступнее и следовательно, – перестали быть инструментом ученых и программистов.

Средний пользователь не хочет видеть на экране терминал из-за отсутствия навыков работы с ним. По этой причине в операционных системах начали внедрять графические интерфейсы для повышения интуитивности и простоты работы. Также, для повышения удобности, начали появляется первые манипуляторы мыши. Чтобы обучить пользователя работы с ними, самым простым решением оказалось создать игру, где пользователь в игровой форме получит навыки работы с новыми устройствам взаимодействия с компьютером.

Одной из старейших и интереснейших игр была игра «Arkanoid». Данная игра появилась ещё в 1986 для вышеупомянутых игровых автоматов, и стала настолько популярной, что именно её название стало нарицательным для класса подобных игр. Портативные компьютеры она также не обошла стороной, и принята была с той же теплотой.

В результате вышеизложенного, было принято решение создать игровое приложение «Arkanoid» для компьютеров под управлением Windows с применением интерфейса программирования Windows API.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Информация об игре «Arkanoid»

Видеоигра игра «Arkanoid» была разработана для игровых автоматов компанией Taito в 1986. Игра основана на играх серии «Breakout» фирмы Atari.

Принцип игры заключается в следующем:

Игрок контролирует небольшую платформу-ракетку, которую можно передвигать горизонтально от одной стенки до другой, подставляя её под шарик, предотвращая его падение вниз. При падении шарика вниз – пользователь теряет одну из данных в начале игры 3-х жизней. Удар шарика по кирпичу приводит к разрушению кирпича. После того как все кирпичи на данном уровне уничтожены, происходит переход на следующий уровень, с новым набором кирпичей. Есть и некоторое разнообразие: определённые кирпичи нужно ударять несколько раз, иногда из разбитых блоков выпадают случайные бонусы, при поимке которых платформой-ракеткой происходит соответствующее изменение игрового процесса.

Данная игра как было сказано выше, не является оригинальным проектом, так принципы игры были заложены ещё в играх серии «Breakout», однако именно реализация в виде «Arkanoid» стала самой популярной, и именно с тех пор начало появляться огромное множество аналогов данной игры, вносящих некоторые коррективы в правила игры.

1.2 Анализ существующих приложений

1.2.1 Игровое приложение «Bricks DEMOLITION»

С данным вариантом реализации принципов игры «Arkanoid», я познакомился не более чем полгода назад. Основная задумка игры была практически нетронута, интерфейс интуитивно понятен и не вызывает сложностей. Данная игра предназначена для мобильных устройств под управлением операционной системы Android и распространяется на бесплатной основе в магазине приложений «Google Play», однако, на условиях наличия рекламы после каждого поражения.

В главном меню игры (см. рисунок 1.1) вам доступны следующие действия:

– начать игру, с помощью кнопки «PLAY»;

– зайти в настройки, с помощью кнопки «OPTIONS»;

– получить информацию по управлению, с помощью кнопки «HELP»;

– выйти из игры, с помощью кнопки «EXIT».

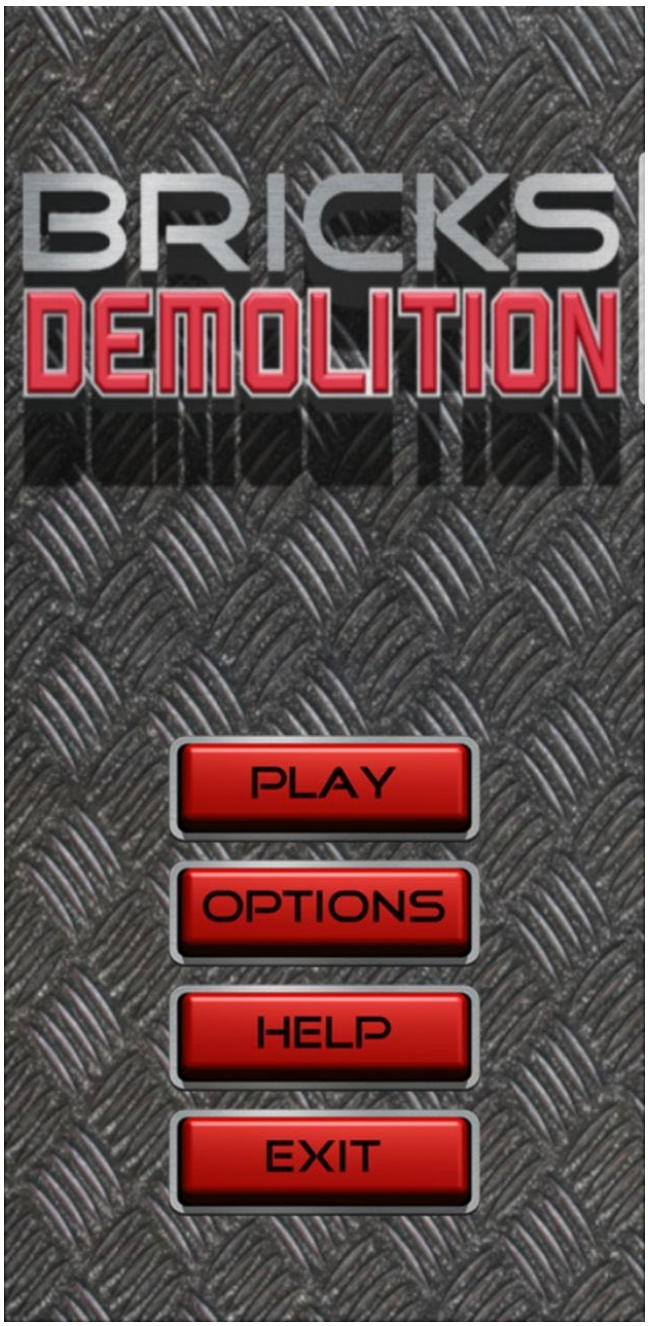

Рисунок 1.1 – Интерфейс главного меню «Bricks DEMOLITION»

При нажатии кнопки «PLAY» игра начинается (см. рисунок 1.2) и на экране появляется следующая информация:

– снизу – платформа-ракетка, управляемая пользователем;
– в нижнем левом углу – количество оставшихся жизней;
– в верхнем левом углу – количество набранных игровых очков;
– сверху по центру – номер текущего уровня;
– в верхнем правом углу — максимальный набранный опыт;
– в игровой зоне – мячик и блоки.

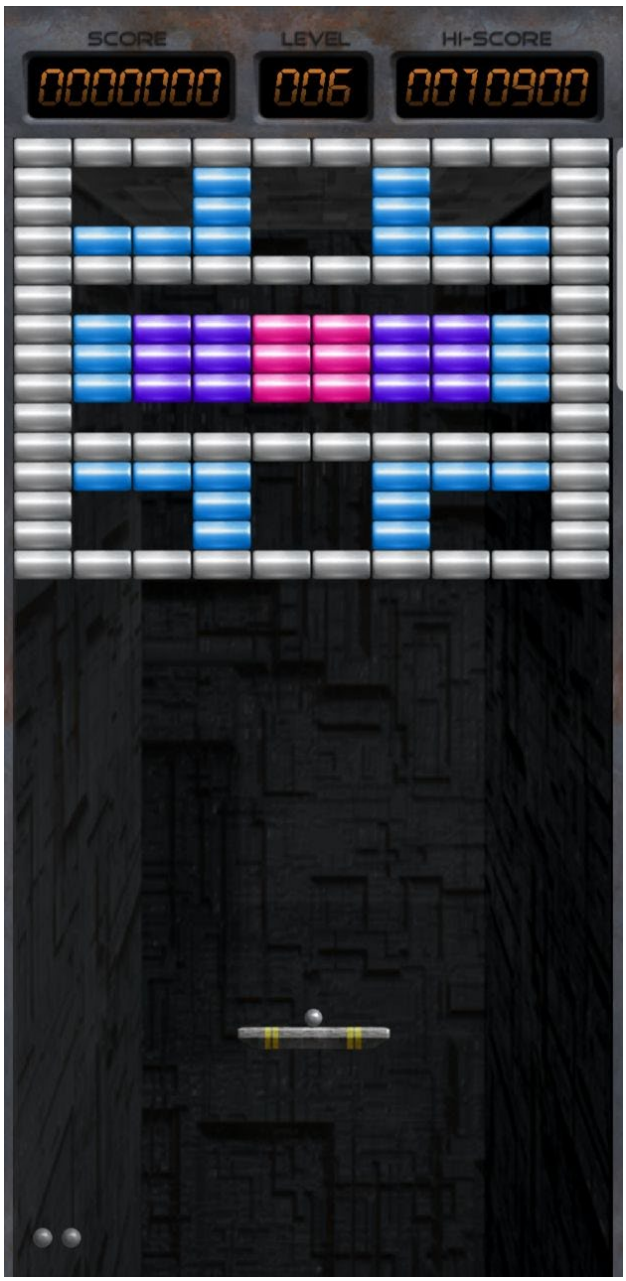

Рисунок 1.2 – Интерфейс перед началом игры в «Bricks DEMOLITION»

В ходе игрового процесса, пользователь разбивает находящиеся выше блоки, с помощью мячика и платформы-ракеты. В результате разрушения блока, может выпасть, случайным образом выбранный, бонус:

– увеличение платформы;
– уменьшение платформы;
– увеличение количества шаров в 2 раза;
– превращение мяча в огненный шар, разбивающий любые блоки;
– бонусные очки;
– добавление по краям платформы-ракетки автоматического оружия;
– магнит для шариков.

Результаты активации некоторых бонусов приведен на рисунке 1.3.

Рисунок 1.3 – Результат применения бонусов «огненный шар» и «оружие» в «Bricks DEMOLITION»

Анализируя данное игровое приложения можно определить следующие недостатки:

– наличие рекламы после поражения;
– перегрузка игровой зоны эффектами, мешающими игровому процессу;
– устаревший дизайн;
– большой вес приложения (23.82 МБ);

К достоинствам данного приложения можно отнести:

– интуитивно понятный интерфейс;
– большое количество бонусов;
– модель бесплатного распространения приложения.

1.2.2 Игровое приложение «Popcorn»

Данная игра была написана в 1988 году французской компанией Lacral Software под операционную систему DOS. Для запуска данной программы, мною использовался эмулятор «DosBox».

В отличие от предыдущего рассмотренного приложения, данная игра являлось коммерческой. Все основы игры «Arkanoid» здесь реализованы и модернизированы. Игра встречает большим количеством интересных, но немного навязчивых анимаций. В главном меню (см. рисунок 1.4) пользователь имеет следующие возможности:

– начать игру, с помощью клавиши «F1»;
– просмотреть демонстрацию игры, с помощью клавиши «F2»;
– выбрать управление мышью, с помощью клавиши «F3»;
– выбрать управление клавиатурой, с помощью клавиши «F4»;
– изменить управление клавиатурой, с помощью клавиши «F5»;
– просмотреть таблицу лидеров, с помощью клавиши «F6»

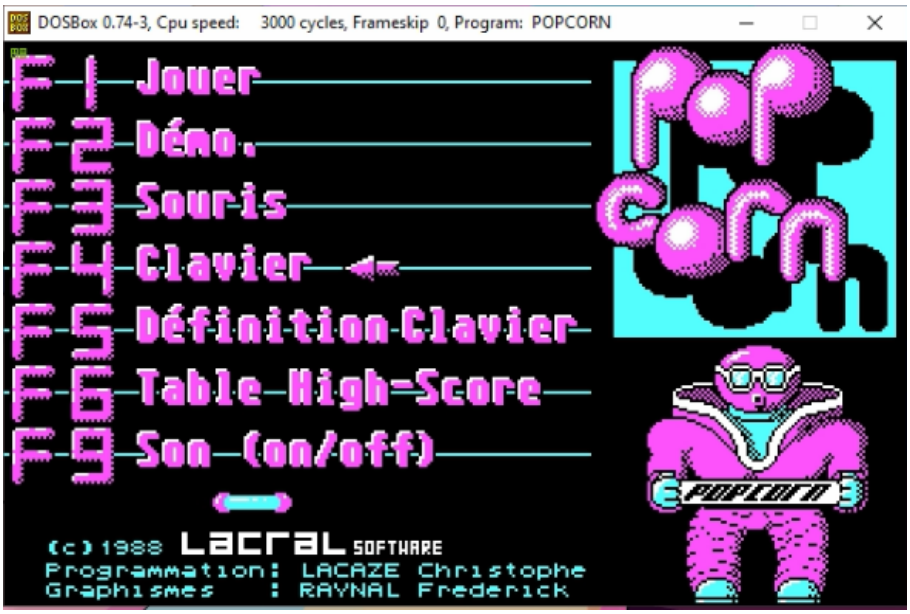

Рисунок 1.4 – Главное меню «Popcorn»

В случае выбора начала игры, пользователь вводит своё имя, а также, имеет возможность ввести имя других игроков, для дальнейшей игры по очереди (см. рисунок 1.5). Данный процесс фактически означает создание локальных аккаунтов, на данную игровую сессию.

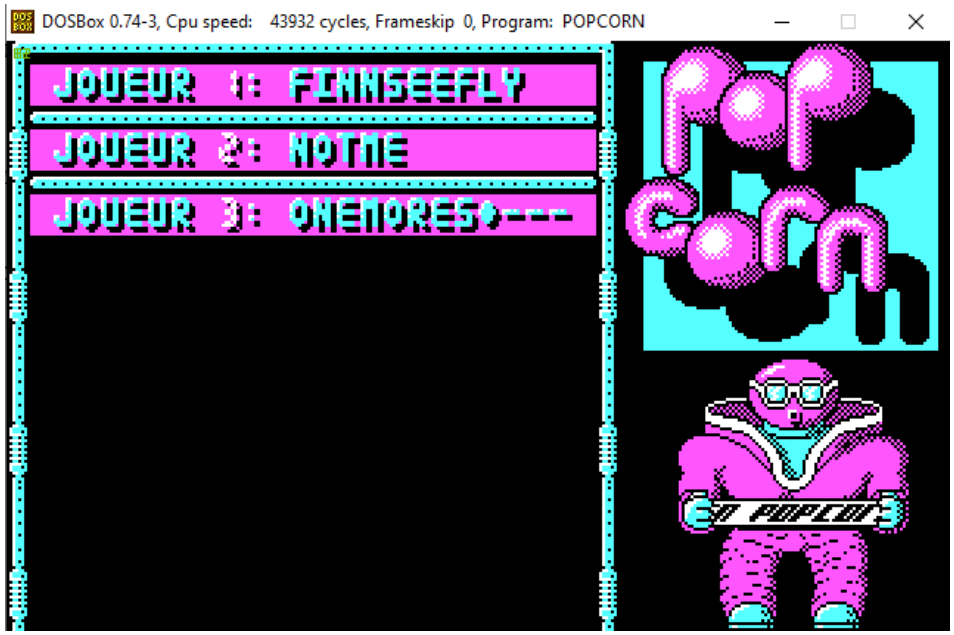

Рисунок 1.5 – Создание аккаунтов в «Popcorn»

Для начала игры необходимо нажать «Enter» в пустом поле для ввода. После этого мы увидим классическую ситуацию, для начала игры, где в правой части интерфейса находятся основная информация, а по центру, располагается игровая зона с платформой-ракеткой, мячиком и блоками (см. рисунок 1.6).

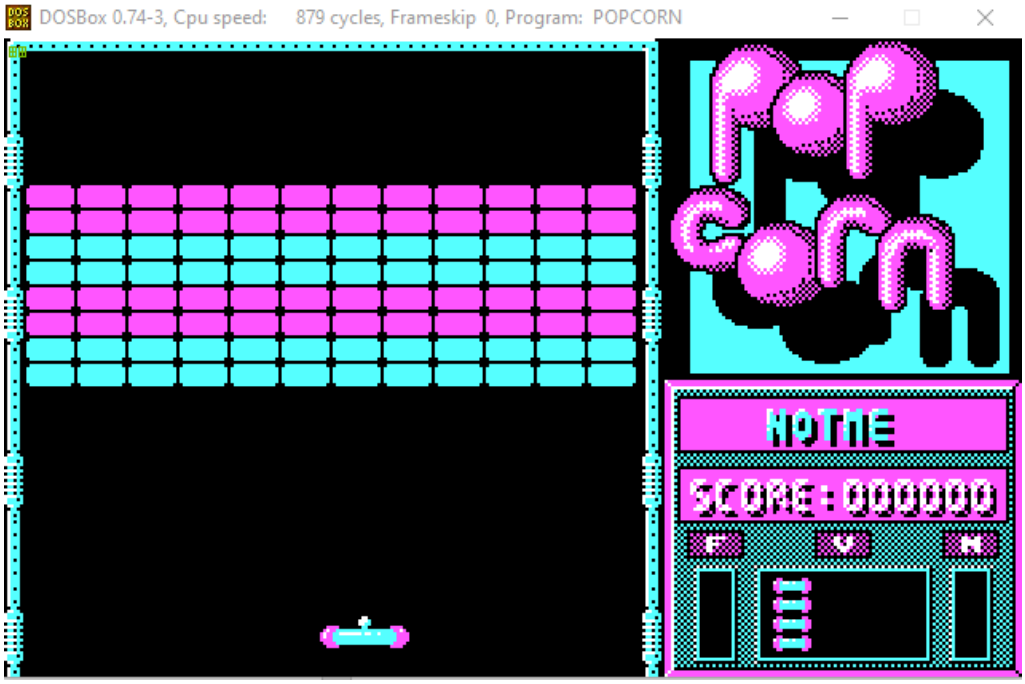

Рисунок 1.6 – Начало игры в «Popcorn»

Дальнейший игровой процесс отличается от предыдущего примера тем, что в игре присутствуют неигровые персонажи, задача которых мешать разбивать блоки (см. рисунок 1.7).

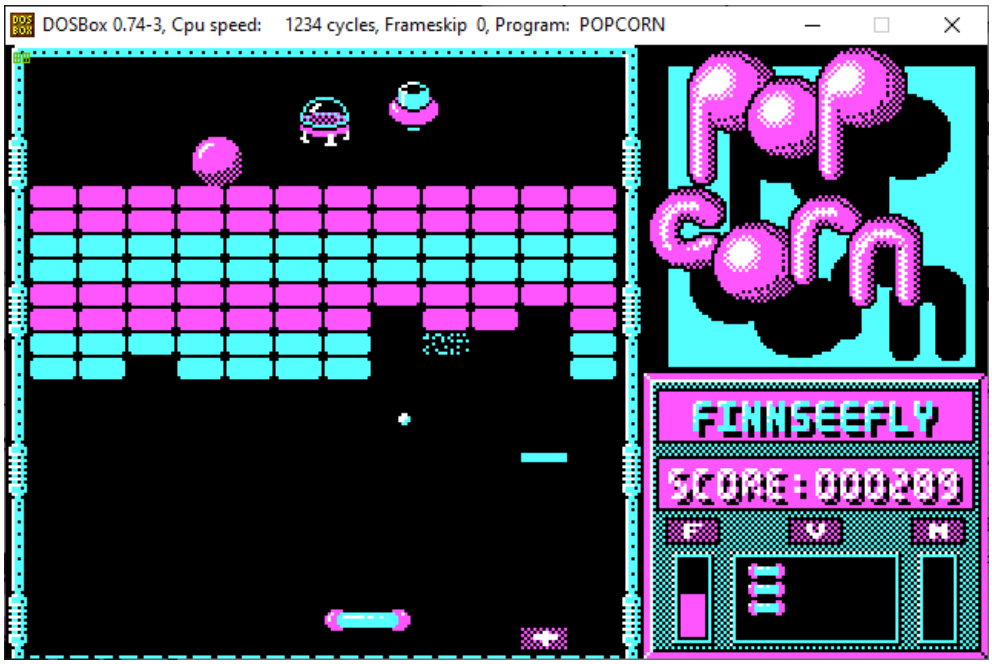

Рисунок 1.7 – Неигровые персонажи в «Popcorn»

Анализируя данное игровое приложения можно определить следующие недостатки:

– слишком навязчивые анимации;
– данное приложение не поддерживается современными ОС;
– единственный язык локализации – французский;

К достоинствам данного приложения можно отнести:

– высокое качество анимаций и графики по меркам 1988 года;
– возможность локального мультиплеера;
– наличие неигровых персонажей;
– хорошо проработанная механика взаимодействия игрового мира;

2 ПОСТАНОВКА ЗАДАЧИ

Разработать игровое приложение на языке c++, с использованием интерфейса программирования Windows API, назначением которого является предоставление пользователю возможности, при наличии компьютера под управлением Windows, сыграть в развивающую реакцию и навыки владения мышью адаптацию игры «Arkanoid». В программе должно быть реализовано:

– платформа-ракетка, управляемая пользователем;
– блоки разных цветов и разной прочности;
– мячик, взаимодействующий с платформой, блоками и игровой зоной;
– возможность сохранения своего результата в таблице лидеров;
– потеря одной игровой жизни в случае падения мяча за нижнюю грань;
– возможность получения случайных бонусов из разбитых блоков;
– возможность поставить игру на паузу;
– управление клавиатурой или мышью;
– просмотр таблицы лидеров;
– поддержка загрузки с дисков новых уровней;
– масштабирование интерфейса и игровой зоны под размеры окна;

При разработки игрового приложения использовать среду разработки JetBrains CLion 2020.2.1 x64, версию стандарта языка c++ 20, систему сборки проекта CMake. При разработке придерживаться концепций объектно-ориентированного программирования.

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Информационная база задачи

3.1.1 Класс GameSession

Данный класс отвечает за хранение данных и состояний сессии, обработку действий пользователя, обработку отрисовки игрового процесса. Выполняет загрузку и выгрузку таблицы лидеров в память. Является ключевым классом данного приложения.

3.1.2 Класс DrawableShape

Данный класс является базой, на основе которой строятся остальные графические элементы всей игры. Содержит в себе информацию обо всех размерах объекта, смещениях в игровой зоне, масштабе, а также всю информация необходимую для отрисовки. В классе определены методы для расчетов всех координат, методы определения столкновений объектов и непосредственно методы отрисовки.

3.1.3 Класс Ball

Данный класс наследуется от класса DrawableShape и является представлением мяча в игре. Содержит в себе информацию о скорости экземпляра, угле наклона относительно координат окна и информацию о статусе объекта. В классе определены методы для вычисления следующего положения объекта.

3.1.4 Перечисление BonusType

Представляет из себя перечисление типов бонусов. В зависимости от данного значения определяется эффект от бонуса.

3.1.5 Класс Bonus

Данный класс наследуется от класса DrawableShape и является представлением бонуса в игре. Содержит в себе информацию о типе бонуса и количестве игрового опыта, получаемого при активации. В классе определены методы для расчета следующего положения и получения свойств объекта.

3.1.6 Перечисление BrickType

Представляет из себя перечисление типов блоков. В зависимости от данного значения определяется количество ударов необходимое для разрушения, количество опыта за разрушения блока и количество опыта, получаемое из бонусов, выпавших из него.

3.1.7 Класс Brick

Данный класс наследуется от класса DrawableShape и является представлением блока в игре. Содержит в себе информацию о типе блока, количестве ударов необходимого для разрушения и количестве опыта за разрушение блока. В классе определены методы для получения информации о свойствах объекта и метод для нанесения удара по блоку.

3.1.8 Класс Platform

Данный класс наследуется от класса DrawableShape и является представлением платформы в игре. Содержит в себе информацию о степени расширения или сжатия данного экземпляра. В классе определены методы для обработки события перемещения платформы мышью или нажатием на клавиши, переопределяет методы класса DrawableShape для расчетов корректных координат с учетом коэффициента растяжения.

3.1.9 Заголовочный файл Complementary

В данном файле с помощью директивы препроцессора #define объявлены необходимые константы и объявлены вспомогательные структуры FloatRect и BoolRect. Содержит функции базовых манипуляций с данными.

3.1.10 Файл исполняемого кода main

Содержит в себе точку входа и функции обработки сообщений к основному окну и диалоговому окну. Практически все сообщения к основному окну перенаправляются к объекту игровой сессии.

3.2 Схема алгоритмов решения задачи по ГОСТ 19.701-90

3.2.1 Схема алгоритма метода GameplayProcessor класса GameSession

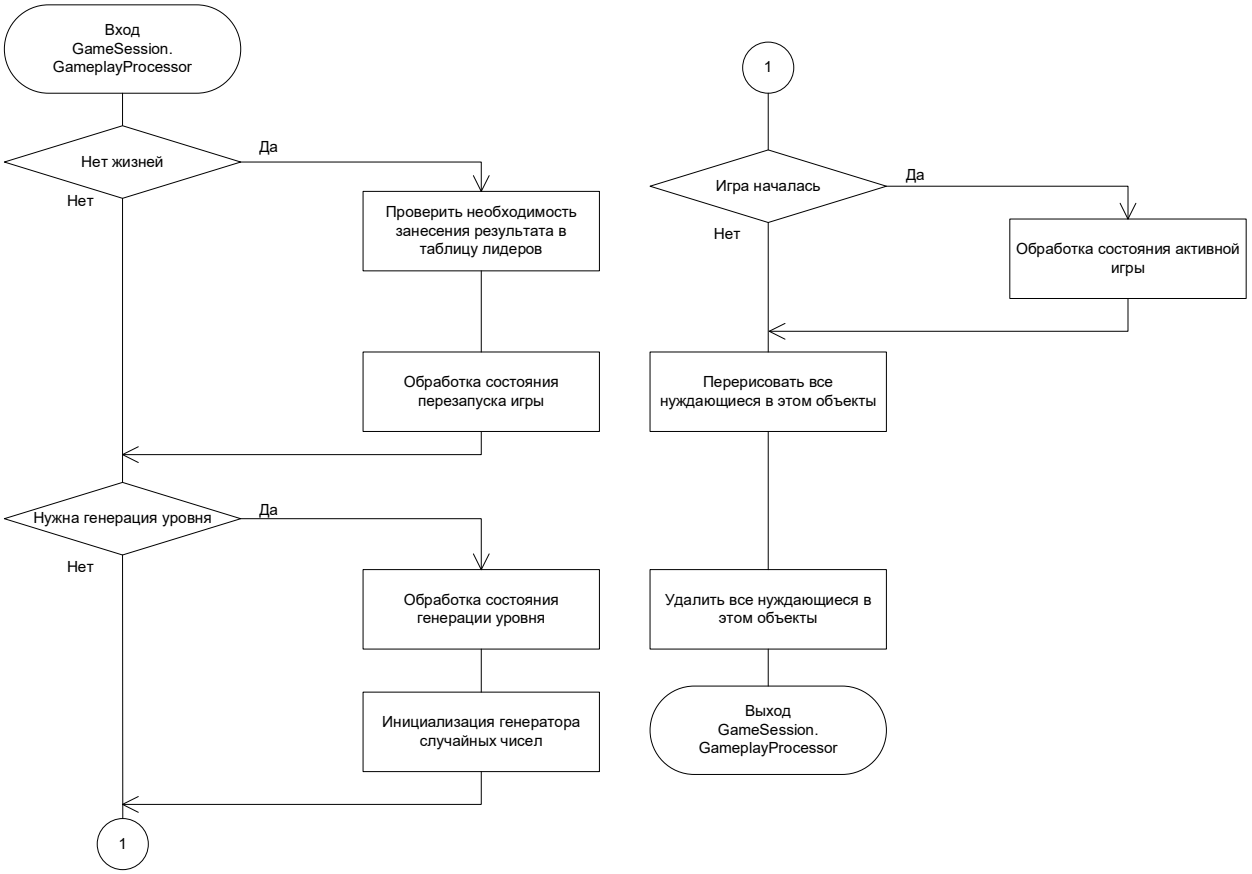

Рисунок 3.1 – Схема алгоритма метода GameplayProcessor класса GameSession

3.2.2 Схема алгоритма метода ResizeEvent класса GameSession

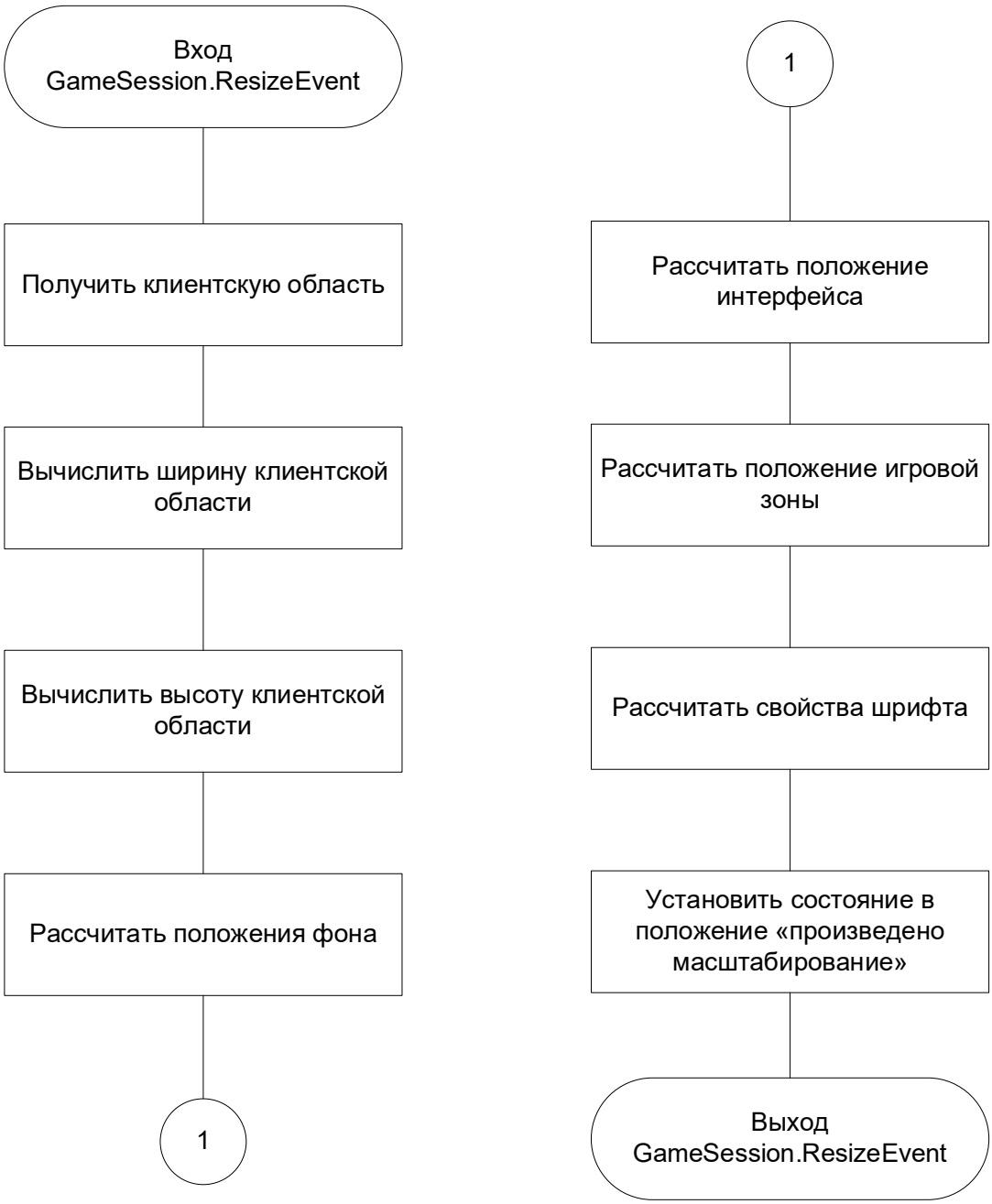

Рисунок 3.2 – Схема алгоритма метода ResizeEvent класса GameSession

3.3 Графический интерфейс

3.3.1 Главная игровая форма в начальном состоянии

При открытии приложения вы увидите главную игровую форму (см. рис. 3.3).

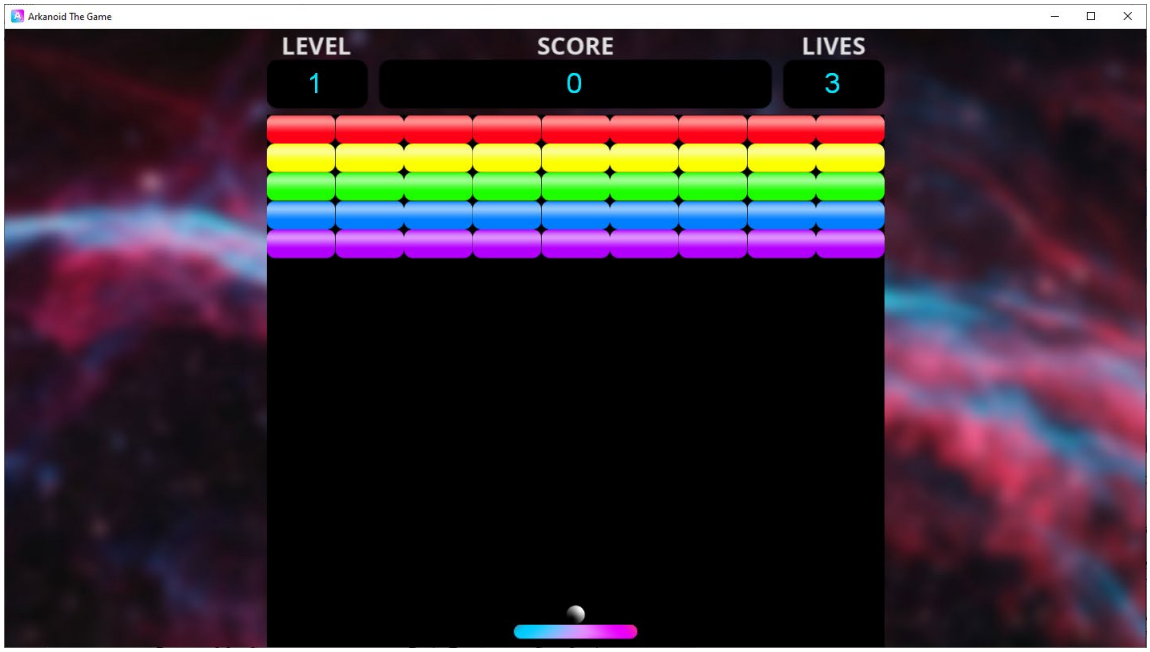

Рисунок 3.3 – Интерфейс главной игровой формы

Составляющие главной игровой формы:

– Фон, масштабирующийся в зависимости от размеров окна (см. рисунок 3.4).

– Игровая зона, подстраивающаяся под размеры окна с сохранением пропорций (см. рисунок 3.4).

– Блоки уровня, расположенные в зависимости от файла конфигурации данного уровня.

– Платформа, расположенная внизу игровой зоны, управляемая пользователем с помощью мыши, либо клавиатуры.

– Мячик, расположенный по центру платформы и следующий за ней до начала игры.

– Показатель номера уровня под обозначением «LEVEL».

– Показатель количества набранных игровых очков под обозначением «SCORE».

– Показатель количества жизней под обозначением «LIVES».

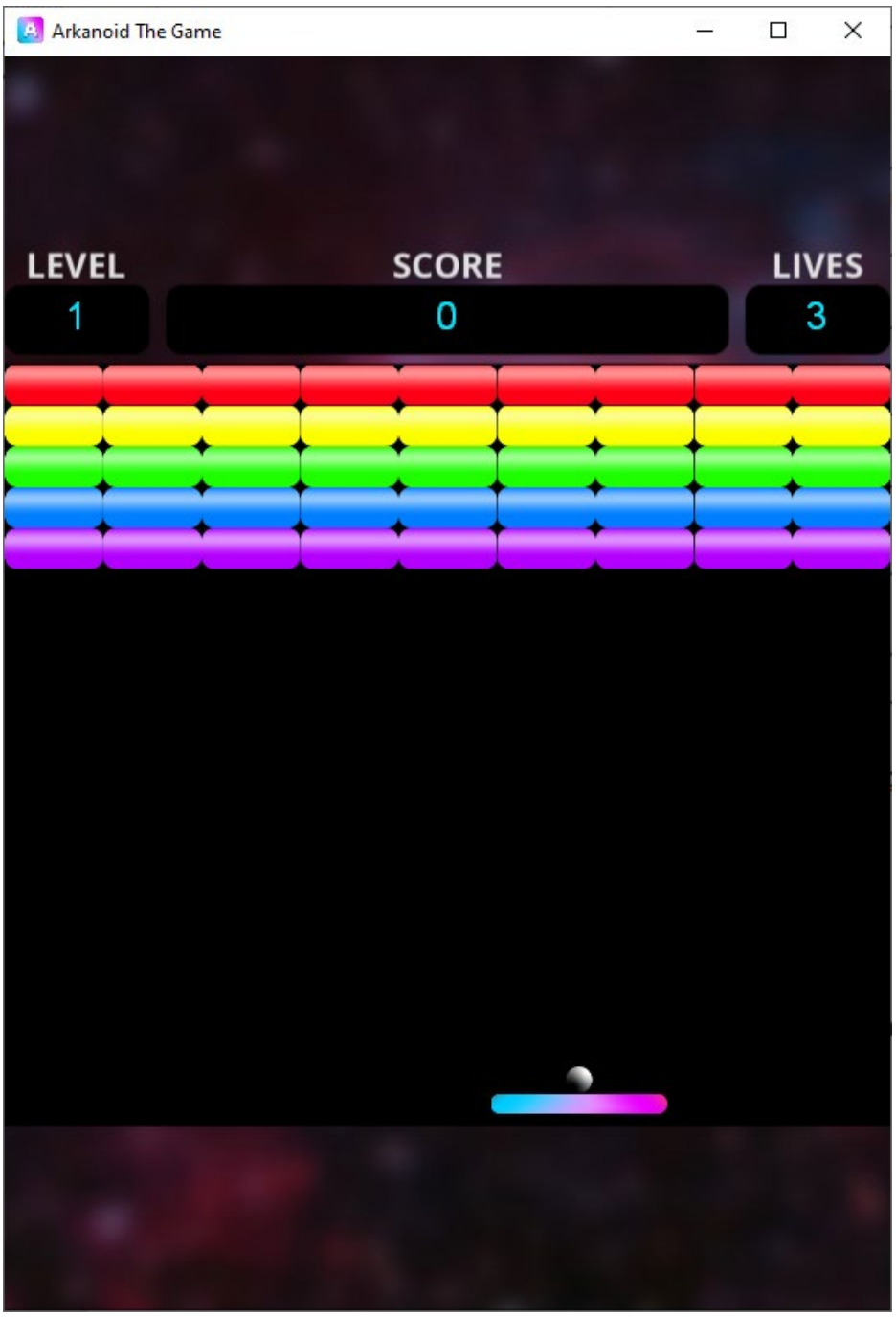

Рисунок 3.4 – Пример подстройки фона и игровой зоны под размеры окна

3.3.2 Главная игровая форма в состоянии паузы

При нажатии клавиши «Esc» главной игровой формы, игра переходит в состояние паузы и меняет соответствующим образом главную игровую форму (см. рисунок 3.5).

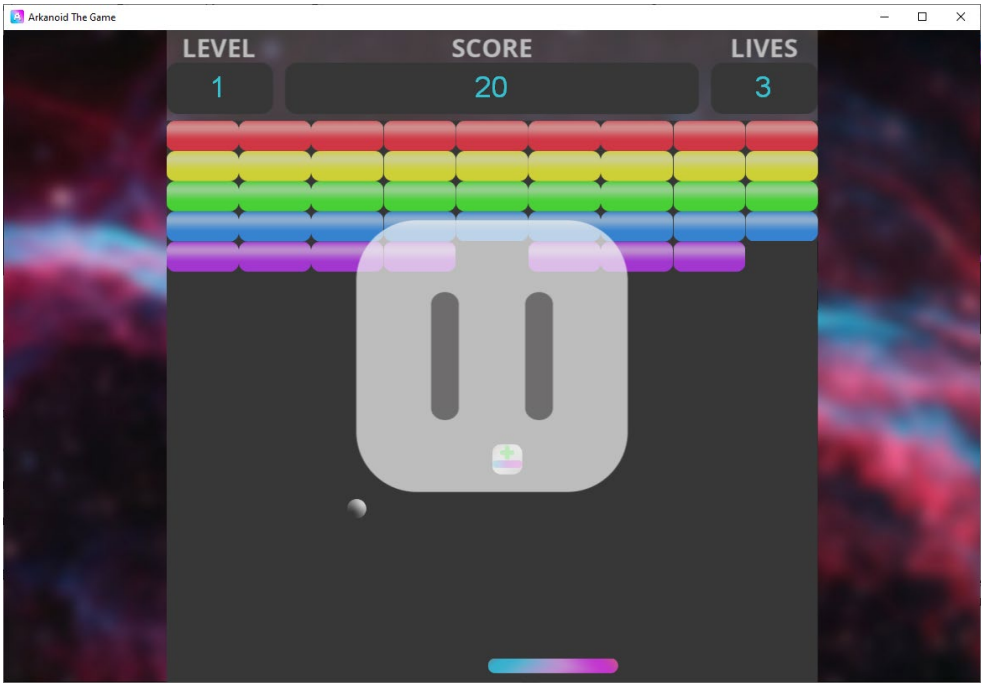

Рисунок 3.5 – Игра в состоянии паузы

В состоянии паузы, в игре приостанавливаются все взаимодействия, и пользователь может не беспокоится, что накопленный игровой прогресс будет потерян.

3.3.3 Главная игровая форма в состоянии просмотра таблицы лидеров

При нажатии клавиши «Tab», игровая форма переходит в состояние просмотра таблицы лидеров и меняет соответствующим образом главную игровую форму (см. рисунок 3.6).

Рисунок 3.6 – Игра в состоянии просмотра таблицы лидеров

В состоянии просмотра таблицы лидеров, в игре, аналогично режиму паузы, приостанавливаются все взаимодействия.

3.3.4 Диалоговое окно «Leaderboard»

Диалоговое окно с полем для ввода имени, необходимое для добавления результата пользователя в таблицу лидеров. (см. рисунок 3.7). Шаблон окна приведен на рисунке 3.8.

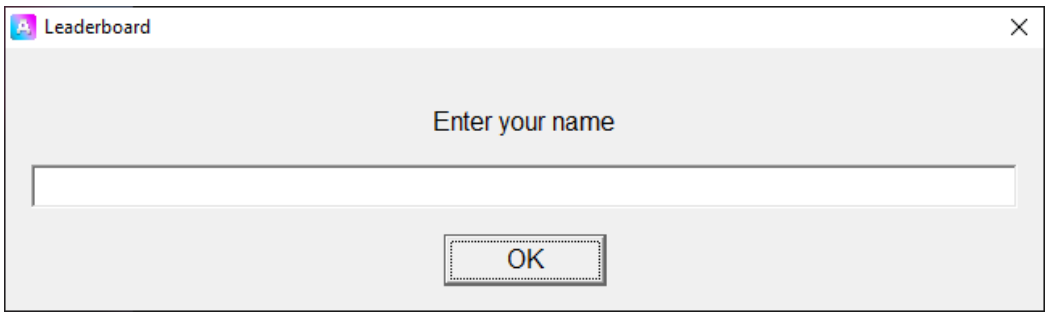

Рисунок 3.7 – Диалоговое окно «Leaderboard»

```
129 DIALOGEX 0, 0, 286, 72
STYLE DS_SETFONT | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Leaderboard"
FONT 12, "Arial", 400, 0, 0x1
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,121,51,45,14
    EDITTEXT        1000,7,32,272,12,ES_AUTOHSCROLL
    CTEXT           "Enter your name",-1,7,16,272,11
END
```

Рисунок 3.8 – Шаблон диалогового окна «Leaderboard»

3.4 Система выпадения бонусов

3.4.1 Описание системы выпадения бонусов

При разрушении блоков, из них с определенной вероятностью могут выпадать бонусы, влияющие на игровой процесс. Система выпадения бонусов построена с использованием последовательностей псевдослучайных чисел, и сделана с расчетом на щедрую выдачу. Это сделано именно таким образом, так как помимо бонусов с положительным эффектом, могут выпадать бонусы и с отрицательным эффектом. Здесь работает контринтуитивная логика, что большое количество бонусов упрощает процесс. На практике же, сложность игры с такой системой выдачи бонусов только возрастает.

3.4.2 Отсутствие бонуса

При разрушении блока, с вероятностью 25% ничего не выпадет. Из данной вероятности, можно сделать вывод, что в трех из четырех случаев должен выпасть бонус. Это достаточно большая вероятность, но сделано это осознанно по причинам, описанным в разделе 3.4.1.

3.4.3 Бонус «Расширение платформы»

При разрушении блока, с вероятностью 18.75% выпадет бонус «расширения платформы» (см. рисунок 3.9). Данный бонус равномерно от центра увеличивает платформу в 1.5 раза. Максимальное увеличение возможно только в 2.25 раза от стандартного размера.

Рисунок 3.9 – Внешний вид бонуса «Расширение платформы»

3.4.4 Бонус «Сжатие платформы»

При разрушении блока, с вероятностью 18.75% выпадет бонус «сжатие платформы» (см. рисунок 3.10). Данный бонус равномерно от центра сжимает платформу в 1.5 раза. Максимальное сжатие возможно только в 2.25 раза от стандартного размера.

Рисунок 3.10 – Внешний вид бонуса «Сжатие платформы»

3.4.5 Бонус «Больше шаров»

При разрушении блока, с вероятностью 18.75% выпадет бонус «Больше шаров» (см. рисунок 3.11). Данный бонус создает дополнительно ещё по одному шару к уже имеющимся с измененным углом направлением на 180 градусов. Максимальное количество шаров не ограничено, однако большое количество шаров обычно больше мешает, чем помогает с прохождением игры.

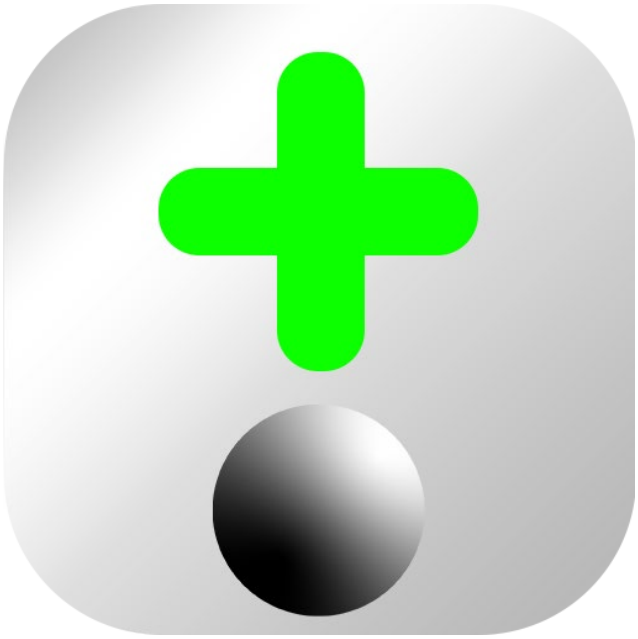

Рисунок 3.11 – Внешний вид бонуса «Больше шаров»

3.4.6 Бонус «Дополнительный опыт»

При разрушении блока, с вероятностью 14% выпадет бонус «Дополнительный опыт». Данный бонус прибавляет к имеющимся у пользователя игровым очкам дополнительные, в размере, зависящем от стоимости блока, из которого бонус выпал. Бонусные очки высчитываются как произведение стоимости блока, из которого выпал бонус и двадцати пяти. Бонус может быть в одной из 5 вариаций, под каждый тип блоков соответственно. Никаких различий в принципе действия у них нет, однако изменение цвета позволяет пользователю выставлять приоритеты по поимке бонусов. Пример бонуса для красного блока представлен на рисунке 3.12.

Рисунок 3.12 – Внешний вид бонуса «Дополнительный опыт» в вариации для красного блока

3.4.7 Бонус «Огненный шар»

При разрушении блока, с вероятностью 4.7% выпадет бонус «Огненный шар» (см. рисунок 3.13). Данный бонус превращает все шары в огненные. На практике это дает возможность шарикам уничтожать все блоки проходя насквозь без отскока, вне зависимости от изначальной прочности блока. Данный бонус дает существенное упрощение игры, поэтому его вероятность выпадения достаточно низкая по сравнению с вероятностями остальных бонусов.

Рисунок 3.13 – Внешний вид бонуса «Огненный шар»

3.5 Система построения уровней

Система разрабатывалась с целью сочетания максимальной эффективности при применении в программном средстве и удобочитаемости человеком. Уровни располагаются в каталоге ресурсов в папке «lvl». Нумерация уровней от 1 до n, – где n максимальный размер переменной int в скомпилированной программе. Имя текстового файла формируется по следующему шаблону: «{Номер уровня}.txt». Так как в один ряд блоков может поместится ограниченное количество блоков, то значит можно задавать шаблон в форме матрицы. В начале файла конфигурации уровня указывается в виде двух чисел ширина и высота матрицы. Далее значениями от 0 до 5 в виде матрицы указываются типы блоков, которые необходимо разместить на игровом уровне. Номера соответствуют следующим блокам:

– 0 – блок отсутствует;
– 1 – фиолетовый блок;
– 2 – синий блок;
– 3 – зеленый блок;
– 4 – желтый блок;
– 5 – красный блок;

Было проверено на практике, что данный механизм конфигурации уровней максимально прост для любого не разбирающегося в программировании человека, и может быть освоен при необходимости даже ребенком. В качестве примера на рисунке 3.14 представлен конфигурационный файл первого уровня.

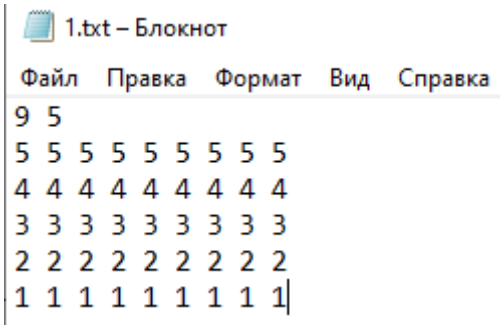

Рисунок 3.14 – Конфигурационный файл первого уровня

3.6 Принцип работы таблицы лидеров

При создании экземпляра объекта класса GameSession, с жесткого диска считывается таблица лидеров. В случае её отсутствия, – файл создается.

Занесение в таблицу лидеров производится в конце игры, который может быть объявлен либо из-за поражения по исчерпанию всех жизней, либо в случае победы, т.е. если закончились уровни. В случае победы, пользователю даются бонусные очки, рассчитываемые как произведение тысячи, количества оставшихся жизней и номера последнего уровня. В случае же поражения, никаких бонусов пользователь не получает. Далее набранный опыт сравнивается с имеющимся в таблице лидеров. Если пользователь набрал опыта меньше, чем последний в таблице лидеров, то тогда игра не будет предлагать записать своё имя. В другом случае, пользователю будет выдано показано модальное диалоговое окно (см. рисунок 3.7), в котором он сможет оставить своё имя. Если пользователь закроет окно, то результат записан не будет. Пользователь также может оставить поле пустым и нажать «ОК». В таком случае, результат будет добавлен в таблицу, но со стандартным именем «Player».

В конце работы программы, измененная таблица лидеров сохраняется на жесткий диск.

3.7 Механика анимации

Существует много вариантов выполнения анимации, однако самым эффективным был найден следующий метод: в момент начала игры создается отметка времени StartTick. В момент отрисовки создается отметка времени EndTick. Все расчеты анимации делаются исходя из времени прошедшего между двумя временными промежутками. В конце завершения расчетов и отрисовки, создается новая временная отметка StartTick и отправляется сообщение WM_PAINT. Таким образом, анимация не зависит ни от каких таймеров, и в случае временной потери производительности, анимация будет пропускать кадры, но не будет замедлятся. В случае же, если с производительностью не имеется проблем – анимация будет настолько плавной, насколько это возможно.

4 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА

4.1 Системные требования

Для нормальной работы программного средства необходимы следующие минимальные системные требования:

– Операционная система: Windows XP, Windows 7, Windows 8 или Windows 10;

– Процессор: Intel Pentium Silver N5000 с тактовой частотой 1.1Ггц или эквивалентный;

– Оперативная память 256 МБ;

– Свободное место на жестком диске: 5,7 МБ.

4.2 Установка

Для запуска программы необходима предварительная установка.

Шаг 1. Открыть ярлык установочного файла, находящийся на диске программы. Этот ярлык изображен на рисунке 4.1.

Рисунок 4.1 – Приложение Arkanoid Setup.exe

Шаг 2. Выбрать язык, на котором будет производится установка (см. рисунок 4.2).

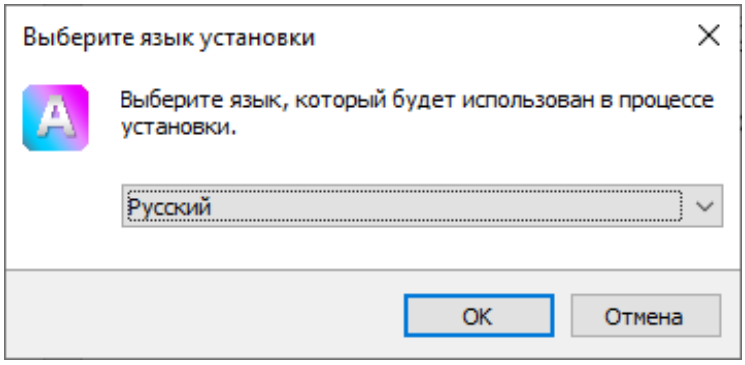

Рисунок 4.2 – Окно «Выберите язык установки»

Шаг 3. Изучить информацию окна «Выбор папки для установки программы», изображенную на рисунке 4.3. Следовать дальнейшим инструкциям. После этого нажать на кнопку «Далее».

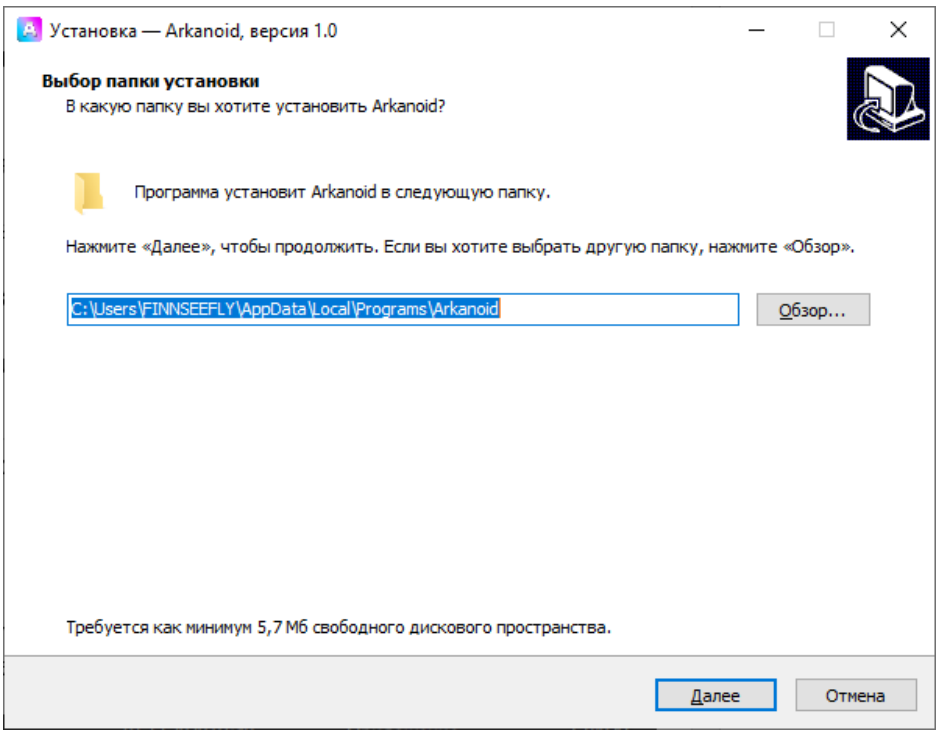

Рисунок 4.3 – Окно «Выберите папку»

Шаг 4. Изучить информацию окна «Дополнительные задачи», изображенную на рисунке 4.4. Следовать дальнейшим инструкциям. После этого нажать на кнопку «Далее».

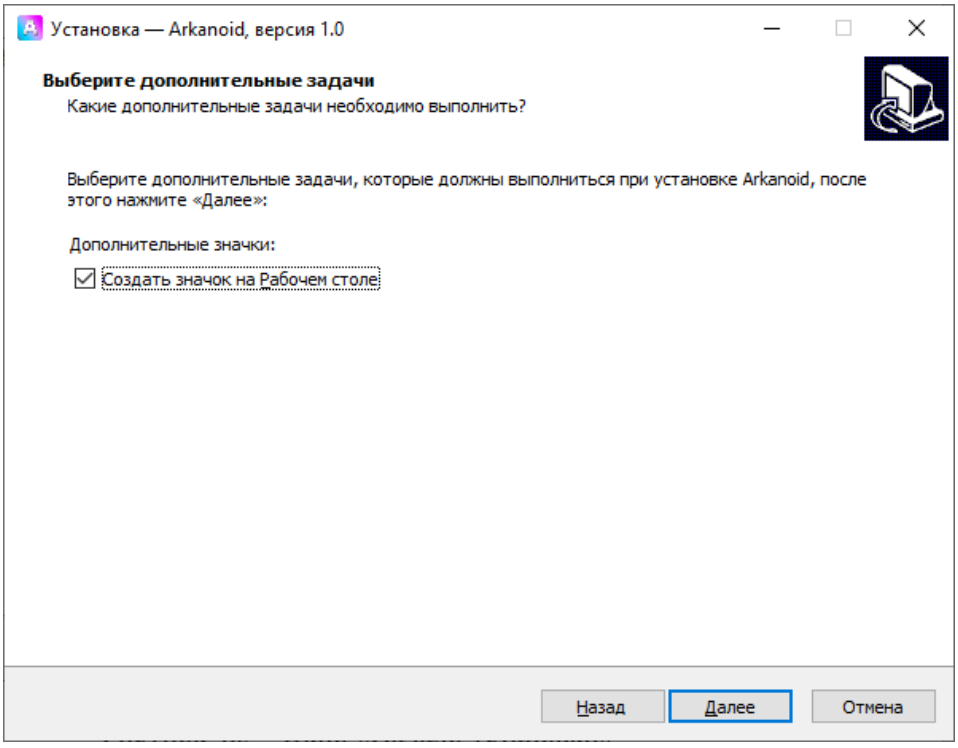

Рисунок 4.4 – Окно «Дополнительные задачи»

Шаг 5. Изучить информацию окна «Начало установки», изображенную на рисунке 4.5. После этого нажать на кнопку «Установить».

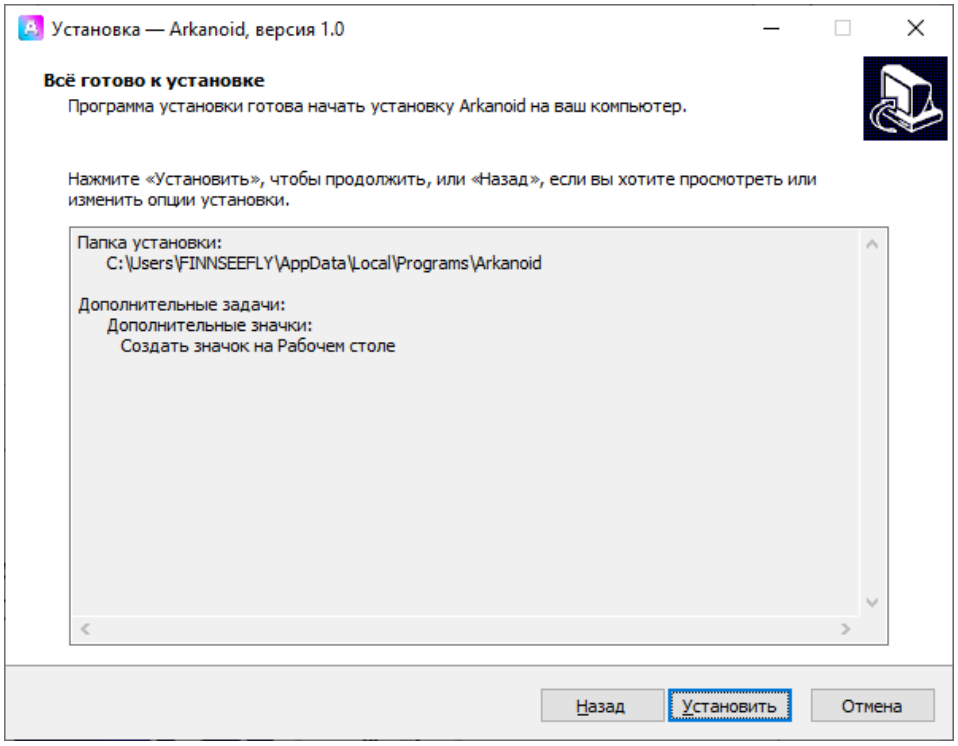

Рисунок 4.5 – Окно «Начало установки»

Шаг 6. Изучить информацию окна «Завершение установки», изображенную на рисунке 4.6. После этого нажать на кнопку «Завершить».

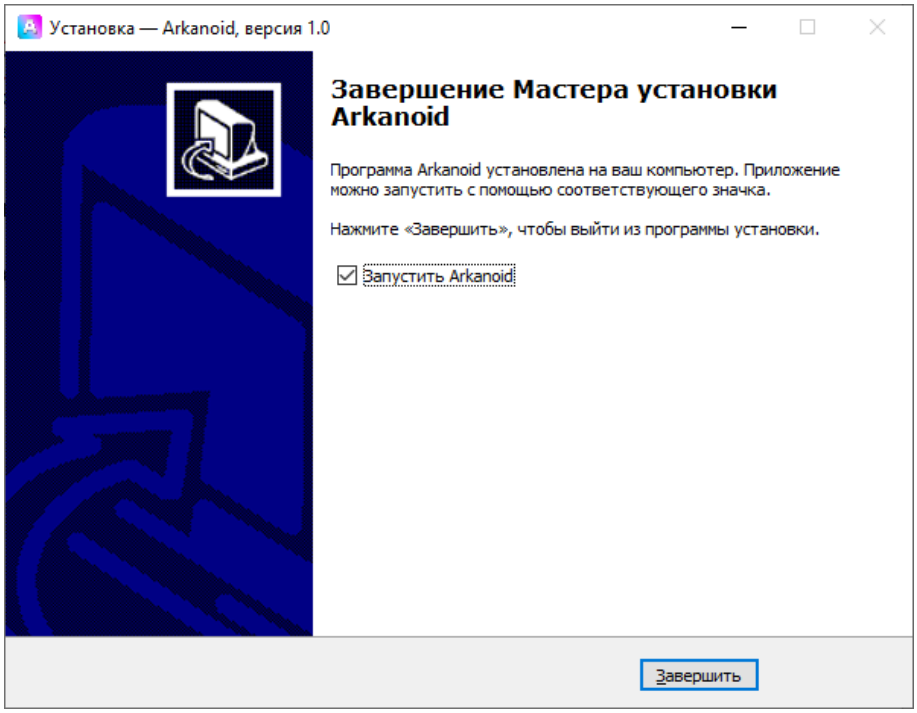

Рисунок 4.6 – Окно «Завершение установки»

4.3 Работа с программным средством

4.3.1 Подготовительный этап

Для начала использования программного средства необходимо воспользоваться ярлыком приложения (см. рисунок 4.7) или исполняемым файлом с расширением .exe (см. рисунок 4.8).

Рисунок 4.7 – Ярлык программного средства на рабочем столе

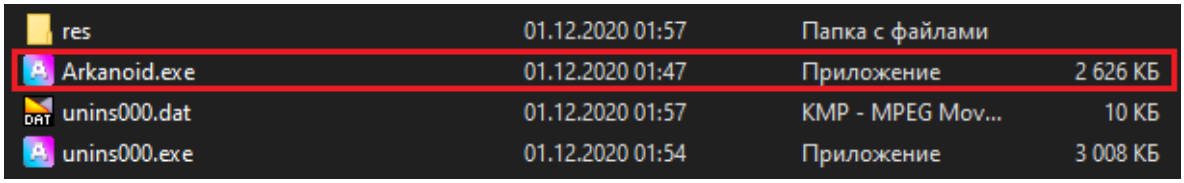

Рисунок 4.8 – Исполняемый файл в папке программного средства

После запуска появится главная игровая форма в начальном состоянии, изображенное на рисунке 4.9.

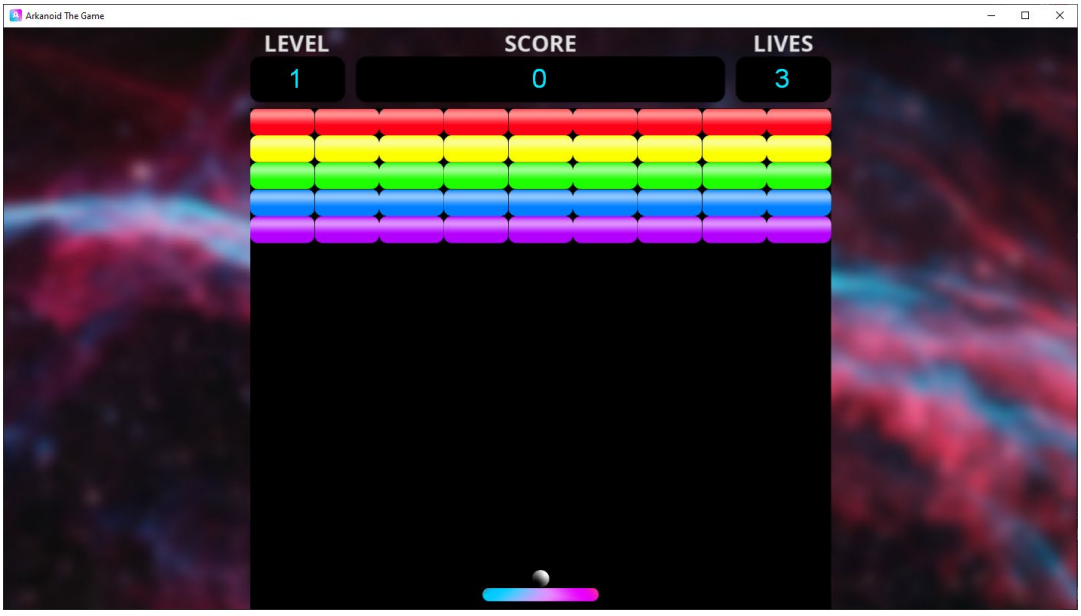

Рисунок 4.9 – Главная игровая форма в начальном состоянии

На данном этапе работы, с приложением пользователь может двигать платформу и шарик с помощью однократного нажатия на левую кнопку мыши и дальнейшего движения ею, либо стрелок на клавиатуре. По нажатию на «Tab» откроется таблица лидеров, а по нажатию «Esc» активируется пауза. Режим просмотра таблицы лидеров и режима паузы являются взаимоисключающими и не могут работать одновременно. Для начала игры необходимо нажать на клавишу «Space», после чего мяч полетит под углом 45 градусов в сторону блоков

4.3.2 Игровой процесс

В ходе игры вам предстоит сохранять в игровой зоне минимум один мячик путем отбивания его платформой. В случае потери мячика – игрок теряет одну жизнь. Цель игры – набрать максимальное количество очков. Цель каждого уровня – разрушить все блоки. Для достижения обеих целей, можно и рекомендуется собирать бонусы, выпадающие из разбитых блоков. Некоторые блоки могут не разбиться от первого удара так запас прочности может быть до 3-х единиц. Примеры игрового процесса приведены на рисунках 4.10 – 4.14

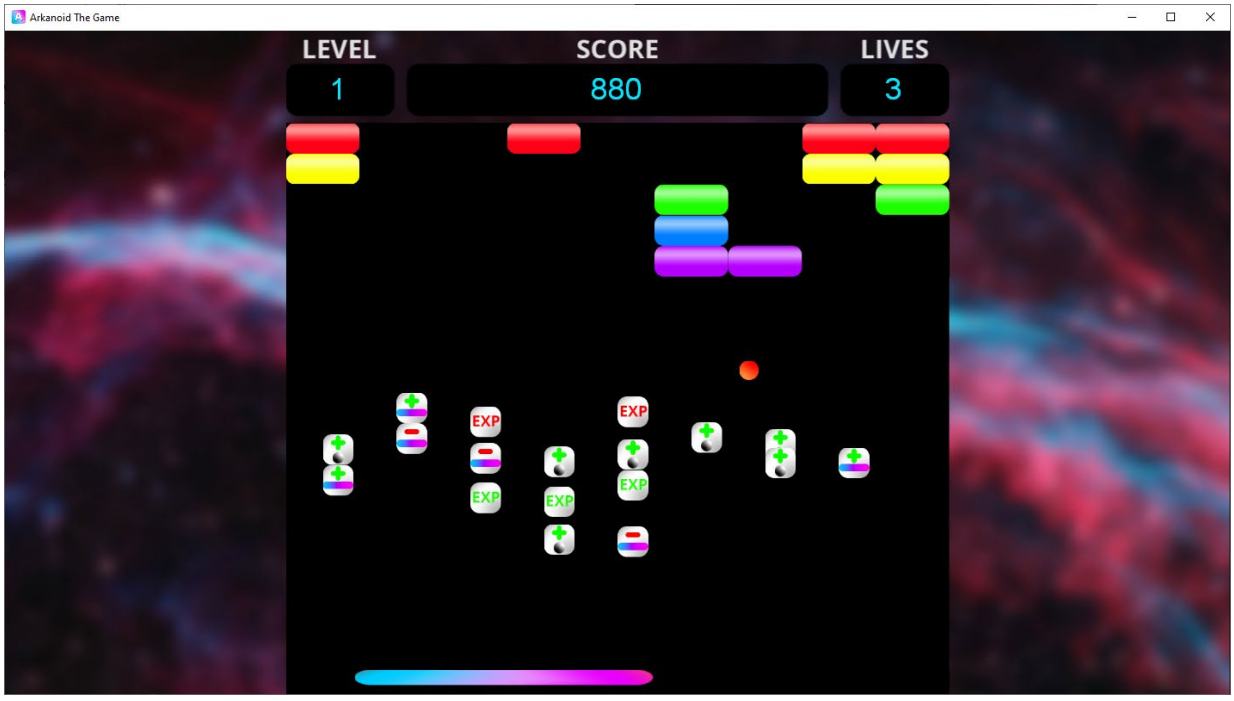

Рисунок 4.10 – Прохождение первого уровня

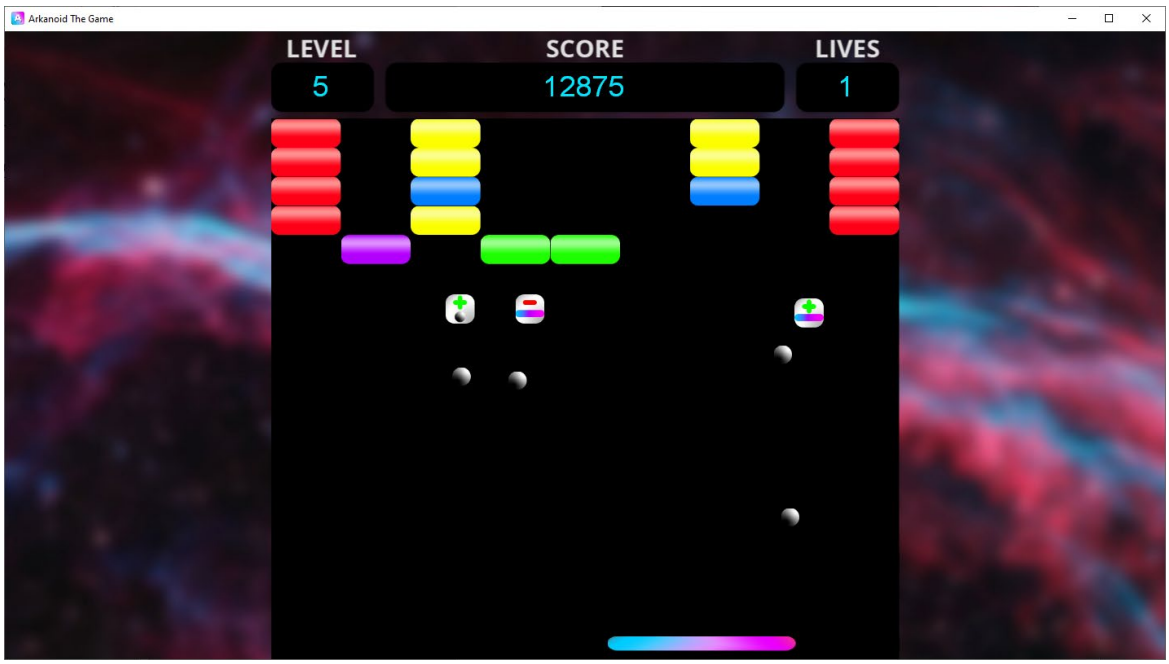

Рисунок 4.11 – Прохождение пятого уровня

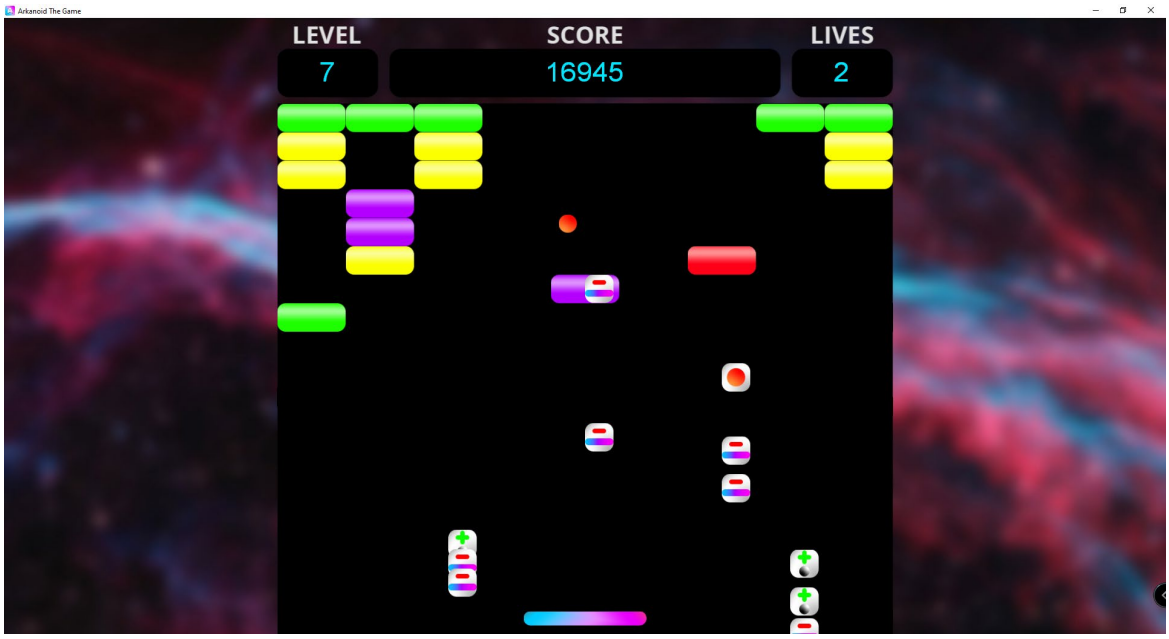

Рисунок 4.12 – Прохождение седьмого уровня

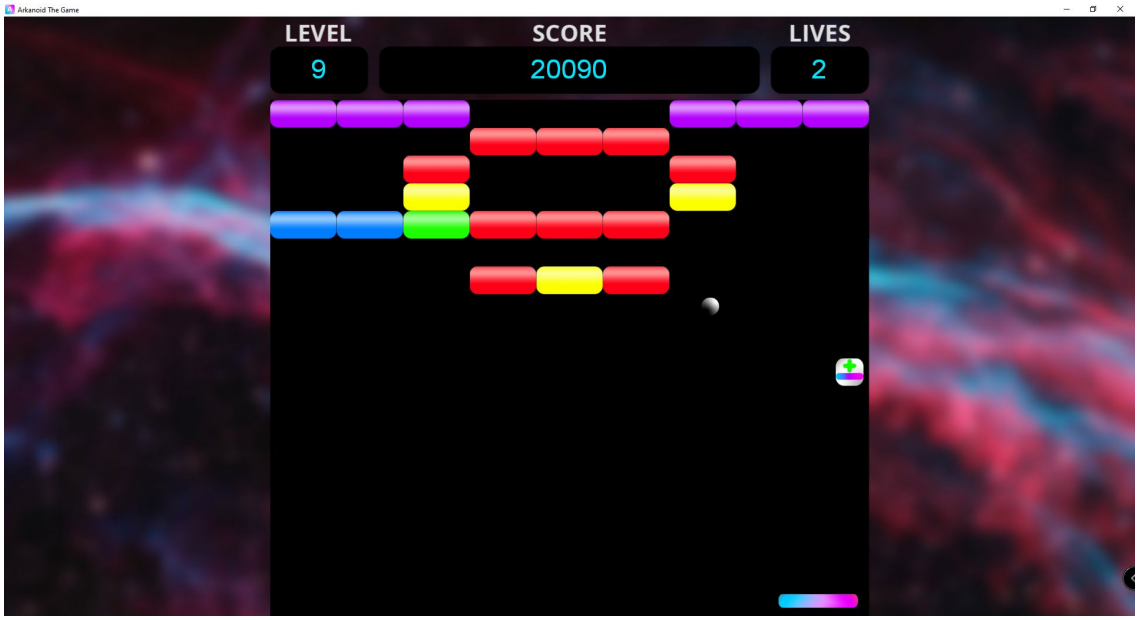

Рисунок 4.13 – Прохождение девятого уровня

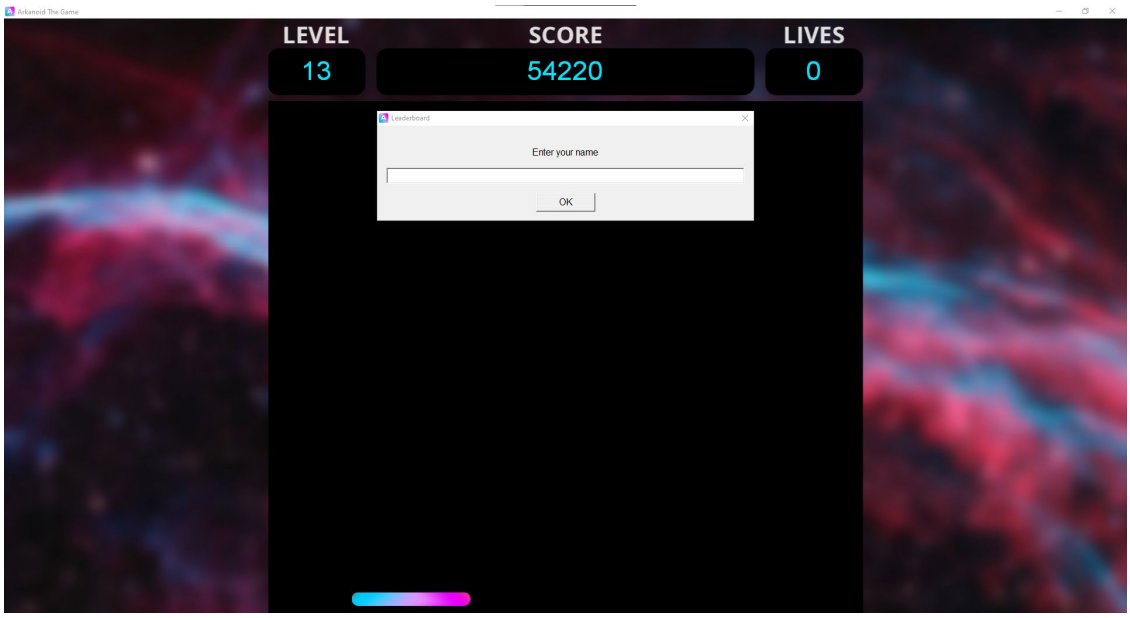

Рисунок 4.14 – Игра пройдена

4.3.3 Оценка результатов работы с программой

Во время работы программы было замерено потребление ресурсов. Согласно полученным данным, программа максимум использовала 5.8 МБ оперативной памяти (см. рисунок 4.15). Также хочется отметить, что так как программа написана на C++ с использованием интерфейса программирования Windows API, в обработке графики вообще не участвует графический ускоритель (см. рисунок 4.16). Данные показатели характеризуют программу как не требовательную по современным меркам, что позволяет программе быть запущенной и комфортно использоваться на непроизводительном портативном компьютере под управлением Windows.

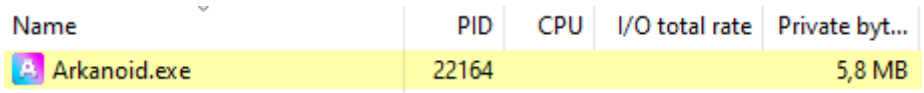

Рисунок 4.15 – Использование ресурсов компьютера приложением

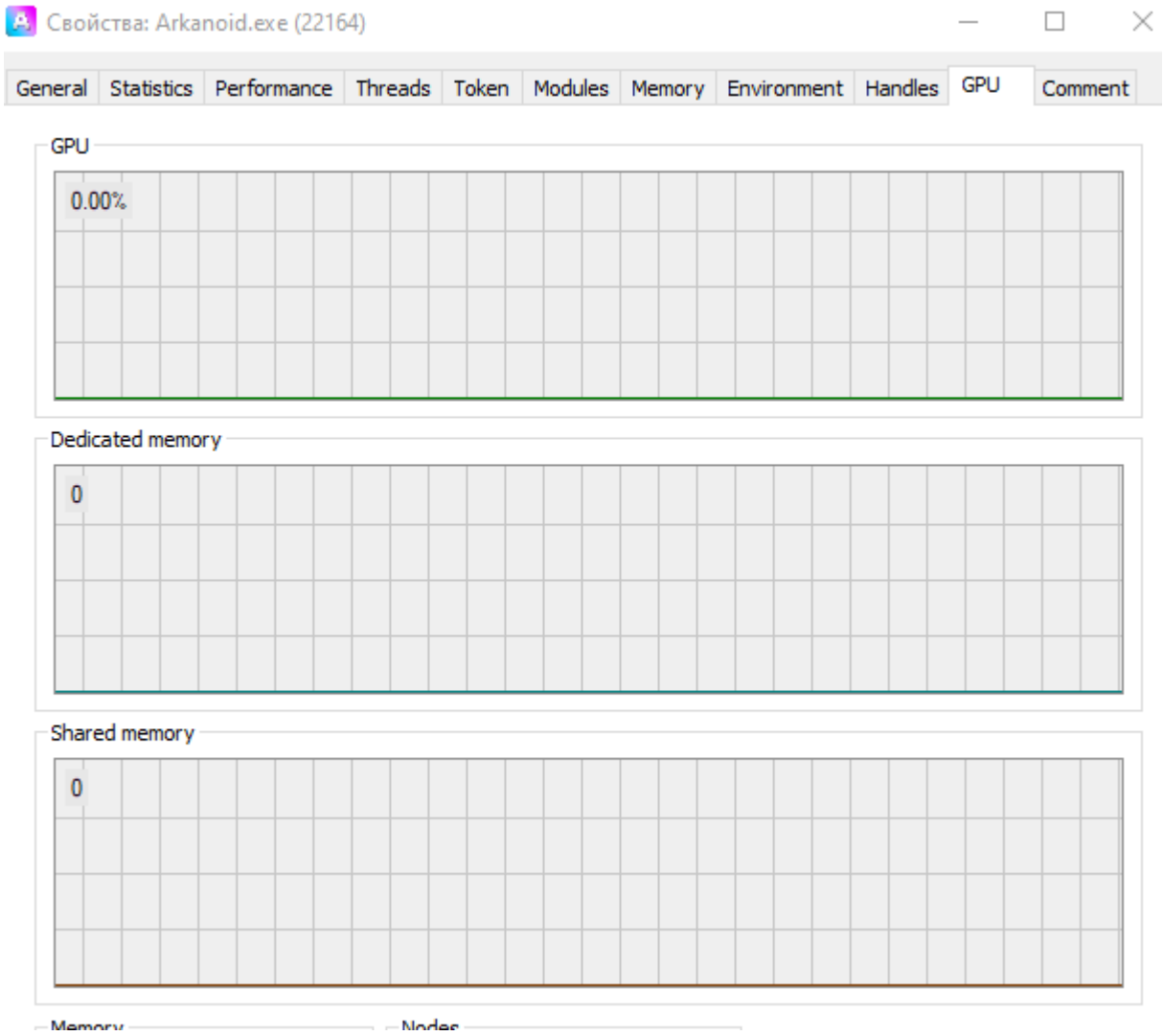

Рисунок 4.16 – Использование GPU приложением

ЗАКЛЮЧЕНИЕ

В ходе разработки приложения был создан продукт, сочетающий в себе лучшие стороны от аналогов, с устраненными проблемными местами и свежим дизайном. С помощью использования современных сред разработки, систем сборки и компиляторов, удалось добиться достаточно высокой производительности и отзывчивости. Немаловажно что, во время реализации данного проекта мне удалось развить навыки разработки приложений на языке c++, получить знания в создании качественной анимации и научится работать с системой сборок проектов CMake. Данное приложение справляется со своими основными задачами, а именно: развлекать и развивать у пользователя реакцию.

В процессе работы было установлено низкое потребление ресурсов ОЗУ и нулевое использование GPU. Благодаря данному факту, игровое программное средство «Arkanoid» будет работать без каких-либо проблем на любом современном компьютере, и с большой долей вероятности, даже на немного устаревшем портативном компьютере под управлением Windows.

Хочется отметить, что данное приложение может быть модернизировано, путем увеличения количества бонусов, разработки более продуманных уровней, оптимизации алгоритмов и добавления новых блоков с новыми механиками. Возможно также имеется смысл в переносе данного программного продукта на более современные фреймворки, однако в этом случае, есть риск усложнения продукта и следовательно, повышения минимальных системных требований.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Microsoft Documentation [Электронный ресурс] Режим доступа: https://docs.microsoft.com/en-us/windows/win32/ – Дата доступа 28.10.2020.

[2] Основы программирования для Win32 API [Электронный ресурс] Режим доступа: https://dims.karelia.ru/win32/ – Дата доступа 20.11.2020.

[3] Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста – СПб.: Питер, 2013. – 464 с.: ил. – (Серия «Библиотека программиста»). ISBN 978-5–496-00487-9.

[4] MSDN – Windows API по-русски [Электронный ресурс] Режим доступа: http://narovol.narod.ru/_tbkp/New_MSDN_API/index_msdn.htm – Дата доступа 29.10.2020.

[5] Win 32 API по шагам [Электронный ресурс] Режим доступа: https://www.firststeps.ru/ – Дата доступа 02.11.2020.

[6] Справочник по функциям Windows API [Электронный ресурс] Режим доступа: http://rusproject.narod.ru/winapi/winapi.htm – Дата доступа 02.11.2020.

[7] Виртуальный компьютерный музей, Арканоид [Электронный ресурс] Режим доступа: https://www.computer-museum.ru/games/arcanoid.html Дата доступа 29.10.2020.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код программы

Файл Ball.h:

```
#ifndef ARKANOID_BALL_H
#define ARKANOID_BALL_H


#include "DrawableShape.h"

class Ball : public DrawableShape {
private:
    float speed;
    float angle;
    bool destroyed = false;
public:
    void CalculateNextPoint(int t);

    Ball(float &gameZoneX0, float &gameZoneY0, Gdiplus::Image *&image, float &scale,
float offsetX, float offsetY,
         float speed, float angle);

    void SetOffsetX(float x);

    void SetOffsetY(float y);

    void SetOffsetX2(float x);

    void SetOffsetY2(float y);

    [[nodiscard]] float GetAngle() const;

    [[nodiscard]] float GetSpeed() const;

    [[nodiscard]] bool IsDestroyed() const;

    void SetDestroyed();

    void SetAngle(float newAngle);
};


#endif //ARKANOID_BALL_H
```

Файл Ball.cpp:

```
#include "Ball.h"

void Ball::CalculateNextPoint(int t) {
    float r = speed * t;
    float h = std::sin(ConvertDegToRad(angle)) * r;
    float s = std::cos(ConvertDegToRad(angle)) * r;
    SetOffsetX(offsetX + s);
    offsetY += h;
    CalculateRECT();
}
```

```
void Ball::SetOffsetX(float x) {
    PrepareToRelocate();
    offsetX = x;
}

float Ball::GetAngle() const {
    return angle;
}

void Ball::SetAngle(float newAngle) {
    this->angle = newAngle;
}

void Ball::SetOffsetY(float y) {
    PrepareToRelocate();
    offsetY = y;
}

void Ball::SetOffsetX2(float x) {
    PrepareToRelocate();
    offsetX = x - GetWidth();
}

void Ball::SetOffsetY2(float y) {
    PrepareToRelocate();
    offsetY = y - GetHeight();
}

bool Ball::IsDestroyed() const {
    return destroyed;
}

void Ball::SetDestroyed() {
    destroyed = true;
}

float Ball::GetSpeed() const {
    return speed;
}

Ball::Ball(float &gameZoneX0, float &gameZoneY0, Gdiplus::Image *&image, float
&scale, float offsetX, float offsetY,
           float speed, float angle) : DrawableShape(gameZoneX0, gameZoneY0, image,
scale, offsetX, offsetY),
                                       speed(speed), angle(angle) {
    CalculateRECT();
}
```

Файл Bouns.h:

```
#ifndef ARKANOID_BONUS_H
#define ARKANOID_BONUS_H


#include "DrawableShape.h"

enum BonusType {
    BONUS_NONE,
```

```
    BONUS_EXPAND,
    BONUS_CUT,
    BONUS_EXPERIENCE,
    BONUS_FIREBALL,
    BONUS_MORE_BALLS
};

#define DEFAULT_BONUS_SPEED 0.2


class Bonus : public DrawableShape {
private:
    BonusType bonusType;
    int price = 0;
    bool destroyed = false;
public:
    bool IsDestroyed();

    void SetDestroyed();

    void CalculateNextPoint(int t);

    BonusType GetBonusType();

    int GetPrice();

    Bonus(float &gameZoneX0, float &gameZoneY0, Gdiplus::Image *&image, float &scale,
float offsetX, float offsetY,
          BonusType bonusType, int price);
};


#endif //ARKANOID_BONUS_H
```

Файл Bonus.cpp:

```
#include "Bonus.h"

bool Bonus::IsDestroyed() {
    return destroyed;
}

void Bonus::SetDestroyed() {
    destroyed = true;
}

void Bonus::CalculateNextPoint(int t) {
    PrepareToRelocate();
    offsetY += DEFAULT_BONUS_SPEED * t;
    CalculateRECT();
}

BonusType Bonus::GetBonusType() {
    return bonusType;
}

int Bonus::GetPrice() {
    return price;
```

```
}

Bonus::Bonus(float &gameZoneX0, float &gameZoneY0, Gdiplus::Image *&image, float
&scale, float offsetX, float offsetY,
             BonusType bonusType, int price) : DrawableShape(gameZoneX0, gameZoneY0,
image, scale, offsetX, offsetY),
                                               bonusType(bonusType), price(price) {}
```

Файл Brick.h:

```
#ifndef ARKANOID_BRICK_H
#define ARKANOID_BRICK_H


#include "DrawableShape.h"
#include "Complementary.h"
#include "Ball.h"

#define BRICK_NUM_NULL 0
#define BRICK_NUM_PURPLE 1
#define BRICK_NUM_BLUE 2
#define BRICK_NUM_GREEN 3
#define BRICK_NUM_YELLOW 4
#define BRICK_NUM_RED 5

#define BRICK_HITS_PURPLE 1
#define BRICK_HITS_BLUE 1
#define BRICK_HITS_GREEN 1
#define BRICK_HITS_YELLOW 2
#define BRICK_HITS_RED 3

#define BRICK_PRICE_PURPLE 10
#define BRICK_PRICE_BLUE 15
#define BRICK_PRICE_GREEN 20
#define BRICK_PRICE_YELLOW 25
#define BRICK_PRICE_RED 30

#define BRICK_WIDTH 120
#define BRICK_HEIGHT 50

enum BrickType {
    BRICK_TYPE_PURPLE,
    BRICK_TYPE_GREEN,
    BRICK_TYPE_BLUE,
    BRICK_TYPE_RED,
    BRICK_TYPE_YELLOW,
};

class Brick : public DrawableShape {
private:
    int hitsBeforeDestruction;
    int price;
    BrickType brickType;
    bool isDestroyed = false;
public:
    Brick(float &gameZoneX0, float &gameZoneY0, Gdiplus::Image *&image, float &scale,
float offsetX, float offsetY,
          int hitsBeforeDestruction, int price, BrickType brickType);

    [[nodiscard]] bool IsDestroyed() const;
```

```
    bool HitTheBrick(bool isFireball);

    [[nodiscard]] int GetPrice() const;

    BrickType GetBrickType();
};


#endif //ARKANOID_BRICK_H
```

Файл Brick.cpp:

```
#include "Brick.h"

bool Brick::IsDestroyed() const {
    return isDestroyed;
}

bool Brick::HitTheBrick(bool isFireball) {
    hitsBeforeDestruction--;
    if (hitsBeforeDestruction == 0 || isFireball) {
        isDestroyed = true;
        wasFilled = true;
        needRepaint = false;
        return true;
    }
    return false;
}

int Brick::GetPrice() const {
    return price;
}

BrickType Brick::GetBrickType() {
    return brickType;
}

Brick::Brick(float &gameZoneX0, float &gameZoneY0, Gdiplus::Image *&image, float
&scale, float offsetX, float offsetY,
             int hitsBeforeDestruction, int price, BrickType brickType) :
DrawableShape(gameZoneX0, gameZoneY0, image,

scale, offsetX, offsetY),

hitsBeforeDestruction(hitsBeforeDestruction),

price(price), brickType(brickType) {
    CalculateRECT();
}
```

Файл Complementary.h:

```
#ifndef ARKANOID_COMPLEMENTARY_H
#define ARKANOID_COMPLEMENTARY_H

#define RESOURCE_ROOT "res\\"
#define LVL_DIR "lvl\\"
#define LVL_EXTENSION ".txt"
#define LEADER_BOARD_PATH "res\\data\\lb.txt"
```

```
#define BACKGROUND_PIC_PATH L"res\\interface\\SpaceBlurred720p.jpg"
#define PAUSE_PIC_PATH L"res\\interface\\Pause.png"
#define LEADER_BOARD_PIC_PATH L"res\\interface\\LeaderBoard.png"
#define GAME_BOX_PIC_PATH L"res\\interface\\GamingZone.png"
#define PLATFORM_PIC_PATH L"res\\platform\\platform.png"
#define BALL_PIC_PATH L"res\\ball\\Ball32x32.png"
#define BLUE_BRICK_PIC_PATH L"res\\bricks\\BlueBrick.png"
#define GREEN_BRICK_PIC_PATH L"res\\bricks\\GreenBrick.png"
#define PURPLE_BRICK_PIC_PATH L"res\\bricks\\PurpleBrick.png"
#define RED_BRICK_PIC_PATH L"res\\bricks\\RedBrick.png"
#define YELLOW_BRICK_PIC_PATH L"res\\bricks\\YellowBrick.png"
#define FIREBALL_PIC_PATH L"res\\ball\\FireBall32x32.png"

#define BONUS_BALL_PATH L"res\\bonus\\BonusBall.png"
#define BONUS_CUT_PATH L"res\\bonus\\BonusCut.png"
#define BONUS_EXPAND_PATH L"res\\bonus\\BonusExpand.png"
#define BONUS_FIREBALL_PATH L"res\\bonus\\BonusFireBall.png"
#define BONUS_EXP1_PATH L"res\\bonus\\BonusEXP1.png"
#define BONUS_EXP2_PATH L"res\\bonus\\BonusEXP2.png"
#define BONUS_EXP3_PATH L"res\\bonus\\BonusEXP3.png"
#define BONUS_EXP4_PATH L"res\\bonus\\BonusEXP4.png"
#define BONUS_EXP5_PATH L"res\\bonus\\BonusEXP5.png"

#define TEXT_TOP 64
#define TEXT_BOTTOM 128
#define LEVEL_LEFT 20
#define LEVEL_RIGHT 156
#define SCORE_LEFT 217
#define SCORE_RIGHT 863
#define LIVES_LEFT 923
#define LIVES_RIGHT 1059

#define BALL_SIZE 32

#define DEFAULT_FONT_HEIGHT 55
#define DEFAULT_FONT_WIDTH 23
#define DEFAULT_FONT_WEIGHT 32
#define DEFAULT_FONT_ESCAPEMENT 0
#define DEFAULT_FONT_UNDERLINE false
#define DEFAULT_FONT_STRIKE_OUT false
#define DEFAULT_FONT_ITALIC false

#define DEFAULT_GAME_ZONE_TOP 150
#define DEFAULT_GAME_ZONE_WIDTH 1080
#define MIN_GAME_ZONE_SIDE 540

#define DEFAULT_PLATFORM_OFFSET_X 432
#define DEFAULT_PLATFORM_OFFSET_Y 890

#define DEFAULT_SPEED 1
#define DEFAULT_ANGLE 315
#define DEFAULT_TIME 1

#define INTERSECTION_LEFT 1
#define INTERSECTION_LEFT_AND_UP 2
#define INTERSECTION_UP 3
#define INTERSECTION_RIGHT_AND_UP 4
#define INTERSECTION_RIGHT 5
#define INTERSECTION_RIGHT_AND_DOWN 6
#define INTERSECTION_DOWN 7
#define INTERSECTION_LEFT_AND_DOWN 8
```

```
#define INTERSECTION_INSIDE 9
#define INTERSECTION_NONE 0

#define FLOAT_MAX_VALUE 100000;
#define FLOAT_MIN_VALUE -100000;

#define LEFT_SIDE_LEFT FLOAT_MIN_VALUE;
#define LEFT_SIDE_RIGHT -1
#define LEFT_SIDE_TOP 0
#define LEFT_SIDE_BOTTOM 931

#define RIGHT_SIDE_LEFT DEFAULT_GAME_ZONE_WIDTH
#define RIGHT_SIDE_RIGHT FLOAT_MAX_VALUE
#define RIGHT_SIDE_TOP 0
#define RIGHT_SIDE_BOTTOM 931

#define DOWN_SIDE_LEFT -1
#define DOWN_SIDE_RIGHT DEFAULT_GAME_ZONE_WIDTH
#define DOWN_SIDE_TOP 930 + BALL_SIZE
#define DOWN_SIDE_BOTTOM FLOAT_MAX_VALUE

#define UP_SIDE_LEFT -1
#define UP_SIDE_RIGHT DEFAULT_GAME_ZONE_WIDTH
#define UP_SIDE_TOP FLOAT_MIN_VALUE
#define UP_SIDE_BOTTOM 0

#define DEFAULT_PLATFORM_MOVE 70

#define NUM_OF_LEAD_OFFSET_X 20
#define NUM_OF_LEAD_WIDTH 63
#define LEAD_OFFSET_Y 265
#define LEAD_HEIGHT 54
#define NAME_OF_LEAD_OFFSET_X 118
#define NAME_OF_LEAD_WIDTH 477
#define SCORE_OF_LEAD_OFFSET_X 632
#define SCORE_OF_LEAD_WIDTH 428
#define LEAD_VERTICAL_INTERVAL 72

#define WM_NEED_A_DIALOG_BOX 1337

#include "iostream"
#include "Windows.h"
#include "sstream"

struct FloatRECT {
    float left;
    float top;
    float right;
    float bottom;
};

struct BoolRECT {
    bool leftUp = false;
    bool leftDown = false;
    bool rightUp = false;
    bool rightDown = false;
};

bool IsDotInRect(float x, float y, FloatRECT rect);

BoolRECT FindOccurrences(FloatRECT target, FloatRECT incoming);
```

```
std::string ConvertIntToString(int value);

float ConvertDegToRad(float value);

long ConvertStringToLong(std::string str);


#endif //ARKANOID_COMPLEMENTARY_H
```

Файл Complementary.cpp:

```
#include "Complementary.h"
#include "math.h"

bool IsDotInRect(float x, float y, FloatRECT rect) {
    return x >= rect.left && x <= rect.right && y >= rect.top && y <= rect.bottom;
}

BoolRECT FindOccurrences(FloatRECT target, FloatRECT incoming) {
    BoolRECT result;
    result.leftUp = IsDotInRect(incoming.left, incoming.top, target);
    result.leftDown = IsDotInRect(incoming.left, incoming.bottom, target);
    result.rightUp = IsDotInRect(incoming.right, incoming.top, target);
    result.rightDown = IsDotInRect(incoming.right, incoming.bottom, target);
    return result;
}

std::string ConvertIntToString(int value) {
    std::stringstream s;
    s << std::scientific << value;
    return s.str();
}

float ConvertDegToRad(float value) {
    return value * M_PI / 180;
}

long ConvertStringToLong(std::string str) {
    if (str == "") return 0;
    return std::stol(str);
}
```

Файл DrawableShape.h:

```
#ifndef ARKANOID_DRAWABLESHAPE_H
#define ARKANOID_DRAWABLESHAPE_H

#include "windows.h"
#include "gdiplus.h"
#include "Complementary.h"

class DrawableShape {
protected:
    float &gameZoneX0;
    float &gameZoneY0;
    Gdiplus::Image *&image;
```

```
    float &scale;
    float offsetX;
    float offsetY;
    bool needRepaint = true;
    bool wasFilled = false;
    FloatRECT rect;


    void EndPaint();

    virtual void SetRepaintRECT();

public:

    void SetOffsetX(float offset);

    void SetOffsetY(float offset);

    void PrepareToRelocate();

    int GetWidth();

    virtual void CalculateRECT();

    int GetHeight();

    FloatRECT GetRECT();

    void SetNeedRepaint();

    [[nodiscard]] bool IsNeedRepaint() const;

    [[nodiscard]] bool IsWasFilled() const;

    RECT repaintRect;

    virtual void PaintOnGraphics(Gdiplus::Graphics &graphics);

    DrawableShape(float &gameZoneX0, float &gameZoneY0, Gdiplus::Image *&image, float
&scale, float offsetX,
                  float offsetY);

    int GetNumOfIntersection(FloatRECT TargetRect);
};


#endif //ARKANOID_DRAWABLESHAPE_H
```

Файл DrawableShape.cpp:

```
#include "DrawableShape.h"
#include "iostream"

int DrawableShape::GetHeight() {
    return image->GetHeight();
}

int DrawableShape::GetWidth() {
    return image->GetWidth();
}
```

```
void DrawableShape::PaintOnGraphics(Gdiplus::Graphics &graphics) {
    graphics.DrawImage(image, gameZoneX0 + offsetX * scale, gameZoneY0 + offsetY *
scale, this->GetWidth() * scale,
                       this->GetHeight() * scale);
    EndPaint();
}


void DrawableShape::EndPaint() {
    needRepaint = false;
    SetRepaintRECT();
    CalculateRECT();
    wasFilled = false;
}


void DrawableShape::CalculateRECT() {
    rect.left = offsetX;
    rect.right = offsetX + image->GetWidth();
    rect.bottom = offsetY + image->GetHeight();
    rect.top = offsetY;
}

void DrawableShape::SetNeedRepaint() {
    needRepaint = true;
}

bool DrawableShape::IsNeedRepaint() const {
    return needRepaint;
}

void DrawableShape::SetRepaintRECT() {
    repaintRect.left = round(offsetX * scale + gameZoneX0);
    repaintRect.right = ceil((offsetX + image->GetWidth()) * scale + gameZoneX0);
    repaintRect.bottom = ceil((offsetY + image->GetHeight()) * scale + gameZoneY0);
    repaintRect.top = round(offsetY * scale + gameZoneY0);
    wasFilled = true;
}

bool DrawableShape::IsWasFilled() const {
    return wasFilled;
}

FloatRECT DrawableShape::GetRECT() {
    return rect;
}

int DrawableShape::GetNumOfIntersection(FloatRECT TargetRect) {
    BoolRECT occurrences = FindOccurrences(TargetRect, rect);
    int numOfOccurrences = occurrences.leftUp + occurrences.leftDown +
occurrences.rightDown + occurrences.rightUp;
    switch (numOfOccurrences) {
        case 0:
            return INTERSECTION_NONE;
        case 1: {
            if (occurrences.rightDown) {
                if (rect.bottom - TargetRect.top < rect.right - TargetRect.left)
return INTERSECTION_UP;
                if (rect.bottom - TargetRect.top > rect.right - TargetRect.left)
return INTERSECTION_LEFT;
```

```
                return INTERSECTION_LEFT_AND_UP;
            }
            if (occurrences.leftDown) {
                if (rect.bottom - TargetRect.top < TargetRect.right - rect.left)
return INTERSECTION_UP;
                if (rect.bottom - TargetRect.top > TargetRect.right - rect.left)
return INTERSECTION_RIGHT;
                return INTERSECTION_RIGHT_AND_UP;
            }
            if (occurrences.leftUp) {
                if (TargetRect.right - rect.left < TargetRect.bottom - rect.top)
return INTERSECTION_RIGHT;
                if (TargetRect.right - rect.left > TargetRect.bottom - rect.top)
return INTERSECTION_DOWN;
                return INTERSECTION_RIGHT_AND_DOWN;
            }
            if (occurrences.rightUp) {
                if (rect.right - TargetRect.left > TargetRect.bottom - rect.top)
return INTERSECTION_DOWN;
                if (rect.right - TargetRect.left < TargetRect.bottom - rect.top)
return INTERSECTION_LEFT;
                if (rect.right - TargetRect.left < TargetRect.bottom - rect.top)
return INTERSECTION_LEFT;
                return INTERSECTION_LEFT_AND_DOWN;
            }
        }
        break;
        case 2: {
            if (occurrences.rightDown && occurrences.rightUp) return
INTERSECTION_LEFT;
            if (occurrences.leftDown && occurrences.rightDown) return
INTERSECTION_UP;
            if (occurrences.leftDown && occurrences.leftUp) return
INTERSECTION_RIGHT;
            if (occurrences.leftUp && occurrences.rightUp) return INTERSECTION_DOWN;
        }
            break;
        default:
            throw std::exception();
    }
    return 0;
}

void DrawableShape::PrepareToRelocate() {
    if (!IsWasFilled()) {
        SetRepaintRECT();
    }
    SetNeedRepaint();
}

void DrawableShape::SetOffsetX(float offset) {
    PrepareToRelocate();
    offsetX = offset;
}

void DrawableShape::SetOffsetY(float offset) {
    PrepareToRelocate();
    offsetY = offset;
}

DrawableShape::DrawableShape(float &gameZoneX0, float &gameZoneY0, Gdiplus::Image
```

```
*&image, float &scale, float offsetX,
                                float offsetY) : gameZoneX0(gameZoneX0),
gameZoneY0(gameZoneY0), image(image),
                                                 scale(scale), offsetX(offsetX),
offsetY(offsetY) {}
```

Файл GameSession.h:

```
#ifndef ARKANOID_GAMESESSION_H
#define ARKANOID_GAMESESSION_H

#include "Windows.h"
#include "vector"
#include "DrawableShape.h"
#include "gdiplus.h"
#include "Ball.h"
#include "Brick.h"
#include "Platform.h"
#include "Bonus.h"
#include "TextBox.h"


class GameSession {
private:
    HWND hWnd;
    HDC hdc, memDC;
    PAINTSTRUCT ps;
    HBITMAP oldBmp, hBM;
    HBRUSH brush;
    RECT clientRect;

    float clientWidth, clientHeight;
    float scale;

    float backgroundX0;
    float backgroundY0;
    float backgroundWidth;
    float backgroundHeight;

    float gameBoxX0;
    float gameBoxY0;
    float gameBoxSide;

    float gameZoneX0;
    float gameZoneY0;

    LOGFONT lf;

    RECT levelTextRect;
    RECT scoreTextRect;
    RECT livesTextRect;

    int level, oldLevel;
    long score, oldScore, winScore;
    int lives, oldLives;


    Gdiplus::Graphics *graphics;

    Gdiplus::Image *backgroundPic;
```

```
Gdiplus::Image *pausePic;
Gdiplus::Image *leaderBoardPic;
Gdiplus::Image *gameZonePic;
Gdiplus::Image *platformPic;
Gdiplus::Image *ballPic;
Gdiplus::Image *defaultBallPic;
Gdiplus::Image *blueBrickPic;
Gdiplus::Image *greenBrickPic;
Gdiplus::Image *purpleBrickPic;
Gdiplus::Image *redBrickPic;
Gdiplus::Image *yellowBrickPic;

Gdiplus::Image *fireBallPic;
Gdiplus::Image *bonusBallPic;
Gdiplus::Image *bonusCutPic;
Gdiplus::Image *bonusExpandPic;
Gdiplus::Image *bonusFireBallPic;
Gdiplus::Image *bonusEXP1Pic;
Gdiplus::Image *bonusEXP2Pic;
Gdiplus::Image *bonusEXP3Pic;
Gdiplus::Image *bonusEXP4Pic;
Gdiplus::Image *bonusEXP5Pic;

std::vector<Ball *> balls;
std::vector<Brick *> bricks;
std::vector<Bonus *> bonuses;
std::vector<TextBox *> places;
std::vector<TextBox *> names;
std::vector<TextBox *> scores;
Platform *platform;

HFONT hFont;

bool isGameStarted = false;
bool isGamePaused = false;
bool isShowingLB = false;
bool isNeedGeneration = true;
bool isWaitForStarted = false;
bool isNeedRepaintBackground = true;
bool isNeedRepaintLevel = true;
bool isNeedRepaintScore = true;
bool isNeedRepaintLives = true;

bool isFireBall = false;

DWORD startTick;
DWORD endTick;

int numOfBricks;
int numOfBalls;
FloatRECT leftSide;
FloatRECT rightSide;
FloatRECT upSide;
FloatRECT downSide;


int GenerateBricks(int numOfLevel);

Brick *BrickFactory(int brickPosX, int brickPosY, int brickType);

Bonus *BonusFactory(float offsetX, float offsetY, BonusType bonusType, BrickType
```

```
brickType);

    TextBox *TextBoxFactory(float offsetX, float offsetY, float width, float height,
std::string value, COLORREF color);

    void LoadLeaderBoard();

    void SaveLeaderBoard();

    void CalculateBackground(float &backgroundX0, float &backgroundY0, float
&backgroundWidth, float &backgroundHeight);

    void CalculateGameBox(float &gameBoxX0, float &gameBoxY0, float &gameBoxSide,
float &scale);

    void CalculateGameZone();

    void ReleaseGraphicsResources();

    void InitPaint();

    void CalculateFontProperties();

    void PrepareFontDrawing(HFONT &hfont);

    void ReleaseFontResources(HFONT &hfont);

    void GameplayProcessor();

    void DeleteBalls();

    void DeleteBricks();

    void DeleteBonuses();

    void RepaintWhatsNeeded();

    void FillWhatsNeed();

    void PaintWhatsNeed();

    static void CorrectOffsetAndAngle(Ball *ball, FloatRECT barrierRect, int
numOfIntersection);

    void DeleteWhatsNeeded();

    void CorrectOffsetAndAngleByPlatform(Ball *ball, FloatRECT platform, int
numOfIntersection);

    static BonusType RandomizeBonus();

    void BeginAgainThisLevel();

    void UseBonus(Bonus *bonus);

    void ProcessingGameCondition();

    void ResetPlatform();

    void SetBonusesNeedDelete();
```

```
    void SetBallsNeedDelete();
    void ProcessingWinCondition();
    void ProcessingRestartCondition();
    void ProcessingGenerationCondition();
    void SetUsingFireBall(bool fireball);
    void DeleteLeaderBoardData();
    void SwapPlaces(int firstNum, int secondNum);
    void DefaultInitScoreBoard(int startIndex, COLORREF color);
    void RepaintGameInfo();
    void HighScoreCheck();
public:
    void SetResized();
    explicit GameSession(HWND hWnd);
    ~GameSession();
    void ResizeEvent();
    static void PreparerResize(LPMINMAXINFO &lpminmaxinfo);
    void Repaint();
    void MovePlatform(float center);
    void SwitchPause();
    [[nodiscard]] bool IsShowingLB() const;
    void SwitchShowingLB();
    bool IsPaused();
    void TryToStartGame();
    void MovePlatformLeft();
    void MovePlatformRight();
    void SetAllNeedRepaint(bool background);
    void ShowLeaderBoard();
    void AddToScoreBoard(std::string *name);
};

#endif //ARKANOID_GAMESESSION_H
```

Файл GameSession.cpp:

```
#include <fstream>
#include <utility>
#include "GameSession.h"
#include "iostream"
#include "algorithm"


GameSession::GameSession(const HWND hwnd) : hWnd(hwnd) {
    backgroundPic = new Gdiplus::Image(BACKGROUND_PIC_PATH);
    pausePic = new Gdiplus::Image(PAUSE_PIC_PATH);
    leaderBoardPic = new Gdiplus::Image(LEADER_BOARD_PIC_PATH);
    gameZonePic = new Gdiplus::Image(GAME_BOX_PIC_PATH);
    platformPic = new Gdiplus::Image(PLATFORM_PIC_PATH);
    defaultBallPic = new Gdiplus::Image(BALL_PIC_PATH);
    blueBrickPic = new Gdiplus::Image(BLUE_BRICK_PIC_PATH);
    greenBrickPic = new Gdiplus::Image(GREEN_BRICK_PIC_PATH);
    purpleBrickPic = new Gdiplus::Image(PURPLE_BRICK_PIC_PATH);
    redBrickPic = new Gdiplus::Image(RED_BRICK_PIC_PATH);
    yellowBrickPic = new Gdiplus::Image(YELLOW_BRICK_PIC_PATH);

    fireBallPic = new Gdiplus::Image(FIREBALL_PIC_PATH);
    bonusBallPic = new Gdiplus::Image(BONUS_BALL_PATH);
    bonusCutPic = new Gdiplus::Image(BONUS_CUT_PATH);
    bonusExpandPic = new Gdiplus::Image(BONUS_EXPAND_PATH);
    bonusFireBallPic = new Gdiplus::Image(BONUS_FIREBALL_PATH);
    bonusEXP1Pic = new Gdiplus::Image(BONUS_EXP1_PATH);
    bonusEXP2Pic = new Gdiplus::Image(BONUS_EXP2_PATH);
    bonusEXP3Pic = new Gdiplus::Image(BONUS_EXP3_PATH);
    bonusEXP4Pic = new Gdiplus::Image(BONUS_EXP4_PATH);
    bonusEXP5Pic = new Gdiplus::Image(BONUS_EXP5_PATH);

    ballPic = defaultBallPic;

    level = 1;
    lives = 3;
    score = 0;
    numOfBalls = 1;

    lf.lfCharSet = DEFAULT_CHARSET;
    lf.lfPitchAndFamily = DEFAULT_PITCH;
    strcpy(lf.lfFaceName, "Arial");
    lf.lfHeight = DEFAULT_FONT_HEIGHT;
    lf.lfWidth = DEFAULT_FONT_WIDTH;
    lf.lfWeight = DEFAULT_FONT_WEIGHT;
    lf.lfEscapement = DEFAULT_FONT_ESCAPEMENT;
    lf.lfUnderline = DEFAULT_FONT_UNDERLINE;
    lf.lfStrikeOut = DEFAULT_FONT_STRIKE_OUT;
    lf.lfClipPrecision = CLIP_DEFAULT_PRECIS;
    lf.lfItalic = DEFAULT_FONT_ITALIC;
    lf.lfQuality = ANTIALIASED_QUALITY;

    levelTextRect.bottom = TEXT_BOTTOM;
    levelTextRect.left = LEVEL_LEFT;
    levelTextRect.right = LEVEL_RIGHT;
    levelTextRect.top = TEXT_TOP;

    scoreTextRect.bottom = TEXT_BOTTOM;
    scoreTextRect.left = SCORE_LEFT;
```

```
    scoreTextRect.right = SCORE_RIGHT;
    scoreTextRect.top = TEXT_TOP;

    livesTextRect.bottom = TEXT_BOTTOM;
    livesTextRect.left = LIVES_LEFT;
    livesTextRect.right = LIVES_RIGHT;
    livesTextRect.top = TEXT_TOP;

    platform = new Platform(gameZoneX0, gameZoneY0, platformPic, scale,
DEFAULT_PLATFORM_OFFSET_X,
                            DEFAULT_PLATFORM_OFFSET_Y);

    brush = CreateSolidBrush(RGB(0, 0, 0));

    leftSide.left = LEFT_SIDE_LEFT;
    leftSide.right = LEFT_SIDE_RIGHT;
    leftSide.top = LEFT_SIDE_TOP;
    leftSide.bottom = LEFT_SIDE_BOTTOM;

    rightSide.left = RIGHT_SIDE_LEFT;
    rightSide.right = RIGHT_SIDE_RIGHT;
    rightSide.top = RIGHT_SIDE_TOP;
    rightSide.bottom = RIGHT_SIDE_BOTTOM;

    downSide.left = DOWN_SIDE_LEFT;
    downSide.right = DOWN_SIDE_RIGHT;
    downSide.top = DOWN_SIDE_TOP;
    downSide.bottom = DOWN_SIDE_BOTTOM;

    upSide.left = UP_SIDE_LEFT;
    upSide.right = UP_SIDE_RIGHT;
    upSide.top = UP_SIDE_TOP;
    upSide.bottom = UP_SIDE_BOTTOM;

    LoadLeaderBoard();

}

GameSession::~GameSession() {
    delete backgroundPic;
    delete pausePic;
    delete leaderBoardPic;
    delete gameZonePic;
    delete platformPic;
    delete defaultBallPic;
    delete blueBrickPic;
    delete greenBrickPic;
    delete purpleBrickPic;
    delete redBrickPic;
    delete yellowBrickPic;
    delete platform;

    delete fireBallPic;
    delete bonusBallPic;
    delete bonusCutPic;
    delete bonusExpandPic;
    delete bonusFireBallPic;
    delete bonusEXP1Pic;
    delete bonusEXP2Pic;
    delete bonusEXP3Pic;
    delete bonusEXP4Pic;
```

```
    delete bonusEXP5Pic;

    SaveLeaderBoard();
    DeleteBricks();
    DeleteBalls();
    DeleteBonuses();
    DeleteLeaderBoardData();
    DeleteObject(brush);
}

void GameSession::ResizeEvent() {
    GetClientRect(hWnd, &clientRect);
    clientWidth = clientRect.right - clientRect.left;
    clientHeight = clientRect.bottom - clientRect.top;

    CalculateBackground(backgroundX0, backgroundY0, backgroundWidth,
backgroundHeight);
    CalculateGameBox(gameBoxX0, gameBoxY0, gameBoxSide, scale);
    CalculateGameZone();

    CalculateFontProperties();
    SetResized();
}

void GameSession::Repaint() {
    InitPaint();
    PrepareFontDrawing(hFont);
    if (isNeedRepaintBackground) {
        graphics->DrawImage(backgroundPic, backgroundX0, backgroundY0,
backgroundWidth, backgroundHeight);
        graphics->DrawImage(gameZonePic, gameBoxX0, gameBoxY0, gameBoxSide,
gameBoxSide);
        SetAllNeedRepaint(false);
    }
    GameplayProcessor();
    RepaintGameInfo();
    if (isGamePaused) {
        graphics->DrawImage(pausePic, gameBoxX0, gameBoxY0, gameBoxSide,
gameBoxSide);
    }
    if (isShowingLB) {
        ShowLeaderBoard();
    }
    ReleaseFontResources(hFont);
    ReleaseGraphicsResources();
    if (!isShowingLB && !isGamePaused && isGameStarted && !isWaitForStarted) {
        InvalidateRect(hWnd, NULL, false);
    }
}

void GameSession::GameplayProcessor() {
    if (lives == 0) {
        HighScoreCheck();
        ProcessingRestartCondition();
    }
    if (isNeedGeneration) {
        ProcessingGenerationCondition();
        srand(GetTickCount());
    }
    if (isGameStarted && !isGamePaused && !isShowingLB) {
        ProcessingGameCondition();
```

```
    }
    RepaintWhatsNeeded();
    DeleteWhatsNeeded();
}

void GameSession::ProcessingGameCondition() {
    endTick = GetTickCount();
    if (startTick == endTick) return;
    for (; startTick <= endTick; startTick++) {
        for (auto ball: balls) {
            if (ball->IsDestroyed()) continue;
            ball->CalculateNextPoint(DEFAULT_TIME);
            CorrectOffsetAndAngleByPlatform(ball, platform->GetRECT(),
                                            ball->GetNumOfIntersection(platform-
>GetRECT()));
            CorrectOffsetAndAngle(ball, leftSide, ball-
>GetNumOfIntersection(leftSide));
            CorrectOffsetAndAngle(ball, rightSide, ball-
>GetNumOfIntersection(rightSide));
            CorrectOffsetAndAngle(ball, upSide, ball->GetNumOfIntersection(upSide));
            if (ball->GetNumOfIntersection(downSide) != 0) {
                numOfBalls--;
                ball->SetDestroyed();
                if (numOfBalls != 0) continue;
                else {
                    lives--;
                    SetUsingFireBall(false);
                    ResetPlatform();
                    SetBonusesNeedDelete();
                    SetBallsNeedDelete();
                    if (lives == 0) break;
                    else {
                        BeginAgainThisLevel();
                    }
                }
            }
            for (auto brick: bricks) {
                if (brick->IsDestroyed()) continue;
                int numOfIntersection = ball->GetNumOfIntersection(brick->GetRECT());
                if (numOfIntersection == 0) continue;
                numOfBricks -= brick->HitTheBrick(isFireBall);
                FloatRECT brickRect = brick->GetRECT();
                if (brick->IsDestroyed()) {
                    score += brick->GetPrice();

                    Bonus *bonus = BonusFactory((brickRect.right - brickRect.left) /
2 + brickRect.left,
                                                (brickRect.bottom - brickRect.top) /
2 + brickRect.top,
                                                RandomizeBonus(), brick-
>GetBrickType());
                    if (bonus != nullptr) {
                        bonuses.push_back(bonus);
                    }
                }
                if (!isFireBall) {
                    CorrectOffsetAndAngle(ball, brickRect, numOfIntersection);
                }
                if (numOfBricks == 0) {
                    ProcessingWinCondition();
                }
```

```
            }
        }
        for (auto bonus:bonuses) {
            if (bonus->IsDestroyed()) continue;
            bonus->CalculateNextPoint(DEFAULT_TIME);
            if (bonus->GetNumOfIntersection(platform->GetRECT())) {
                UseBonus(bonus);
                bonus->PrepareToRelocate();
                continue;
            }
            if (bonus->GetNumOfIntersection(downSide) == INTERSECTION_UP) {
                bonus->SetDestroyed();
                bonus->PrepareToRelocate();
                continue;
            }
            for (auto brick:bricks) {
                int numOfIntersection = bonus->GetNumOfIntersection(brick-
>GetRECT());
                if (numOfIntersection == 0) continue;
                brick->PrepareToRelocate();
            }
        }
    }

}

void GameSession::BeginAgainThisLevel() {
    SetUsingFireBall(false);
    balls.push_back(new Ball(gameZoneX0, gameZoneY0, ballPic, scale,
                             platform->GetRealWidth() / 2 + platform-
>GetRealOffsetX() -
                             ballPic->GetWidth() / 2,
                             platform->GetOffsetY() - 1 - ballPic->GetHeight(),
DEFAULT_SPEED,
                             DEFAULT_ANGLE));
    isGameStarted = false;
    isGamePaused = false;
    isNeedGeneration = false;
    isWaitForStarted = true;
    numOfBalls = 1;
    ResetPlatform();
    SetBonusesNeedDelete();
    RepaintWhatsNeeded();
    DeleteWhatsNeeded();
}

void GameSession::DeleteBalls() {
    for (Ball *ball: balls) {
        delete ball;
    }
    balls.clear();
}

void GameSession::DeleteBricks() {
    for (Brick *brick: bricks) {
        delete brick;
    }
    bricks.clear();
}

void GameSession::DeleteBonuses() {
```

```
    for (Bonus *bonus: bonuses) {
        delete bonus;
    }
    bonuses.clear();
}

void GameSession::PreparerResize(LPMINMAXINFO &lpminmaxinfo) {
    lpminmaxinfo->ptMinTrackSize.x = MIN_GAME_ZONE_SIDE + 20;
    lpminmaxinfo->ptMinTrackSize.y = MIN_GAME_ZONE_SIDE + 40;
}

void GameSession::InitPaint() {
    hdc = BeginPaint(hWnd, &ps);
    memDC = CreateCompatibleDC(hdc);
    hBM = CreateCompatibleBitmap(hdc, clientWidth, clientHeight);
    oldBmp = (HBITMAP) SelectObject(memDC, hBM);
    graphics = new Gdiplus::Graphics(memDC);
    BitBlt(memDC, 0, 0, clientWidth, clientHeight, hdc, 0, 0, SRCCOPY);
}

void GameSession::ReleaseGraphicsResources() {
    BitBlt(hdc, 0, 0, clientWidth, clientHeight, memDC, 0, 0, SRCCOPY);
    ValidateRect(hWnd, &ps.rcPaint);
    DeleteObject(SelectObject(memDC, oldBmp));
    DeleteObject(hBM);
    DeleteDC(memDC);
    EndPaint(hWnd, &ps);
    delete graphics;
}

void GameSession::CalculateBackground(float &backgroundX0, float &backgroundY0, float
&backgroundWidth,
                                      float &backgroundHeight) {
    float horizontalCoeff = backgroundPic->GetWidth() / clientWidth;
    float verticalCoeff = backgroundPic->GetHeight() / clientHeight;
    if (horizontalCoeff < verticalCoeff) {
        backgroundWidth = backgroundPic->GetWidth() / horizontalCoeff;
        backgroundHeight = backgroundPic->GetHeight() / horizontalCoeff;
        backgroundY0 = 0 - (backgroundHeight - clientHeight) / 2;
        backgroundX0 = 0;
    } else {
        backgroundWidth = backgroundPic->GetWidth() / verticalCoeff;
        backgroundHeight = backgroundPic->GetHeight() / verticalCoeff;
        backgroundX0 = 0 - (backgroundWidth - clientWidth) / 2;
        backgroundY0 = 0;
    }
}

void GameSession::CalculateGameBox(float &gameBoxX0, float &gameBoxY0, float
&gameBoxSide, float &scale) {
    if (clientWidth < clientHeight) {
        gameBoxSide = clientWidth;
        gameBoxY0 = (clientHeight - gameBoxSide) / 2;
        gameBoxX0 = 0;
        scale = gameBoxSide / (float) gameZonePic->GetWidth();
    } else {
        gameBoxSide = clientHeight;
        gameBoxX0 = (clientWidth - gameBoxSide) / 2;
        gameBoxY0 = 0;
        scale = gameBoxSide / (float) gameZonePic->GetHeight();
    }
```

```
}

void GameSession::CalculateFontProperties() {
    lf.lfHeight = DEFAULT_FONT_HEIGHT * scale;
    lf.lfWidth = DEFAULT_FONT_WIDTH * scale;
    lf.lfWeight = DEFAULT_FONT_WEIGHT * scale;

    levelTextRect.bottom = TEXT_BOTTOM * scale + gameBoxY0;
    levelTextRect.left = LEVEL_LEFT * scale + gameBoxX0;
    levelTextRect.right = LEVEL_RIGHT * scale + gameBoxX0;
    levelTextRect.top = TEXT_TOP * scale + gameBoxY0;

    scoreTextRect.bottom = TEXT_BOTTOM * scale + gameBoxY0;
    scoreTextRect.left = SCORE_LEFT * scale + gameBoxX0;
    scoreTextRect.right = SCORE_RIGHT * scale + gameBoxX0;
    scoreTextRect.top = TEXT_TOP * scale + gameBoxY0;

    livesTextRect.bottom = TEXT_BOTTOM * scale + gameBoxY0;
    livesTextRect.left = LIVES_LEFT * scale + gameBoxX0;
    livesTextRect.right = LIVES_RIGHT * scale + gameBoxX0;
    livesTextRect.top = TEXT_TOP * scale + gameBoxY0;
}

void GameSession::PrepareFontDrawing(HFONT &hfont) {
    hfont = CreateFontIndirect(&lf);
    DeleteObject(SelectObject(memDC, hfont));
    SetTextColor(memDC, RGB(0, 230, 255));
    SetBkColor(memDC, RGB(0, 0, 0));
}

void GameSession::ReleaseFontResources(HFONT &hfont) {
    DeleteObject(hfont);
}

void GameSession::CalculateGameZone() {
    gameZoneY0 = gameBoxY0 + DEFAULT_GAME_ZONE_TOP * scale;
    gameZoneX0 = gameBoxX0;
}

int GameSession::GenerateBricks(int numOfLevel) {
    int Width;
    int Height;
    int numOfBlocks = 0;
    std::string filename = RESOURCE_ROOT;
    filename += LVL_DIR;
    filename += ConvertIntToString(numOfLevel);
    filename += LVL_EXTENSION;
    std::ifstream reader(filename);
    if (!reader.is_open()) return -1;
    reader >> Width;
    reader >> Height;
    for (int i = 0; i < Height; i++) {
        for (int j = 0; j < Width; j++) {
            int brickType;
            reader >> brickType;
            Brick *brick;
            brick = BrickFactory(j, i, brickType);
            if (brick == nullptr) continue;
            bricks.push_back(brick);
            numOfBlocks++;
        }
```

```
    }
    reader.close();
    return numOfBlocks;
}

Brick *GameSession::BrickFactory(int brickPosX, int brickPosY, int brickType) {
    switch (brickType) {
        case BRICK_NUM_NULL:
            return nullptr;
        case BRICK_NUM_PURPLE:
            return new Brick(gameZoneX0, gameZoneY0, purpleBrickPic, scale, brickPosX
* BRICK_WIDTH,
                             brickPosY * BRICK_HEIGHT, BRICK_HITS_PURPLE,
BRICK_PRICE_PURPLE, BRICK_TYPE_PURPLE);
        case BRICK_NUM_BLUE:
            return new Brick(gameZoneX0, gameZoneY0, blueBrickPic, scale, brickPosX *
BRICK_WIDTH,
                             brickPosY * BRICK_HEIGHT, BRICK_HITS_BLUE,
BRICK_PRICE_BLUE, BRICK_TYPE_BLUE);
        case BRICK_NUM_GREEN:
            return new Brick(gameZoneX0, gameZoneY0, greenBrickPic, scale, brickPosX
* BRICK_WIDTH,
                             brickPosY * BRICK_HEIGHT, BRICK_HITS_GREEN,
BRICK_PRICE_GREEN, BRICK_TYPE_GREEN);
        case BRICK_NUM_YELLOW:
            return new Brick(gameZoneX0, gameZoneY0, yellowBrickPic, scale, brickPosX
* BRICK_WIDTH,
                             brickPosY * BRICK_HEIGHT, BRICK_HITS_YELLOW,
BRICK_PRICE_YELLOW, BRICK_TYPE_YELLOW);
        case BRICK_NUM_RED:
            return new Brick(gameZoneX0, gameZoneY0, redBrickPic, scale, brickPosX *
BRICK_WIDTH,
                             brickPosY * BRICK_HEIGHT, BRICK_HITS_RED,
BRICK_PRICE_RED, BRICK_TYPE_RED);
        default:
            return nullptr;
    }
}

void GameSession::MovePlatform(float center) {
    platform->Move(center);
    if (isWaitForStarted) {
        balls[0]->SetOffsetX((platform->GetRealWidth() / 2 + platform-
>GetRealOffsetX() - ballPic->GetWidth() / 2));
    }
}

void GameSession::MovePlatformLeft() {
    platform->MoveLeft();
    if (isWaitForStarted) {
        balls[0]->SetOffsetX((platform->GetRealWidth() / 2 + platform-
>GetRealOffsetX() - ballPic->GetWidth() / 2));
    }
}

void GameSession::MovePlatformRight() {
    platform->MoveRight();
    if (isWaitForStarted) {
        balls[0]->SetOffsetX((platform->GetRealWidth() / 2 + platform-
>GetRealOffsetX() - ballPic->GetWidth() / 2));
    }
```

```
}

void GameSession::SetResized() {
    SetAllNeedRepaint(true);
}

void GameSession::SwitchPause() {
    isGamePaused = !isGamePaused;
    if (!isGamePaused) {
        SetAllNeedRepaint(true);
        startTick = GetTickCount();
    }
    InvalidateRect(hWnd, NULL, false);
}

bool GameSession::IsPaused() {
    return isGamePaused;
}

void GameSession::SetAllNeedRepaint(bool background) {
    isNeedRepaintBackground = background;
    isNeedRepaintLevel = true;
    isNeedRepaintScore = true;
    isNeedRepaintLives = true;
    for (auto ball: balls) {
        if (!ball->IsDestroyed()) {
            ball->SetNeedRepaint();
        }
    }
    for (auto brick: bricks) {
        if (!brick->IsDestroyed()) {
            brick->SetNeedRepaint();
        }
    }
    for (auto bonus: bonuses) {
        bonus->SetNeedRepaint();
    }
    platform->SetNeedRepaint();
}

void GameSession::TryToStartGame() {
    if (isWaitForStarted && !isGameStarted && !isGamePaused && !isShowingLB) {
        isWaitForStarted = false;
        isGameStarted = true;
        InvalidateRect(hWnd, NULL, false);
        startTick = GetTickCount();
    }
}

void GameSession::RepaintWhatsNeeded() {
    FillWhatsNeed();
    PaintWhatsNeed();
}

void GameSession::FillWhatsNeed() {
    for (auto ball: balls) {
        if (ball->IsWasFilled()) {
            FillRect(memDC, &ball->repaintRect, brush);
        }

    }
```

```
    for (auto brick: bricks) {
        if (brick->IsWasFilled()) {
            FillRect(memDC, &brick->repaintRect, brush);
        }
    }
    for (auto bonus: bonuses) {
        if (bonus->IsWasFilled()) {
            FillRect(memDC, &bonus->repaintRect, brush);
        }
    }

    if (platform->IsWasFilled()) {
        FillRect(memDC, &platform->repaintRect, brush);
    }
}

void GameSession::PaintWhatsNeed() {
    for (auto ball: balls) {
        if (ball->IsNeedRepaint() && !ball->IsDestroyed()) {
            ball->PaintOnGraphics(*graphics);
        }

    }
    for (auto brick: bricks) {
        if (brick->IsNeedRepaint() && !brick->IsDestroyed()) {
            brick->PaintOnGraphics(*graphics);
        }
    }
    for (auto bonus: bonuses) {
        if (bonus->IsNeedRepaint() && !bonus->IsDestroyed()) {
            bonus->PaintOnGraphics(*graphics);
        }
    }
    if (platform->IsNeedRepaint()) {
        platform->PaintOnGraphics(*graphics);
    }
}

void GameSession::CorrectOffsetAndAngle(Ball *ball, FloatRECT barrierRect, int
numOfIntersection) {
    float angle = ball->GetAngle();
    switch (numOfIntersection) {
        case INTERSECTION_NONE:
            break;
        case INTERSECTION_DOWN: {
            ball->SetOffsetY(barrierRect.bottom);
            if (angle == 270) {
                ball->SetAngle(fmod(angle + 180, 360));
            } else if (angle > 180 && angle < 270) {
                ball->SetAngle(360 - angle);
            } else if (angle > 270 && angle < 360) {
                ball->SetAngle(360 - angle);
            }
        }
            break;
        case INTERSECTION_LEFT: {
            ball->SetOffsetX2(barrierRect.left);
            if (angle == 0) {
                ball->SetAngle(fmod(angle + 180, 360));
            } else if (angle > 0 && angle < 90) {
                ball->SetAngle(180 - angle);
```

```
    } else if (angle > 270 && angle < 360) {
        ball->SetAngle(540 - angle);
    }
}
    break;
case INTERSECTION_UP: {
    ball->SetOffsetY2(barrierRect.top);
    if (angle == 90) {
        ball->SetAngle(fmod(angle + 180, 360));
    } else if (angle > 0 && angle < 90) {
        ball->SetAngle(360 - angle);
    } else if (angle > 90 && angle < 180) {
        ball->SetAngle(360 - angle);
    }

}
    break;
case INTERSECTION_RIGHT: {
    ball->SetOffsetX(barrierRect.right);
    if (angle == 180) {
        ball->SetAngle(fmod(angle + 180, 360));
    } else if (angle > 90 && angle < 180) {
        ball->SetAngle(180 - angle);
    } else if (angle > 180 && angle < 270) {
        ball->SetAngle(540 - angle);
    }
}
    break;
case INTERSECTION_LEFT_AND_UP: {
    ball->SetOffsetX2(barrierRect.left);
    ball->SetOffsetY2(barrierRect.top);
    if (angle == 45) {
        ball->SetAngle(fmod(angle + 180, 360));
    }

}
    break;

case INTERSECTION_RIGHT_AND_UP: {
    ball->SetOffsetX(barrierRect.right);
    ball->SetOffsetY2(barrierRect.top);
    if (angle == 135) {
        ball->SetAngle(fmod(angle + 180, 360));
    }
}
    break;
case INTERSECTION_RIGHT_AND_DOWN: {
    ball->SetOffsetX(barrierRect.right);
    ball->SetOffsetY(barrierRect.bottom);
    if (angle == 225) {
        ball->SetAngle(fmod(angle + 180, 360));
    }
}
    break;

case INTERSECTION_LEFT_AND_DOWN: {
    ball->SetOffsetX2(barrierRect.left);
    ball->SetOffsetY(barrierRect.bottom);
    if (angle == 315) {
        ball->SetAngle(fmod(angle + 180, 360));
    }
```

```
        }
            break;
    }
}

void GameSession::DeleteWhatsNeeded() {
    for (int i = 0; i < balls.size(); i++) {
        if (balls[i]->IsDestroyed()) {
            Ball *ball = balls[i];
            auto newEnd = std::remove(balls.begin(), balls.end(), ball);
            balls.erase(newEnd, balls.end());
            delete ball;
        }
    }
    for (int i = 0; i < bricks.size(); i++) {
        if (bricks[i]->IsDestroyed()) {
            Brick *brick = bricks[i];
            auto newEnd = std::remove(bricks.begin(), bricks.end(), brick);
            bricks.erase(newEnd, bricks.end());
            delete brick;
            i--;
        }
    }
    for (int i = 0; i < bonuses.size(); i++) {
        if (bonuses[i]->IsDestroyed()) {
            Bonus *bonus = bonuses[i];
            auto newEnd = std::remove(bonuses.begin(), bonuses.end(), bonus);
            bonuses.erase(newEnd, bonuses.end());
            delete bonus;
        }
    }
}

void GameSession::CorrectOffsetAndAngleByPlatform(Ball *ball, FloatRECT platform, int
numOfIntersection) {
    float angle = ball->GetAngle();
    switch (numOfIntersection) {
        case INTERSECTION_NONE:
            break;
        case INTERSECTION_UP: {
            srand(GetTickCount());
            ball->SetOffsetY2(platform.top);
            FloatRECT ballRect = ball->GetRECT();
            float ballCenter = (ballRect.right - ballRect.left) / 2;
            float platformCenter = (platform.right - platform.left) / 2;
            float distanceBtwBallAndPlatform = abs(ballCenter - platformCenter);
            float dotCoefficient = distanceBtwBallAndPlatform / platformCenter;
            if (angle > 0 && angle < 90) {
                ball->SetAngle(360 - angle * dotCoefficient * (float) (500 + rand() %
1000) / 1000);
                if (ball->GetAngle() > 330) ball->SetAngle(330);
            } else if (angle > 90 && angle < 180) {
                ball->SetAngle(360 - angle * dotCoefficient * (float) (500 + rand() %
1000) / 1000);
                if (ball->GetAngle() < 210) ball->SetAngle(210);
            }
        }
        default:
            this->platform->PrepareToRelocate();
    }
}
```

```
BonusType GameSession::RandomizeBonus() {
    int random = rand() % 4;
    if (random == 0) return BONUS_NONE;
    random = rand() % 16 + 1;
    if (random >= 1 && random <= 4) return BONUS_EXPAND;
    if (random >= 5 && random <= 8) return BONUS_CUT;
    if (random >= 9 && random <= 12) return BONUS_MORE_BALLS;
    if (random >= 13 && random <= 15) return BONUS_EXPERIENCE;
    if (random >= 16) return BONUS_FIREBALL;
    return BONUS_NONE;
}

Bonus *GameSession::BonusFactory(float offsetX, float offsetY, BonusType bonusType,
BrickType brickType) {
    switch (bonusType) {
        case BONUS_NONE:
            return nullptr;
        case BONUS_MORE_BALLS:
            return new Bonus(gameZoneX0, gameZoneY0, bonusBallPic, scale, offsetX,
offsetY, BONUS_MORE_BALLS, 0);
        case BONUS_CUT:
            return new Bonus(gameZoneX0, gameZoneY0, bonusCutPic, scale, offsetX,
offsetY, BONUS_CUT, 0);
        case BONUS_FIREBALL:
            return new Bonus(gameZoneX0, gameZoneY0, bonusFireBallPic, scale,
offsetX, offsetY, BONUS_FIREBALL, 0);
        case BONUS_EXPAND:
            return new Bonus(gameZoneX0, gameZoneY0, bonusExpandPic, scale, offsetX,
offsetY, BONUS_EXPAND, 0);
        case BONUS_EXPERIENCE: {
            switch (brickType) {
                case BRICK_TYPE_PURPLE:
                    return new Bonus(gameZoneX0, gameZoneY0, bonusEXP1Pic, scale,
offsetX, offsetY, BONUS_EXPERIENCE,
                                     BRICK_PRICE_PURPLE * 25);
                case BRICK_TYPE_GREEN:
                    return new Bonus(gameZoneX0, gameZoneY0, bonusEXP3Pic, scale,
offsetX, offsetY, BONUS_EXPERIENCE,
                                     BRICK_PRICE_GREEN * 25);
                case BRICK_TYPE_BLUE:
                    return new Bonus(gameZoneX0, gameZoneY0, bonusEXP2Pic, scale,
offsetX, offsetY, BONUS_EXPERIENCE,
                                     BRICK_PRICE_BLUE * 25);
                case BRICK_TYPE_RED:
                    return new Bonus(gameZoneX0, gameZoneY0, bonusEXP5Pic, scale,
offsetX, offsetY, BONUS_EXPERIENCE,
                                     BRICK_PRICE_RED * 25);
                case BRICK_TYPE_YELLOW:
                    return new Bonus(gameZoneX0, gameZoneY0, bonusEXP4Pic, scale,
offsetX, offsetY, BONUS_EXPERIENCE,
                                     BRICK_PRICE_YELLOW * 25);
                default:
                    return nullptr;
            }
        }
        default:
            return nullptr;
    }
```

```
}

void GameSession::UseBonus(Bonus *bonus) {
    switch (bonus->GetBonusType()) {
        case BONUS_EXPERIENCE: {
            score += bonus->GetPrice();
        }
            break;
        case BONUS_EXPAND: {
            platform->IncSizeCoefficient();
        }
            break;
        case BONUS_FIREBALL: {
            SetUsingFireBall(true);
        }
            break;
        case BONUS_CUT: {
            platform->DecSizeCoefficient();
        }
            break;
        case BONUS_MORE_BALLS: {
            int localNumOfBalls = balls.size();
            for (int i = 0; i < localNumOfBalls; i++) {
                FloatRECT ballRect = balls[i]->GetRECT();
                balls.push_back(new Ball(gameZoneX0, gameZoneY0, ballPic, scale,
ballRect.left, ballRect.top,
                                         balls[i]->GetSpeed(), fmod(balls[i]-
>GetAngle() + 180, 360)));
                numOfBalls++;
            }
        }
            break;
    }
    bonus->SetDestroyed();
}

void GameSession::ResetPlatform() {
    platform->SetOffsetX(DEFAULT_PLATFORM_OFFSET_X);
    platform->SetOffsetY(DEFAULT_PLATFORM_OFFSET_Y);
    platform->SetDefaultSizeCoefficient();
}

void GameSession::SetBonusesNeedDelete() {
    for (auto bonus: bonuses) {
        bonus->PrepareToRelocate();
        bonus->SetDestroyed();
    }
}

void GameSession::SetBallsNeedDelete() {
    for (auto ball: balls) {
        ball->PrepareToRelocate();
        ball->SetDestroyed();
    }
}

void GameSession::ProcessingWinCondition() {
    isGameStarted = false;
    isGamePaused = false;
    isNeedGeneration = true;
    isWaitForStarted = false;
```

```
    SetUsingFireBall(false);
    numOfBalls = 1;
    level += 1;
    ResetPlatform();
    SetBallsNeedDelete();
    SetBonusesNeedDelete();
    RepaintWhatsNeeded();
    DeleteWhatsNeeded();
    InvalidateRect(hWnd, NULL, false);
}

void GameSession::ProcessingRestartCondition() {
    isGameStarted = false;
    isGamePaused = false;
    isNeedGeneration = true;
    isWaitForStarted = false;
    isNeedRepaintBackground = true;
    score = 0;
    lives = 3;
    level = 1;
    numOfBalls = 1;

}

void GameSession::ProcessingGenerationCondition() {
    DeleteBalls();
    DeleteBricks();
    DeleteBonuses();
    numOfBricks = GenerateBricks(level);
    if (numOfBricks == -1) {
        score += 1000 * lives * (level - 1);
        lives = 0;
        InvalidateRect(hWnd, NULL, false);
        SetAllNeedRepaint(true);
        return;
    }
    balls.push_back(new Ball(gameZoneX0, gameZoneY0, ballPic, scale,
                             platform->GetRealWidth() / 2 + platform-
>GetRealOffsetX() - ballPic->GetWidth() / 2,
                             platform->GetOffsetY() - 1 - ballPic->GetHeight(),
DEFAULT_SPEED, DEFAULT_ANGLE));
    numOfBalls = 1;
    isNeedGeneration = false;
    isWaitForStarted = true;
    SetUsingFireBall(false);
    ResetPlatform();
    SetAllNeedRepaint(true);
}

void GameSession::SetUsingFireBall(bool fireball) {
    if (fireball) {
        ballPic = fireBallPic;
    } else {
        ballPic = defaultBallPic;
    }
    isFireBall = fireball;
}

bool GameSession::IsShowingLB() const {
    return isShowingLB;
}
```

```
void GameSession::SwitchShowingLB() {
    isShowingLB = !isShowingLB;
    if (!isShowingLB) {
        SetAllNeedRepaint(true);
        startTick = GetTickCount();
    }
    InvalidateRect(hWnd, NULL, false);
}

void GameSession::ShowLeaderBoard() {
    graphics->DrawImage(leaderBoardPic, gameBoxX0, gameBoxY0, gameBoxSide,
gameBoxSide);
    for (auto place: places) {
        place->DrawOnDC(memDC);
    }
    for (auto name: names) {
        name->DrawOnDC(memDC);
    }
    for (auto scoreItem: scores) {
        scoreItem->DrawOnDC(memDC);
    }
}

void GameSession::DeleteLeaderBoardData() {
    for (auto place: places) {
        delete place;
    }
    places.clear();
    for (auto name: names) {
        delete name;
    }
    names.clear();
    for (auto scoreItem:scores) {
        delete scoreItem;
    }
    scores.clear();
}

void GameSession::LoadLeaderBoard() {
    COLORREF color = RGB(99, 97, 97);
    std::ifstream reader(LEADER_BOARD_PATH);
    if (!reader.is_open()) {
        std::ofstream out(LEADER_BOARD_PATH);
        out.close();
        DefaultInitScoreBoard(0, color);
        return;
    }
    int numOfLeaders;
    numOfLeaders = 0;
    std::string name;
    int scoreItem;
    getline(reader, name);
    while (!reader.eof() && numOfLeaders < 10) {
        numOfLeaders++;
        std::string scoreStr;
        getline(reader, scoreStr);
        scoreItem = ConvertStringToLong(scoreStr);
        places.push_back(
                TextBoxFactory(NUM_OF_LEAD_OFFSET_X, LEAD_OFFSET_Y + (numOfLeaders -
1) * LEAD_VERTICAL_INTERVAL,
```

```
                                NUM_OF_LEAD_WIDTH,
                                LEAD_HEIGHT, ConvertIntToString(numOfLeaders),
color));
        names.push_back(
                TextBoxFactory(NAME_OF_LEAD_OFFSET_X, LEAD_OFFSET_Y + (numOfLeaders -
1) * LEAD_VERTICAL_INTERVAL,
                                NAME_OF_LEAD_WIDTH,
                                LEAD_HEIGHT, name, color));
        scores.push_back(
                TextBoxFactory(SCORE_OF_LEAD_OFFSET_X, LEAD_OFFSET_Y + (numOfLeaders
- 1) * LEAD_VERTICAL_INTERVAL,
                                SCORE_OF_LEAD_WIDTH,
                                LEAD_HEIGHT, ConvertIntToString(scoreItem), color));
        getline(reader, name);
    }
    DefaultInitScoreBoard(numOfLeaders, color);
    reader.close();
}

TextBox *
GameSession::TextBoxFactory(float offsetX, float offsetY, float width, float height,
std::string value,
                            COLORREF color) {
    return new TextBox(gameBoxX0, gameBoxY0, scale, offsetX, offsetY, width, height,
std::move(value), color);
}

void GameSession::AddToScoreBoard(std::string *name) {
    bool readyToAdd = false;
    for (int i = 0; i < 10; i++) {
        if (winScore > ConvertStringToLong(scores[i]->GetValue())) {
            for (int j = 9; j >= i; j--) {
                if (ConvertStringToLong(scores[j]->GetValue()) != 0 && j != 8 + 1) {
                    SwapPlaces(j, j + 1);
                }
                readyToAdd = true;
            }
        }
        if (readyToAdd) {
            scores[i]->SetValue(ConvertIntToString(winScore));
            names[i]->SetValue(*name);
            delete name;
            break;
        }

    }
}

void GameSession::SwapPlaces(int firstNum, int secondNum) {
    std::string tmpScore = scores[secondNum]->GetValue();
    std::string tmpName = names[secondNum]->GetValue();
    scores[secondNum]->SetValue(scores[firstNum]->GetValue());
    names[secondNum]->SetValue(names[firstNum]->GetValue());
    if (ConvertStringToLong(tmpScore) != 0) {
        scores[firstNum]->SetValue(tmpScore);
        names[firstNum]->SetValue(tmpName);
    }
}

void GameSession::SaveLeaderBoard() {
    std::ofstream out;
```

```
    out.open(LEADER_BOARD_PATH);
    if (out.is_open()) {
        for (int i = 0; i < 10; i++) {
            if (ConvertStringToLong(scores[i]->GetValue()) != 0)
                out << names[i]->GetValue() << std::endl << scores[i]->GetValue() <<
std::endl;
            else
                break;
        }
    }
    out.close();
}

void GameSession::DefaultInitScoreBoard(int startIndex, COLORREF color) {
    for (int i = startIndex; i < 10; i++) {
        std::string name = "";
        std::string score = "";
        places.push_back(
                TextBoxFactory(NUM_OF_LEAD_OFFSET_X, LEAD_OFFSET_Y + i *
LEAD_VERTICAL_INTERVAL, NUM_OF_LEAD_WIDTH,
                               LEAD_HEIGHT, ConvertIntToString(i + 1), color));
        names.push_back(
                TextBoxFactory(NAME_OF_LEAD_OFFSET_X, LEAD_OFFSET_Y + i *
LEAD_VERTICAL_INTERVAL, NAME_OF_LEAD_WIDTH,
                               LEAD_HEIGHT, name, color));
        scores.push_back(
                TextBoxFactory(SCORE_OF_LEAD_OFFSET_X, LEAD_OFFSET_Y + i *
LEAD_VERTICAL_INTERVAL, SCORE_OF_LEAD_WIDTH,
                               LEAD_HEIGHT, score, color));
    }
}

void GameSession::RepaintGameInfo() {
    if (oldLevel != level || isNeedRepaintLevel) {
        std::string levelStr = ConvertIntToString(level);
        FillRect(memDC, &levelTextRect, brush);
        DrawTextA(memDC, levelStr.c_str(), -1, (LPRECT) &levelTextRect,
                  DT_CENTER | DT_SINGLELINE | DT_VCENTER);
        oldLevel = level;
        isNeedRepaintLevel = false;
    }
    if (oldScore != score || isNeedRepaintScore) {
        std::string scoreStr = ConvertIntToString(score);
        FillRect(memDC, &scoreTextRect, brush);
        DrawTextA(memDC, scoreStr.c_str(), -1, (LPRECT) &scoreTextRect,
                  DT_CENTER | DT_SINGLELINE | DT_VCENTER);
        oldScore = score;
        isNeedRepaintScore = false;
    }
    if (oldLives != lives || isNeedRepaintLives) {
        std::string livesStr = ConvertIntToString(lives);
        FillRect(memDC, &livesTextRect, brush);
        DrawTextA(memDC, livesStr.c_str(), -1, (LPRECT) &livesTextRect,
                  DT_CENTER | DT_SINGLELINE | DT_VCENTER);
        oldLives = lives;
        isNeedRepaintLives = false;
    }

}

void GameSession::HighScoreCheck() {
```

```
    for (int i = 0; i < 10; i++) {
        if (score > ConvertStringToLong(scores[i]->GetValue())) {
            winScore = score;
            SendMessageA(hWnd, WM_NEED_A_DIALOG_BOX, 0, 0);
            break;
        }
    }
}
```

Файл Platform.h:

```
#ifndef ARKANOID_PLATFORM_H
#define ARKANOID_PLATFORM_H


#include "DrawableShape.h"

class Platform : public DrawableShape {
private:
    float sizeCoefficient = 1;
public:
    void IncSizeCoefficient();

    void DecSizeCoefficient();

    void Move(float centerX);

    float GetRealWidth();

    float GetRealOffsetX();

    float GetOffsetY();

    Platform(float &gameZoneX0, float &gameZoneY0, Gdiplus::Image *&image, float
&scale, float offsetX, float offsetY);

    void PaintOnGraphics(Gdiplus::Graphics &graphics) override;

    void SetRepaintRECT() override;

    void CalculateRECT() override;

    void MoveLeft();

    void MoveRight();

    void SetDefaultSizeCoefficient();
};


#endif //ARKANOID_PLATFORM_H
```

Файл Platform.cpp:

```
#include "Platform.h"


void Platform::IncSizeCoefficient() {
```

```
    if (sizeCoefficient < 2.24) {
        PrepareToRelocate();
        sizeCoefficient *= 1.5;
        if (GetRealOffsetX() < 0) {
            float imageWidth = image->GetWidth();
            offsetX = (imageWidth * sizeCoefficient - imageWidth) / 2;
        }
        if (GetRealOffsetX() + GetRealWidth() >= DEFAULT_GAME_ZONE_WIDTH) {
            offsetX = DEFAULT_GAME_ZONE_WIDTH - GetRealWidth();
        }
    }
}

void Platform::DecSizeCoefficient() {
    if (sizeCoefficient > 0.45) {
        PrepareToRelocate();
        sizeCoefficient /= 1.5;
        if (GetRealOffsetX() < 0) {
            float imageWidth = image->GetWidth();
            offsetX = (imageWidth * sizeCoefficient - imageWidth) / 2;
        }
        if (GetRealOffsetX() + GetRealWidth() >= DEFAULT_GAME_ZONE_WIDTH) {
            offsetX = DEFAULT_GAME_ZONE_WIDTH - GetRealWidth();
        }
    }
}

void Platform::SetDefaultSizeCoefficient() {
    PrepareToRelocate();
    sizeCoefficient = 1;
}

void Platform::PaintOnGraphics(Gdiplus::Graphics &graphics) {
    graphics.DrawImage(image, gameZoneX0 + GetRealOffsetX() * scale, gameZoneY0 +
offsetY * scale,
                       GetRealWidth() * scale,
                       this->GetHeight() * scale);
    EndPaint();
}

float Platform::GetRealWidth() {
    return image->GetWidth() * sizeCoefficient;
}

float Platform::GetRealOffsetX() {
    float imageWidth = image->GetWidth();
    return offsetX - (imageWidth * sizeCoefficient - imageWidth) / 2;
}

void Platform::Move(float centerX) {
    if (!IsWasFilled()) {
        SetRepaintRECT();
    }

    float imageWidth = image->GetWidth();
    float realWidth = GetRealWidth();
    if (((centerX - realWidth / 2 * scale)) < gameZoneX0) {
        offsetX = (imageWidth * sizeCoefficient - imageWidth) / 2;
    } else if ((centerX + realWidth / 2 * scale) > gameZoneX0 +
DEFAULT_GAME_ZONE_WIDTH * scale) {
        offsetX = DEFAULT_GAME_ZONE_WIDTH - realWidth + (realWidth - imageWidth) / 2;
```

```
    } else {
        offsetX = centerX / scale - gameZoneX0 / scale - imageWidth / 2;
    }
    needRepaint = true;
}

float Platform::GetOffsetY() {
    return offsetY;
}

void Platform::SetRepaintRECT() {
    repaintRect.left = round(gameZoneX0 + GetRealOffsetX() * scale);
    repaintRect.right = ceil(gameZoneX0 + (GetRealOffsetX() + GetRealWidth()) *
scale);
    repaintRect.bottom = ceil(gameZoneY0 + (offsetY + GetHeight()) * scale);
    repaintRect.top = round(gameZoneY0 + offsetY * scale);
    wasFilled = true;
}

void Platform::CalculateRECT() {
    rect.left = GetRealOffsetX();
    rect.right = GetRealOffsetX() + GetRealWidth();
    rect.bottom = offsetY + image->GetHeight();
    rect.top = offsetY;
}

void Platform::MoveLeft() {
    if (!IsWasFilled()) {
        SetRepaintRECT();
    }

    offsetX -= DEFAULT_PLATFORM_MOVE;
    if (GetRealOffsetX() < 0) {
        float imageWidth = image->GetWidth();
        offsetX = (imageWidth * sizeCoefficient - imageWidth) / 2;
    }

    needRepaint = true;
}

void Platform::MoveRight() {
    if (!IsWasFilled()) {
        SetRepaintRECT();
    }

    offsetX += DEFAULT_PLATFORM_MOVE;
    if (GetRealOffsetX() + GetRealWidth() >= DEFAULT_GAME_ZONE_WIDTH) {
        offsetX = DEFAULT_GAME_ZONE_WIDTH - GetRealWidth();
    }


    needRepaint = true;
}

Platform::Platform(float &gameZoneX0, float &gameZoneY0, Gdiplus::Image *&image,
float &scale, float offsetX,
                   float offsetY) : DrawableShape(gameZoneX0, gameZoneY0, image,
scale, offsetX, offsetY) {}
```

Файл TextBox.h:

```
#ifndef ARKANOID_TEXTBOX_H
#define ARKANOID_TEXTBOX_H

#include <string>
#include "Windows.h"

class TextBox {
private:
    float &gameBoxX0;
    float &gameBoxY0;
    float &scale;
    float offsetX;
    float offsetY;
    float width;
    float height;
    RECT textZone;
    std::string value;
    COLORREF backgroundColor;

    void PrepareTextZone();

public:

    TextBox(float &gameBoxX0, float &gameBoxY0, float &scale, float offsetX, float
offsetY, float width, float height,
            std::string value, COLORREF backgroundColor);

    [[nodiscard]] std::string GetValue() const;

    void SetValue(std::string string);

    void DrawOnDC(HDC hdc);
};


#endif //ARKANOID_TEXTBOX_H
```

Файл TextBox.cpp:

```
#include "TextBox.h"

std::string TextBox::GetValue() const {
    return value;
}

void TextBox::SetValue(std::string string) {
    TextBox::value = std::move(string);
}

void TextBox::PrepareTextZone() {
    textZone.left = gameBoxX0 + offsetX * scale;
    textZone.right = gameBoxX0 + scale * (offsetX + width);
    textZone.top = gameBoxY0 + offsetY * scale;
    textZone.bottom = gameBoxY0 + scale * (offsetY + height);
}

void TextBox::DrawOnDC(HDC hdc) {
```

```
    auto prevValue = SetBkColor(hdc, backgroundColor);
    PrepareTextZone();
    DrawTextA(hdc, value.c_str(), -1, (LPRECT) &textZone, DT_CENTER | DT_SINGLELINE |
DT_VCENTER);
    SetBkColor(hdc, prevValue);
}

TextBox::TextBox(float &gameBoxX0, float &gameBoxY0, float &scale, float offsetX,
float offsetY, float width,
                 float height, std::string value, COLORREF backgroundColor) :
gameBoxX0(gameBoxX0),

gameBoxY0(gameBoxY0),

scale(scale), offsetX(offsetX),

offsetY(offsetY), width(width),

height(height),

value(std::move(value)),

backgroundColor(backgroundColor) {}
```

Файл main.cpp:

```
#include <iostream>
#include <windows.h>
#include "gdiplus.h"
#include "GameSession.h"


HINSTANCE mainInstance;
GameSession *gameSession;
bool isLeftButtonDown = false;
WNDCLASSEX mainWindowClass;


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

INT_PTR CALLBACK DialogProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPTSTR lpCmdLine,
                     int nCmdShow) {
    Gdiplus::GdiplusStartupInput gdiplusStartupInput;
    ULONG_PTR gdiplusToken;
    GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, nullptr);

    WNDCLASSEX wcex;
    HWND hWnd;
    MSG msg;
    mainInstance = hInstance;
    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = WndProc;
    wcex.cbClsExtra = 0;
```

```
    wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance;
    wcex.hIcon = (HICON) LoadImageA(hInstance, "IDI_ICON1", IMAGE_ICON, 256, 256, 0);
    wcex.hIconSm = wcex.hIcon;
    wcex.hCursor = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH) GetStockObject(BLACK_BRUSH);
    wcex.lpszMenuName = NULL;
    wcex.lpszClassName = "ArkanoidWindowClass";
    wcex.hIconSm = wcex.hIcon;
    mainWindowClass = wcex;

    RegisterClassEx(&wcex);


    hWnd = CreateWindow("ArkanoidWindowClass",  // Указатель на зарегистрированное
имя класса
                        "Arkanoid The Game",    // Указатель на имя окна
                        WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU | WS_THICKFRAME |
WS_MINIMIZEBOX |
                        WS_MAXIMIZEBOX,         // Стиль окна
                        CW_USEDEFAULT,          // Горизонтальная позиция окна
                        0,                      // Вертикальная позиция окна
                        1440,                   // Ширина окна
                        810,                    // Высота окна
                        NULL,                   // Дескриптор родительского или окна
владельца
                        NULL,                   // Дескриптор меню или идентификатор
дочернего окна
                        hInstance,              // Дескриптор экземпляра приложения
                        NULL);                  // Указатель на данные создания окна

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    Gdiplus::GdiplusShutdown(gdiplusToken);
    return (int) msg.wParam;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message,
                         WPARAM wParam, LPARAM lParam) {
    switch (message) {
        case WM_PAINT:
            gameSession->Repaint();
            break;
        case WM_LBUTTONDOWN: {
            isLeftButtonDown = !isLeftButtonDown;
            if (!gameSession->IsPaused() && !gameSession->IsShowingLB()) {
                int offsetX = LOWORD(lParam);
                gameSession->MovePlatform(offsetX);
                InvalidateRect(hWnd, NULL, false);
            }
        }
            break;
        case WM_MOUSEMOVE:
            if (isLeftButtonDown && !gameSession->IsPaused() && !gameSession-
>IsShowingLB()) {
```

```
                int offsetX = LOWORD(lParam);
                gameSession->MovePlatform(offsetX);
                InvalidateRect(hWnd, NULL, false);
            }
            break;
        case WM_KEYDOWN:
            switch (wParam) {
                case VK_LEFT:
                    if (!gameSession->IsPaused() && !gameSession->IsShowingLB()) {
                        gameSession->MovePlatformLeft();
                        InvalidateRect(hWnd, NULL, false);
                    }
                    break;
                case VK_RIGHT:
                    if (!gameSession->IsPaused() && !gameSession->IsShowingLB()) {
                        gameSession->MovePlatformRight();
                        InvalidateRect(hWnd, NULL, false);
                    }
                    break;
                case VK_ESCAPE:
                    if (!gameSession->IsShowingLB()) {
                        gameSession->SwitchPause();
                    }
                    break;
                case VK_SPACE:
                    gameSession->TryToStartGame();
                    break;
                case VK_TAB:
                    if (!gameSession->IsPaused()) {
                        gameSession->SwitchShowingLB();
                    }
            }
            break;
        case WM_SIZE:
            gameSession->ResizeEvent();
            break;
        case WM_GETMINMAXINFO: {
            auto lpMMI = (LPMINMAXINFO) lParam;
            gameSession->PreparerResize(lpMMI);
        }
            break;
        case WM_MOVE:
            gameSession->SetAllNeedRepaint(true);
            break;
        case WM_ACTIVATE:
            gameSession->SetAllNeedRepaint(true);
            break;
        case WM_NEED_A_DIALOG_BOX: {
            INT_PTR result = DialogBoxA(mainInstance, MAKEINTRESOURCE(129), hWnd,
DialogProc);
            if (result != 0) {
                auto *str = (std::string *) result;
                if (!str->empty())
                    gameSession->AddToScoreBoard(str);
                else {
                    str = new std::string("Player");
                    gameSession->AddToScoreBoard(str);
                }
            }
        }
            break;
```

```
        case WM_CREATE:
            gameSession = new GameSession(hWnd);
            break;
        case WM_DESTROY:
            delete gameSession;
            PostQuitMessage(0);
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

LRESULT CALLBACK DialogProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam) {
    switch (message) {
        case WM_INITDIALOG:
            HICON hIcon;
            hIcon = mainWindowClass.hIcon;
            if (hIcon) {
                SendMessage(hDlg, WM_SETICON, ICON_SMALL, (LPARAM) hIcon);
            }
            return (INT_PTR) TRUE;
        case WM_COMMAND:
            if (LOWORD(wParam) == IDOK) {
                TCHAR *buf;
                int len;
                HWND MainInputEdit = GetDlgItem(hDlg, 1000);
                buf = (TCHAR *) malloc(len = ((GetWindowTextLength(MainInputEdit) +
1) * sizeof(TCHAR)));
                if (buf != NULL) {
                    GetWindowText(MainInputEdit, buf, len);
                }
                std::string editStr = std::string(buf);
                free(buf);
                std::string *tmpStr = new std::string(editStr);
                EndDialog(hDlg, (INT_PTR) tmpStr);
                return (INT_PTR) TRUE;
            } else if (LOWORD(wParam) == IDCANCEL) {
                EndDialog(hDlg, 0);
                return (INT_PTR) TRUE;
            }
            break;
    }
    return (INT_PTR) FALSE;
};
```

ВЕДОМОСТЬ ДОКУМЕНТОВ

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КП 1–40 01 01 616 ПЗ	Пояснительная записка	78 с.
	Графические документы	
ГУИР.851006-01 СА	Перерисовка	Формат А1
	Схема алгоритма	

Изм	Лист	№ докум.	Подп.	Дата
Разраб.		Матюшонок М.С		
Провер.		Жиденко А.Л.		

БГУИР КП I- 40 01 01 616 ПЗ

Игровое приложение «Arkanoid»

Ведомость курсового проекта

Литера	Лист	Листов
Т	78	78

Кафедра ПОИТ
гр. 851006