

Министерство науки и высшего образования Российской Федерации

Рязанский государственный радиотехнический университет  
им. В.Ф. Уткина

Кафедра «Вычислительная и прикладная математика» (ВПМ)

**Пояснительная записка  
к курсовому проекту**

на тему: **Игровая программа: Battle City**

по курсу

**«Конструирование программного обеспечения»**

Выполнил:

студент группы №143

Соколов Е.А.

Проверил:

ст. преподаватель кафедры ВПМ

Васин А.О.

Рязань, 2024

## Оглавление

Введение.....	3
1 Анализ задачи .....	4
1.1 Разработка иерархии классов .....	4
1.1.1 Выделение сущностей.....	6
1.1.2 Зависимости между классами. Диаграмма классов.....	6
1.2 Алгоритмы.....	9
1.2.1 Алгоритм интеллекта вражеских танков .....	9
1.2.2 Поиск кратчайшего пути до базы .....	10
1.2.3 Алгоритм взаимодействия снаряда с объектами на игровой карте .....	11
1.3 Разработка интерфейса программы.....	12
2 Написание программы.....	17
2.1.1 Описание разработанных процедур и функций .....	17
2.2 Разработка программы.....	26
2.2.1 Описание классов, перечислений и интерфейсов проекта .....	26
2.3 Описание шаблонов проектирования, которые использовались при написании программы.....	30
2.3.1 Модель–Представление–Контроллер, Model–View–Controller, MVC.....	30
2.3.2 Одиночка (Singleton).....	30
2.3.3 Абстрактная фабрика (Abstract Factory).....	30
2.4 Описание методов рефакторинга, которые использовались при оптимизации исходного кода программы .....	31
2.4.1 Разбиение метода.....	31
2.4.2 Удаление дубликатов кода .....	35
2.4.3 Замена магических чисел константами .....	37
2.4.4 Удаление мертвого кода .....	37
2.5 Разработка тестов.....	38
2.5.1 Test Cases .....	38
2.5.2 Модульные тесты.....	43
3 Результат работы программы .....	45
3.1 Консольная версия .....	45
3.2 WPF версия .....	50
Заключение.....	54
Приложения .....	54

## Введение

Данная работа представляет собой готовую игровую программу «Battle City», похожую на оригинальную игру. Основная задача разработки программного обеспечения — закрепить теоретические основы, изученные в ходе курса, и применить их на практике при создании полнофункционального программного продукта.

Проект будет реализован на языке программирования C#. Основной паттерн проектирования MVC. Финальный продукт должен иметь два режима работы: консольный и графический.

# 1 Анализ задачи

## 1.1 Разработка иерархии классов

В основе архитектуры разрабатываемого приложения лежит шаблон проектирования MVC, состоящий из:

- model (модель) – класс, ответственный за логику работы Игры, существует в единственном экземпляре. Модель включает в себя как движок Игры, так и вспомогательные утилиты (классы для работы с рекордами, загрузки уровней), модели различных для меню, ресурсы уровней,

- view (вид/представление) – класс, отображающий пользователю текущее состояние модели. У каждого меню существует два вида: консольный и графический (используется технология WPF),

- controller (контроллер) – класс, позволяющий пользователю изменять состояние модели.

Дополнительно стоит отметить, что:

- старт программы начинается с запуска контроллера, который создаёт свое представление, которое уже и создает окно, показывает его пользователю,

- вид знает о существовании модели, а модель не является зависимой от других классов,

- модель создает события, на которые подписывается вид,

- для каждого окна приложения реализован своя собственная реализация шаблона модель-представление-контроллер. При переходе с одного экрана Игры на другой контроллер первого экрана передает управление контроллеру второго.

Высокоуровневая схема архитектуры приложения представлена на рисунке 1.

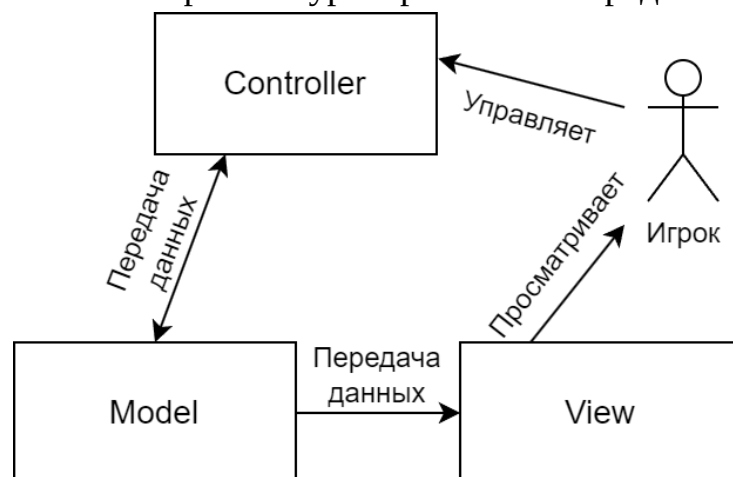


Рисунок 1 – Высокоуровневая схема архитектуры приложения

Диаграмма последовательности, отражающая данный подход на примере главного меню, представлена на рисунке 2.

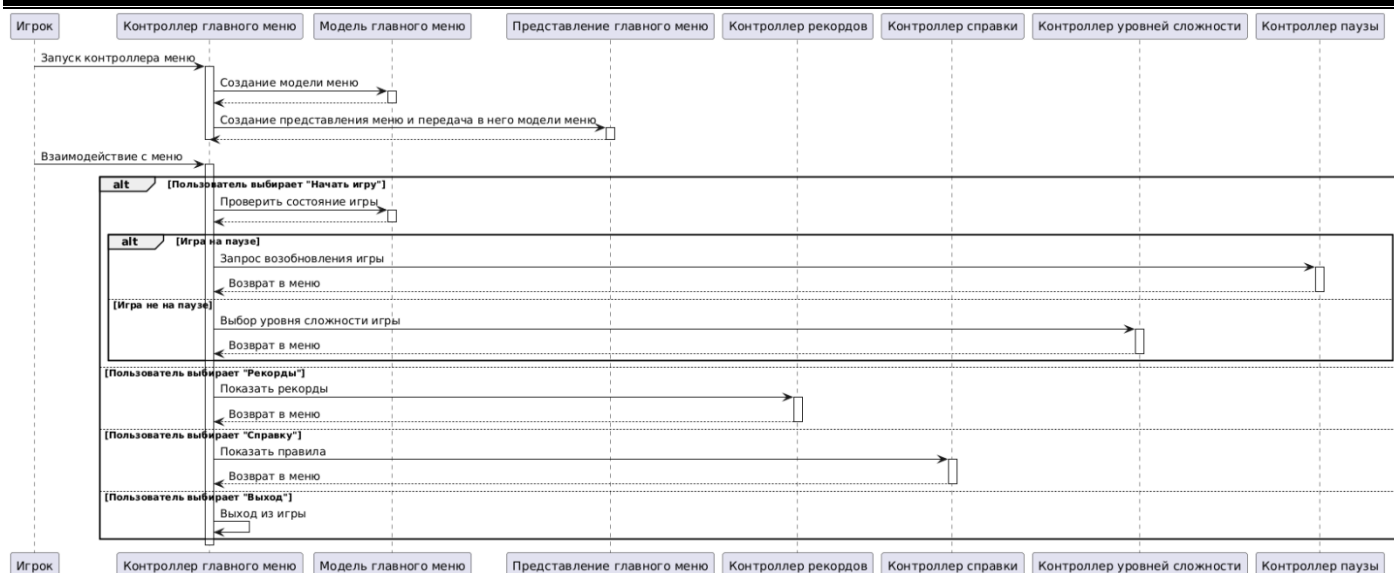


Рисунок 2 – Алгоритм работы главного меню приложения

Диаграмма последовательности, отражающая общий подход взаимодействия между контроллерами, моделью и представлениями, представлена на рисунке 3.

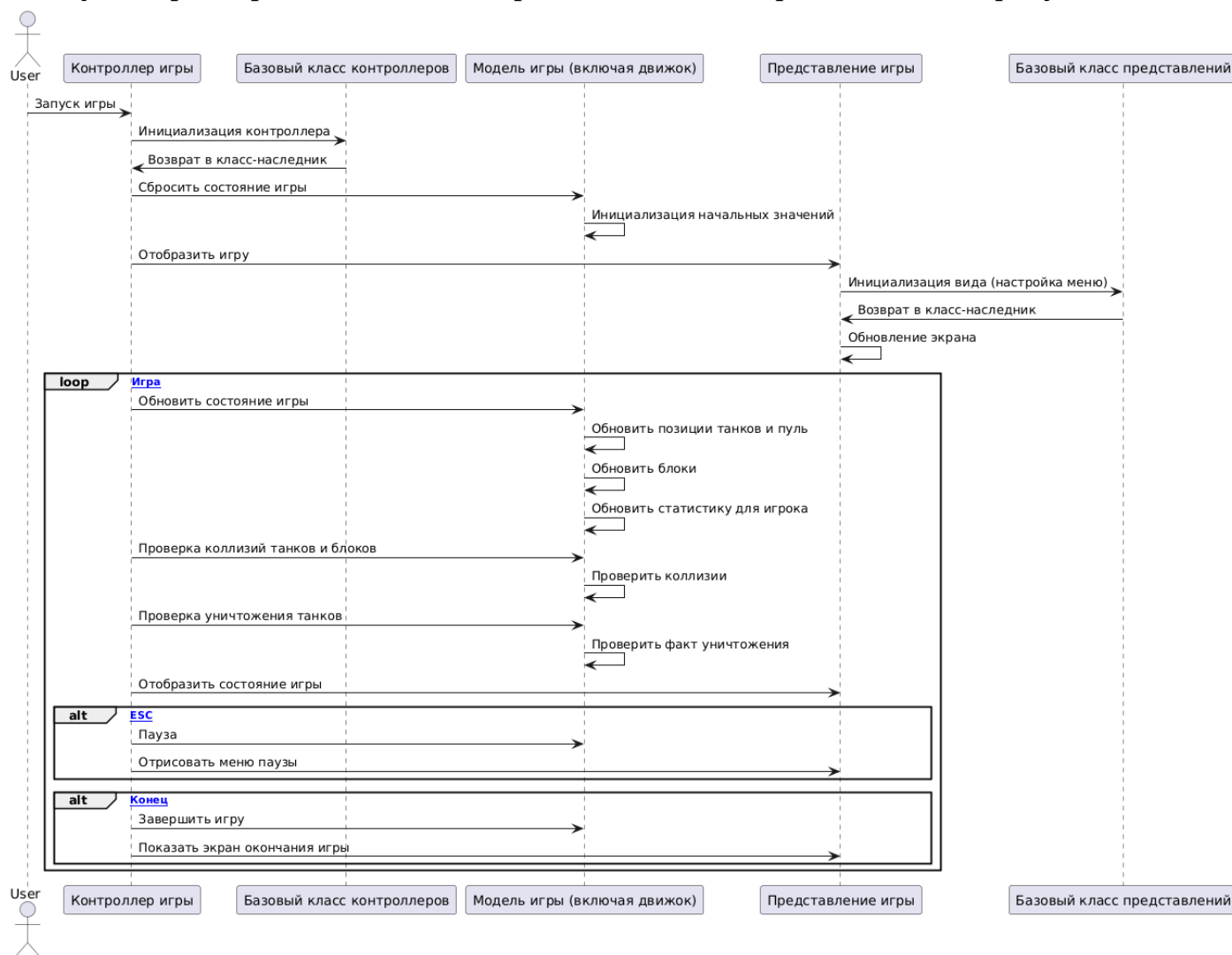


Рисунок 3 – Общий принцип работы шаблона MVC для приложения

### 1.1.1 Выделение сущностей

В результате декомпозиции предметной области выделены следующие сущности:

- Танк – модель игрового танка, которым управляет либо пользователь Игры, либо компьютер,
- Пуля (снаряд) – игровой объект, выпускаемый танками, который может поражать другие танки или другие препятствия,
- Блок – модель препятствия в Игре (стена, кусты, вода и т.п.), который может пропускать или не пропускать танки и пули. Некоторые блоки являются разрушаемыми, а другие – нет.

Дополнительно были выделены такие вспомогательные сущности, как:

- Игровое поле – модель пространства, на котором происходит танковый бой,
- Ячейка – единица пространства игрового поля, на которой может быть расположен блок,
- Модель Игры, Пользователя и настроек Игры – сущность для хранения текущего состояния и параметров Игры,
- Уровень – конфигурация игрового поля с определенным расположением блоков, расположением, количеством и параметрами танков противника, настроек танка Игрока.

### 1.1.2 Зависимости между классами. Диаграмма классов

Основные сущности Игры – это динамические (танки, пули) и статические (блоки) объекты. Диаграмма классов для основных сущностей движка игры представлена на рисунке 4.

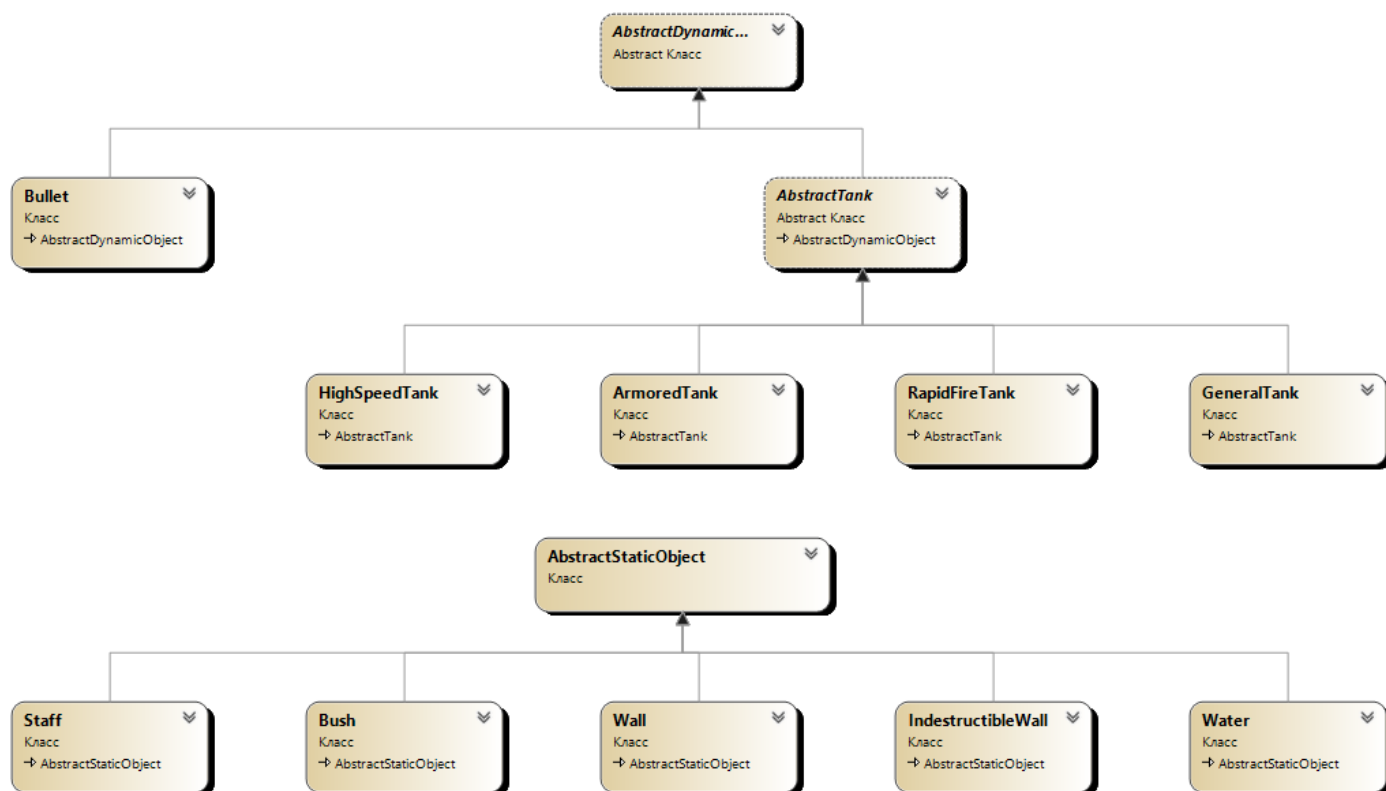


Рисунок 4 – Диаграмма классов для основных сущностей движка Игры

Для обеспечения непосредственно работы Игры необходимы следующие дополнительные классы, описанные в пункте 1.1.1 документа и представленные на рисунке 5.

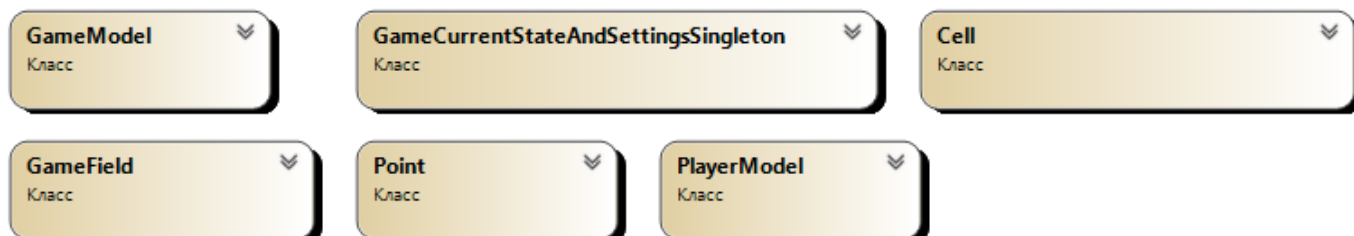


Рисунок 5 – Дополнительные классы движка Игры

Диаграмма вспомогательных классов для чтения файла уровня и преобразования его в формат модели представлена на рисунке 6.

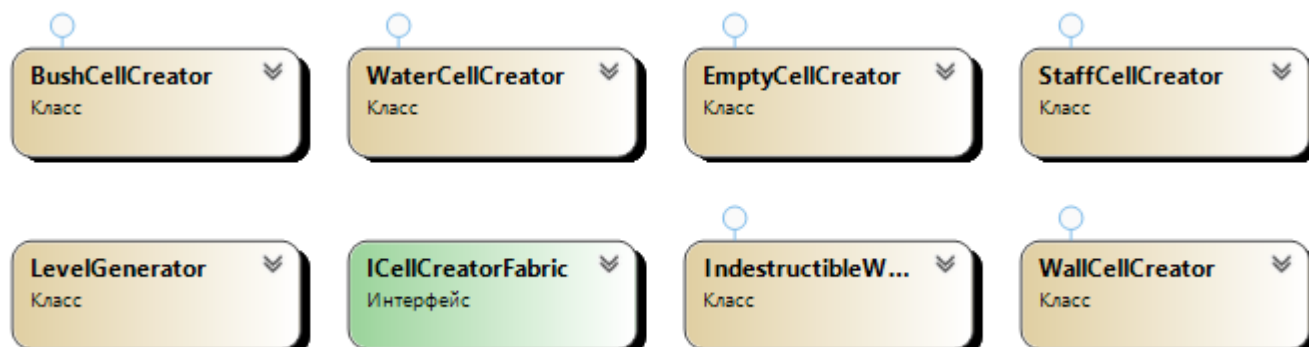


Рисунок 6 – Вспомогательные классы для загрузки уровней

Диаграмма классов для контроллеров приложения представлена на рисунке 7 (в качестве дочерних классов взяты реализации из WPF, для консольного режима иерархия такая же). Классы `ControllerBase` и `ControllerMenu` хранятся в модели.

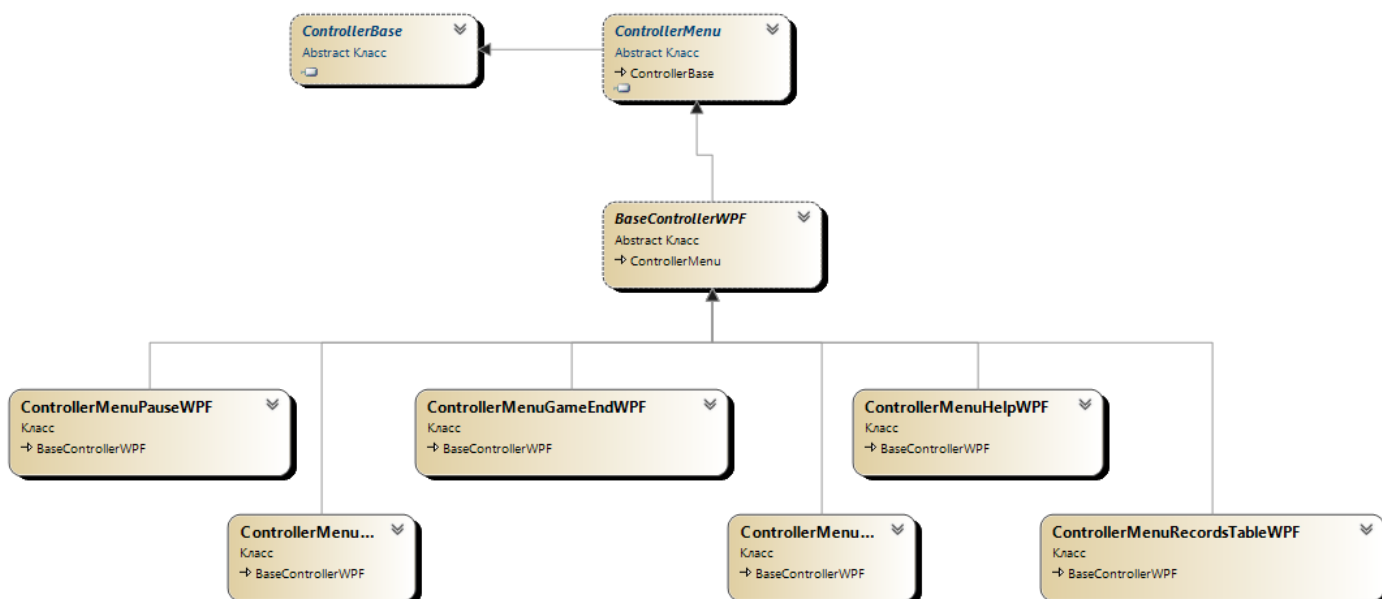


Рисунок 7 – Диаграмма классов для контроллеров приложения (на примере WPF)

Диаграмма классов для представлений приложения представлена на рисунке 8 (в качестве дочерних классов также взяты реализации из WPF). Базовые абстрактные классы хранятся в модели.

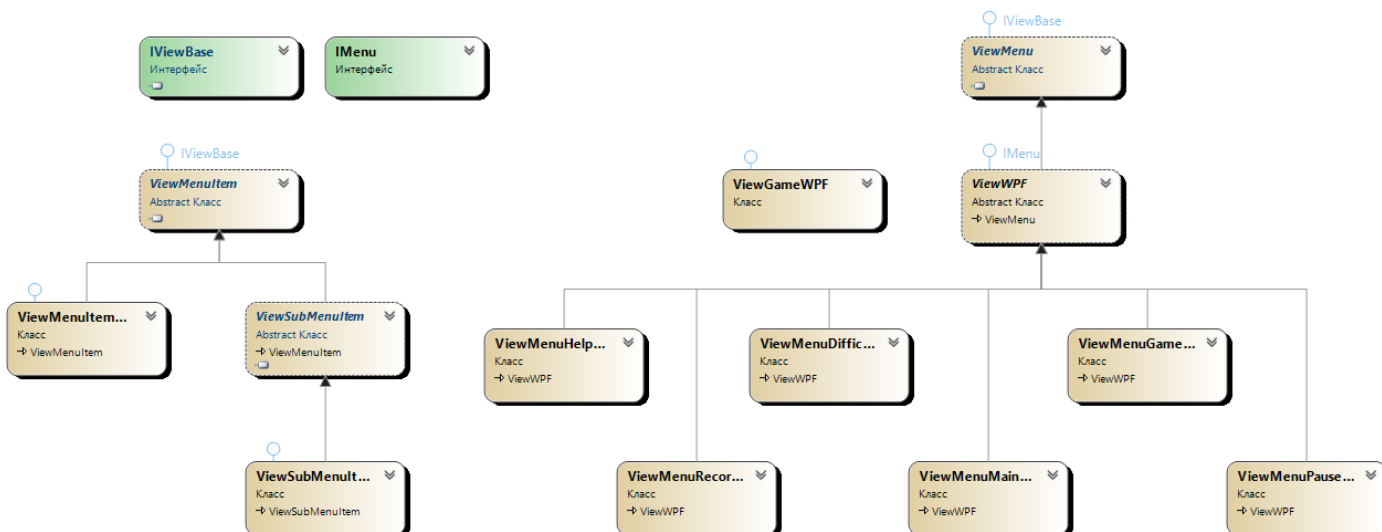


Рисунок 8 – Диаграмма классов для представлений приложения (на примере WPF)



## 1.2 Алгоритмы

### 1.2.1 Алгоритм интеллекта вражеских танков

Вражеский танк стремится уничтожить игрока или базу. Он вычисляет кратчайший путь до танка Игрока и начинает двигаться к нему, периодически проверяя досягаемость цели (танка Игрока или блока базы) для снарядов. Если цель может быть поражена, танк открывает огонь. Если цель не находится в зоне досягаемости, то танк с вероятностью 25% на данном шаге может переключиться на новую цель (приоритетно на танк Игрока).

Данный алгоритм в виде диаграммы последовательности представлен на рисунке 9.

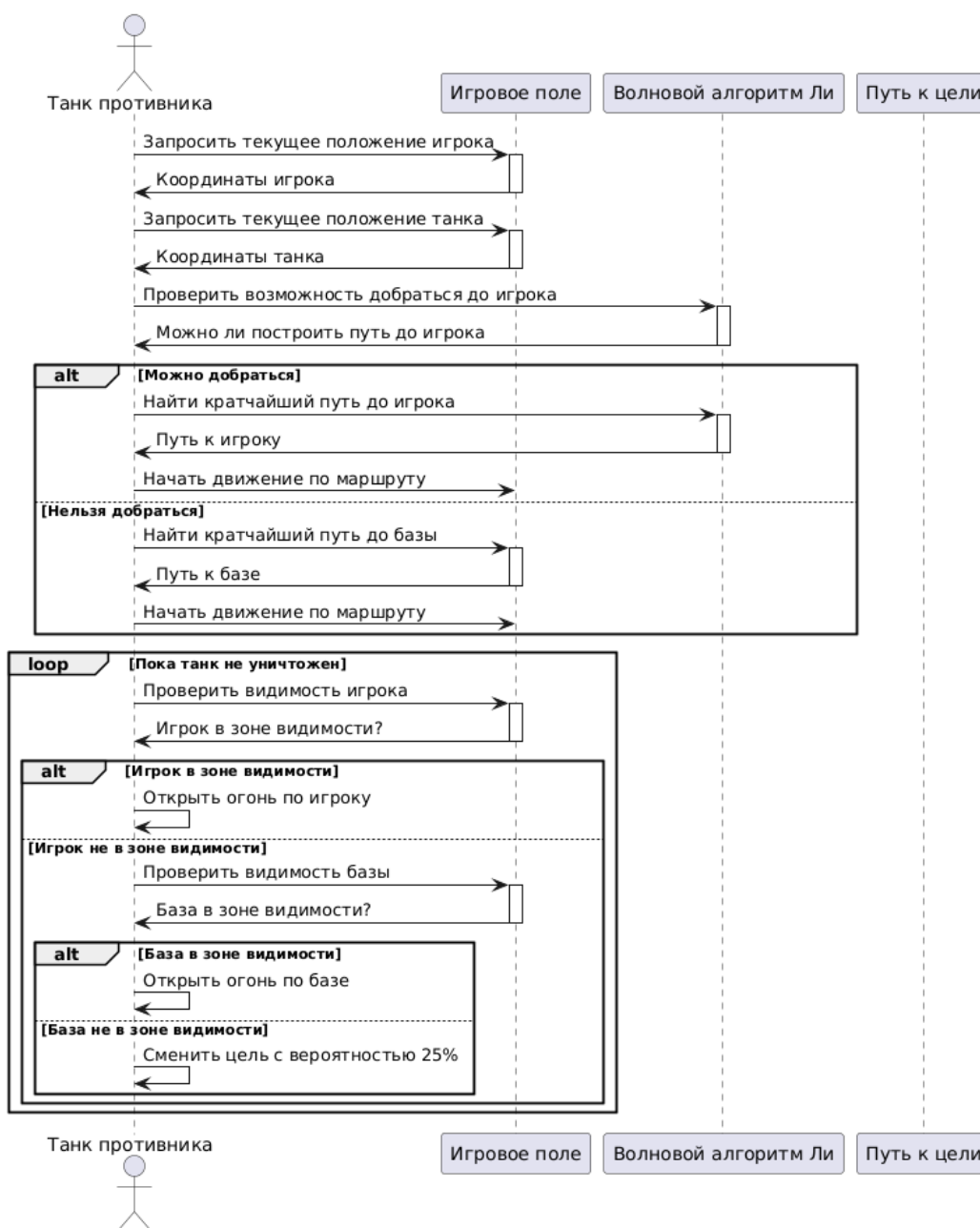


Рисунок 9 – Алгоритм интеллекта вражеских танков

### 1.2.2 Поиск кратчайшего пути до базы

Для поиска танками противника кратчайшего пути до базы используется волновой алгоритм Ли, работающий по принципу постепенного «распространения волны» от стартовой точки (где располагается танк) к целевой (базе), заполняя доступные ячейки шаг за шагом, пока не достигнет цели или не заполнит все возможные пути.

Сам алгоритм можно кратко описать следующим образом:

инициализация: всем ячейкам игрового поля соответствует значение -1, также задается начальная ячейка, ей присваивается значение 0,

распространение волны: на каждом шаге алгоритм помечает соседние ячейки, которые не были еще посещены (т.е. им соответствует -1) значением, увеличенным на 1 по сравнению с текущей,

проверка завершения и восстановление пути: если числовая «волна» достигла целевой точки, то путь существует; восстановить его кратчайшую версию можно, двигаясь в обратном направлении по ячейкам с убывающими значениями.

Блок-схема алгоритма поиска кратчайшего пути танками противника представлена на рисунке 10.

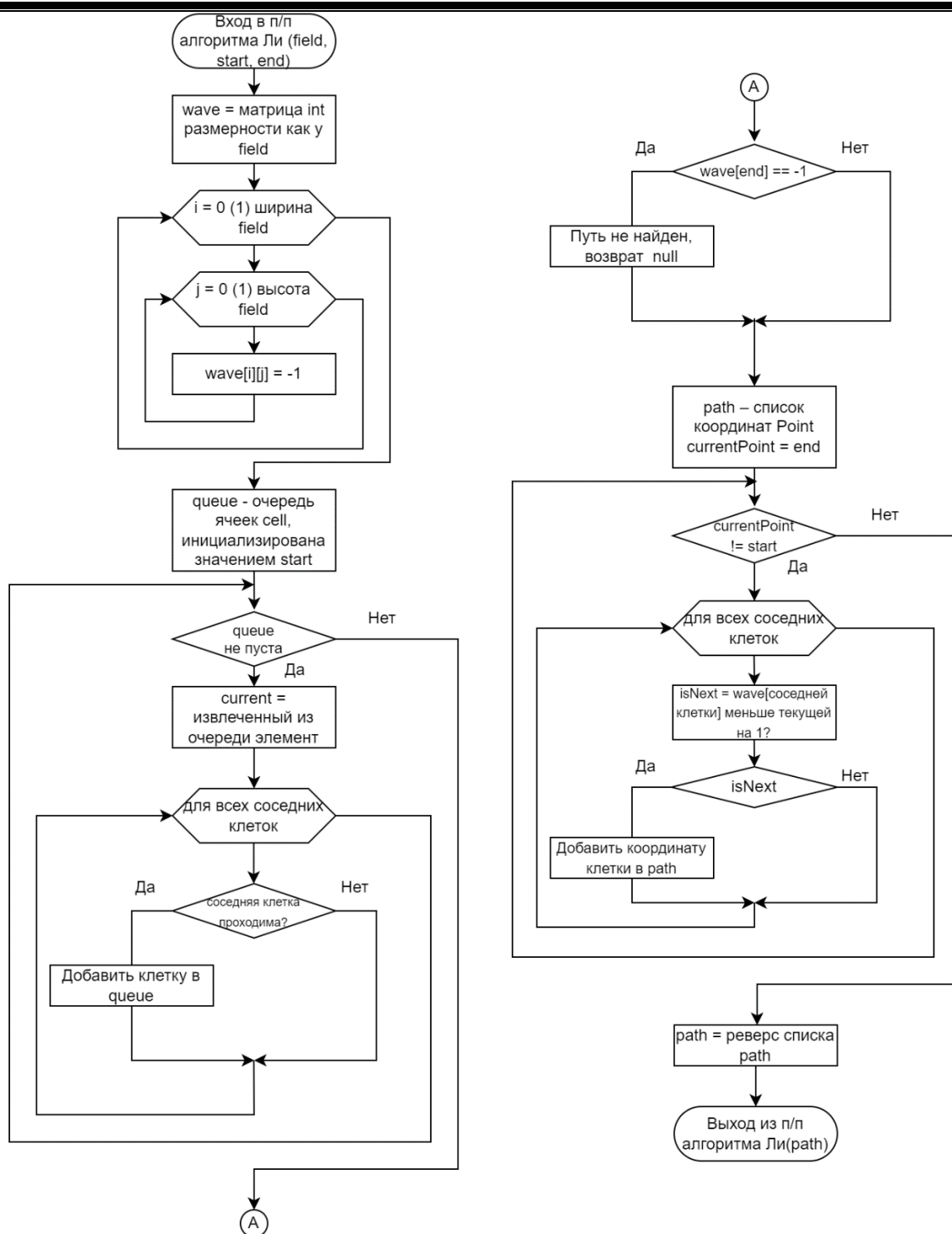


Рисунок 10 – Волновой алгоритм Ли

### 1.2.3 Алгоритм взаимодействия снаряда с объектами на игровой карте

Снаряды, выпущенные танком, могут поражать другие танки, базу и разрушимые препятствия. Некоторые неразрушимые препятствия снаряд может преодолеть, а некоторые – нет. При встрече с любым непреодолимым препятствием, а также танком или базой снаряд уничтожается, нанося урон (если снаряд попал в блок и этот блок разрушимый, то он уничтожается от первого попадания пули, а если попал в танк, то у него отнимается здоровье на величину урона, хранимого в объекте пули). Снаряды никак не взаимодействуют с друг с другом.

Алгоритм взаимодействия снаряда с объектами на игровой карте в виде диаграммы последовательности представлен на рисунке 11.

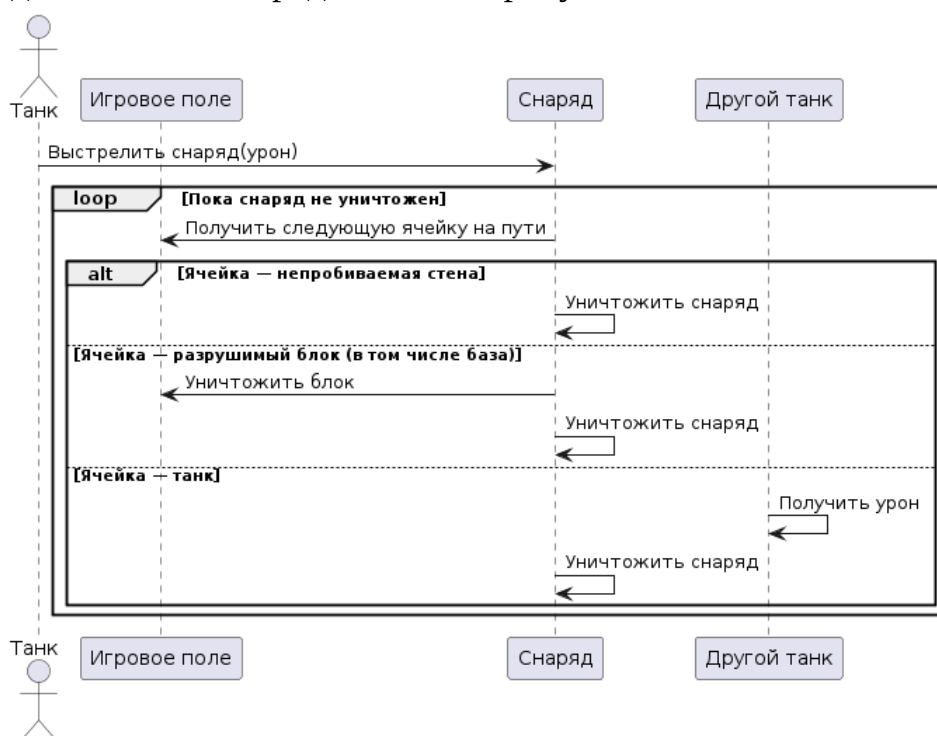


Рисунок 11 – Алгоритм взаимодействия снаряда с объектами на игровой карте

### 1.3 Разработка интерфейса программы

Эскиз главного меню приложения представлен на рисунке 12.



Рисунок 12 – Главное меню Игры

Эскиз таблицы рекордов представлен на рисунке 13.

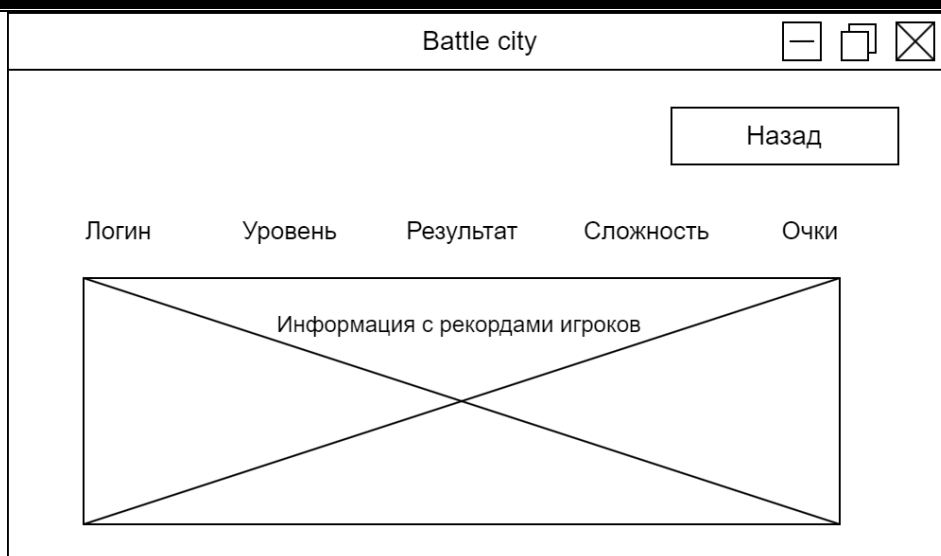


Рисунок 13 – Таблица рекордов

Эскиз справки Игры представлен на рисунке 14.

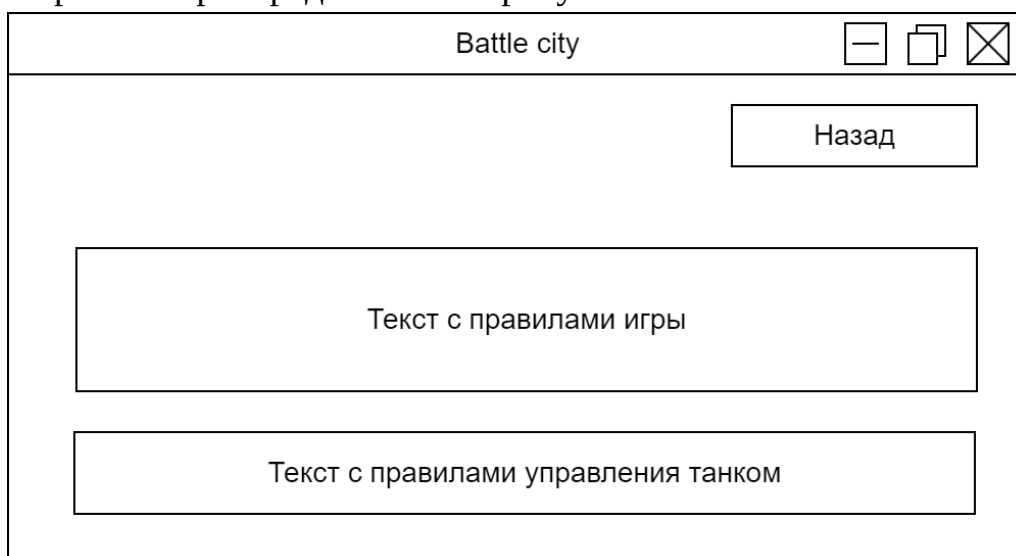


Рисунок 14 – Справка Игры

Эскиз окна с выбором уровня сложности представлен на рисунке 15.

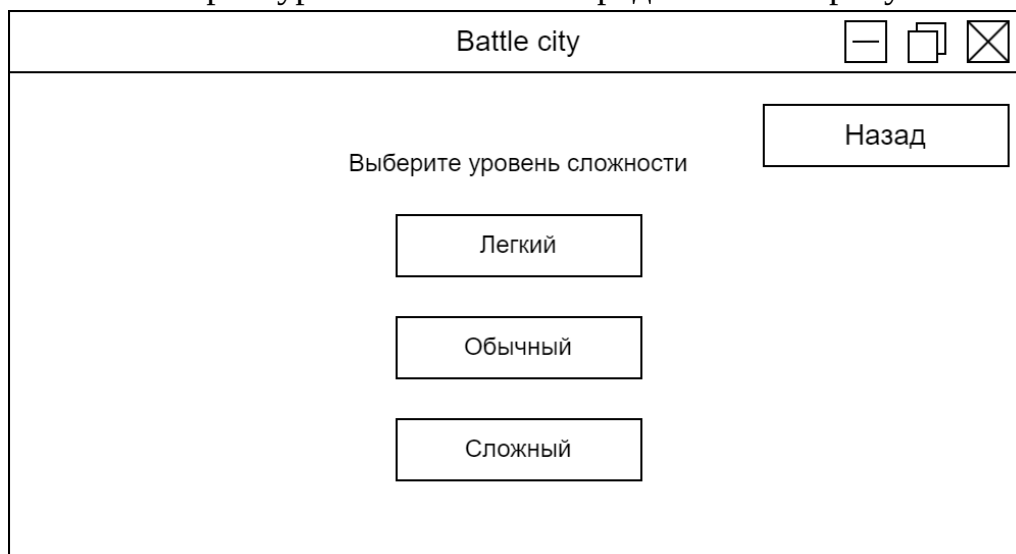


Рисунок 15 – Окно с выбором уровня сложности

Эскиз окна с игровым полем представлен на рисунке 16.

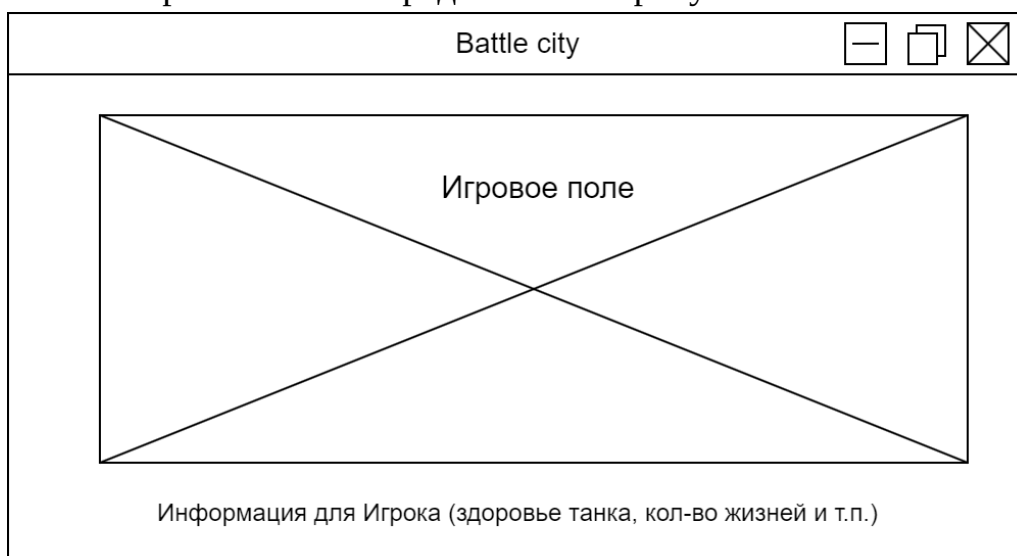


Рисунок 16 – Эскиз окна с игровым полем

Эскиз окна с паузой представлен на рисунке 17.

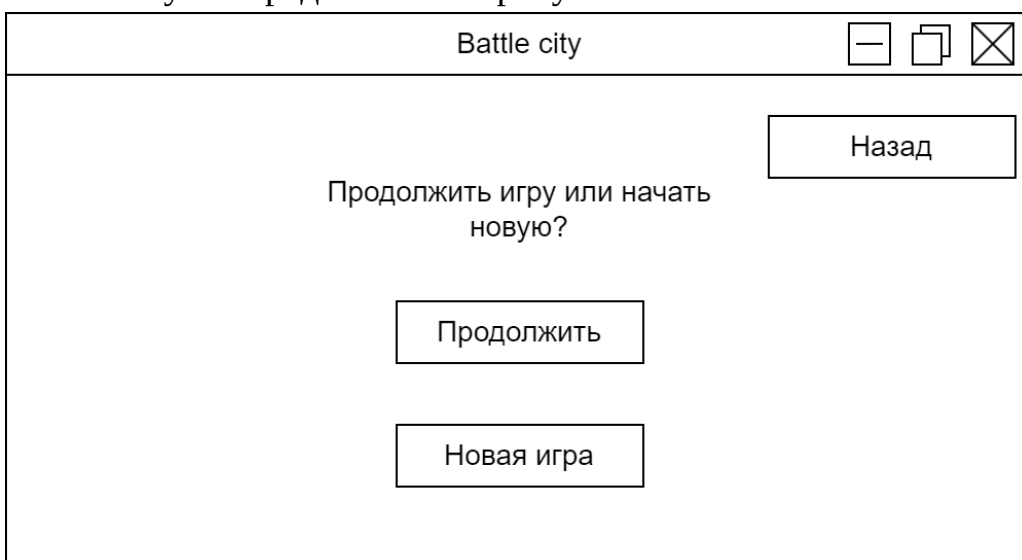
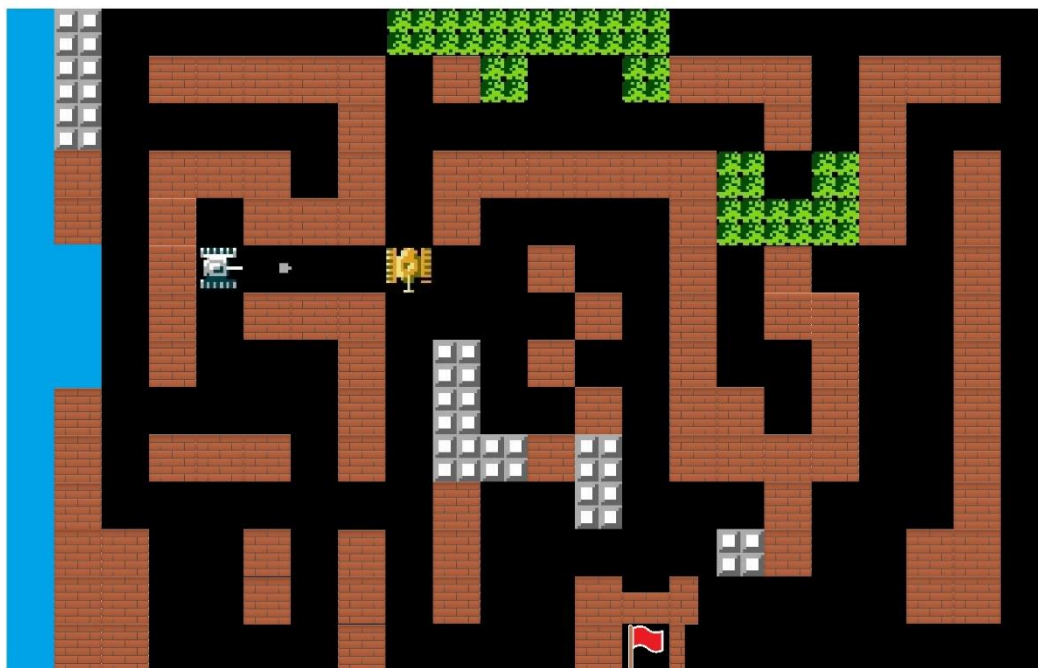


Рисунок 17 – Эскиз окна с паузой

Эскиз самого игрового поля представлен на рисунке 18.



Здоровье = 80, Жизней = 1, Уровень = First, Счет = 200

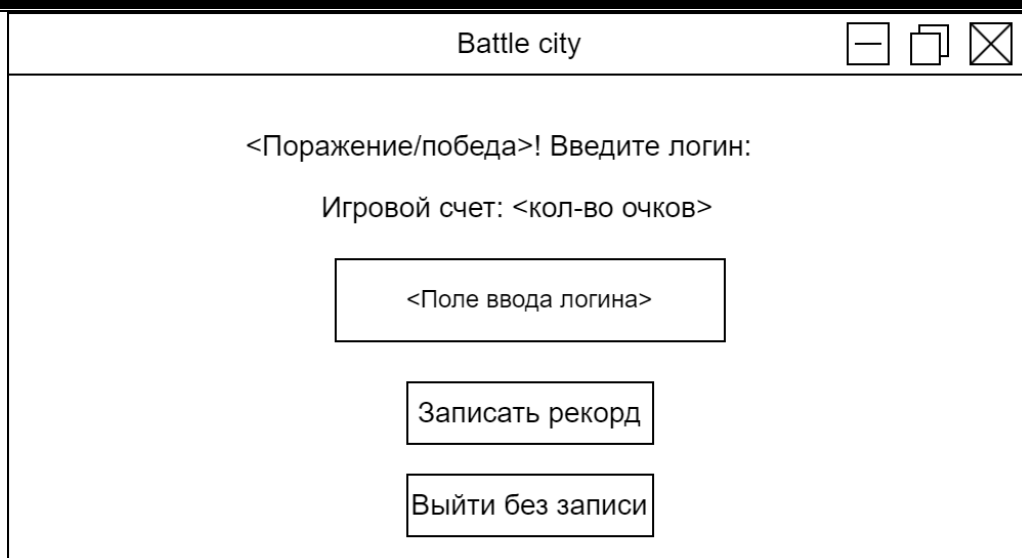
Рисунок 18 – Эскиз игрового поля

Эскизы элементов игрового поля представлены на рисунке 19.



Рисунок 19 – Эскизы элементов игрового поля

Эскиз окна с информацией о результате завершения Игры и предложением сохранить результаты представлен на рисунке 20.



The image shows a window titled "Battle city" with standard window controls (minimize, maximize, close). The main content area displays the following text and elements:

- <Поражение/победа>! Введите логин:
- Игровой счет: <кол-во очков>
- A text input field containing the placeholder text "<Поле ввода логина>".
- A button labeled "Записать рекорд".
- A button labeled "Выйти без записи".

Рисунок 20 – Окно с информацией о результате завершения Игры  
Полная совокупная диаграмма взаимодействия представлена на рисунке 21.



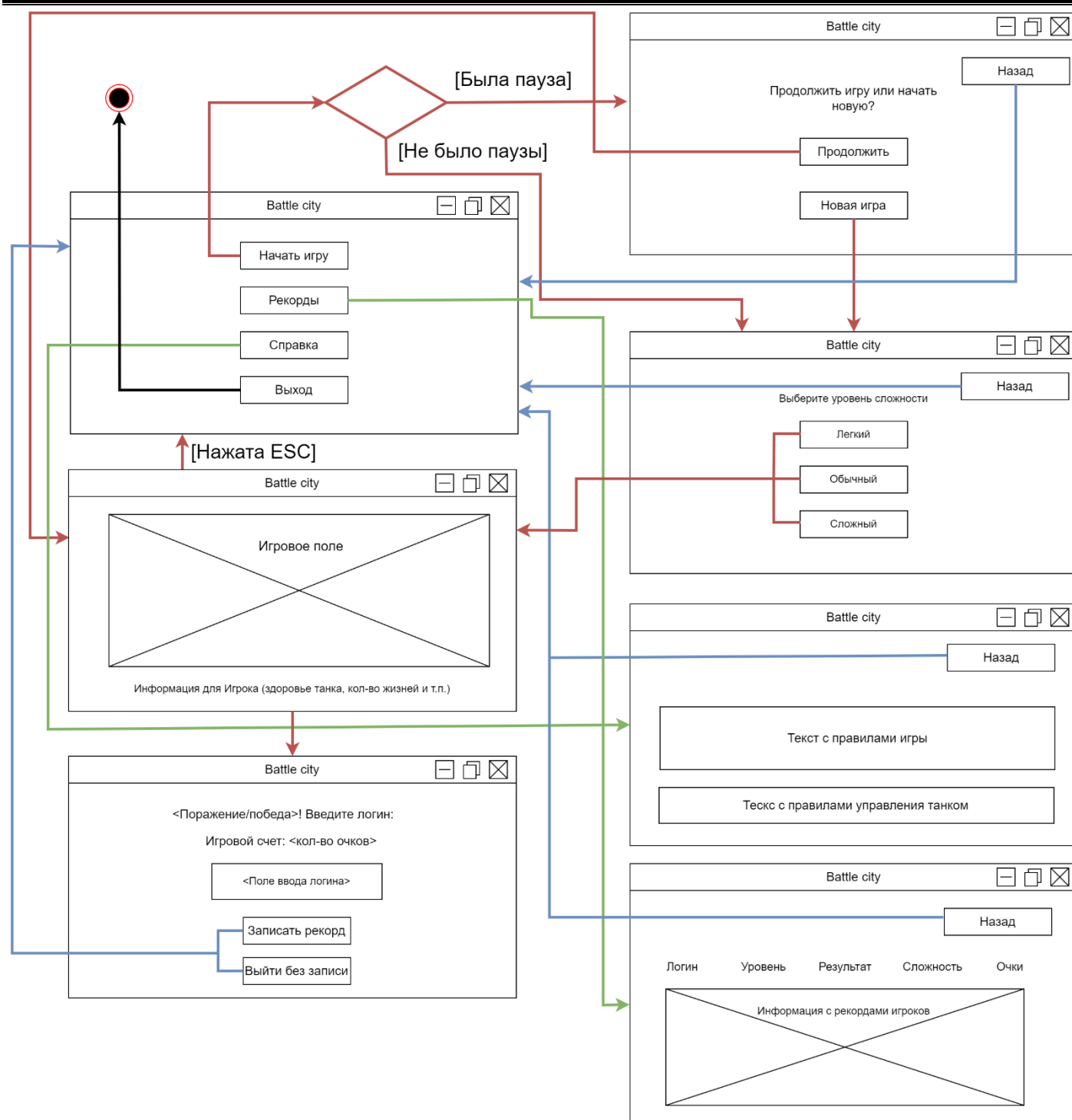


Рисунок 21 – Полная совокупная диаграмма взаимодействия для Игры

## 2 Написание программы

### 2.1.1 Описание разработанных процедур и функций

Таблица 1 - GameControllerBase

GameControllerBase(GameModelBase model, GameViewBase view, IHighScoresController highScoresController)	Инициализирует контроллер с моделью, представлением и контроллером таблицы рекордов.
StartGame()	Запускает игру, сбрасывает модель и запускает игровой цикл.

GameLoop()	Абстрактный метод для реализации игрового цикла.
GameOver()	Завершает игру, отображает результат и проверяет новый рекорд.
GetCurrentTime()	Абстрактный метод для получения текущего времени.
HandleNewHighScore(int score)	Обрабатывает новый рекорд, если он является самым высоким.

Таблица 2 - IGameController

StartGame()	Запускает игру.
GameLoop()	Выполняет игровой цикл.
GameOver()	Завершает игру.

Таблица 3 - IHighScoresController

Show()	Отображает таблицу рекордов.
AddScore(string name, int score)	Добавляет новый рекорд с указанным именем игрока и количеством очков.
IsNewHighScore(int score)	Проверяет, является ли счет новым рекордом, возвращает true, если это так.

Таблица 4 - IMenuController

Start()	Запускает отображение меню.
Stop()	Останавливает работу меню.

Таблица 5 - IRulesController

Show()	Отображает правила игры.
--------	--------------------------

Таблица 6 - MenuBase

MenuBase(IEnumerable<MenuItemBase> items)	Инициализирует новый экземпляр меню с переданным списком пунктов меню.
Items	Получает список пунктов меню.
MoveUp()	Перемещает выбор вверх по списку пунктов меню.
MoveDown()	Перемещает выбор вниз по списку пунктов меню.
ExecuteSelected()	Выполняет действие для выбранного пункта меню.

Таблица 7 - BaseObstacle

BaseObstacle(float x, float y, ObstacleTypes type)	Инициализирует новое препятствие с заданными координатами и типом.
--	--

Update(float deltaTime)	Обновляет состояние препятствия (по умолчанию ничего не делает).
IsPartOfObstacle(float y)	Проверяет, является ли данная точка частью препятствия.
SetX(float x)	Устанавливает значение координаты X для препятствия.

Таблица 8 - Bullet

Bullet(float x, float y, Direction Direction)	Инициализирует пулю с заданными координатами и направлением.
Update(float deltaTime)	Обновляет положение пули в зависимости от времени и направления.
IsPartOfObstacle(float y)	Проверяет, является ли заданная точка частью пули (по вертикали).

Таблица 9 - Enemy

Enemy(float x, float y)	Инициализирует вражеский танк с заданными координатами.
Update(float deltaTime)	Обновляет положение врага и обрабатывает его действия, включая выстрелы и движение пуль.
Shoot()	Создает пули, которые враг стреляет (вперед, вверх и вниз).
IsPartOfObstacle(float y)	Проверяет, является ли заданная точка частью врага.
Bullets	Возвращает список пуль, выпущенных врагом.

Таблица 10 - StaticObstacle

StaticObstacle(float x, ObstaclePosition position)	Инициализирует статическое препятствие с заданными координатами и позицией.
GetYPosition(ObstaclePosition position)	Возвращает координату Y для статического препятствия в зависимости от его позиции.
IsPartOfObstacle(float y)	Проверяет, является ли заданная точка частью статического препятствия в зависимости от его позиции.

Таблица 11 - GameModelBase

GameModelBase(GameStateBase gameState)	Инициализирует новый экземпляр модели игры с состоянием игры.
Jump()	Выполняет прыжок игрока.
Update(float deltaTime)	Обновляет состояние игры (положение игрока, препятствия, счет).
UpdatePlayerPosition(float deltaTime)	Обновляет позицию игрока, учитывая гравитацию и скорость.

UpdateObstacles(float deltaTime)	Обновляет положение препятствий и генерирует новые.
GenerateObstacle()	Генерирует новое препятствие в зависимости от текущего счета.
UpdateScore()	Обновляет счет игры.
CheckCollisions()	Проверяет столкновения игрока с препятствиями.
GetGameState()	Возвращает текущее состояние игры.
Reset()	Сбрасывает состояние игры к начальным значениям.
Shoot()	Выстрел игрока, создавая пулю.

Таблица 12 - GameStateBase

GameStateBase()	Инициализирует новое состояние игры и очищает список препятствий.
Reset()	Сбрасывает состояние игры к начальным значениям

Таблица 13 - HighScoreRecord

HighScoreRecord()	Инициализирует новую запись таблицы рекордов с именем игрока и количеством очков.
-------------------	---

Таблица 14 - HighScoresManager

HighScoresManager()	Инициализирует новый экземпляр менеджера таблицы рекордов.
GetScores()	Получает список рекордов.
IsNewHighScore(int score)	Проверяет, является ли счет новым рекордом.
AddScore(string name, int score)	Добавляет новый рекорд.
LoadScores()	Загружает рекорды из файла.
SaveScores(List<HighScoreRecord> scores)	Сохраняет рекорды в файл.

Таблица 15 - RulesModel

Instance	Получает единственный экземпляр модели правил.
----------	--

Таблица 16 - GameViewBase

Render(GameStateBase gameState)	Отрисовывает состояние игры
ShowGameOver(int finalScore)	Отображает экран окончания игры

Таблица 17 - HighScoresViewBase

Render(IReadOnlyList<HighScoreRecord> scores)	Отрисовывает таблицу рекордов
---	-------------------------------

FormatScoreEntry(int position, string name, int score)	Форматирует строку записи рекорда
--	-----------------------------------

Таблица 18 - MenuViewBase

Render(MenuBase menu)	Отрисовывает меню
-----------------------	-------------------

Таблица 19 - RulesViewBase

Render()	Отрисовывает правила
----------	----------------------

Таблица 20 - AddHighScoreController

AddHighScoreController(int score)	Инициализирует новый экземпляр контроллера добавления рекорда
Show()	Отображает форму добавления рекорда

Таблица 21 - GameController

GameController(IHighScoresController highScoresController)	Инициализирует новый экземпляр консольного контроллера игры
StartGame()	Запускает игру
StartInputThread()	Запускает поток обработки ввода
GameLoop()	Выполняет игровой цикл
ProcessInput()	Обрабатывает пользовательский ввод
TogglePause()	Переключает состояние паузы
RenderGame()	Отрисовка игры
GameOver()	Завершает игру
GetCurrentTime()	Получает текущее время
HandleNewHighScore(int score)	Обрабатывает новый рекорд

Таблица 22 - HighScoresController

HighScoresController()	Консольный контроллер таблицы рекордов
Show()	Отображает таблицу рекордов
AddScore(string name, int score)	Добавляет новый рекорд
IsNewHighScore(int score)	Проверяет, является ли счет новым рекордом

Таблица 23 - MenuController

MenuController()	Консольный контроллер меню
Start()	Запускает отображение меню
Stop()	Останавливает работу меню

Таблица 24 - RulesController

RulesController()	Консольный контроллер правил
Show()	Отображает правила игры

Таблица 25 - Menu

Menu()	Консольная реализация меню
--------	----------------------------

Menu(IEnumerable<MenuItem> items)	Инициализирует новый экземпляр консольного меню
-----------------------------------	---

Таблица 26 - MenuFactory

MenuFactory()	Консольная реализация фабрики меню
MenuFactory(IGameController gameController, IRulesController rulesController, IHighScoresController highScoresController)	Инициализирует новый экземпляр консольной фабрики меню
CreateMainMenu()	Создает главное меню
CreatePlayMenuItem()	Создает пункт меню "Играть"
CreateRulesMenuItem()	Создает пункт меню "Правила"
CreateHighScoresMenuItem()	Создает пункт меню "Таблица рекордов"
CreateExitMenuItem()	Создает пункт меню "Выход"

Таблица 27 - MenuItem

MenuItem()	Консольная реализация пункта меню
MenuItem(string text, Action action)	Инициализирует новый экземпляр пункта меню
Execute()	Выполняет действие пункта меню

Таблица 28 - PlayMenuItem

PlayMenuItem(Action startGame)	Реализация пункта меню "Играть"
RulesMenuItem(Action showRules)	Реализация пункта меню "Правила"
HighScoresMenuItem(Action showHighScores)	Реализация пункта меню "Таблица рекордов"
ExitMenuItem(Action exit)	Реализация пункта меню "Выход"

Таблица 29 - GameModel

GameModel()	Консольная реализация модели игры
-------------	-----------------------------------

Таблица 30 - GameState

GameState()	Консольная реализация состояния игры
-------------	--------------------------------------

Таблица 31 - AddHighScoreView

Render(int score)	Отображает форму добавления рекорда
ReadPlayerName()	Считывает имя игрока

Таблица 32 - ConsoleView

Render(GameStateBase gameState)	Отрисовывает состояние игры
ClearBuffer()	Очищает буфер консоли
DrawObstacles(GameStateBase gameState)	Отрисовывает препятствия
DrawPlayer(GameStateBase gameState)	Отрисовывает игрока
DrawScore(GameStateBase gameState)	Отрисовывает счет
ShowGameOver(int finalScore)	Отображает экран окончания игры
ResetConsole()	Сбрасывает настройки консоли
ShowPauseMenu()	Отображает меню паузы

HidePauseMenu()	Скрывает меню паузы
-----------------	---------------------

Таблица 33 - HighScoresView

Render(IReadOnlyList<HighScoreRecord> scores)	Отрисовывает таблицу рекордов
---	-------------------------------

Таблица 34 - MenuView

Render(MenuBase menu)	Отрисовывает меню
-----------------------	-------------------

Таблица 35 - RulesView

Render()	Отрисовывает правила игры
----------	---------------------------

Таблица 36 - WpfAddHighScoreController

WpfAddHighScoreController()	WPF контроллер добавления нового рекорда
WpfAddHighScoreController(int score)	Инициализирует новый экземпляр WPF контроллера добавления рекорда
Show()	Отображает форму добавления рекорда
Window_KeyDown(object sender, KeyEventArgs e)	Обрабатывает нажатие клавиш в окне

Таблица 37 - WpfGameController

WpfGameController(WpfGameView view)	Инициализирует новый экземпляр WPF контроллера игры
StartGame()	Запускает игру
GameOver()	Завершает игру
Window_KeyDown(object sender, KeyEventArgs e)	Обрабатывает нажатие клавиш в окне
TogglePause()	Переключает паузу игры
ClosePauseMenu()	Закрывает меню паузы
GameLoop_Update(object sender, EventArgs e)	Обновляет состояние игры при отрисовке кадра
GetCurrentTime()	Получает текущее время
GameLoop()	Выполняет игровой цикл
HandleNewHighScore(int score)	Обрабатывает новый рекорд
ExitToMenu()	Выход в меню
Stop()	Останавливает игру
GameOverKeyDown(object sender, KeyEventArgs e)	Обрабатывает нажатие клавиш по завершении игры

Таблица 38 - WpfHighScoresController

WpfHighScoresController(WpfHighScoresView view)	Инициализирует новый экземпляр WPF контроллера таблицы рекордов
Show()	Отображает таблицу рекордов
Window_KeyDown(object sender, KeyEventArgs e)	Обрабатывает нажатие клавиш в окне
AddScore(string name, int score)	Добавляет новый рекорд
IsNewHighScore(int score)	Проверяет, является ли счет новым рекордом

Stop()	Останавливает контроллер
--------	--------------------------

Таблица 39 - WpfMenuController

WpfMenuController()	Инициализирует новый экземпляр WPF контроллера меню
Start()	Запускает отображение меню
Stop()	Останавливает работу меню
Window_KeyDown(object sender, KeyEventArgs e)	Обрабатывает нажатие клавиш в окне

Таблица 40 - WpfRulesController

WpfRulesController()	Инициализирует новый экземпляр WPF контроллера правил
Show()	Отображает правила игры
Window_KeyDown(object sender, KeyEventArgs e)	Обрабатывает нажатие клавиш в окне
Stop()	Останавливает работу контроллера

Таблица 41 - WpfMenu

WpfMenu(IEnumerable<WpfMenuItem> items)	Инициализирует новый экземпляр WPF меню
---	---

Таблица 42 - WpfMenuFactory

WpfMenuFactory(IGameController gameController, IRulesController rulesController, IHighScoresController highScoresController)	Инициализирует новый экземпляр WPF фабрики меню
CreateMainMenu()	Создает главное меню
CreatePlayMenuItem()	Создает пункт меню "Играть"
CreateRulesMenuItem()	Создает пункт меню "Правила"
CreateHighScoresMenuItem()	Создает пункт меню "Таблица рекордов"
CreateExitMenuItem()	Создает пункт меню "Выход"

Таблица 43 - WpfMenuItem

WpfMenuItem(string text, Action action)	Инициализирует новый экземпляр WPF пункта меню
Execute()	Выполняет действие пункта меню

Таблица 44 - WpfPlayMenuItem

WpfPlayMenuItem(Action startGame)	Инициализирует новый экземпляр пункта меню "Играть"
WpfRulesMenuItem	WPF реализация пункта меню "Правила"
WpfRulesMenuItem(Action showRules)	Инициализирует новый экземпляр пункта меню "Правила"
WpfHighScoresMenuItem	WPF реализация пункта меню "Таблица рекордов"



WpfHighScoresMenuItem(Action showHighScores)	Инициализирует новый экземпляр пункта меню "Таблица рекордов"
WpfExitMenuItem	WPF реализация пункта меню "Выход"
WpfExitMenuItem(Action exit)	Инициализирует новый экземпляр пункта меню "Выход"

Таблица 45 - WpfGameModel

WpfGameModel()	Инициализирует новый экземпляр WPF модели игры
----------------	--

Таблица 46 - WpfGameState

WpfGameState()	Пустой конструктор, наследующий от GameStateBase
----------------	--

Таблица 47 - MainScreen

MainScreen()	Конструктор, инициализирует окно и панель для размещения элементов
GetInstance()	Метод для получения единственного экземпляра главного окна

Таблица 48 - WpfAddHighScoreView

WpfAddHighScoreView()	Конструктор, инициализирует элементы интерфейса
Show()	Метод для отображения формы с вводом имени и результатом
GetPlayerName()	Метод для получения введенного имени игрока

Таблица 49 - WpfRulesView

WpfRulesView()	Конструктор, инициализирует элементы интерфейса
Show()	Отображает окно правил
Render()	Отрисовывает правила игры

Таблица 50 - WpfMenuView

WpfMenuView()	Конструктор, инициализирует элементы интерфейса
Show()	Отображает меню
Render(MenuBase menu)	Отрисовывает меню

Таблица 51 - WpfHighScoresView

WpfHighScoresView()	Конструктор, инициализирует элементы интерфейса
Show()	Отображает таблицу рекордов

Render(IReadOnlyList<HighScoreRecord> scores)	Отрисовывает таблицу рекордов
---	-------------------------------

Таблица 52 - WpfGameView

WpfGameView()	Конструктор, инициализирует элементы интерфейса
Show()	Отображает игровое окно
Render(GameStateBase gameState)	Отрисовывает текущее состояние игры
CreateObstacle(BaseObstacle obstacle)	Создает препятствие
CreateDoubleObstacle(bool hasTop, bool hasBottom)	Создает двойное препятствие
ShowGameOver(int finalScore)	Отображает экран окончания игры
LoadImage(string path)	Загружает изображение по указанному пути
ShowPauseMenu(bool show)	Отображает или скрывает паузу

## 2.2 Разработка программы

Разработка программы в соответствии с Заданием осуществлялась по следующим этапам:

1. Анализ предметной области
2. Составление требований и документации
3. Проектирование программного обеспечения
4. Написание кода
5. Тестирование
6. Исправление ошибок
7. Подготовка документации

### 2.2.1 Описание классов, перечислений и интерфейсов проекта

Пространство имен	Название класса / интерфейса / перечисления	Назначение
Base.Controllers	IGameController	Интерфейс контроллера игры
	IHighScoresController	Интерфейс контроллера таблицы рекордов
	IMenuController	Интерфейс контроллера меню
	IRulesController	Интерфейс контроллера правил
	GameControllerBase	Базовый класс контроллера игры
Base.Models.Menu	MenuItemBase	Базовый класс пункта меню
	MenuBase	Базовый класс меню

	MenuFactoryBase	Базовый класс фабрики меню
Base.Models.Obstacle	BaseObstacle	Базовый класс блока
	StaticObstacle	Класс неподвижного препятствия
	MovingObstacle	Класс движущегося препятствия
	Tank	Класс вражеского танка
	Player	Класс игрока
	Bullet	Класс патрона
	PlayerBullet	Класс патрона игрока
Base.Models	GameModelBase	Базовый класс модели игры
	GameStateBase	Базовый класс состояния игры
	HighScoreRecord	Запись таблицы рекордов
	HighScoresManager	Менеджер таблицы рекордов
	RulesModel	Модель правил игры
Base.Views	GameViewBase	Базовый класс представления игры
	HighScoresViewBase	Базовый класс представления таблицы рекордов
	MenuViewBase	Базовый класс представления меню
	RulesViewBase	Базовый класс представления правил
Console.ConsoleControllers	AddHighScoreController	Контроллер добавления нового рекорда
	GameController	Консольный контроллер игры
	HighScoresController	Консольный контроллер таблицы рекордов

	MenuController	Консольный контроллер меню
	RulesController	Консольный контроллер правил
Console.ConsoleModels.Menu	MenuItem	Консольная реализация пункта меню
	PlayMenuItem	Реализация пункта меню "Играть"
	RulesMenuItem	Реализация пункта меню "Правила"
	HighScoresMenuItem	Реализация пункта меню "Таблица рекордов"
	ExitMenuItem	Реализация пункта меню "Выход"
	Menu	Консольная реализация меню
	MenuFactory	Консольная реализация фабрики меню
Console.ConsoleModels	GameModel	Консольная реализация модели игры
	GameState	Консольная реализация состояния игры
Console.ConsoleViews	AddHighScoreView	Консольное представление добавления нового рекорда
	ConsoleView	Консольное представление игры
	HighScoresView	Консольное представление таблицы рекордов
	MenuView	Консольное представление меню
	RulesView	Консольное представление правил игры
Wpf.WpfControllers	WpfAddHighScoreController	WPF контроллер добавления нового рекорда

	WpfGameController	WPF контроллер игры
	WpfHighScoresController	WPF контроллер таблицы рекордов
	WpfMenuController	WPF контроллер меню
	WpfRulesController	WPF контроллер правил
Wpf.WpfModels.Menu	WpfMenuItem	WPF реализация пункта меню
	WpfPlayMenuItem	WPF реализация пункта меню "Играть"
	WpfRulesMenuItem	WPF реализация пункта меню "Правила"
	WpfHighScoresMenuItem	WPF реализация пункта меню "Таблица рекордов"
	WpfExitMenuItem	WPF реализация пункта меню "Выход"
	WpfMenu	WPF реализация меню
	WpfMenuFactory	WPF реализация фабрики меню
Wpf.WpfModels	WpfGameModel	WPF реализация модели игры
	WpfGameState	WPF реализация состояния игры
Wpf.WpfViews	MainScreen	Главное окно приложения
	WpfAddHighScoreView	WPF представление добавления нового рекорда
	WpfGameView	WPF представление игры
	WpfHighScoresView	WPF представление таблицы рекордов
	WpfMenuView	WPF представление меню

	WpfRulesView	WPF представление правил игры
--	--------------	-------------------------------------

## 2.3 Описание шаблонов проектирования, которые использовались при написании программы

В данной работе использовались три шаблона проектирования: Модель–Представление–Контроллер, Одиночка, Абстрактная фабрика, которые будут более подробно рассмотрены в текущем разделе.

### 2.3.1 Модель–Представление–Контроллер, Model–View–Controller, MVC

MVC — это архитектурный шаблон проектирования, разделяющий приложение на три основных компонента:

- модель (данные приложения и его логика),
- представление (отображение данных, полученных от модели),
- контроллер (управление взаимодействием между моделью и представлением, ввод данных от пользователя).

Так как каждая часть шаблона реализует строго определенный функционал, то с каждой из них можно работать независимо.

Данный шаблон подробно рассматривался в пункте 1.1.1 документа.

### 2.3.2 Одиночка (Singleton)

Одиночка — это порождающий паттерн, гарантирующий создание только одного экземпляра класса и предоставляющий единую точку доступа к этому объекту из любого места программы.

В данном проекте шаблон Одиночка используется в классе настроек и текущего состояния Игры в классе GameCurrentStateAndSettingsSingleton. У данного класса есть статическое поле `_instance` типа класса, которое инициализируется при первом вызове статического метода `GetInstance()`, возвращающего ссылку на `_instance`. Для обеспечения создания только одного объекта данного класса его конструктор сделан приватным.

### 2.3.3 Абстрактная фабрика (Abstract Factory)

Абстрактная фабрика — это порождающий шаблон проектирования, предоставляющий интерфейс для создания семейств связанных объектов без указания их конкретных классов. Шаблон реализуется созданием абстрактного класса `Factory`, который представляет собой интерфейс для создания компонентов системы, который реализуют дочерние классы.

В данном проекте абстрактная фабрика представлена интерфейсом `ICellCreatorFabric` для создания конкретных блоков на уровне при его загрузке из файла. В данном интерфейсе находится метод `Cell CreateCell()`, возвращающий ячейку

игрового поля с данным блоком. У данного интерфейса есть 5 реализаций (WallCellCreator, IndestructibleWallCellCreator, BushCellCreator, WaterCellCreator, EmptyCellCreator), реализующих метод CreateCell() и возвращающих объекты классов-наследников Cell. Объекты фабрик создаются в LevelGenerator, где хранятся в словаре с ключом двоичного представления блока. При считывании файла уровней по ключу достаётся конкретная фабрика, у которой вызывается метод CreateCell().

## **2.4 Описание методов рефакторинга, которые использовались при оптимизации исходного кода программы**

Рефакторинг — это процесс улучшения внутренней структуры кода без изменения его внешнего поведения. Основная цель — сделать код более читаемым, поддерживаемым и расширяемым.

### **2.4.1 Разбиение метода**

Большой по объёму метод разбивается на несколько более коротких, каждый из которых выполняет одну конкретную задачу, что улучшает читаемость кода и облегчает его тестирование.

В данной работе был разбит метод UpdateAI танка, определяющий его поведение на конкретном игровом тике. Данный метод был одним из самых объёмных в проекте, его код до рефакторинга представлен ниже:

```
/// <summary>
/// Обновить поведение танка
/// </summary>
/// <param name="parGameModel">Модель игры</param>
public void UpdateAI(GameModel parGameModel)
{
    if (parGameModel.Player.PlayerTank == null) return;

    var copyPlayerPositionCells = new Point(parGameModel.Player.PlayerTank.Point);
    copyPlayerPositionCells.X /= parGameModel.Field.CellSize;
    copyPlayerPositionCells.Y /= parGameModel.Field.CellSize;

    var copyTankPositionCells = new Point(this.Point);
    copyTankPositionCells.X /= parGameModel.Field.CellSize;
    copyTankPositionCells.Y /= parGameModel.Field.CellSize;

    // Проверка на прямую видимость
    List<Point> targetPoints = new();
    targetPoints.Add(parGameModel.Player.PlayerTank.Point);
    targetPoints.AddRange(parGameModel.StaffBlocksCoordinates.Values.ToList());
    for (int i = 0; i < targetPoints.Count; i++)
    {
        Point elTargetPoint = targetPoints[i];

        // Прямая видимость
        double closeCoeff = i != 0 ? 0.2 : 0.5;
        int dx = Math.Sign(elTargetPoint.X - this.Point.X) *
```

```
(Math.Abs(elTargetPoint.X - this.Point.X) /
parGameModel.Field.CellSize > closeCoeff ? 1 : 0);
    int dy = Math.Sign(elTargetPoint.Y - this.Point.Y) *
        (Math.Abs(elTargetPoint.Y - this.Point.Y) /
parGameModel.Field.CellSize > closeCoeff ? 1 : 0);

    if (dx != 0 || dy != 0)
    {
        float currentX = this.Point.X;
        float currentY = this.Point.Y;

        bool hasLineOfSight = true;
        while (Math.Abs(currentX - elTargetPoint.X) > parGameModel.Field.CellSize
||
            Math.Abs(currentY - elTargetPoint.Y) > parGameModel.Field.CellSize)
        {
            if (currentX < 0 || currentY < 0
                || currentY >= parGameModel.Field.GetHeightInPixels() || currentX
>= parGameModel.Field.GetWidthInPixels()
                || !(parGameModel.Field.GetCellByCoordinates((int)currentX,
(int)currentY).StaticObject?.IsPassableByBullet ?? true))
            {
                hasLineOfSight = false;
                break;
            }

            if (currentX != elTargetPoint.X) currentX += dx;
            if (currentY != elTargetPoint.Y) currentY += dy;
        }

        if (hasLineOfSight)
        {
            var directionToTarget = GetDirectionToPoint(elTargetPoint);
            if (Direction != directionToTarget && (DateTime.Now -
_lastDirectionChangeTime >= _directionChangeCooldown))
            {
                _lastDirectionChangeTime = DateTime.Now;
                Direction = directionToTarget; // Поворот танка к цели
                parGameModel.OnTankDirectionUpdated(this);
                parGameModel.Fire(this); // Стрельба
                return;
            }
            parGameModel.Fire(this); // Стрельба
            if (i == 0 || _random.Next(4) != 0)
            {
                return;
            }
        }
    }
}

// Вычисление пути
var field = parGameModel.Field;
var width = field.GetWidthInCells();
var height = field.GetHeightInCells();
var wave = new int[width, height];
```



```
for (int x = 0; x < width; x++)
    for (int y = 0; y < height; y++)
        wave[x, y] = -1;

var queue = new Queue<Point>();
queue.Enqueue(copyTankPositionCells);
wave[copyTankPositionCells.X, copyTankPositionCells.Y] = 0;

var directions = new[]
{
    new Point(-1, 0), new Point(1, 0),
    new Point(0, -1), new Point(0, 1)
};

while (queue.Count > 0)
{
    var current = queue.Dequeue();
    foreach (var elDir in directions)
    {
        var next = new Point(current.X + elDir.X, current.Y + elDir.Y);
        if (IsValidCellForTank(field, next, Width, Height) && wave[next.X, next.Y]
== -1)
        {
            wave[next.X, next.Y] = wave[current.X, current.Y] + 1;
            queue.Enqueue(next);

            if (next.X == copyPlayerPositionCells.X && next.Y ==
copyPlayerPositionCells.Y)
            {
                queue.Clear();
                break;
            }
        }
    }
}

// Построение пути
var path = new List<Point>();
if (wave[copyPlayerPositionCells.X, copyPlayerPositionCells.Y] != -1)
{
    var currentPoint = copyPlayerPositionCells;
    while (!IsCellsPointEquals(currentPoint, copyTankPositionCells))
    {
        path.Add(currentPoint);
        foreach (var elDir in directions)
        {
            var next = new Point(currentPoint.X - elDir.X, currentPoint.Y -
elDir.Y);

            if (next.X >= 0 && next.X < width &&
                next.Y >= 0 && next.Y < height &&
                wave[next.X, next.Y] == wave[currentPoint.X, currentPoint.Y] - 1)
            {
                currentPoint = next;
                break;
            }
        }
    }
}
```

```
}

if (path.Count > 1)
{
    var nextPointInCells = path[1];
    var nextPointInPixels = new Point(nextPointInCells.X *
parGameModel.Field.CellSize + _random.Next(-12, 12),
    nextPointInCells.Y * parGameModel.Field.CellSize + _random.Next(-12, 13));
    ChangeDirection(nextPointInPixels);
    parGameModel.OnTankDirectionUpdated(this);
    IsMoving = true;
}
else
{
    IsMoving = false;
}
}
```

В результате рефакторинга был выделен ряд методов:

- ChangeDirection с логикой смены направления танка,
- HasDirectLineOfSight для проверки видимости цели,
- FindPathWithLee для поиска кратчайшего пути методом Ли.

Реализация данных методов дана в приложении, а в тексте записки приведена реализация обновленного метода UpdateAI, который стал заметно короче:

```
/// <summary>
/// Обновить поведение танка
/// </summary>
/// <param name="parGameModel">Модель игры</param>
public void UpdateAI(GameModel parGameModel)
{
    if (parGameModel.Player.PlayerTank == null) return;

    var copyPlayerPositionCells = new Point(parGameModel.Player.PlayerTank.Point);
    copyPlayerPositionCells.X /= (parGameModel.Field.CellSize);
    copyPlayerPositionCells.Y /= (parGameModel.Field.CellSize);
    var copyTankPositionCells = new Point(this.Point);
    copyTankPositionCells.X /= (parGameModel.Field.CellSize);
    copyTankPositionCells.Y /= (parGameModel.Field.CellSize);

    // Проверка на прямую видимость
    List<Point> targetPoints = new();
    targetPoints.Add(parGameModel.Player.PlayerTank.Point);
    targetPoints.AddRange(parGameModel.StaffBlocksCoordinates.Values.ToList());
    for (int i = 0; i < targetPoints.Count; i++)
    {
        Point elTargetPoint = targetPoints[i];
        if (HasDirectLineOfSight(parGameModel.Field, this.Point, elTargetPoint, i !=
0))
        {
            var directionToTarget = GetDirectionToPoint(elTargetPoint);
            if (Direction != directionToTarget && (DateTime.Now -
_lastDirectionChangeTime >= _directionChangeCooldown))
            {
                _lastDirectionChangeTime = DateTime.Now;
```

```
        Direction = directionToTarget; // Поворот танка к цели
        parGameModel.OnTankDirectionUpdated(this);
        parGameModel.Fire(this); // Стрельба
        return;
    }
    parGameModel.Fire(this); // Стрельба
    if (i == 0 || _random.Next(4) != 0)
    {
        return;
    }
}

var field = parGameModel.Field;

// Вычисление пути с помощью алгоритма Ли
var path = FindPathWithLee(field, copyTankPositionCells, copyPlayerPositionCells,
this.Width, this.Height);

if (path != null && path.Count > 1)
{
    var nextPointInCells = path[1];
    var nextPointInPixels = new Point(nextPointInCells.X *
parGameModel.Field.CellSize + _random.Next(-12, 12), nextPointInCells.Y *
parGameModel.Field.CellSize + _random.Next(-12, 13));
    ChangeDirection(nextPointInPixels);
    parGameModel.OnTankDirectionUpdated(this);
    IsMoving = true;
}
else
{
    IsMoving = false;
}
}
```

#### 2.4.2 Удаление дубликатов кода

Повторяющийся код, выполняющий одинаковые или похожие задачи, выделяется в отдельный метод.

Данный метод рефакторинга был применен для метода `IsColliding`, имеющего две перегрузки для проверки коллизий танков с танками и танков с пулями, что представлено ниже:

```
/// <summary>
/// Проверка коллизий танков друг с другом
/// </summary>
/// <param name="parTank">Проверяемый танк номер 1</param>
/// <param name="parNewPosition">Проверяемая позиция</param>
/// <param name="parOtherTank">Проверяемый танк номер 2</param>
/// <returns>Истина, если танки сталкиваются</returns>
private bool IsColliding(AbstractTank parTank, Point parNewPosition, AbstractTank
parOtherTank)
{
    int tankWidth = (int)parTank.Width;
    int tankHeight = (int)parTank.Height;
    int otherTankWidth = (int)parOtherTank.Width;
    int otherTankHeight = (int)parOtherTank.Height;
```

```
var tankBounds = new Rectangle((int)parNewPosition.X - tankWidth / 2,
    (int)parNewPosition.Y - tankHeight / 2, tankWidth, tankHeight);
var otherTankBounds = new Rectangle((int)parOtherTank.Point.X - otherTankWidth / 2,
    (int)parOtherTank.Point.Y - otherTankHeight / 2, otherTankWidth,
otherTankHeight);
return tankBounds.Intersects(otherTankBounds);
}

/// <summary>
/// Проверка коллизий пуля с танками
/// </summary>
/// <param name="parBullet">Проверяемая пуля</param>
/// <param name="parNewPosition">Проверяемая позиция</param>
/// <param name="parOtherTank">Проверяемый танк номер 2</param>
/// <returns>Истина, если снаряд попадает в танк</returns>
private bool IsColliding(Bullet parBullet, Point parNewPosition, AbstractTank
parOtherTank)
{
    if (parBullet.ParentTankId == parOtherTank.Id)
    {
        return false;
    }
    int tankWidth = (int)parOtherTank.Width;
    int tankHeight = (int)parOtherTank.Height;
    int bulletWidth = (int)parBullet.Width;
    int bulletHeight = (int)parBullet.Height;

    var bulletBounds = new Rectangle((int)parNewPosition.X - bulletWidth / 2,
        (int)parNewPosition.Y - bulletHeight / 2, bulletWidth, bulletHeight);
    var tankBounds = new Rectangle((int)parOtherTank.Point.X - tankWidth / 2,
        (int)parOtherTank.Point.Y - tankHeight / 2, tankWidth, tankHeight);
    return bulletBounds.Intersects(tankBounds);
}
```

Поскольку и танки, и пули наследуются от общего класса, то два данных метода были объединены в один общий:

```
/// <summary>
/// Проверка коллизий танков с танками и пуля с танками
/// </summary>
/// <param name="parDynamicObject">Проверяемый танк или пуля</param>
/// <param name="parNewPosition">Проверяемая позиция</param>
/// <param name="parOtherTank">Другой проверяемый танк</param>
/// <returns>Истина, если объекты сталкиваются</returns>
private bool IsColliding(AbstractDynamicObject parDynamicObject, Point parNewPosition,
AbstractTank parOtherTank)
{
    // Обработка частного случая для пули
    if (parDynamicObject is Bullet && ((Bullet)parDynamicObject).ParentTankId ==
parOtherTank.Id)
    {
        return false;
    }
    int tankWidth = (int)parDynamicObject.Width;
    int tankHeight = (int)parDynamicObject.Height;
    int otherTankWidth = (int)parOtherTank.Width;
    int otherTankHeight = (int)parOtherTank.Height;
    var tankBounds = new Rectangle((int)parNewPosition.X - tankWidth / 2,
        (int)parNewPosition.Y - tankHeight / 2, tankWidth, tankHeight);
```

```
var otherTankBounds = new Rectangle((int)parOtherTank.Point.X - otherTankWidth / 2,
    (int)parOtherTank.Point.Y - otherTankHeight / 2, otherTankWidth,
otherTankHeight);
return tankBounds.IntersectsWith(otherTankBounds);
}
```

### 2.4.3 Замена магических чисел константами

При проведении данного метода рефакторинга магические числа (смысл которых сходу не понятен при чтении исходного кода) заменяются константами с осмысленными именами, что значительно упрощает понимание.

В реализованном проекте такие константы как 50, 40 (ширина и высота игрового поля в блоках), 16 (ширина ячейки в пикселях) были вынесены из непосредственного использования в методах в константы:

```
/// <summary>
/// Класс для создания уровней
/// </summary>
public class LevelGenerator
{
    /// <summary>
    /// Ширина поля в клетках
    /// </summary>
    private const int FIELD_WIDTH = 50;

    /// <summary>
    /// Высота поля в клетках
    /// </summary>
    private const int FIELD_HEIGHT = 40;

    /// <summary>
    /// Сторона клетки в пикселях
    /// </summary>
    private const int CELL_SIZE = 16;

    // Дальнейший код не приводится для экономии места
}
```

### 2.4.4 Удаление мертвого кода

При проведении данного метода устаревший код, который уже не используется, удаляется, что позволяет упростить понимание исходного кода и предотвратить путаницу.

В данной программе в классе ячеек Cell сначала хранилась информация в том числе и для динамических объектов, располагающихся в данной ячейке. Затем было решено хранить и обрабатывать эти сведения в классе игрового поля, после чего следующие поля и свойства из класса Cell были удалены:

```
/// <summary>
/// Ячейка игрового поля
/// </summary>
public class Cell
{
    /// <summary>
    /// Динамический объект, связанный с ячейкой
```

```

/// </summary>
AbstractDynamicObject? _dynamicObject;

/// <summary>
/// Статический объект, связанный с ячейкой
/// </summary>
public AbstractStaticObject? StaticObject
{
    get
    {
        return _staticObject;
    }
    set
    {
        _staticObject = value;
    }
}

// Другие поля и методы
}

```

## 2.5 Разработка тестов

Тесты были разработаны для класса танков.

### 2.5.1 Test Cases

№	Название теста	Описание теста	Действия	Ожидаемый результат
1	DefaultPropertiesAreSetCorrectly	Проверяет факт создания танка с валидными параметрами по умолчанию	Создать объект tank типа GeneralTank и сравнить его характеристики с эталонными	Скорость, скорострельность, наносимый урон и здоровье танка равны эталонным константным параметрам
2	ApplyDamageReducesHealthCorrectly	Проверяет факт нанесения урона танку снарядом	Создать объект tank типа GeneralTank, bullet типа Bullet с уроном в 20 единиц, вызвать у танка tank метод ApplyDamage(bullet) для применения урона пули к танку, сравнить текущее здоровье танка с эталонным параметром минус 20	У танка отнялось 20 единиц здоровья, так как эталонный параметр здоровья танка равен 100, то текущее значение здоровья танка Health должно быть равно 80
3	ApplyDamageByParentTankShouldNotBe	Проверяет, что снаряд не может поразить родительский танк	Создать объект tank типа GeneralTank, bullet типа Bullet с уроном в 20 единиц и id родительского танка, равным tank.Id, вызвать у танка tank метод ApplyDamage(bullet) для применения урона пули к танку, сравнить текущее здоровье танка с эталонным	У танка осталось исходное количество здоровья Health, так как коллизии снарядов с родительскими танками не проверяются
4	ApplyDamageDoesNotReduceHealthBelowZero	Проверяет, что здоровье танка после нанесения урона не может стать меньше 0	Создать объект tank типа GeneralTank, присвоить свойству Health значение 5, bullet типа Bullet с уроном в 10 единиц, вызвать у танка tank	Здоровье танка Health равно 0, поскольку даже при уроне, большим текущего значения

			метод ApplyDamage(bullet) для применения урона пули к танку, сравнить текущее здоровье танка с 0	здоровья, здоровье танка не может стать отрицательным
5	FireCreatesBulletWhenCooldownAllows	Проверка на то, что танк может выпустить хотя бы один снаряд без ограничений на максимальную скорострельность	Создать объект tank типа GeneralTank, вызвать у него метод Fire(), и проверить, что он вернул не null	Объект bullet не равен null, так как первый выстрел танку разрешается сделать без предварительных ограничений
6	FireShouldNotCreateBulletWhenCooldown	Проверка на ограничение скорострельности танка (стрельба не чаще чем один раз в FireCooldown секунд)	Создать объект tank типа GeneralTank, вызвать у него метод Fire(), затем еще раз вызвать метод Fire(), занести возвращенное значение в переменную bulletSecond и проверить, что bulletSecond равна null	Объект bulletSecond равен null, так как попытка второго выстрела была произведена менее чем через FireCooldown секунд
7	FireShouldCreateBulletAfterCooldown	Проверка на то, что спустя FireCooldown времени танк может выстрелить снова	Создать объект tank типа GeneralTank, вызвать у него метод Fire(), занести возвращенное значение в переменную bulletFirst, у которой вызвать метод OnObjectDestroyed() для уничтожения первой пули, затем подождать не менее tank.FireCooldown секунд, затем еще раз вызвать метод Fire(), занести возвращенное значение в переменную bulletSecond и проверить, что bulletSecond не равна null	Объект bulletSecond не равен null, так как попытка второго выстрела была произведена не менее чем через FireCooldown секунд
8	DestroyedTankCannotFire	Проверяет, что танк не стреляет, если он уничтожен	Создать объект tank типа GeneralTank, установить у него свойство Health в 0 (танк уничтожен), вызвать у него метод Fire(), результат которого поместить в переменную bullet и проверить, что bullet равен null	Объект bullet равен null, так как родительский танк имеет нулевое здоровье
9	UpdateActiveBulletCounterDecreasesBulletCount	Проверка на ограничение максимального количества активных (летающих) снарядов для танка	Создать объект tank типа GeneralTank, вызвать у него метод Fire(), занести возвращенное значение в переменную bullet, затем подождать не менее tank.FireCooldown секунд, затем у bullet вызвать метод OnObjectDestroyed() для уничтожения пули, затем еще раз вызвать метод Fire(), занести возвращенное значение в переменную bullet, затем подождать не менее tank.FireCooldown секунд, затем еще раз вызвать метод Fire(),	Объект bulletSecond не равен null, так как в момент стрельбы танком было достигнуто максимальное значение все еще летающих снарядов

			занести результат в bullet и проверить, что bullet равен null	
10	GetDirectionToPointRightDirection	Проверка на то, что танк правильно определяет направление к заданной точке	Создать объект tank типа GeneralTank с координатами (2;2), создать переменную direction, в которую поместить возвращенное значение метода tank.GetDirectionToPoint, в который передать объект класса Point с координатами (5; 2) и проверить, что direction равно DirectionEnum.Right	Переменная direction равна DirectionEnum.Right, т.е. танк определил, что цель находится справа
11	FindPathWithLeePathFound	Проверка на то, что танк может находить путь к цели на пустом поле	Создать объект tank типа GeneralTank, создать объект Field игрового поля размером 10 на 10, вызвать у tank метод FindPathWithLee, куда передать field, начальную точку с координатами (0;0) и конечную точку с координатами (5;5), результат работы поместить в переменную path и проверить, что она не равна null	Переменная path не равна null, т.е. кратчайший путь был найден
12	FindPathWithLeePathFoundWithPartialObstacle	Проверка на то, что танк может находить путь к цели при наличии преграды с брешью	Создать объект tank типа GeneralTank, создать объект field игрового поля размером 10 на 10, в ячейки с координатами (1;0), (1;1), (1;3), (1;4) поместить объекты типа Water, вызвать у tank метод FindPathWithLee, куда передать field, начальную точку с координатами (0;0) и конечную точку с координатами (5;5), результат работы поместить в переменную path и проверить, что она не равна null	Переменная path не равна null, т.е. кратчайший путь был найден, поскольку между начальной и конечной точкой есть проход через пустую клетку с координатами (1; 2)
13	FindPathWithLeeNoPathFound()	Проверка на то, что танк не может находить путь к цели при наличии сплошной преграды	Создать объект tank типа GeneralTank, создать объект field игрового поля размером 10 на 10, в ячейки с координатами (1;0), (1;1), (1;2), (1;3), (1;4) поместить объекты типа Water, вызвать у tank метод FindPathWithLee, куда передать field, начальную точку с координатами (0;0) и конечную точку с координатами (5;5), результат работы поместить в переменную path и проверить, что она равна null	Переменная path равна null, т.е. кратчайший путь не был найден, поскольку между начальной и конечной точкой сплошная непроходимая область ячеек с водой без разрывов
14	IsValidCellForTankValidCell()	Проверка на то, что пустая ячейка является валидной для размещения танка	Создать объект tank типа GeneralTank, создать объект field игрового поля размером 10 на 10 с размером стороны клетки в 1 пиксель, объявить переменную result, в которую	Переменная result равна true, так как танк может размещаться на пустой ячейке



			поместить результат вызова метода IsValidCellForTank, куда передать field, координату (5;5) и 1 как ширину танка в пикселях, аналогично 1 для высоты. Проверить, что result равен true	
15	IsValidCellForTankInvalidCellOutOfBounds()	Проверка на то, что ячейка за пределами поля не является валидной для размещения танка	Создать объект tank типа GeneralTank, создать объект field игрового поля размером 10 на 10 с размером стороны клетки в 1 пиксель, объявить переменную result, в которую поместить результат вызова метода IsValidCellForTank, куда передать field, координату (15;15) и 1 как ширину танка в пикселях, аналогично 1 для высоты. Проверить, что result равен false	Переменная result равна false, так как танк не может размещаться за границами игрового поля
16	IsValidCellForTankInvalidCellObstacle()	Проверка на то, что препятствие не является валидной клеткой для размещения танка	Создать объект tank типа GeneralTank, создать объект field игрового поля размером 10 на 10 с размером стороны клетки в 1 пиксель, поместить в точку (5;5) объект Wall(), объявить переменную result, в которую поместить результат вызова метода IsValidCellForTank, куда передать field, координату (5;5) и 1 как ширину танка в пикселях, аналогично 1 для высоты. Проверить, что result равен false	Переменная result равна false, так как танк не может размещаться поверх непроходимого блока
17	HasDirectLineOfSightClearLine()	Проверка на то, что другая точка находится в зоне поражения, если между точками нет препятствий	Создать объект tank типа GeneralTank, создать объект field игрового поля размером 10 на 10 с размером стороны клетки в 1 пиксель, объявить переменную result, в которую поместить результат вызова метода HasDirectLineOfSight, куда передать field, координаты (0;0) и (5;5). Проверить, что result равен true	Переменная result равна true, так как от начальной до конечной точки пустое пространство и объект, находящийся в конечной точке, может быть поражен снарядом, вылетающим из начальной точки
18	HasDirectLineOfSightDoesNotBlockByWall()	Проверка на то, что разрушимая стена на прямом пути к цели может потенциально быть уничтожена танком, т.е цель за ней может быть поражена	Создать объект tank типа GeneralTank, создать объект field игрового поля размером 10 на 10 с размером стороны клетки в 1 пиксель, поместить в точку (0;2) объект Wall разрушимой стены, объявить переменную result, в которую поместить результат вызова метода HasDirectLineOfSight, куда передать field, координаты	Переменная result равна true, так как от начальной до конечной точки пустое пространство плюс блок разрушимой стены и объект, находящийся в конечной точке, может быть поражен снарядом, вылетающим из

			(0;0) и (5;5). Проверить, что result равен true	начальной точки после уничтожения предыдущим снарядом разрушимой стены
19	HasDirectLineOfSightBlockedLine()	Проверка на то, что неразрушимая стена на прямом пути к цели сигнализирует о невозможности поражения ее пулей	Создать объект tank типа GeneralTank, создать объект field игрового поля размером 10 на 10 с размером стороны клетки в 1 пиксель, поместить в точку (0;2) объект IndestructibleWall неразрушимой стены, объявить переменную result, в которую поместить результат вызова метода HasDirectLineOfSight, куда передать field, координаты (0;0) и (5;5). Проверить, что result равен false	Переменная result равна false, так как на отрезке от начальной до конечной точки кроме пустого пространства находится блок неразрушимой стены, который не позволит пуле долететь до точки назначения
20	ChangeDirectionChangesToNewDirection()	Изменение направления для танка выполняется правильно	Создать объект tank типа GeneralTank с начальными координатами (5;5), вызвать у него метод ChangeDirection с параметром (5; 8) и проверить, что свойство tank.Direction равно DirectionEnum.Down	Свойство tank.Direction равно DirectionEnum.Down, так как точка (5; 8) находится относительно точки (5;5) внизу
21	TestTankCanChangeDirectionAfterPause()	Тест на то, что после задержки танк может поменять направление	Создать объект tank типа GeneralTank с начальными координатами (0;0), вызвать у него метод ChangeDirection с параметром (10; 10), поместить текущее значение свойства tank.Direction в переменную initialDirection, затем подождать не менее tank.DirectionChangeCooldown секунд, затем снова вызвать у него метод ChangeDirection с параметром (-10; -10), поместив результат в переменную directionAfterChange и проверить, что переменные initialDirection и directionAfterChange не равны	Переменные initialDirection и directionAfterChange не равны, так как между двумя изменениями направления в противоположные стороны была сделана пауза на количество секунд, достаточное для разрешения изменить направление еще раз
22	UpdateActiveBulletsCounterSubtractToZero()	Счетчик активных снарядов у танка не может стать меньше 0	Создать объект tank типа GeneralTank, присвоить свойству ActiveBullets значение 1, вызвать у объекта tank метод UpdateActiveBulletsCounter() трижды для уменьшения значения счетчика, затем проверить, что tank.ActiveBullets равно 0	Свойство tank.ActiveBullets равно 0, так как счетчик активных снарядов не может принимать отрицательное значение

## 2.5.2 Модульные тесты

На основе предыдущего пункта были разработаны модульные тесты, первые 10 штук которых приведены ниже с краткими комментариями:

```
/// <summary>
/// Проверяет факт создания танка с валидными параметрами по умолчанию
/// </summary>
[Fact]
public void DefaultPropertiesAreSetCorrectly()
{
    var tank = new GeneralTank(new Point(0, 0));

    Assert.Equal(AbstractTank.DEFAULT_SPEED, tank.Speed);
    Assert.Equal(AbstractTank.DEFAULT_FIRING_RATE, tank.FiringRate);
    Assert.Equal(AbstractTank.DEFAULT_DAMAGE, tank.Damage);
    Assert.Equal(AbstractTank.DEFAULT_HEALTH, tank.Health);
}

/// <summary>
/// Проверяет факт нанесения урона танку снарядом
/// </summary>
[Fact]
public void ApplyDamageReducesHealthCorrectly()
{
    var tank = new GeneralTank(new Point(0, 0));
    var bullet = new Bullet(new Point(0, 0), DirectionEnum.Up, Guid.NewGuid(), 20);
    tank.ApplyDamage(bullet);
    Assert.Equal(80, tank.Health);
}

/// <summary>
/// Проверяет, что снаряд не может поразить родительский танк
/// </summary>
[Fact]
public void ApplyDamageByParentTankShouldNotBe()
{
    var tank = new GeneralTank(new Point(0, 0));
    tank.Health = 100;
    var bullet = new Bullet(new Point(), DirectionEnum.Right, tank.Id, 20);
    if (bullet == null)
    {
        Assert.Fail();
    }
    tank.ApplyDamage(bullet);
    Assert.Equal(100, tank.Health);
}

/// <summary>
/// Проверяет, что здоровье танка после нанесения урона не может стать меньше 0
/// </summary>
[Fact]
public void ApplyDamageDoesNotReduceHealthBelowZero()
{
    var tank = new GeneralTank(new Point(0, 0));
```

```
tank.Health = 5;
var bullet = new Bullet(new Point(0, 0), DirectionEnum.Up, Guid.NewGuid(), 200);
bullet.Damage = 10;
tank.ApplyDamage(bullet);
Assert.Equal(0, tank.Health);
}

/// <summary>
/// Проверка на то, что танк может выпустить хотя бы один снаряд без ограничений на
/// максимальную скорострельность
/// </summary>
[Fact]
public void FireCreatesBulletWhenCooldownAllows()
{
    var tank = new GeneralTank(new Point(0, 0));
    var bullet = tank.Fire();
    Assert.NotNull(bullet);
}

/// <summary>
/// Проверка на ограничение скорострельности танка
/// </summary>
[Fact]
public void FireShouldNotCreateBulletWhenCooldown()
{
    var tank = new GeneralTank(new Point(0, 0));
    tank.Fire();
    var bulletSecond = tank.Fire();
    Assert.Null(bulletSecond);
}

/// <summary>
/// Проверка на то, что спустя FireCooldown времени танк может выстрелить снова
/// </summary>
[Fact]
public void FireShouldCreateBulletAfterCooldown()
{
    var tank = new GeneralTank(new Point(0, 0));
    var bulletFirst = tank.Fire();
    Assert.NotNull(bulletFirst);
    bulletFirst.OnObjectDestroyed();
    Thread.Sleep((int)(tank.FireCooldown.TotalMilliseconds * 1.1));
    var bulletSecond = tank.Fire();
    Assert.NotNull(bulletSecond);
}

/// <summary>
/// Проверяет, что танк не стреляет, если он уничтожен
/// </summary>
[Fact]
public void DestroyedTankCannotFire()
{
    var tank = new GeneralTank(new Point(5, 5)) { Health = 0 };
    var bullet = tank.Fire();
    Assert.Null(bullet);
}
```

```
/// <summary>
/// Проверка на ограничение максимального количества активных (летающих) снарядов для
танка
/// </summary>
[Fact]
public void UpdateActiveBulletsCounterDecreasesBulletCount()
{
    var tank = new GeneralTank(new Point(0, 0));
    var bullet = tank.Fire();
    Assert.NotNull(bullet);
    Thread.Sleep((int)(tank.FireCooldown.TotalMilliseconds * 1.1));
    bullet.OnObjectDestroyed();
    bullet = tank.Fire();
    Assert.NotNull(bullet);
    Thread.Sleep((int)(tank.FireCooldown.TotalMilliseconds * 1.1));
    bullet = tank.Fire();
    Assert.Null(bullet);
}

/// <summary>
/// Проверка на то, что танк правильно определяет направление к заданной точке
/// </summary>
[Fact]
public void GetDirectionToPointRightDirection()
{
    var tank = new GeneralTank(new Point(2, 2));
    var direction = tank.GetDirectionToPoint(new Point(5, 2));
    Assert.Equal(DirectionEnum.Right, direction);
}
```

## 3 Результат работы программы

### 3.1 Консольная версия

Результаты работы программы в консольной версии приведены на рисунках 22 – 28. Выбор пунктов меню и переход к соответствующим меню производится через стрелки на клавиатуре.

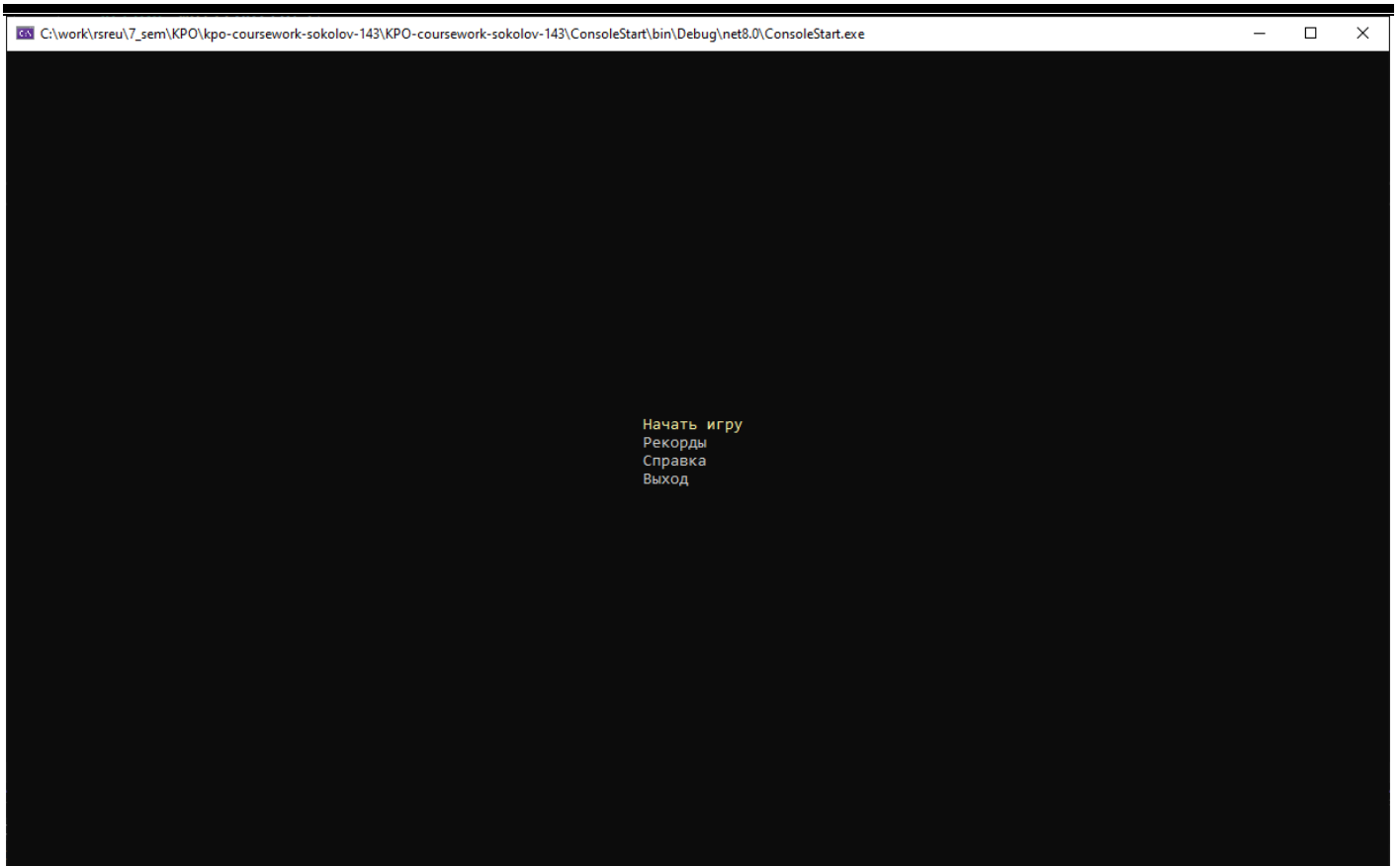


Рисунок 22 – Главное меню

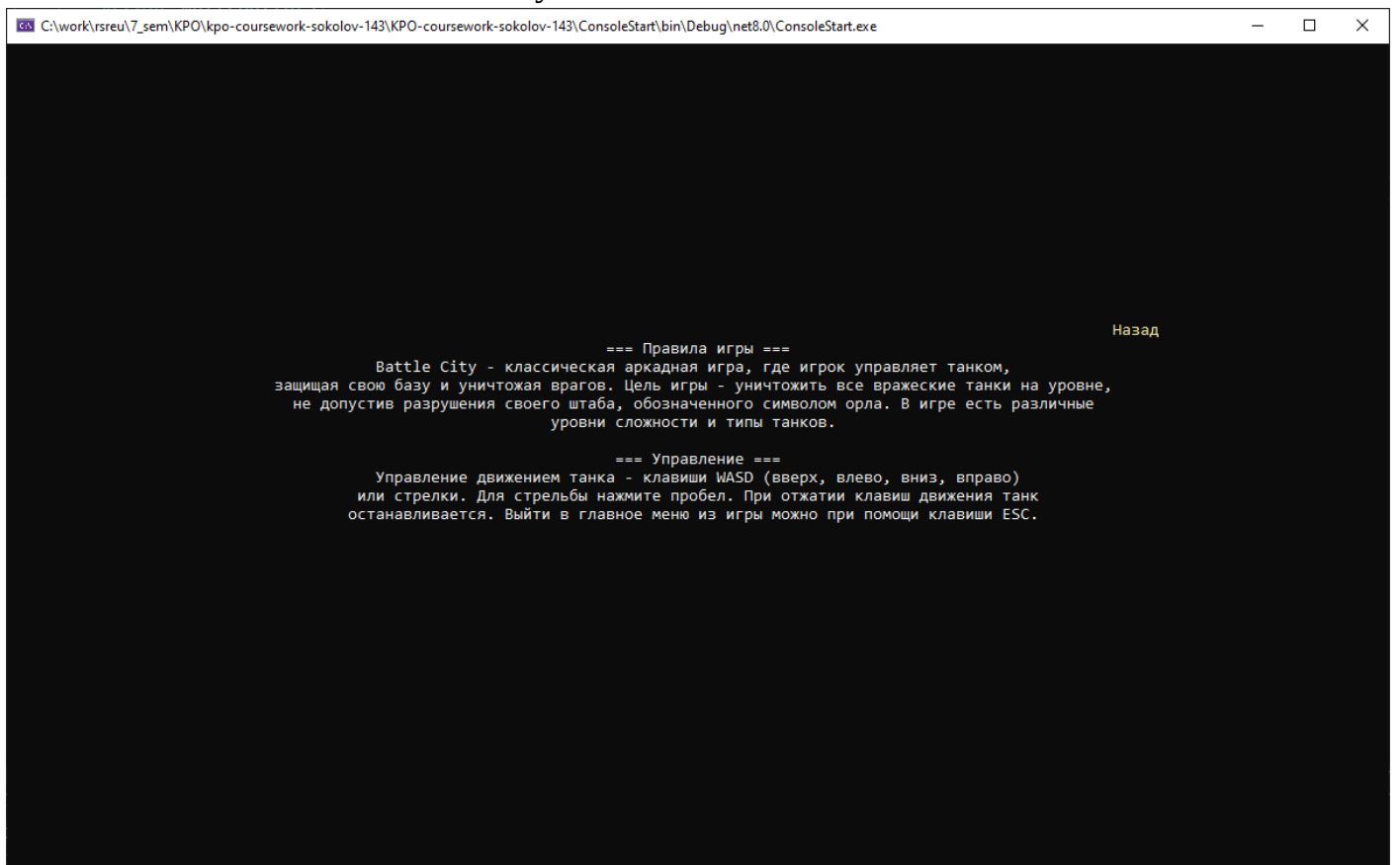


Рисунок 23 – Справка

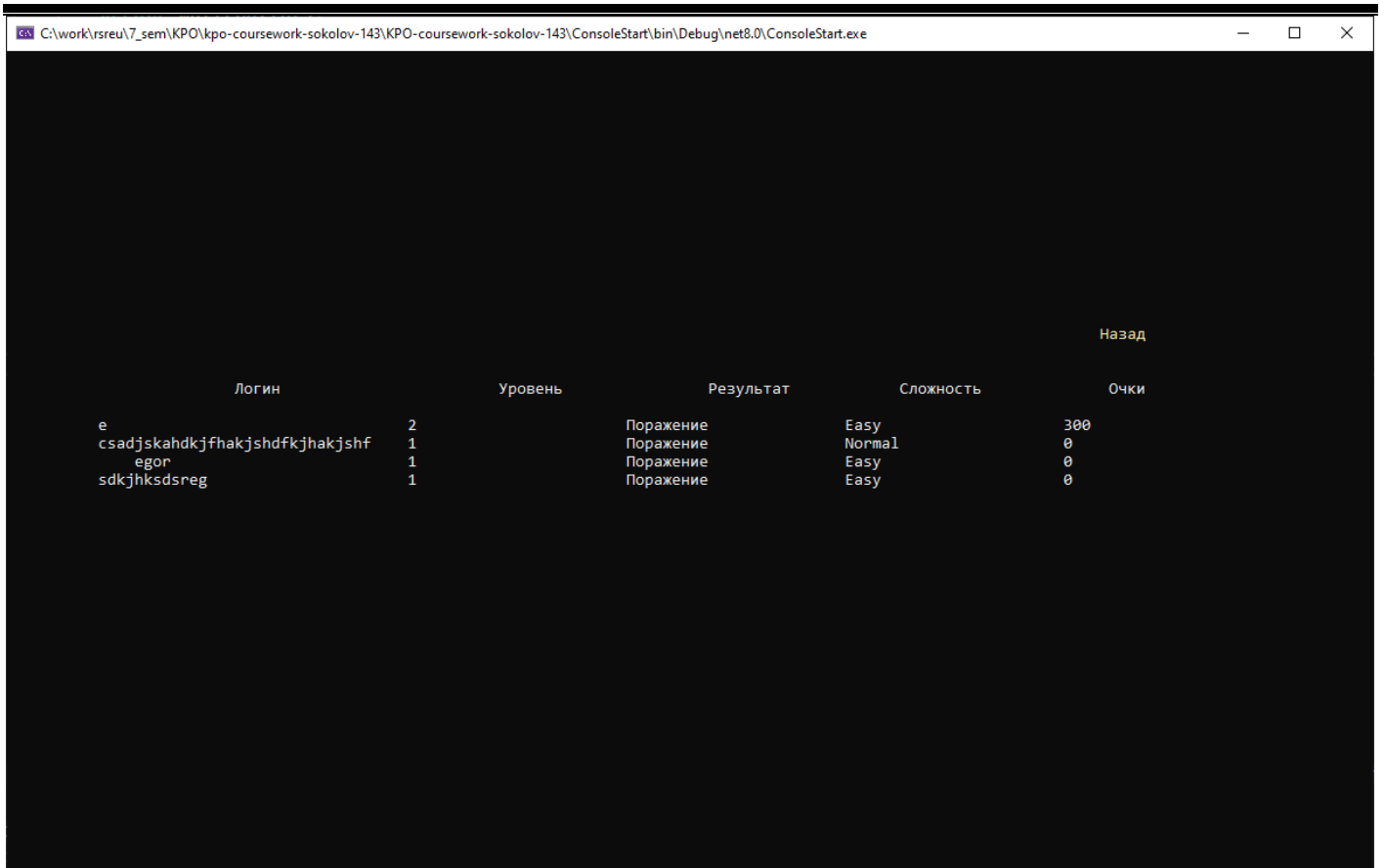


Рисунок 24 – Рекорды



Рисунок 25 – Выбор уровня сложности Игры



Рисунок 26 – Игровой процесс

Если Игрок проходит все урони Игры и выигрывает, либо на каком-то моменте потратил все свои жизни, либо его база была уничтожена, то она попадает в меню записи рекордов, где приводится информация о поражении или победе и о количестве набранных пользователем очков, где можно записать рекорд в таблицу рекордов либо выйти в главное меню без записи.



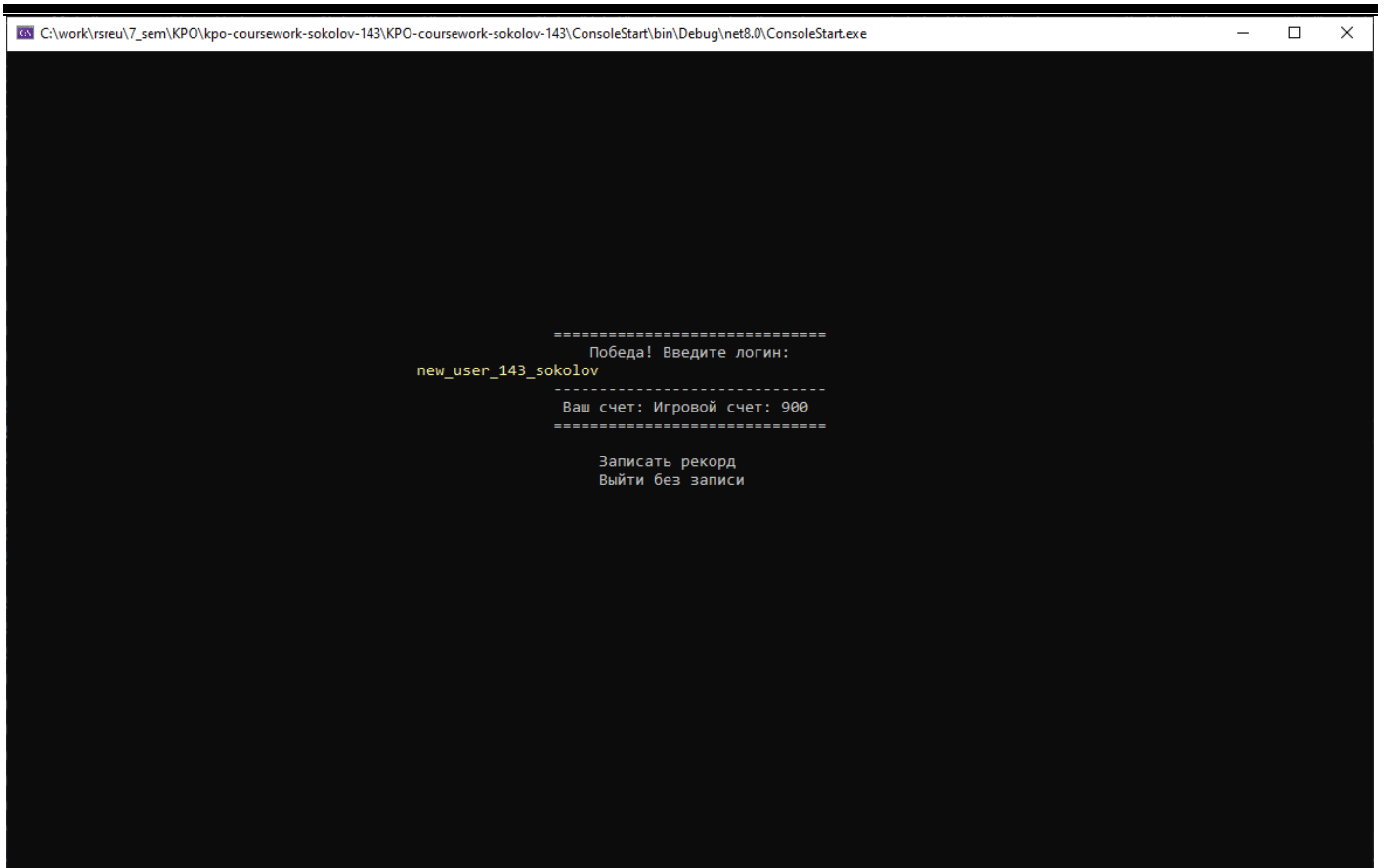


Рисунок 27 – Меню записи рекордов

В любой момент игрового процесса при нажатии клавиши ESC пользователь попадает в главное меню приложения, откуда при выборе пункта меню с новой игрой Пользователю будет предложен выбор либо продолжить старую игру, либо начать новую.

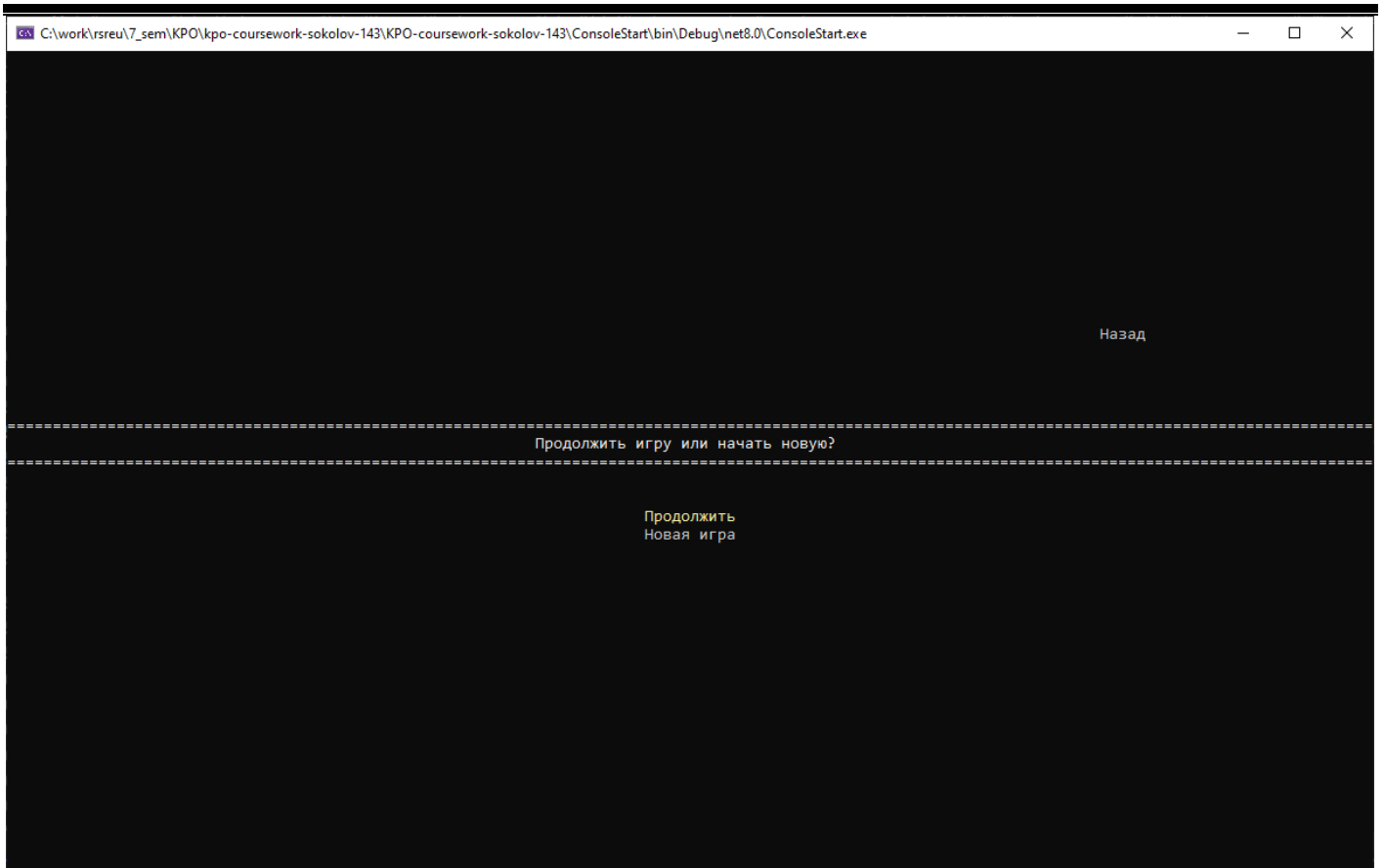


Рисунок 28 – Возврат из паузы

### 3.2 WPF версия

Результаты работы программы в консольной версии приведены на рисунках 29 – 36. Выбор пунктов меню и переход к соответствующим меню производится через стрелки на клавиатуре либо через клик левой кнопкой мыши.

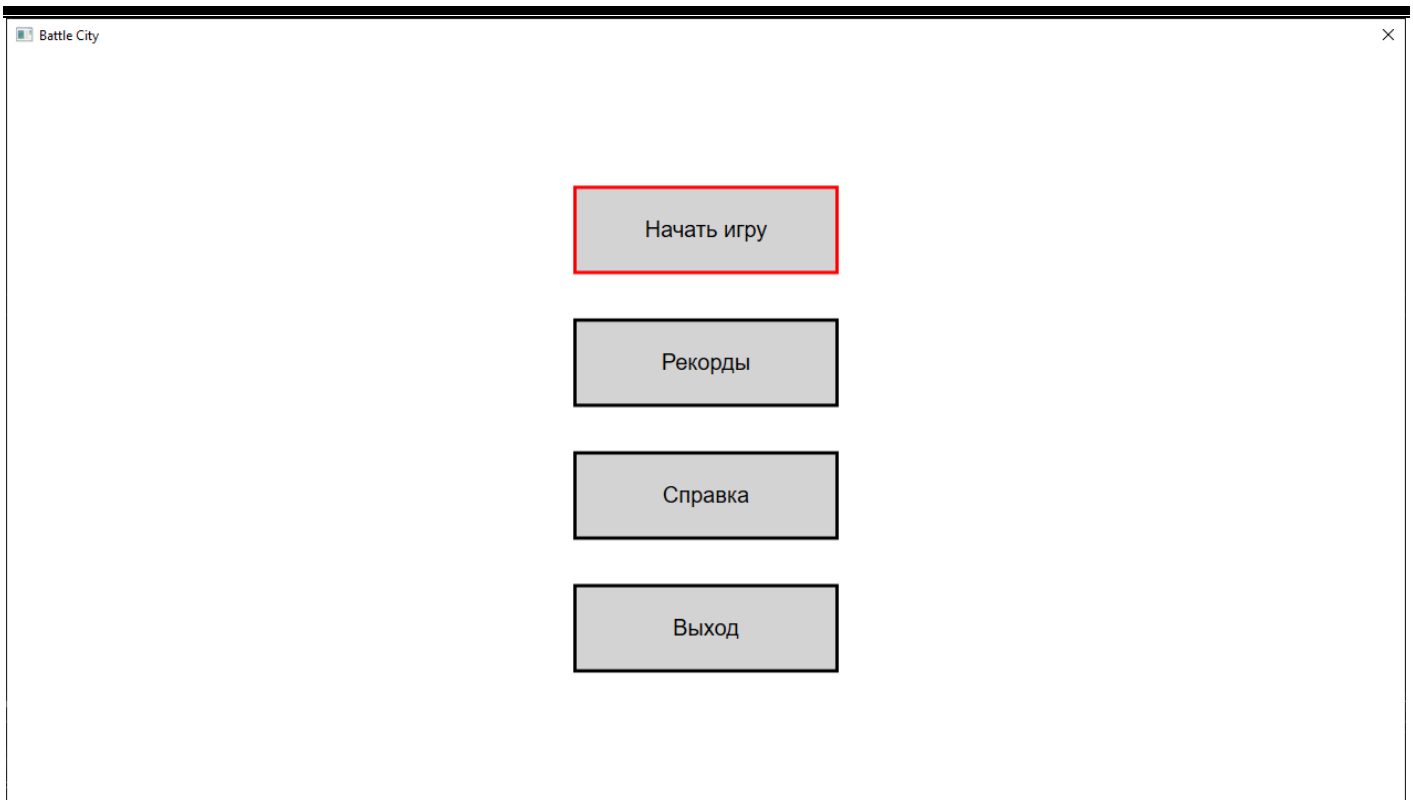


Рисунок 29 – Главное меню

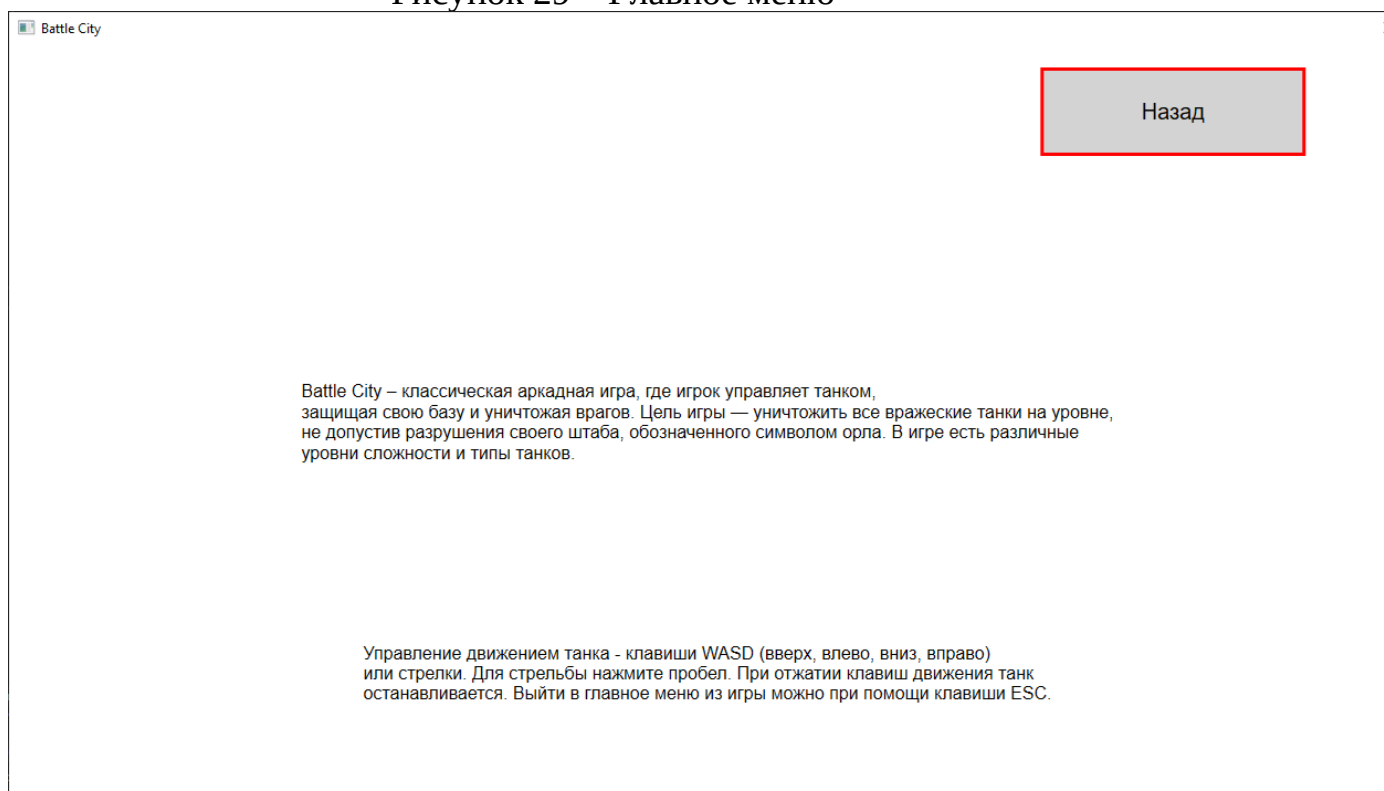


Рисунок 30 – Справка

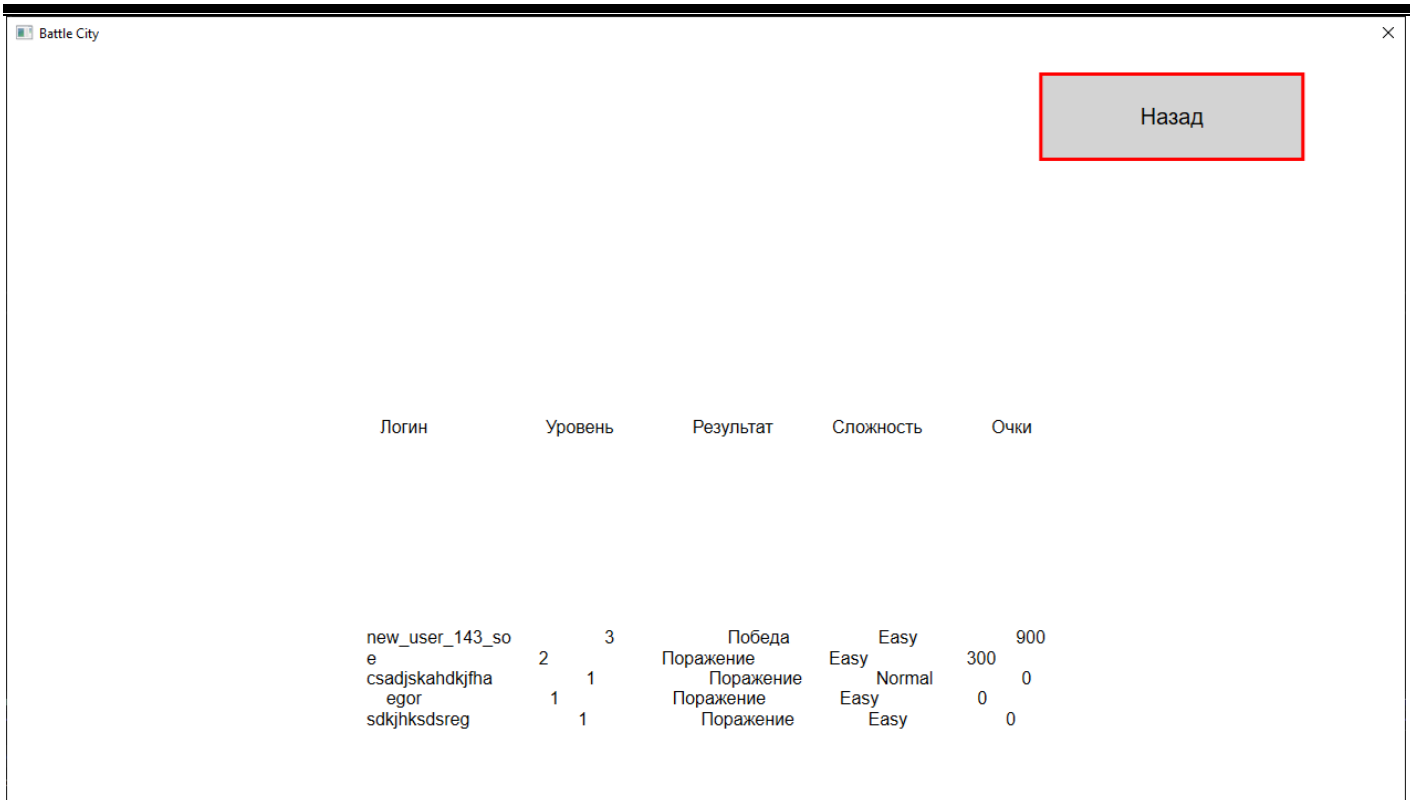


Рисунок 31 – Рекорды

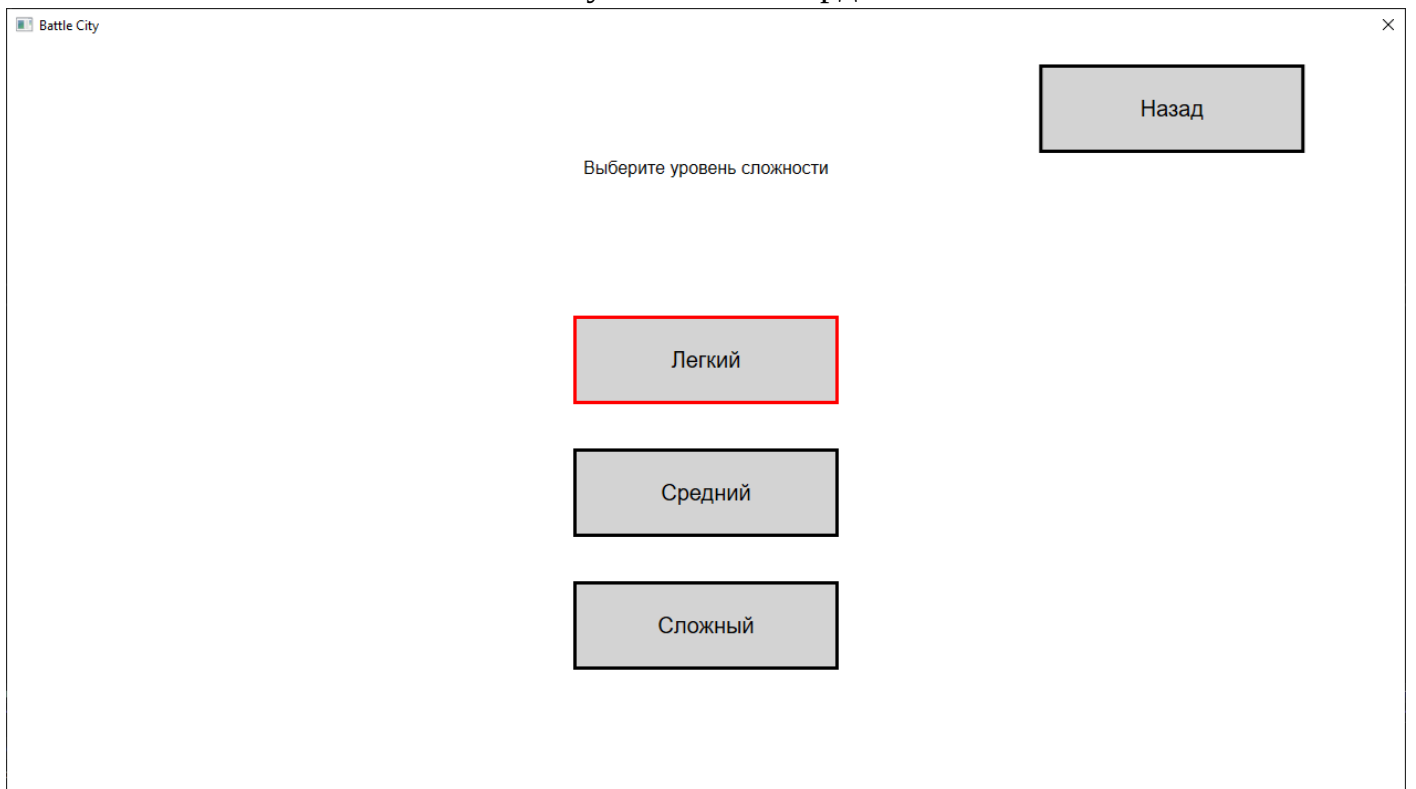


Рисунок 32 – Выбор уровня сложности Игры

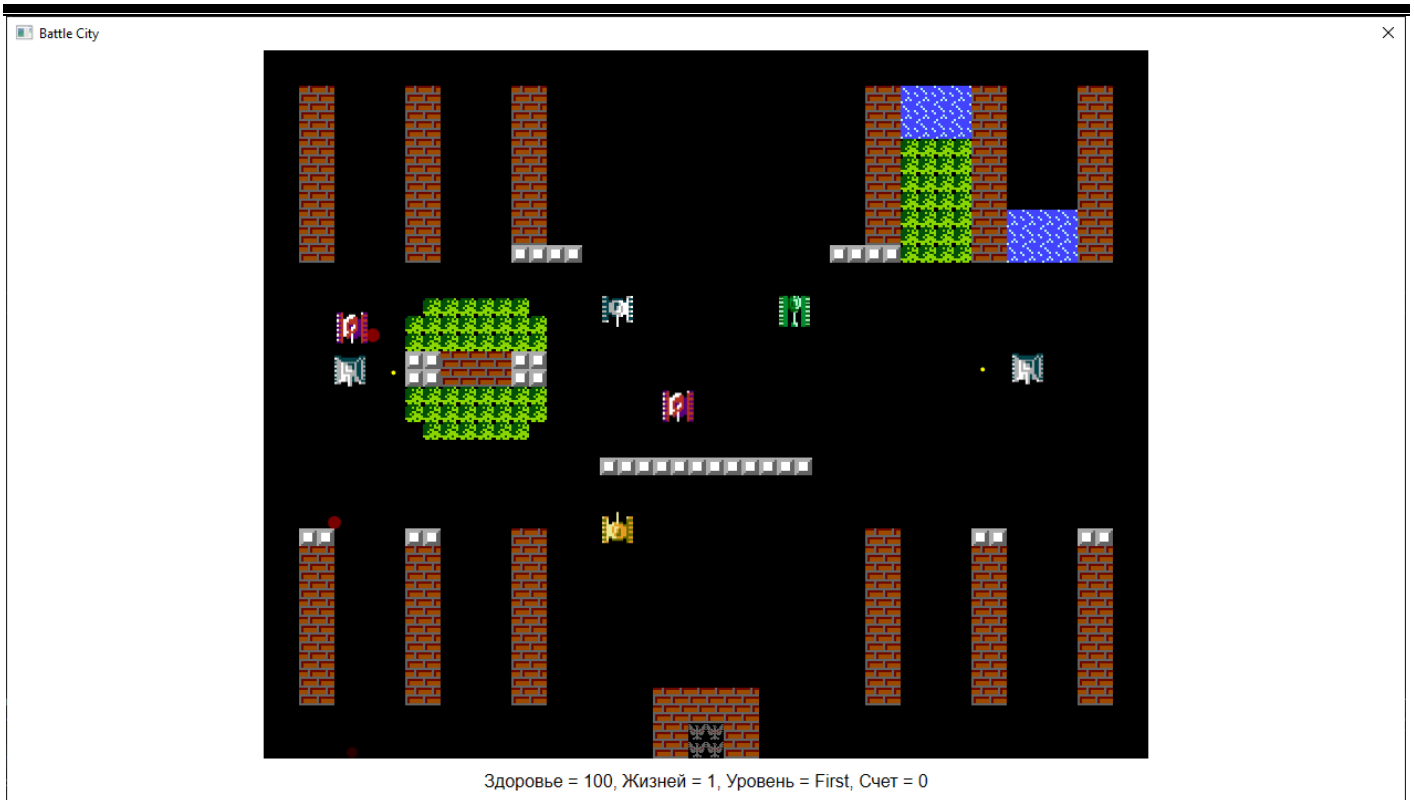


Рисунок 33 – Игровой процесс

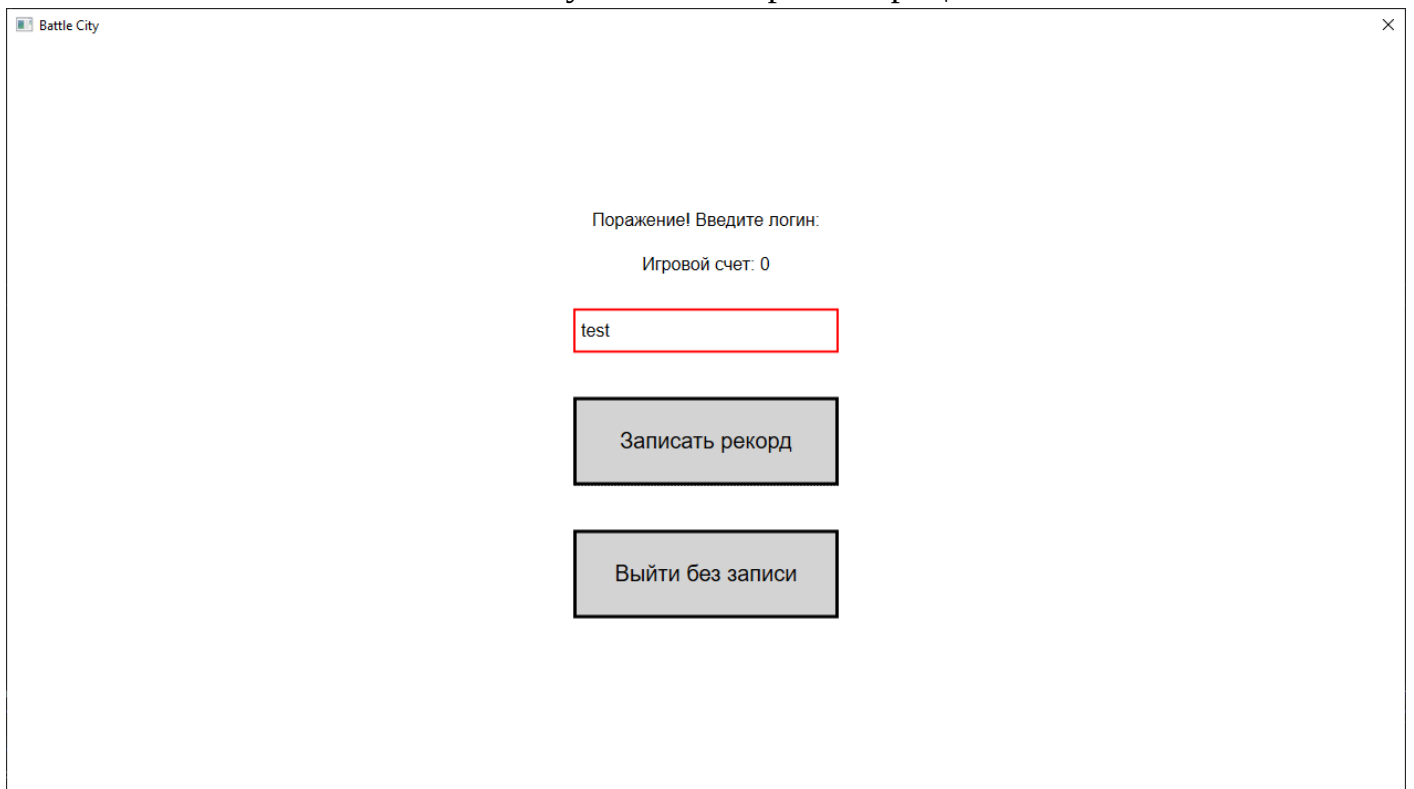


Рисунок 34 – Меню записи рекордов

В любой момент игрового процесса при нажатии клавиши ESC пользователь попадает в главное меню приложения, откуда при выборе пункта меню с новой игрой Пользователю будет предложен выбор либо продолжить старую игру, либо начать новую.

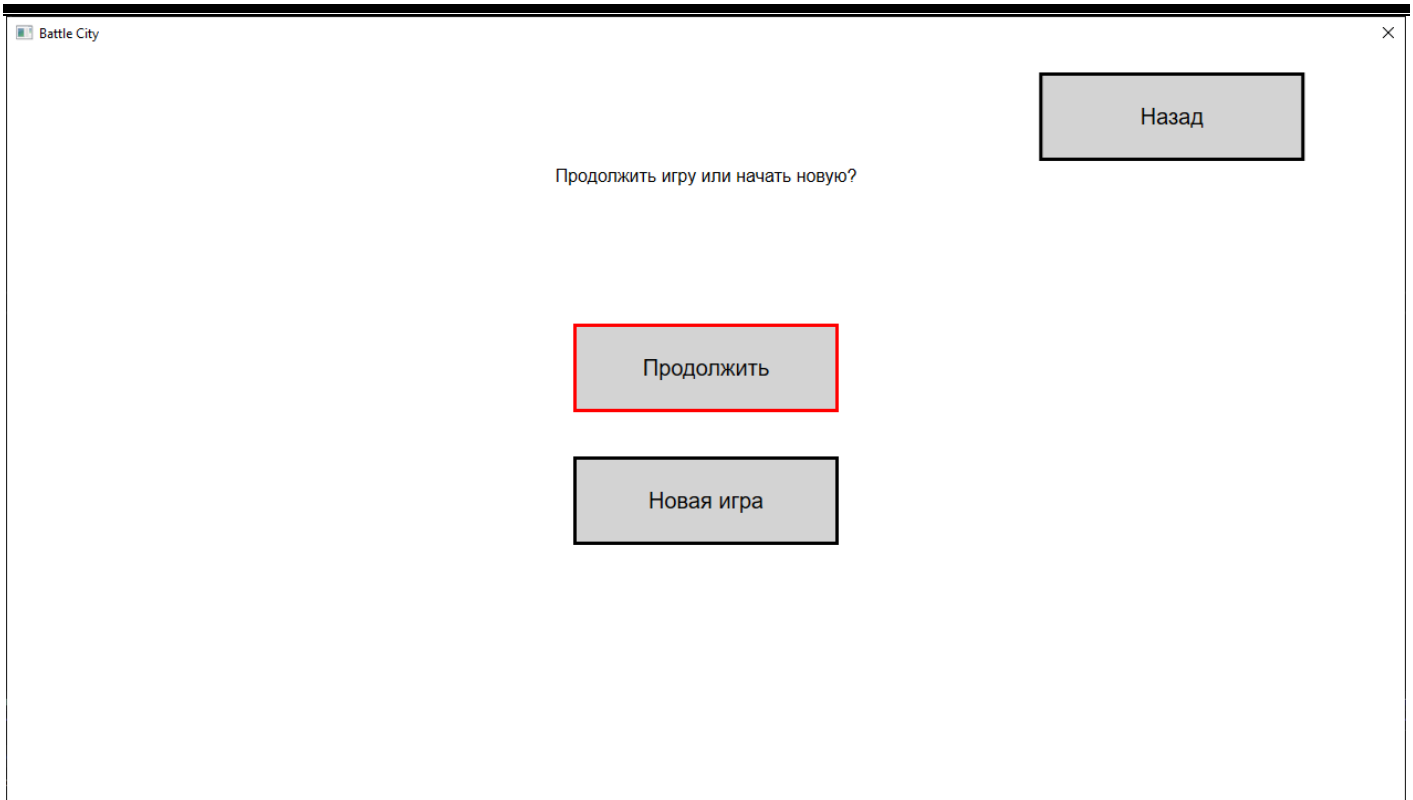


Рисунок 35 – Возврат из паузы

## Заключение

В ходе выполнения данной работы были продемонстрированы умения программирования на языке C# с использованием WPF. В процессе разработки программы применялись различные шаблоны проектирования, методы многозадачности и инструменты для работы с системами контроля версий. Разработанный программный продукт работает стабильно и полностью соответствует заданным требованиям.

## Приложения

### Методы, появившиеся после рефакторинга метода UpdateAI для раздела 2.4.1

```

/// <summary>
/// Волновой алгоритм Ли с учетом размеров танка
/// </summary>
/// <param name="field">Игровое поле</param>
/// <param name="start">Точка начала поиска</param>
/// <param name="end">Конечная точка</param>
/// <param name="tankWidthInCells">Ширина танка в клетках</param>
/// <param name="tankHeightInCells">Высота танка в клетках</param>
/// <returns></returns>
public List<Point> FindPathWithLee(GameField field, Point start, Point end, int
tankWidthInCells, int tankHeightInCells)
{
    int width = field.GetWidthInCells();
    int height = field.GetHeightInCells();
    var wave = new int[width, height];

    for (int x = 0; x < width; x++)
    {

```

```
        for (int y = 0; y < height; y++)
        {
            wave[x, y] = -1;
        }
    }

    var queue = new Queue<Point>();
    queue.Enqueue(start);
    wave[(int)(start.X), (int)(start.Y)] = 0;

    var directions = new[]
    {
        new Point(-1, 0), new Point(1, 0),
        new Point(0, -1), new Point(0, 1)
    };

    // Волновое распространение
    while (queue.Count > 0)
    {
        var current = queue.Dequeue();

        foreach (var elDir in directions)
        {
            var next = new Point(current.X + elDir.X, current.Y + elDir.Y);

            if (IsValidCellForTank(field, next, tankWidthInCells,
tankHeightInCells) &&
                wave[(int)(next.X), (int)(next.Y)] == -1)
            {
                wave[(int)(next.X), (int)(next.Y)] = wave[(int)(current.X),
(int)(current.Y)] + 1;
                queue.Enqueue(next);

                if (next.X == end.X && next.Y == end.Y)
                {
                    queue.Clear();
                    break;
                }
            }
        }
    }

    // Построение пути
    if (wave[(int)(end.X), (int)(end.Y)] == -1) return null; // Путь не найден

    var path = new List<Point>();
    var currentPoint = end;
    int counter = 0;
    while (!IsCellsPointEquals(currentPoint, start) && (path.Count == 0 ||
!IsCellsPointEquals(currentPoint, path.Last()))) && counter <= 1000)
    {
        counter++;
        path.Add(currentPoint);
        foreach (var elDir in directions)
        {
            var next = new Point(currentPoint.X - elDir.X, currentPoint.Y -
elDir.Y);
```

```
        if ((int)(next.X) >= 50 || (int)(next.Y) >= 40)
        {
            float c = next.X / 2;
        }

        if (next.X >= 0 && next.X < width &&
            next.Y >= 0 && next.Y < height &&
            wave[(int)(next.X), (int)(next.Y)] ==
wave[(int)(currentPoint.X), (int)(currentPoint.Y)] - 1)
        {
            currentPoint = next;
            break;
        }
    }
}

path.Reverse();
return path;
}

/// <summary>
/// Находится ли точка parEnd в прямой видимости (на луче нет неразрушимой стены),
/// т.е. до этой точки с позиции parStart может долететь снаряд
/// </summary>
/// <param name="parGameField">Модель игрового поля</param>
/// <param name="parStart">Начальная точка</param>
/// <param name="parEnd">Целевая конечная точка</param>
/// <param name="parIsCell">Является ли цель клеткой (блоком)</param>
/// <returns>Истина, если точка в досягаемости снарядом</returns>
public bool HasDirectLineOfSight(GameField parGameField, Point parStart, Point parEnd,
bool parIsCell = false)
{
    double closeCoeff = parIsCell ? 0.2 : 0.5;
    int dx = Math.Sign(parEnd.X - parStart.X) * (Math.Abs(parEnd.X - parStart.X) /
parGameField.CellSize > closeCoeff ? 1 : 0);
    int dy = Math.Sign(parEnd.Y - parStart.Y) * (Math.Abs(parEnd.Y - parStart.Y) /
parGameField.CellSize > closeCoeff ? 1 : 0);
    if (dx == 0 && dy == 0)
    {
        return true;
    }
    float currentX = parStart.X;
    float currentY = parStart.Y;

    while (Math.Abs(currentX - parEnd.X) > parGameField.CellSize || Math.Abs(currentY
- parEnd.Y) > parGameField.CellSize)
    {
        if (currentX < 0 || currentY < 0
            || currentY >= parGameField.GetHeightInPixels() || currentX >=
parGameField.GetWidthInPixels() ||
            !IsCellPassableByBulletOrDestructible(parGameField, (int)currentX,
(int)currentY)) return false;

        if (currentX != parEnd.X) currentX += dx;
        if (currentY != parEnd.Y) currentY += dy;
    }
}
```



```
        return true;
    }

    /// <summary>
    /// Изменить направление танка, чтобы он повернулся к целевой точке
    /// </summary>
    /// <param name="parTargetPoint">Целевая точка</param>
    public void ChangeDirection(Point parTargetPoint)
    {
        if (DateTime.Now - _lastDirectionChangeTime < _directionChangeCooldown)
        {
            return; // Слишком частая смена направления запрещена
        }
        int dx = ((int)parTargetPoint.X - (int)Point.X);
        int dy = ((int)parTargetPoint.Y - (int)Point.Y);
        if (dx == 0 && dy == 0) return;
        DirectionEnum newDirection;
        if (Math.Abs(dx) > Math.Abs(dy))
        {
            newDirection = dx > 0 ? DirectionEnum.Right : DirectionEnum.Left;
        }
        else
        {
            newDirection = dy > 0 ? DirectionEnum.Down : DirectionEnum.Up;
        }

        if (Direction != newDirection)
        {
            Direction = newDirection;

            _lastDirectionChangeTime = DateTime.Now;
        }
    }
}
```

### Исходный код диаграммы последовательности для главного меню приложения

```
@startuml
participant "Игрок" as User
participant "Контроллер главного меню" as MenuController
participant "Модель главного меню" as Menu
participant "Представление главного меню" as MenuView
participant "Контроллер рекордов" as AchievementController
participant "Контроллер справки" as HelpController
participant "Контроллер уровней сложности" as GameController
participant "Контроллер паузы" as PauseController

User -> MenuController: Запуск контроллера меню
activate MenuController
    MenuController -> Menu: Создание модели меню
    activate Menu
    return
    MenuController -> MenuView: Создание представления меню и передача в него модели меню
    activate MenuView
    return
deactivate MenuController
```

```
User -> MenuController: Взаимодействие с меню
activate MenuController
    alt Пользователь выбирает "Начать игру"
        MenuController -> Menu: Проверить состояние игры
        activate Menu
        return
    alt Игра на паузе
        MenuController -> PauseController: Запрос возобновления игры
        activate PauseController
        PauseController --> MenuController: Возврат в меню
        deactivate PauseController
    else Игра не на паузе
        MenuController -> GameController: Выбор уровня сложности игры
        activate GameController
        GameController --> MenuController: Возврат в меню
        deactivate GameController
    end
else Пользователь выбирает "Рекорды"
    MenuController -> AchievementController: Показать рекорды
    activate AchievementController
    AchievementController --> MenuController: Возврат в меню
    deactivate AchievementController
else Пользователь выбирает "Справку"
    MenuController -> HelpController: Показать правила
    activate HelpController
    HelpController --> MenuController: Возврат в меню
    deactivate HelpController
else Пользователь выбирает "Выход"
    MenuController -> MenuController: Выход из игры
end
deactivate MenuController
@enduml
```

#### Исходный код диаграммы ИИ танка

```
@startuml
actor "Танк противника" as EnemyTank
participant "Игровое поле" as Field
participant "Волновой алгоритм Ли" as Algorithm
participant "Путь к цели" as Path

EnemyTank -> Field: Запросить текущее положение игрока
activate Field
Field -> EnemyTank: Координаты игрока
deactivate Field

EnemyTank -> Field: Запросить текущее положение танка
activate Field
Field -> EnemyTank: Координаты танка
deactivate Field
```

```
EnemyTank -> Algorithm: Проверить возможность добраться до игрока
activate Algorithm
Algorithm -> EnemyTank: Можно ли построить путь до игрока
deactivate Algorithm

alt Можно добраться
    EnemyTank -> Algorithm: Найти кратчайший путь до игрока
    activate Algorithm
    Algorithm -> EnemyTank: Путь к игроку
    deactivate Algorithm
    EnemyTank -> Field: Начать движение по маршруту
else Нельзя добраться
    EnemyTank -> Field: Найти кратчайший путь до базы
    activate Field
    Field -> EnemyTank: Путь к базе
    deactivate Field
    EnemyTank -> Field: Начать движение по маршруту
end

loop Пока танк не уничтожен
    EnemyTank -> Field: Проверить видимость игрока
    activate Field
    Field -> EnemyTank: Игрок в зоне видимости?
    deactivate Field
    alt Игрок в зоне видимости
        EnemyTank -> EnemyTank: Открыть огонь по игроку
    else Игрок не в зоне видимости
        EnemyTank -> Field: Проверить видимость базы
        activate Field
        Field -> EnemyTank: База в зоне видимости?
        deactivate Field
        alt База в зоне видимости
            EnemyTank -> EnemyTank: Открыть огонь по базе
        else База не в зоне видимости
            EnemyTank -> EnemyTank: Сменить цель с вероятностью 25%
        end
    end
end
end
@enduml
```

**Исходный код диаграммы поиска кратчайшего пути до базы:**

```
@startuml
actor "Танк противника"
participant "Игровое поле" as Field
participant "Волновой алгоритм Ли" as Algorithm
participant "Путь к базе" as Path

"Танк противника" -> Field: Запросить текущее положение базы
```

```
activate Field
Field -> "Танк противника": Координаты базы
deactivate Field

"Танк противника" -> Field: Запросить текущее положение танка
activate Field
Field -> "Танк противника": Координаты танка
deactivate Field

"Танк противника" -> Algorithm: Запуск волнового алгоритма\n(начало с текущих
координат)
activate Algorithm

loop Пока путь не найден
    Algorithm -> Field: Получить соседние ячейки
    activate Field
    Field -> Algorithm: Соседние ячейки
    deactivate Field

    alt Если соседняя ячейка достижима
        Algorithm -> Field: Отметить ячейку с новым шагом волны
        Algorithm -> Algorithm: Добавить ячейку в очередь на обработку
    else Если ячейка непроходима (стена, препятствие)
        Algorithm -> Algorithm: Игнорировать ячейку
    end
end

Algorithm -> "Path": Вернуть кратчайший путь к базе
deactivate Algorithm
"Танк противника" -> "Path": Получить путь к базе
activate Path
"Path" -> "Танк противника": Список координат для перемещения
deactivate Path
"Танк противника" -> Field: Начать движение по маршруту
@enduml
```

**Исходный код диаграммы взаимодействия снаряда с объектами игрового поля:**

```
@startuml
actor "Танк" as Player
participant "Игровое поле" as Field
participant "Снаряд" as Projectile
participant "Другой танк" as Enemy

Player -> Projectile: Выстрелить снаряд(урон)

loop Пока снаряд не уничтожен
    Projectile -> Field: Получить следующую ячейку на пути

    alt Ячейка — непроходимая стена
        Projectile -> Projectile: Уничтожить снаряд
    else Ячейка — разрушимый блок (в том числе база)
        Projectile -> Field: Уничтожить блок
    end
end
```

```
        Projectile -> Projectile: Уничтожить снаряд
    else Ячейка — танк
        Enemy -> Enemy: Получить урон
        Projectile -> Projectile: Уничтожить снаряд
    end
end
end
@enduml
```