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: **«**Разработка игрового приложения The Battle of Midway**»**

по курсу

**«Конструирование программного обеспечения»**

Выполнил:
студентка группы № 9413
Ужегова М.Н.

Проверил:
доцент кафедры ВПМ
Столчнев В. К.

Рязань, 2023 г.

# Оглавление

Введение....3
1 Анализ задачи....4
1.1 Разработка иерархии классов....4
1.1.1 Выделение сущностей....4
1.1.2 Зависимости между классами. Диаграмма классов....5
1.2 Алгоритмы....6
1.2.1 Алгоритм работы приложения во время движения самолета игрока....6
1.2.2 Алгоритм работы приложения при попадании оружия в вражеский самолет....7
1.2.3 Алгоритм работы приложения при использовании молнии....8
1.2.4 Алгоритм работы приложения при попадании оружия самолета игрока в бонус....9
1.2.5 Алгоритм работы навигации в приложении....9
1.3 Разработка интерфейса программы....10
1.3.1 Интерфейс главного меню....10
1.3.2 Интерфейс пункта меню «Новая игра»....10
1.3.3 Интерфейс окна с количеством очков....13
1.3.4 Интерфейс пункта меню «Рекорды»....15
1.3.5 Интерфейс пункта меню «Справка»....15
2 Написание программы....16
2.1.1 Описание разработанных процедур и функций....16
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....29
2.3.1 Модель-Представление-Контроллер (Model-View-Controller)....29
2.3.2 Абстрактная фабрика (Abstract factory)....30
2.3.3 Одиночка (Singleton)....30
2.4 Описание методов рефакторинга, которые использовались при оптимизации исходного кода программы....30
2.4.1 Вынесение констант....30
2.4.2 Выделение метода....31
2.4.3 Выделение локальной переменной....32
2.4.4 Переименование метода....33
2.4.5 Изменение сигнатуры метода....33
2.5 Разработка тестов....34
2.5.1 Test Cases....34
2.5.2 Модульные тесты....36
3 Результат работы программы....36
3.1 Графическая версия....36
3.2 Консольная версия....41
Заключение....46
Приложения....46

# Введение

Необходимо разработать игровую программу The Battle of Midway.

Должны быть реализованы две реализации интерфейса программы, основанные на разработанных общих классах и библиотеках: WPF и консоль. Необходимо использовать шаблон проектирования MVC (Модель-Вид-Контроллер) и еще не менее двух шаблонов проектирования. В приложении обязательно должна использоваться многопоточность и синхронизация между потоками. Для одного из классов приложения должны быть разработаны полноценные модульные тесты. Требования к используемому программному обеспечению: ОС Windows и среда разработки Microsoft Visual Studio C# 2022.

Результатом работы является разработанная игровая программа «The Battle of Midway».

# 1 Анализ задачи

Задача курсового проекта – разработка игровой программы «The Battle of Midway».

The Battle of Midway – компьютерная игра в жанре вертикального скролл-шутера. Действие игры разворачивается над океаном. Игрок, управляя самолетом, должен отразить атаки вражеской авиации, разбить вражеские морские силы. Стартовое игровое меню приложения содержит следующие пункты: новая игра, рекорды, справка, выход.

Самолет игрока вооружен стандартным пулеметом, наносящим 10 единиц урона, а также супероружием – молнией, наносящей 10 единиц урона всем вражеским самолетам на игровом поле и отнимающей 20 единиц энергии. В начале игры у самолета 100 единиц здоровья и 100 единиц энергии. Уничтожая вражеские самолеты можно получить бонус, благодаря которому можно восполнить энергию на 10 единиц, либо получить улучшенное оружие на 20 секунд. Для получения другого вида бонуса нужно стрельнуть в бонус.

Существует 2 типа вражеских самолетов: желтый и белый. Желтые самолеты имеют 10 единиц здоровья, дают 5 очков за уничтожение и просто перемещаются по экрану. Белые самолеты имеют 40 единиц урона, дают 20 очков за уничтожение, перемещаются по экрану, стреляя снарядами.

Цель игрока – набрать как можно больше очков, чтобы попасть в таблицу рекордов. Таблица рекордов имеет не более 10 записей, отсортированных по убыванию. После игры выводится окно с результатом. Если игрок попал в таблицу рекордов, на экране запрашивается имя.

## *1.1 Разработка иерархии классов*

Model-View-Controller («Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо (рисунок 1).

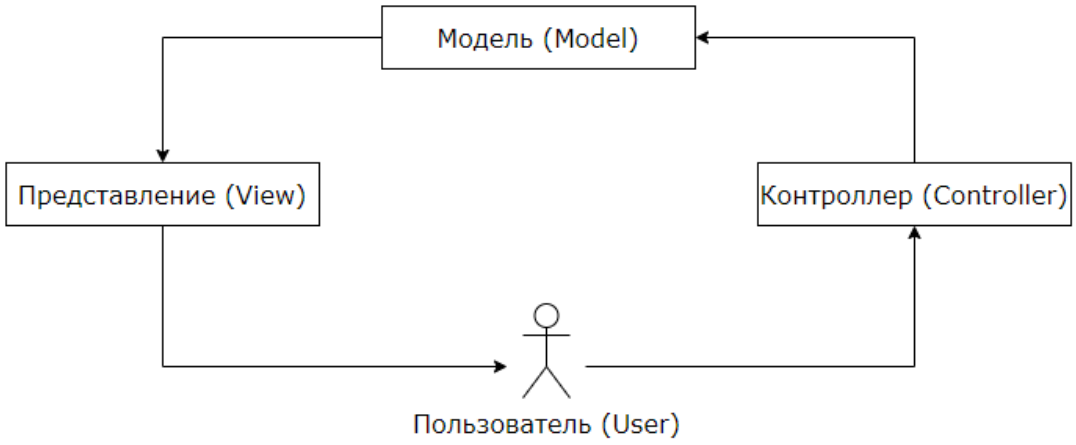


**Рисунок 1 – Визуальное представление шаблона MVC**

Описание компонент:

- Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя свое состояние.
- Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
- Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Разработка иерархии классов строится на основе шаблона MVC.

### 1.1.1 Выделение сущностей

Из предметной области игры The Battle of Midway можно выделить следующие сущности:

- Физический объект – объект, на который распространяется физика и логика игры.

- Самолет игрока – объект, которым управляет пользователь во время игрового процесса.
- Вражеские самолеты – объекты, которых игрок должен уничтожить.
- Оружие – объект, который используется самолетами для нанесения урона другим самолетам.
- Молния – стандартное оружие, уничтожающее всех противников на экране.
- Бонус – энергия или улучшенное оружие.
- Очки – единица измерения, отражающая награду за уничтожение вражеских самолетов.
- Энергия – количественный ресурс, расходуемый молнией.
- Здоровье – количественный ресурс, расходуемый при получении урона вражескими самолетами.
- Таймер – количество времени, в течение которого действует бонусное оружие.

## 1.1.2 Зависимости между классами. Диаграмма классов

### 1.1.2.1 Диаграмма классов контроллеров приложения

Диаграмма классов контроллеров приложения изображена на рисунке 2.

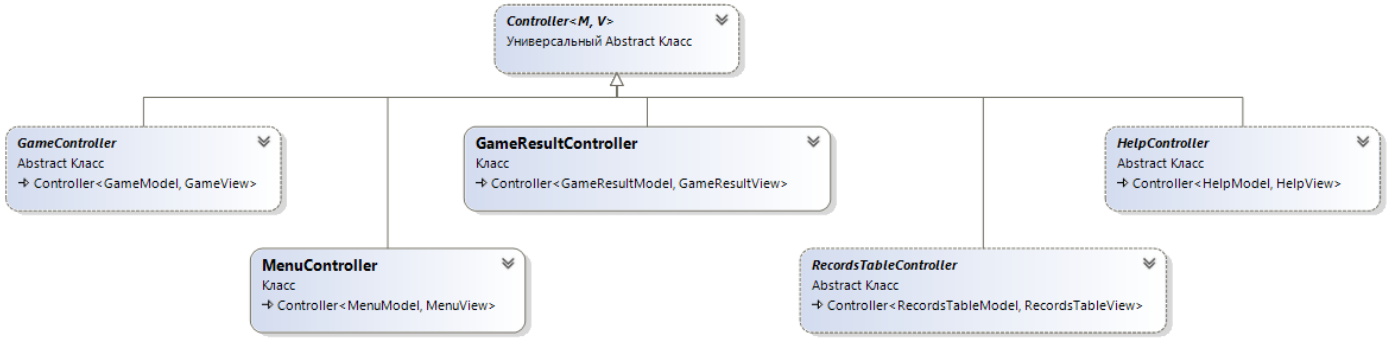


**Рисунок 2 – Диаграмма классов контроллеров**

### 1.1.2.2 Диаграмма классов моделей приложения

Диаграмма классов моделей приложения изображена на рисунке 3.

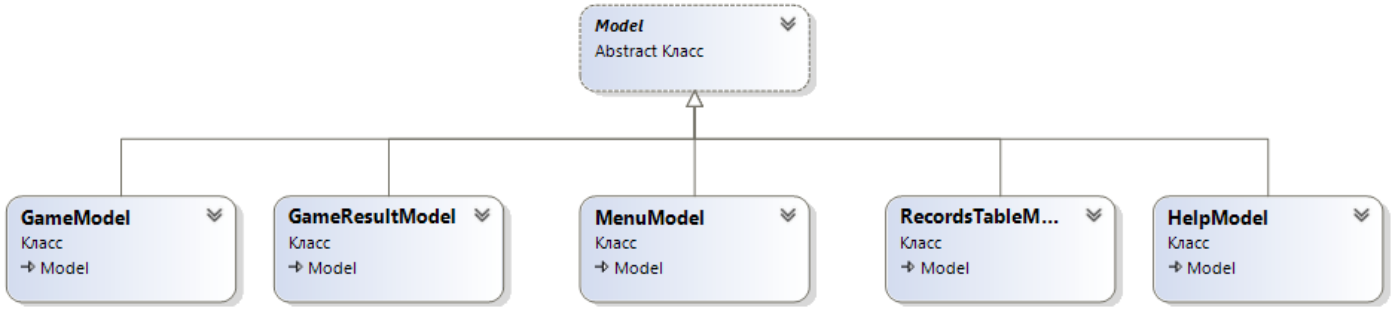


**Рисунок 3 – Диаграмма классов моделей**

### 1.1.2.3 Диаграмма классов представлений приложения

Диаграмма классов представлений приложения изображена на рисунке 4.

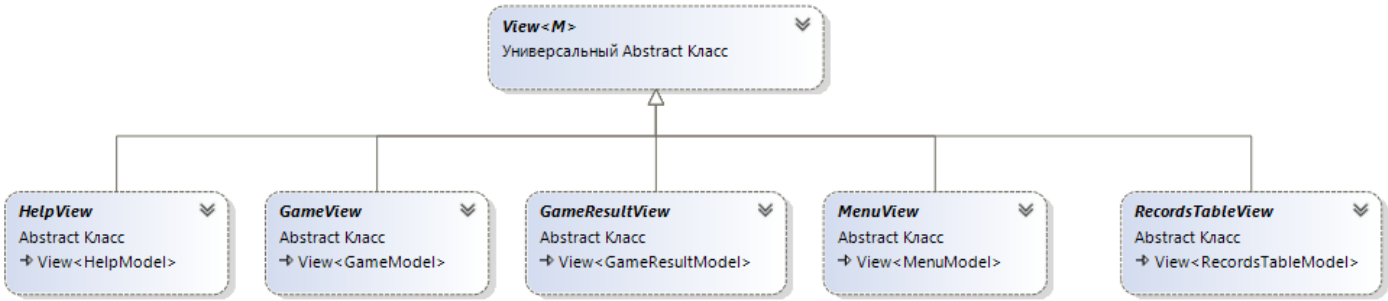


**Рисунок 4 – Диаграмма классов представлений**

### 1.1.2.4 Диаграмма классов игровых объектов

Диаграмма классов игровых объектов приложения изображена на рисунке 5.

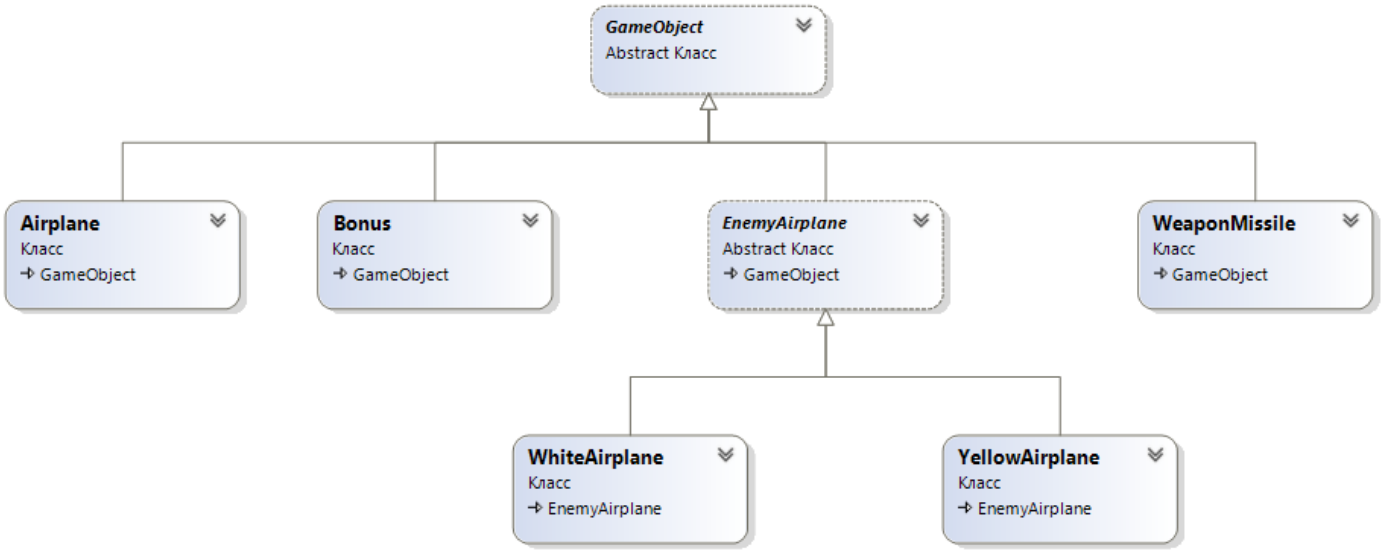


**Рисунок 5 – Диаграмма классов игровых объектов**

### 1.1.2.5 Диаграмма классов оружий самолета

Диаграмма классов оружий самолета приложения изображена на рисунке 6.

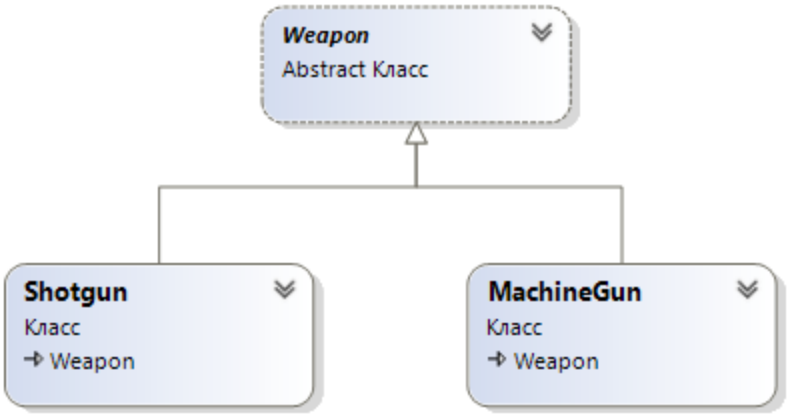


**Рисунок 6 – Диаграмма классов оружий самолета**

## *1.2 Алгоритмы*

### 1.2.1 Алгоритм работы приложения во время движения самолета игрока

Логика работы приложения во время движения самолета игрока представлена на диаграмме последовательности (рисунок 7).

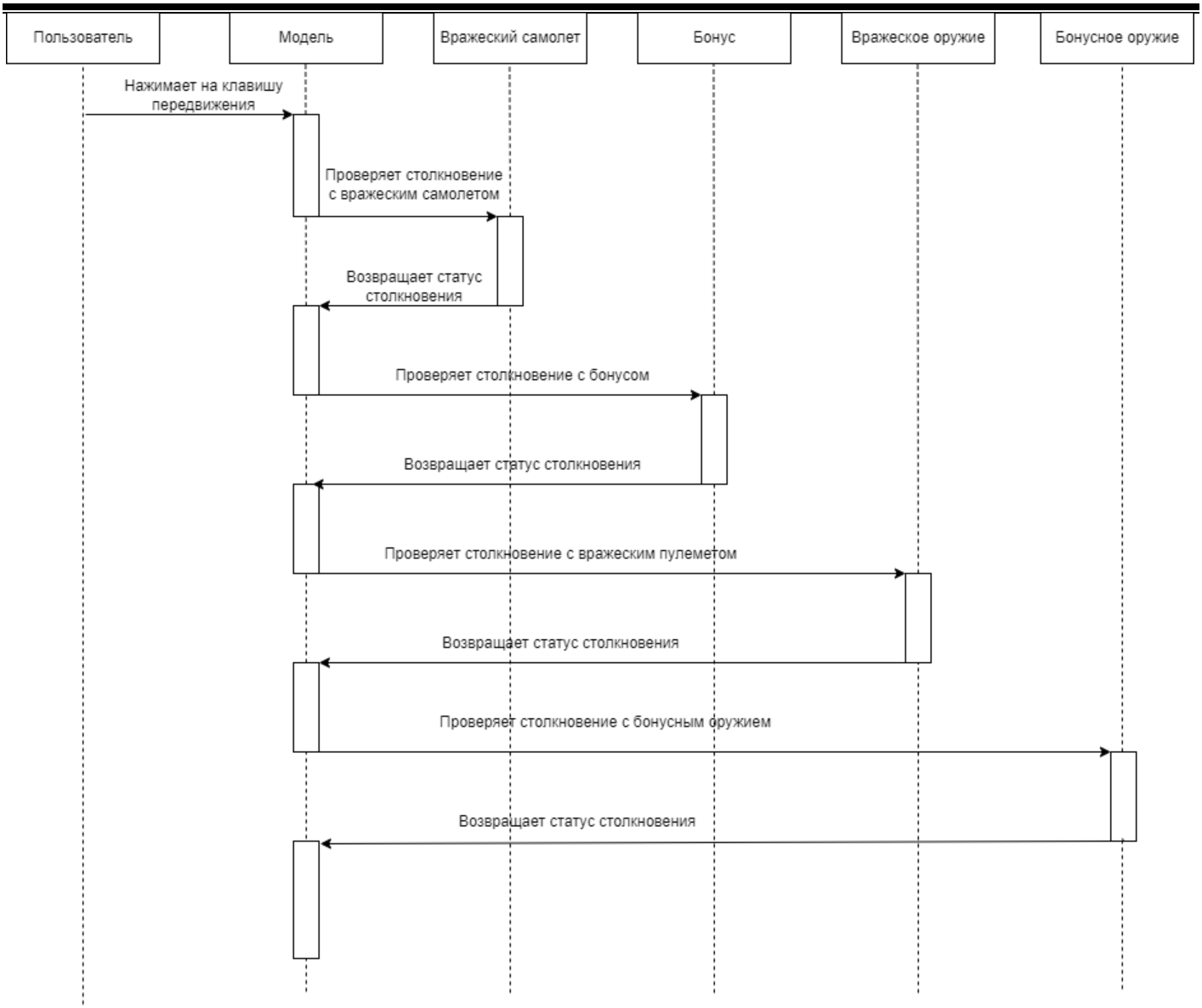


**Рисунок 7 – Диаграмма последовательности для передвижения самолета игрока**

Пользователь нажимает на клавишу передвижения самолета. При столкновении с вражеским самолетом или вражеским оружием самолет игрока получает урон (понижается количество здоровья). При столкновении с бонусом самолет игрока получает бонус и количество энергии увеличивается. При столкновении с бонусным оружием самолет игрока получает оружие; при вторном столкновении – получает улучшенное оружие; запускается таймер действия бонусного оружия.

### 1.2.2 Алгоритм работы приложения при попадании оружия в вражеский самолет

Логика работы приложения при попадании оружия самолета игрока в вражеский самолет описана на рисунке 8.

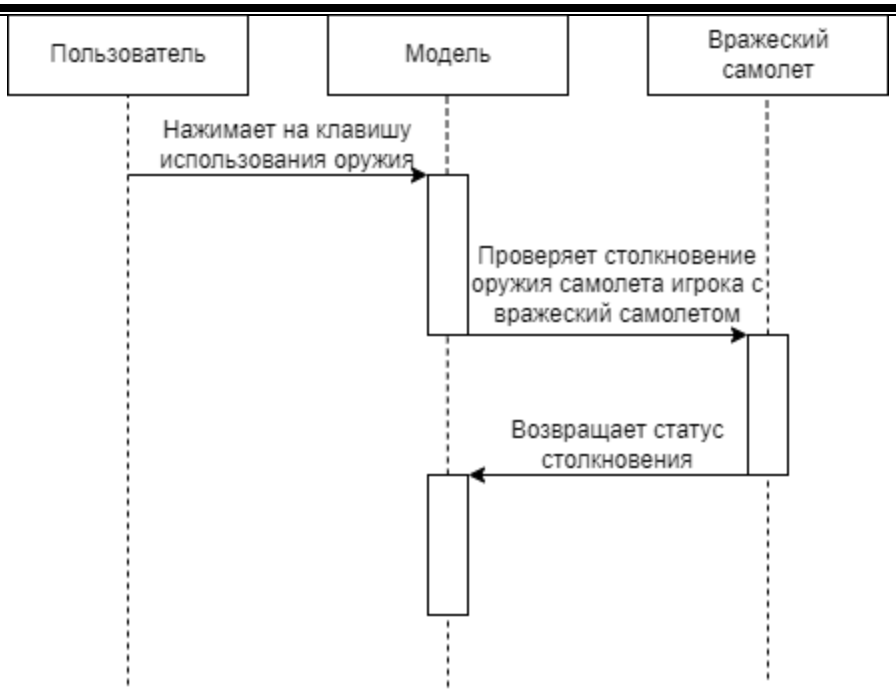


**Рисунок 8 – Диаграмма последовательности для попадания оружия игрока в вражеский самолет**

После нажатия на клавишу использования оружия должна происходить проверка столкновения оружия самолета игрока с вражеским самолетом. Если столкновение произошло, то количество очков увеличивается. Если количество урона равно или превысило количество здоровья вражеского самолета, то вражеский самолет уничтожается.

### 1.2.3 Алгоритм работы приложения при использовании молнии

Логика работы приложения при использовании молнии представлена на рисунке 9.

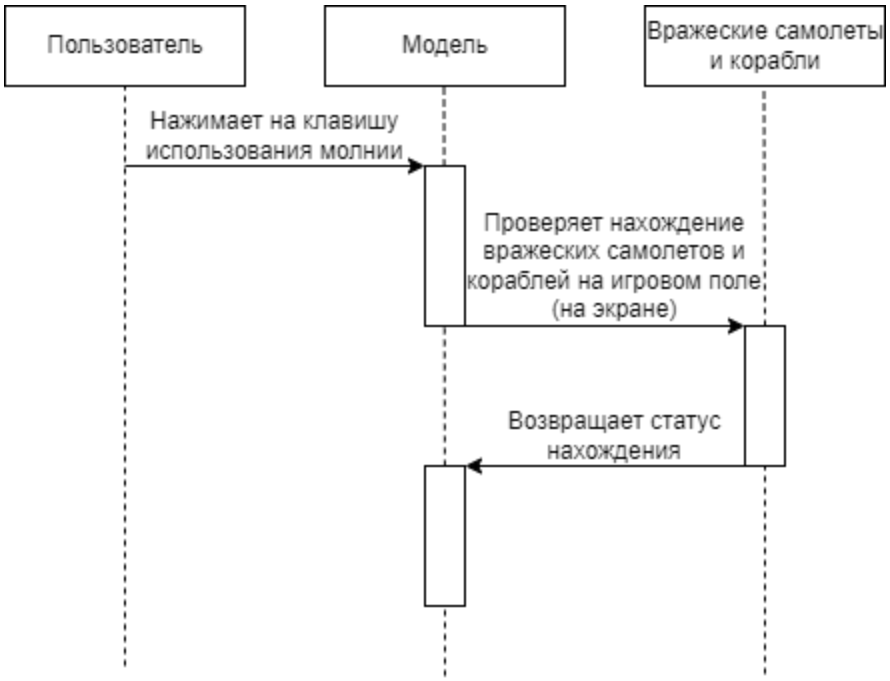


**Рисунок 9 – Диаграмма последовательности для использования молнии**

После нажатия на клавишу использования молнии должна происходить проверка наличия на игровом поле вражеских объектов (самолетов и кораблей). Если вражеские объекты находятся в момент использования молнии на экране, то они получают урон, а количество очков увеличивается. Если количество урона равно или превысило количество здоровья вражеских объектов, то они уничтожаются.

### 1.2.4 Алгоритм работы приложения при попадании оружия самолета игрока в бонус

Логика работы приложения при попадании оружия самолета игрока в бонус описана на рисунке 10.


**Рисунок 10 – Диаграмма последовательности для попадания оружия игрока в бонус**

После нажатия на клавишу использования оружия должна происходить проверка столкновения оружия самолета игрока с бонусом. Если столкновение произошло, то один вид бонуса сменяется на другой.

### 1.2.5 Алгоритм работы навигации в приложении

Навигация в приложении представлена на рисунке 11.

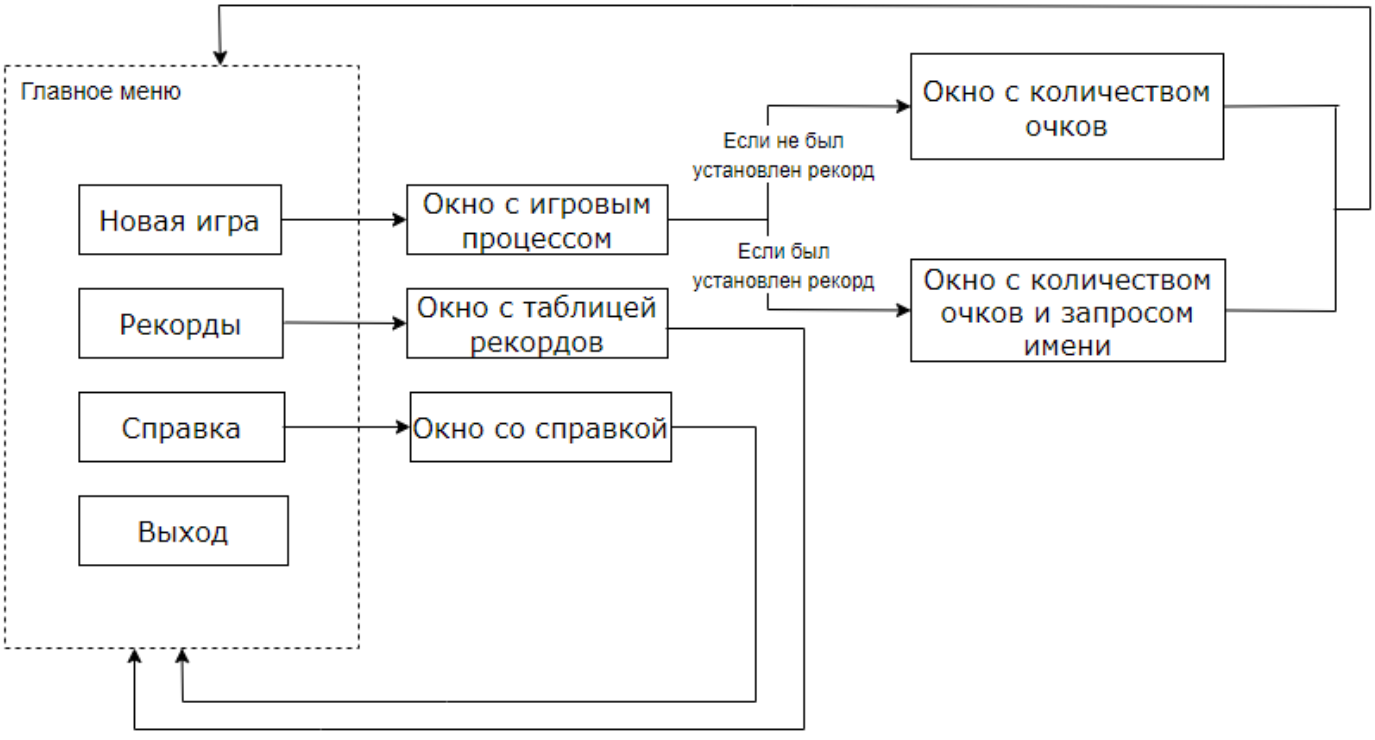


**Рисунок 11 – Навигация в приложении**

Главное меню должно состоять из 4 пунктов: новая игра, рекорды, справка и выход.

По нажатию пункта меню «Новая игра» должно открываться окно с игровым процессом. После выхода из игрового процесса должно открываться либо окно с количеством очков, если не был установлен рекорд, либо окно с количеством очков и запросом имени, если был установлен рекорд. После закрытия этого окна должно открываться окно с главным меню.

По нажатию пункта меню «Рекорды» должно открываться окно с таблицей рекордов. После выхода должно открываться окно с главным меню.

По нажатию пункта меню «Справка» должно открываться окно со справкой. После выхода должно открываться окно с главным меню.

По нажатию пункта меню «Выход» приложение должно закрыться.

## *1.3 Разработка интерфейса программы*

### 1.3.1 Интерфейс главного меню

Интерфейс главного меню изображен на рисунке 12.

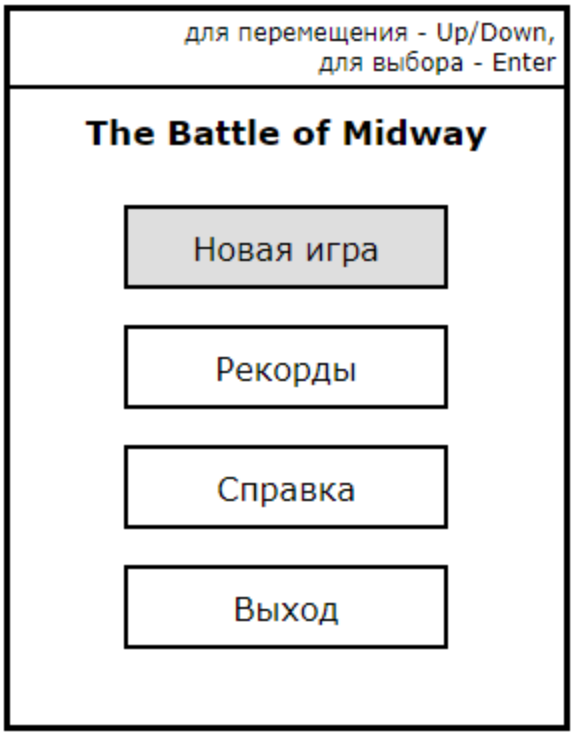


**Рисунок 12 – Главное меню**

Главное меню содержит следующие пункты: «Новая игра», «Рекорды», «Как играть», «Выход». Для навигации по пунктам меню (см.п. 1.2.5) предназначены клавиши стрелок Up/Down. Пункт меню, находящийся в фокусе, выделяется другим цветом. Для подтверждения выбора пункта меню предназначена клавиша Enter. При выборе пункта меню осуществляется мгновенный переход на окно соответствующего пункта.

### 1.3.2 Интерфейс пункта меню «Новая игра»

После выбора пункта меню «Новая игра» начинается игровой процесс.

#### 1.3.2.1 Интерфейс игрового поля

Интерфейс игрового поля представлен на рисунке 13.

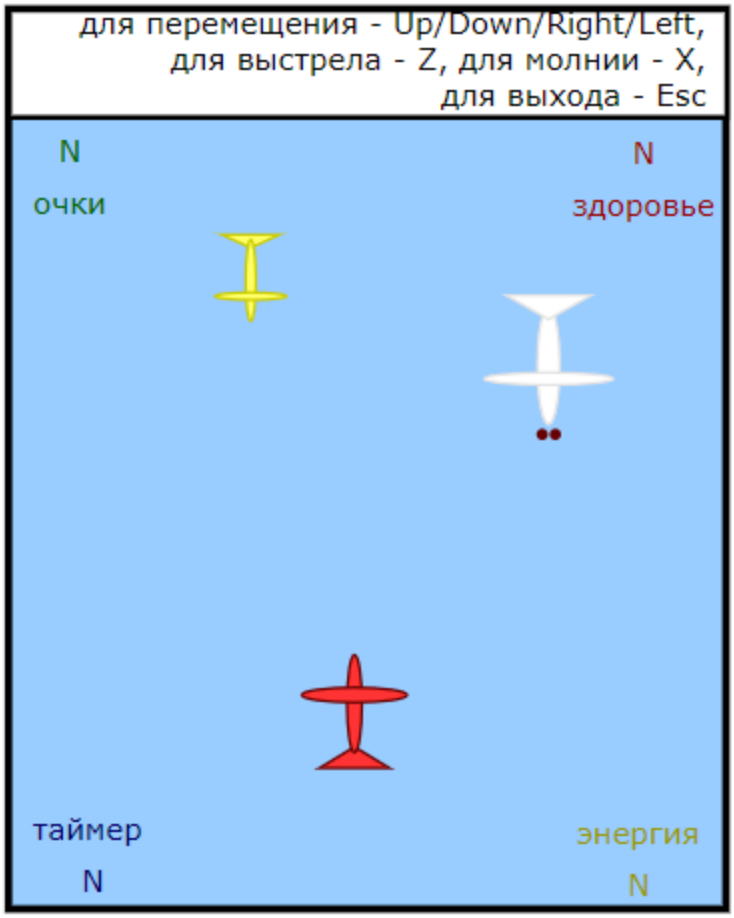


**Рисунок 13 – Игровое поле**

Игровой процесс с видом сверху. На игровом поле должно отображаться:

- в левой верхней части окна – актуальное количество набранных очков;
- в правой верхней части окна – здоровье игрока;
- в правой нижней части окна – актуальное количество энергии;
- в левой нижней части окна – таймер, если получено бонусное оружие.

Для перемещения игрока во время игрового процесса необходимо нажимать на клавиши стрелок Up/Down/Right/Left. Для выстрела из оружия необходимо производить нажатие клавиши Z. Для использования молнии необходимо нажимать на клавишу X.

Игра заканчивается, когда здоровье игрока становится равным нулю. Для выхода из игрового процесса предназначена клавиша Esc.

### 1.3.2.2 Интерфейсы игровых оружий

Ниже представлены интерфейсы для всех видов игрового оружия. Пунктиром показано дальнейшее направление оружия.

Интерфейс с использованием пулемета показан на рисунке 14.

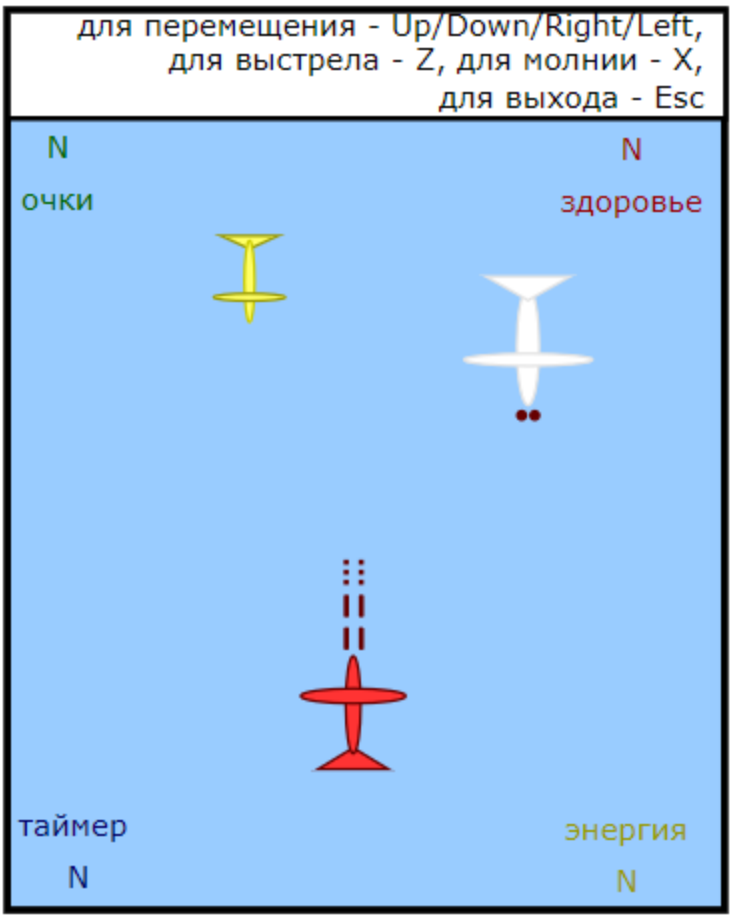


**Рисунок 14 – Применение пулемета**

Интерфейс с использованием дробовика показан на рисунке 15.

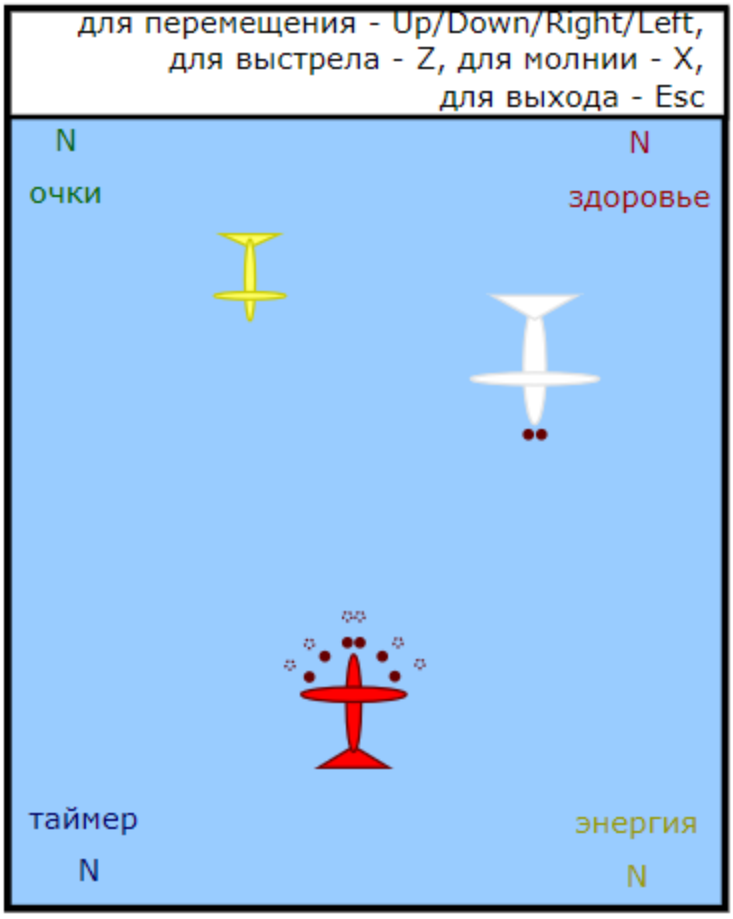


**Рисунок 15 – Применение дробовика**

### 1.3.2.3 Интерфейс молнии

Интерфейс с использованием молнии показан на рисунке 16.

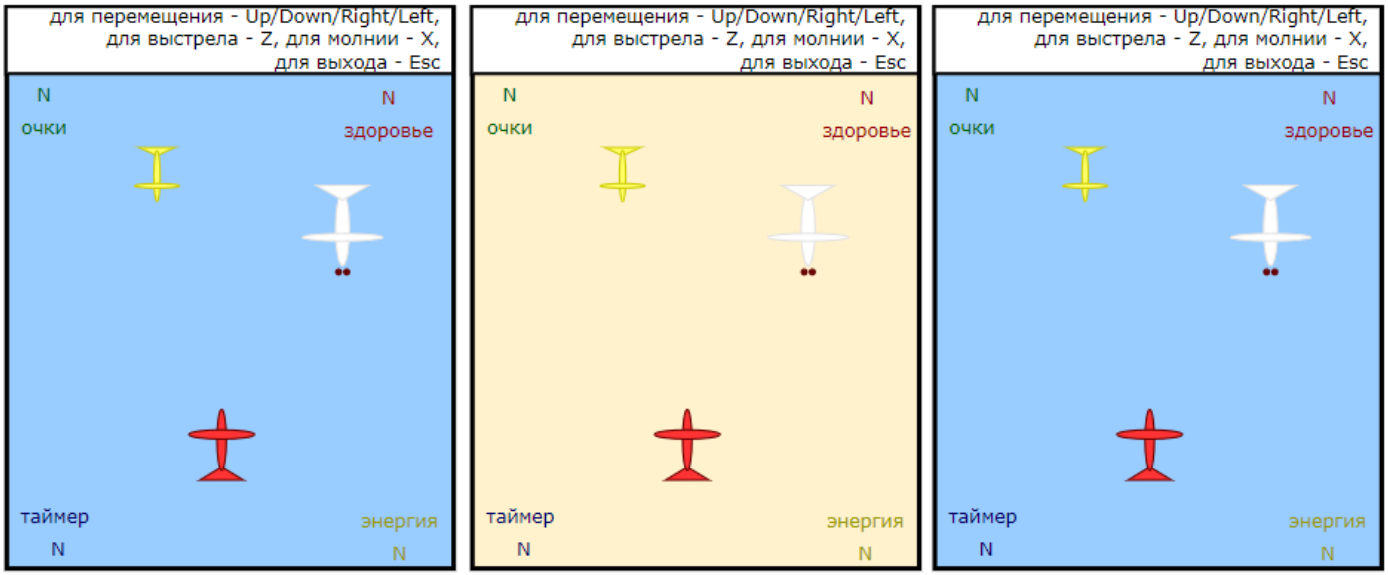


**Рисунок 16 – Применение молнии**

При использовании молнии несколько раз меняется фон.

### 1.3.2.4 Интерфейсы бонусов

После уничтожения вражеских самолетов выпадает бонус. После выстрела в бонус он сменяется другим бонусом. Интерфейсы бонусов представлены на рисунке 17.

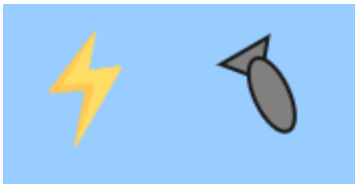

**Рисунок 17 – Бонусы**

### 1.3.3 Интерфейс окна с количеством очков

После завершения игрового процесса появляется окно с количеством набранных очков (рисунок 18).

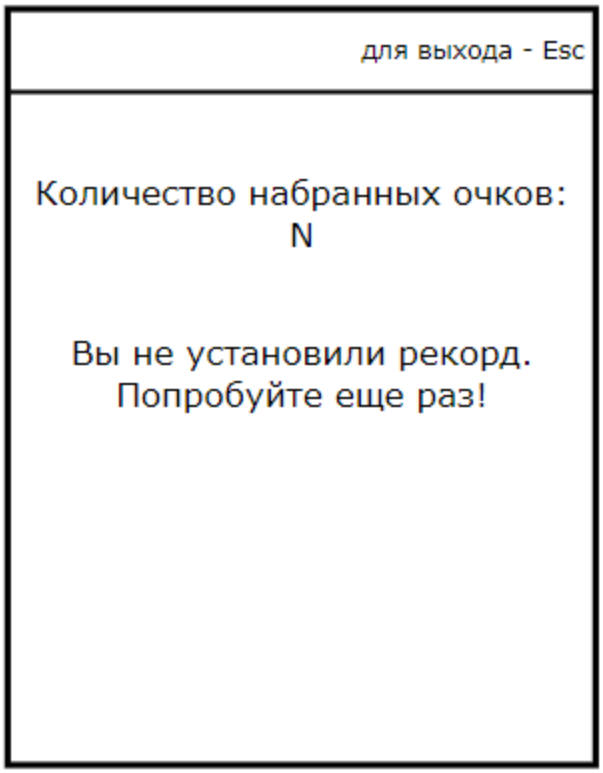


**Рисунок 18 – Количество набранных очков**

Для выхода из окна с количеством набранных очков нужно нажимать Esc. После выхода открывается окно с главным меню. В случае установления рекорда пользователю выводится количество набранных очков и просьба о вводе имени (рисунок 19).

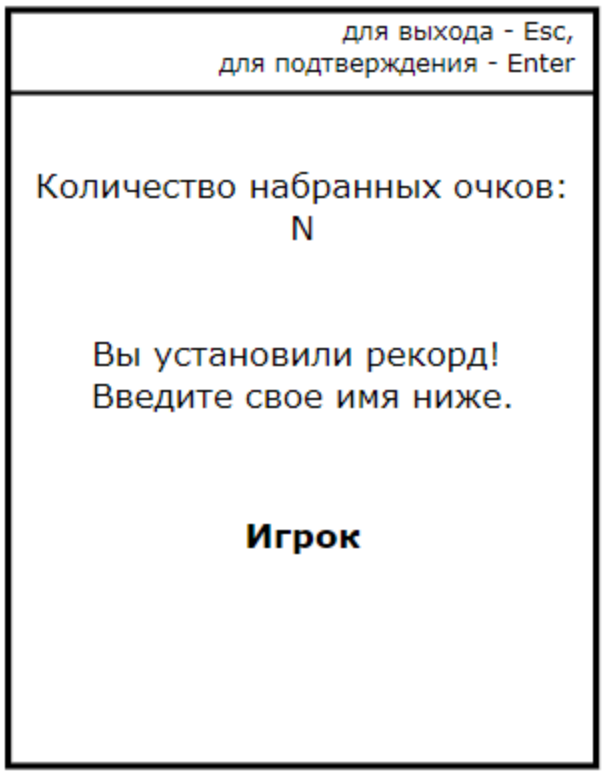


**Рисунок 19 – Количество набранных очков и ввод имени**

Для ввода имени игрока используются любые символы. Максимальное количество символов – 15. Для подтверждения имени предназначена клавиша Enter, для выхода - Escape. После подтверждения открывается окно с главным меню.

### 1.3.4 Интерфейс пункта меню «Рекорды»

Интерфейс пункта меню «Рекорды» изображен на рисунке 20.

| для выхода - Esc | |
|---|---|
| Имя игрока | Очки |
| . . . . . | N |
| . . . . . | N |
| . . . . . | N |
| . . . . . | N |
| . . . . . | N |
| . . . . . | N |
| . . . . . | N |
| . . . . . | N |
| . . . . . | N |
| . . . . . | N |

**Рисунок 20 – Таблица рекордов**

Таблица рекордов содержит две колонки: имя игрока и количество очков. Записи в таблице рекордов должны быть отсортированы по убыванию. Максимальное количество записей в таблице рекордов – 10. Для выхода из таблицы рекордов нужно нажать на Esc. После выхода открывается окно с главным меню.

### 1.3.5 Интерфейс пункта меню «Справка»

Интерфейс пункта меню «Справка» изображен на рисунке 21.

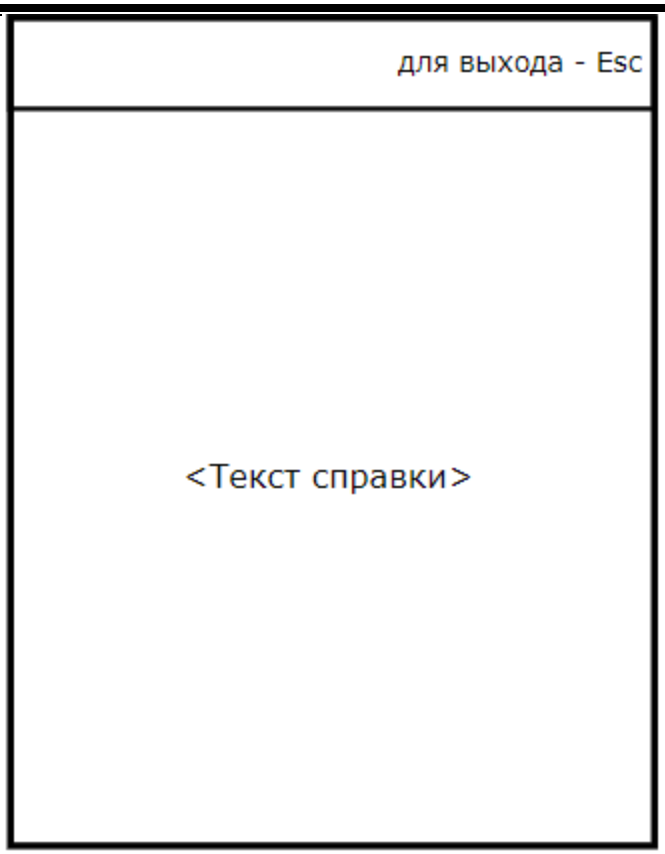


**Рисунок 21 – Справка**

Справка должна содержать краткое описание и правила игры, включая особенности каждого вида оружия и бонусов.

# 2 Написание программы

### 2.1.1 Описание разработанных процедур и функций

**Таблица 1 – Описание разработанных процедур и функций**

| Класс | Метод | Параметры | Назначение |
|---|---|---|---|
| **Core** | | | |
| GameController | `public void BackToMenu()` | - | Возвращает в меню |
| | `public override void Start()` | - | Запускает контроллер игры |
| | `public override void Stop()` | - | Останавливает контроллер игры |
| Airplane | `public void MoveUp()` | - | Перемещает самолет вверх |
| | `public void MoveDown()` | - | Перемещает самолет вниз |
| | `public void MoveRight()` | - | Перемещает самолет вправо |
| | `public void MoveLeft()` | - | Перемещает самолет влево |
| | `public void Shoot()` | - | Самолет стреляет |
| | `public void UseLightning()` | - | Самолет использует молнию |
| | `public void TakeDamage(int parDamage)` | `parDamage - урон` | Самолет получает урон |
| | `public override void Update()` | - | Обновляет состояние самолета |

| | | | |
|---|---|---|---|
| Bonus | public void ToggleBonus() | - | Меняет бонус |
| | public override void Update() | - | Обновляет оружие или количество энергии в зависимости от взятого бонуса |
| EnemyAirplane | public void TakeDamage(int parDamage) | parDamage - урон | Вражеский самолет получает урон |
| GameObject | public static bool CheckCollision (GameObject parObj1, GameObject parObj2) | parObj1 – первый объект, parObj2 – второй объект | Проверяет столкновение двух объектов. Возвращает булево значение |
| | public abstract void Update() | - | Обновляет состояние объекта |
| WhiteAirplane | private void Shoot() | - | Белый вражеский самолет стреляет |
| | private void CheckShoot() | - | Проверяет возможность выстрелить |
| | public override void Update() | - | Обновляет состояние белого вражеского самолета |
| YellowAirplane | public override void Update() | - | Обновляет состояние зеленого вражеского самолета |
| WeaponMissile | public override void Update() | - | Обновляет состояния снаряда |
| MachineGun | public override void Shoot(Airplane parAirplane) | parAirplane - самолет | Пулемет стреляет |
| Shotgun | private void CountDown(object? sender, ElapsedEventArgs e) | - | Отсчитывает время использования дробовика |
| | public override void Shoot(Airplane parAirplane) | parAirplane - самолет | Дробовик стреляет |
| Weapon | public abstract void Shoot(Airplane parAirplane) | parAirplane - самолет | Оружие стреляет |
| GameModel | public void Init() | - | Инициализирует модель игры |
| | private void StartGenerateEnemies() | - | Запускает генерацию вражеских самолетов |
| | public void Start() | - | Начинает игру |
| | public void Stop() | | Останавливает игру |
| | public void AddGameObject(GameObject parGameObject) | parGameObject – игровой объект | Добавляет объект |
| | public void RemoveGameObject(GameObject parGameObject) | parGameObject – игровой объект | Удаляет объект |
| | public void End() | - | Завершает игру |
| GameResultController | public void SetGameResult(int parScore) | parScore - очки | Устанавливает результат игры |
| | public void BackToMenu() | - | Возвращает в меню |
| GameResultModel | public void SetIsRecord() | - | Устанавливает свойство поставлен ли рекорд |
| | public string GetEndText() | - | Получает текст, который выводится в конце игры и возвращает его |

| | | | |
|---|---|---|---|
| | `public bool SaveRecord()` | | Сохраняет рекорд. Возвращает булево значение |
| GameResultView | `private void Redraw()` | - | Перерисовывает представление результата игры |
| HelpController | `public void BackToMenu()` | | Возвращает в меню |
| HelpModel | `public string GetHelpText()` | - | Получает текст справки |
| | `public void NeedRedraw()` | - | Вызывает событие о необходимости перерисовки |
| HelpView | `private void Redraw()` | - | Перерисовывает представление справки |
| MenuController | `public override void Start()` | - | Запускает меню |
| | `public override void Stop()` | - | Останавливает меню |
| | `private void MenuClickHandler(MenuItem parSelectedMenuItem)` | `parSelectedMenuItem` - выбранный пункт меню | Обрабатывает нажатие на выбранный пункт меню |
| MenuItemTextFormer | `public static string GetTitle(MenuItem parMenuItem)` | `parMenuItem` – пункт менб | Получает текст для пункта меню и возвращает его |
| MenuModel | `public void ChooseNextMenuItem()` | - | Выбирает следующий пункт меню |
| | `public void ChoosePreviousMenuItem()` | - | Выбирает предыдущий пункт меню |
| | `public void FocusMenuItem(int parIndex)` | `parIndex` – индекс пункта меню | Выделяет пункт меню |
| | `public void Enter()` | - | Нажимает на выбранный пункт меню |
| | `public void NeedRedraw()` | - | Вызывает событие о необходимости перерисовки |
| MenuView | `private void Redraw()` | - | Перерисовывает меню |
| MainMenu | `public MenuItem GetMenuItemByIndex(int parIndex)` | `parIndex` - индекс пункта меню | Получает пункт меню по индексу и возвращает его |
| Controller<M, V> | `public virtual void Start()` | - | Запускает контроллер MVC |
| | `public virtual void Stop()` | - | Останавливает контроллер MVC |
| View<M> | `public abstract void Draw()` | - | Рисует представление |
| | `public virtual void Start()` | - | Запускает представление |
| | `public virtual void Stop()` | - | Останавливает представление |
| RecordsTableController | `public override void Start()` | | Запускает контроллер таблицы рекордов |
| | `public void BackToMenu()` | - | Возвращает в меню |
| GameRecord | `public override string ToString()` | - | Получает строку |
| RecordsTableModel | `public void NeedRedraw()` | - | Вызывает событие о необходимости перерисовки |

| | | | |
|---|---|---|---|
| | `public List<GameRecord> GetRecords()` | - | Получает таблицу рекордов |
| | `public void UpdateRecords()` | - | Обновляет таблицу рекордов |
| RecordsTableView | `private void Redraw()` | - | Перерисовывает представление таблицы рекордов |
| RecordsTableRepository | `public static bool IsRecord(int parScore)` | `parScore - очки` | Проверяет установлен ли рекорд. Возвращает булево значение |
| | `public static void AddRecord(GameRecord parRecord)` | `parRecord - рекорд` | Добавляет новый рекорд |
| | `public static void SaveRecords(List<GameRecord> parRecords)` | `parRecords - рекорды` | Сохраняет рекорды в файл |
| | `public static List<GameRecord> GetAllRecords()` | - | Получает все рекорды |
| | `private static void SortRecords(List<GameRecord> parRecords)` | `parRecords - рекорды` | Сортирует рекорды по убыванию |
| AbstractControllersFactory | `public abstract MenuController CreateMenuController()` | - | Создает контроллер меню |
| | `public abstract GameController CreateGameController()` | - | Создает контроллер игры |
| | `public abstract RecordsTableController CreateRecordsTableController()` | - | Создает контроллер таблицы рекордов |
| | `public abstract HelpController CreateHelpController()` | - | Создает контроллер справки |
| | `public abstract GameResultController CreateGameResultController()` | - | Создает контроллер результата игры |
| MainApplication | `private void Init()` | - | Инициализирует приложение |
| | `private void InitMenuController()` | - | Инициализирует контроллер меню |
| | `private void InitGameController()` | - | Инициализирует контроллер игры |
| | `private void InitRecordsTableController()` | - | Инициализирует контроллер таблицы рекордов |
| | `private void InitHelpController()` | - | Инициализирует контроллер справки |
| | `private void InitGameResultController()` | - | Инициализирует контроллер результата игры |
| | `public virtual void Start()` | - | Запускает приложение |
| **WPF** | | | |
| WPFGameController | `private void OnClose(object? sender, EventArgs e)` | - | Обрабатывает закрытие окна |
| | `public void KeyDownHandler(object sender, KeyEventArgs e)` | - | Обрабатывает нажатие клавиши клавиатуры |
| | `public void Window_KeyUp(object sender, KeyEventArgs e)` | - | Обрабатывает отпускание клавиши клавиатуры |
| | `public override void Start()` | - | Запускает WPF контроллер игры |
| | `public override void Stop()` | - | Останавливает WPF контроллер игры |

| | | | |
|---|---|---|---|
| AirplaneView | public override void Draw(Canvas parCanvas) | parCanvas - элемент Canvas | Рисует самолет игрока |
| | private void DrawWeaponTimer(Canvas parCanvas) | parCanvas - элемент Canvas | Рисует таймер оружия |
| | private void DrawScore(Canvas parCanvas) | parCanvas - элемент Canvas | Рисует очки |
| | private void DrawEnergy(Canvas parCanvas) | parCanvas - элемент Canvas | Рисует энергию |
| | private void DrawHealth(Canvas parCanvas) | parCanvas - элемент Canvas | Рисует здоровье |
| | private void DrawShape(Canvas parCanvas) | parCanvas - элемент Canvas | Рисует форму самолета игрока |
| BonusView | public override void Draw(Canvas parCanvas) | parCanvas - элемент Canvas | Рисует бонус |
| GameObjectView | public abstract void Draw(Canvas parCanvas) | parCanvas - элемент Canvas | Рисует WPF представление игрового объекта |
| MissileView | public override void Draw(Canvas parCanvas) | parCanvas - элемент Canvas | Рисует представление снаряда |
| WhiteAirplaneView | public override void Draw(Canvas parCanvas) | parCanvas - элемент Canvas | Рисует белый вражеский самолет |
| YellowAirplaneView | public override void Draw(Canvas parCanvas) | parCanvas - элемент Canvas | Рисует желтый вражеский самолет |
| WPFGameView | private void LightningDraw() | - | Рисует молнию |
| | private void ViewInit() | - | Инициализирует WPF представление игры |
| | private void SetWindow() | - | Настраивает окно |
| | private void StartDrawing() | - | Начинает отрисовку |
| | public override void Draw() | - | Рисует карту |
| | private GameObjectView GetObjectView(GameObject parObject) | parObject - Объект | Получает WPF представление объекта |
| | public override void Start() | - | Запускает WPF представление игры |
| | public override void Stop() | - | Останавливает WPF представление игры |
| WPFGameResultController | private void KeyDownHandler(object sender, KeyEventArgs e) | - | Обрабатывает нажатие клавиши клавиатуры |
| | public override void Start() | - | Запускает WPF контроллер результата игры |
| | public override void Stop() | - | Останавливает WPF контроллер результата игры |
| WPFGameResultView | private void SetupCanvas() | - | Настраивает элемент Canvas |
| | public override void Draw() | - | Рисует WPF представление результата игры |
| | public override void Start() | - | Запускает WPF представление результата игры |
| | public override void Stop() | - | Останавливает WPF представление результата игры |
| WPFHelpController | private void KeyDownHandler(object sender, KeyEventArgs e) | - | Обрабатывает нажатие клавиши клавиатуры |
| | public override void Start() | - | Запускает WPF |

| | | | |
|---|---|---|---|
| | | | контроллер справки |
| | `public override void Stop()` | - | Останавливает WPF контроллер справки |
| WPFHelpView | `private void SetupCanvas()` | - | Настраивает элемент Canvas |
| | `public override void Draw()` | - | Рисует WPF представление справки |
| | `public override void Start()` | - | Запускает WPF представление справки |
| | `public override void Stop()` | - | Останавливает WPF представление справки |
| WPFMenuController | `private void KeyDownHandler(object sender, KeyEventArgs e)` | - | Обрабатывает нажатие клавиши клавиатуры |
| | `public override void Start()` | - | Запускает WPF контроллер меню |
| | `public override void Stop()` | - | Останаливает WPF контроллер меню |
| WPFMenuView | `private void SetupCanvas()` | - | Настраивает элемент Canvas |
| | `public override void Draw()` | - | Рисует WPF представление меню |
| | `public override void Start()` | - | Запускает WPF представление меню |
| | `public override void Stop()` | - | Останавливает WPF представление меню |
| WPFRecordsTable Controller | `private void KeyDownHandler(object sender, KeyEventArgs e)` | - | Обрабатывает нажатие клавиши клавиатуры |
| | `public override void Start()` | - | Запускает WPF контроллер таблицы рекордов |
| | `public override void Stop()` | - | Останавливает WPF контроллер таблицы рекордов |
| WPFRecordsTableView | `private void SetupCanvas()` | - | Настраивает элемент Canvas |
| | `public override void Draw()` | - | Рисует WPF представление таблицы рекордов |
| | `public override void Start()` | - | Запускает WPF представление таблицы рекордов |
| | `public override void Stop()` | - | Останавливает WPF представление таблицы рекордов |
| WindowKeeper | `public Window GetWindow()` | - | Получает окно |
| WPFApplication | `private void ConfigureWindow()` | - | Конфигурирует окно приложения |
| | `public override void Start()` | - | Запускает WPF приложение |
| WPFControllersFactory | `public override GameController CreateGameController()` | - | Создает контроллер игры |
| | `public override GameResultController CreateGameResultController()` | - | Создает контроллер результата игры |

| | | | |
|---|---|---|---|
| | `public override HelpController CreateHelpController()` | - | Создает контроллер справки |
| | `public override MenuController CreateMenuController()` | - | Создает контроллер меню |
| | `public override RecordsTableController CreateRecordsTableController()` | - | Создает контроллер таблицы рекордов |
| **Console** | | | |
| ConsoleGameController | `public void ReadKeysStart()` | - | Запускает считывание нажатых клавиш в консоли |
| | `public void ReadKeysStop()` | - | Останавливает считывание клавиш |
| | `public override void Start()` | - | Запускает консольный контроллер игры |
| | `public override void Stop()` | - | Останавливает консольный контроллер игры |
| AirplaneView | `public override void Draw()` | - | Рисует консольное представление самолета игрока |
| BonusView | `public override void Draw()` | - | Рисует консольное представление бонуса |
| GameObjectView | `public abstract void Draw()` | - | Рисует консольное представление игрового объекта |
| MissileView | `public override void Draw()` | - | Рисует консольное представление снаряда |
| WhiteAirplaneView | `public override void Draw()` | - | Рисует белый вражеский самолет |
| YellowAirplaneView | `public override void Draw()` | - | Рисует консольное представление желтого вражеского самолета |
| ConsoleGameView | `private void ViewInit()` | - | Инициализирует консольное представление игры |
| | `private void SetWindow()` | - | Настраивает окно |
| | `private void StartDrawing()` | - | Начинает отрисовку |
| | `private void LightningDraw()` | - | Рисует молнию |
| | `public override void Draw()` | - | Рисует карту |
| | `private GameObjectView GetObjectView(GameObject parObject)` | `parObject` - объект | Получает представление объекта |
| | `public override void Start()` | - | Запускает консольное представление игры |
| | `public override void Stop()` | - | Останавливает консольное представление игры |
| ConsoleGameResult Controller | `public void ReadKeysStart()` | - | Запускает считывание нажатых клавиш в консоли |
| | `public void ReadKeysStop()` | - | Останавливает считывание клавиш |
| | `public override void Start()` | - | Запускает консольный контроллер результата игры |

| | | | |
|---|---|---|---|
| | `public override void Stop()` | - | Останавливает консольный контроллер результата игры |
| ConsoleGameResultView | `public void SetupWindow()` | - | Настраивает окно |
| | `private void InitialDraw()` | - | Инициализирует начальную отрисовку при запуске |
| | `public override void Draw()` | - | Рисует введенное имя победившего игрока |
| | `public override void Start()` | - | Запускает консольное представление результата игры |
| | `public override void Stop()` | - | Останавливает консольное представление результата игры |
| ConsoleHelpController | `public void ReadKeysStart()` | - | Запускает считывание нажатых клавиш в консоли |
| | `public void ReadKeysStop()` | - | Останавлиает считывание клавиш |
| | `public override void Start()` | - | Запускает консольный контроллер справки |
| | `public override void Stop()` | - | Останавливает консольный контроллер справки |
| ConsoleHelpView | `public override void Draw()` | - | Рисует консольное представление справки |
| | `public override void Start()` | - | Запускает консольное представление справки |
| | `public override void Stop()` | - | Останавливает консольное представление справки |
| ConsoleMenuController | `public void ReadKeysStart()` | - | Запускает считывание нажатых клавиш в консоли |
| | `public void ReadKeysStop()` | - | Останавливает считывание клавиш |
| | `public override void Start()` | - | Запускает консольный контроллер меню |
| | `public override void Stop()` | - | Останавливает консольный контроллер меню |
| ConsoleMenuView | `public void SetupWindow()` | - | Настраивает окно |
| | `public override void Draw()` | - | Рисует консольное представление меню |
| | `public override void Start()` | - | Запускает консольное представление меню |
| | `public override void Stop()` | - | Останавливает консольное представление меню |
| ConsoleRecordsTable | `public void ReadKeysStart()` | - | Запускает считывание |

| | | | |
|---|---|---|---|
| Controller | | | нажатых клавиш в консоли |
| | `public void ReadKeysStop()` | - | Останавливает считывание клавиш |
| | `public override void Start()` | - | Запускает консольный контроллер таблицы рекордов |
| | `public override void Stop()` | - | Останавливает консольный контроллер таблицы рекордов |
| ConsoleRecordsTableView | `public override void Draw()` | - | Рисует консольное представление таблицы рекордов |
| | `public override void Start()` | - | Запускает консольное представление таблицы рекордов |
| | `public override void Stop()` | - | Останавливает консольное представление таблицы рекордов |
| ConsoleApplication | `private void ConfigureWindow()` | - | Конфигурирует окно консольного приложения |
| ConsoleControllersFactory | `public override GameController CreateGameController()` | - | Создает контроллер игры |
| | `public override GameResultController CreateGameResultController()` | - | Создает контроллер результата игры |
| | `public override HelpController CreateHelpController()` | - | Создает контроллер справки |
| | `public override MenuController CreateMenuController()` | - | Создает контроллер меню |
| | `public override RecordsTableController CreateRecordsTableController()` | - | Создает контроллер таблицы рекордов |
| ConsoleShapeDrawer | `public static void DrawRectangle(int parX, int parY, int parWidth, int parHeight, ConsoleColor color)` | `parX - координата X, parY - координата Y, parWidth – ширина прямоугольника, parHeight – высота прямоугольника, color - цвет` | Рисует прямоугольник |
| FastConsoleOutput | `public static void SetPixel(int parX, int parY, short attribute)` | - | Устанавливает пиксель |
| | `public static void SetString(int parX, int parY, short parAttribute, string parStr)` | `parX - координата X, parY - координата Y, parAttribute – цвет, parStr - строка` | Пишет строку |
| | `public static void SetRectangle(int parX, int parY, int parWidth, int parHeight, short parAttribute)` | `parX - координата X, parY - координата Y, parWidth – ширина, parHeight – высота, parAttribute – цвет` | Рисует прямоугольник в буфере |
| | `public static void Init()` | - | Инициализирует |
| | `public static void Draw()` | - | Рисует все символы из буфера в консоли |

| | | | |
|---|---|---|---|
| | `public static void DrawRectangle(int parX, int parY, int parWidth, int parHeight, short parColor)` | `parX - координата X, parY - координата Y, parWidth – ширина, parHeight – высота, parColor – цвет` | Рисует прямоуголник в буфере |
| **AirplaneTests** | | | |
| AirplaneTests | `public void WallMoveRightTest()` | - | Тестирует перемещение самолета вправо, когда он находится у края экрана |
| | `public void NearToWallMoveRightTest()` | - | Тестирует перемещение самолета вправо, когда он находится рядом с границей экрана на расстоянии, меньшем чем его минимальное передвижение |
| | `public void FreeMoveLeftTest()` | - | Тестирует свободное перемещение самолета влево |
| | `public void WallMoveLeftTest()` | - | Тестирует перемещение самолета влево, когда он находится у края экрана |
| | `public void NearToWallMoveLeftTest()` | - | Тестирует перемещение самолета влево, когда он находится рядом с границей экрана на расстоянии, меньшем чем его минимальное передвижение |
| | `public void FreeMoveUpTest()` | - | Тестирует свободное перемещение самолета вверх |
| | `public void WallMoveUpTest()` | - | Тестирует перемещение самолета вверх, когда он находится у края экрана |
| | `public void NearToWallMoveUpTest()` | - | Тестирует перемещение самолета вверх, когда он находится рядом с границей экрана на расстоянии, меньшем чем его минимальное передвижение |
| | `public void FreeMoveDownTest()` | - | Тестирует свободное перемещение самолета вниз |
| | `public void WallMoveDownTest()` | - | Тестирует перемещение самолета |

| | | | |
|---|---|---|---|
| | | | вниз, когда он находится у края экрана |
| | `public void NearToWallMoveDownTest()` | - | Тестирует перемещение самолета вниз, когда он находится рядом с границей экрана на расстоянии, меньшем чем его минимальное передвижение |
| | `public void ShotgunBonusTakeTest()` | - | Тестирует взятие бонуса оружия |
| | `public void ShotgunBonusNotTakeTest()` | - | Тестирует взятие бонуса энергии |
| | `public void EnergyBonusNotTakeTest()` | - | Тестирует не взятие бонуса оружия |
| | `public void EnergyBonusNotTakeTest()` | - | Тестирует не взятие бонуса энергии |
| | `public void LightningEnergyWastingTest()` | - | Тестирует расход энергии молнией |
| | `public void LightningEnergyNotWastingTest()` | - | Тестирует использование молнии при недостаточном количестве энергии |
| | `public void EnemyMissileDamageTakeTest()` | - | Тестирует получение урона от вражеского снаряда |
| | `public void OwnMissileDamageNotTakeTest()` | - | Тестирует отсутствие получения урона от собственных снарядов |

## *2.2 Разработка программы*

Описание последовательности разработки:

1. Составление технического задания.
2. Проектирование игры.
3. Разработка игры.
4. Тестирование программы.
5. Написание документации.

### 2.2.1 Описание классов, перечислений и интерфейсов проекта

| Пространство имен | Название класса / интерфейса / перечисления | Назначение |
|---|---|---|
| Core.Controller | GameController | Контроллер игры |
| Core.Game | Airplane | Самолет игрока |
| Core.Game | Bonus | Бонус |
| Core.Game | BonusType | Типы бонусов |
| Core.Game | EnemyAirplane | Вражеский самолет |
| Core.Game | GameObject | Игровой объект |
| Core.Game | WhiteAirplane | Белый вражеский самолет |

| | | |
|---|---|---|
| Core.Game | YellowAirplane | Зеленый вражеский самолет |
| Core.Game.Missile | MissileType | Тип снаряда |
| Core.Game.Missile | WeaponMissile | Снаряд |
| Core.Game.Weapons | MachineGun | Пулемет |
| Core.Game.Weapons | Shotgun | Дробовик |
| Core.Game | Weapon | Оружие |
| Core.Game | GameModel | Модель игры |
| Core.View | GameView | Представление игры |
| Core.GameResult.Controller | GameResultController | Контроллер результата игры |
| Core.GameResult.Model | GameResultModel | Модель результата игры |
| Core.GameResult.View | GameResultView | Представление результата игры |
| Core.Help.Controller | HelpController | Контроллер справки |
| Core.Help.Model | HelpModel | Модель справки |
| Core.Help.View | HelpView | Представление справки |
| Core.Menu.Controller | MenuController | Контроллер меню |
| Core.Menu.Model | MenuItemTextFormer | Формирователь текста для пунктов меню |
| Core.Menu.Model | MenuModel | Модель меню |
| Core.Menu.View | MenuView | Представление меню |
| Core.Menu | MainMenu | Меню |
| Core.Menu | MenuItem | Пункты меню |
| Core.MVC | Controller<M, V> | Контроллер MVC |
| Core.MVC | Model | Модель |
| Core.MVC | View<M> | Представление |
| Core.RecordsTable.Controller | RecordsTableController | Контроллер таблицы рекордов |
| Core.RecordsTable.Model | GameRecord | Игровой рекорд |
| Core.RecordsTable.Model | RecordsTableModel | Модель таблицы рекордов |
| ore.RecordsTable.View | RecordsTableView | Представление таблицы рекордов |
| Core.RecordsTable | RecordsTableRepository | Репозиторий таблицы рекордов |
| Core | AbstractControllersFactory | Фабрика контроллеров |
| Core | MainApplication | Приложение |
| WPF.WPFController | WPFGameController | WPF контроллер игры |
| WPF.Game.View.ObjectsView | AirplaneView | WPF представление самолета игрока |
| WPF.Game.View.ObjectsView | BonusView | WPF представление бонуса |
| WPF.Game.View.ObjectsView | GameObjectView | WPF представление игрового объекта |
| WPF.Game.View.ObjectsView | MissileView | WPF представление снаряда |

| | | |
|---|---|---|
| WPF.Game.View.ObjectsView | WhiteAirplaneView | WPF представление белого вражеского самолета |
| WPF.Game.View.ObjectsView | YellowAirplaneView | WPF представление желтого вражеского самолета |
| WPF | WPFGameView | WPF представление игры |
| WPF.GameResult.Controller | WPFGameResultController | WPF контроллер результата игры |
| WPF.GameResult.Model | WPFGameResultView | WPF представление резульата игры |
| WPF.Help.Controller | WPFHelpController | WPF контроллер справки |
| WPF.Help.View | WPFHelpView | WPF представление справки |
| WPF.Menu.Controller | WPFMenuController | WPF контроллер меню |
| WPF.Menu.View | WPFMenuView | WPF представление меню |
| WPF.RecordsTable.Controller | WPFRecordsTableController | WPF контроллер таблицы рекордов |
| WPF.RecordsTable.View | WPFRecordsTableView | WPF представление таблицы рекордов |
| WPF | WindowKeeper | Хранитель окон |
| WPF | WPFApplication | WPF приложение |
| WPF | WPFControllersFactory | Фабрика WPF контроллеров |
| Console.ControllerConsole | ConsoleGameController | Консольный контроллер игры |
| Console.Game.View.ObjectsView | AirplaneView | Консольное представление самолета игрока |
| Console.Game.View.ObjectsView | BonusView | Консольное представление бонуса |
| Console.Game.View.ObjectsView | GameObjectView | Консольное представление игрового объекта |
| Console.Game.View.ObjectsView | MissileView | Консольное представление снаряда |
| Console.Game.View.ObjectsView | WhiteAirplaneView | Консольное представление белого вражеского самолета |
| Console.Game.View.ObjectsView | YellowAirplaneView | Консольное представление желтого вражеского самолета |
| Console | ConsoleGameView | Консольное представление игры |

| | | |
|---|---|---|
| Console.GameResult.Controller | ConsoleGameResultController | Консольный контроллер результата игры |
| Console.GameResult.View | ConsoleGameResultView | Консольное представление результата игры |
| Console.Help.Controller | ConsoleHelpController | Консольный контроллер справки |
| Console.Help.View | ConsoleHelpView | Консольное представление справки |
| Console.Menu.Controller | ConsoleMenuController | Консольный контроллер меню |
| Console.Menu.View | ConsoleMenuView | Консольное представление меню |
| Console.RecordsTable .Controller | ConsoleRecordsTable Controller | Консольный контроллер таблицы рекордов |
| Console.RecordsTable.View | ConsoleRecordsTableView | Консольное представление таблицы рекордов |
| Console | ConsoleApplication | Консольное приложение |
| Console | ConsoleControllersFactory | Фабрика консольных контроллеров |
| Console | ConsoleShapeDrawer | Класс, рисующий фигуры в консоли |
| Console | FastConsole | Средства для работы с быстрым выводом в консоль |
| Console | FastConsoleOutput | Помощник по работе с быстрым выводом в консоль |

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

### 2.3.1 Модель-Представление-Контроллер (Model-View-Controller)

Модель-Представление-Контроллер – схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя свое состояние. Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели. Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

В проекте создаются абстрактные классы Controller, Model, View. Их наследуют все базовые контроллеры, модели и представления; консольные и графические контроллеры и представления.

### 2.3.2 Абстрактная фабрика (Abstract factory)

Абстрактная фабрика – это порождающий паттерн проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов. Использовать Абстрактную Фабрику целесообразно, когда появляется нужда создать несколько зависимых объектов без обязательного присвоения конкретных классов.

Шаблон реализуется в классах *AbstractControllersFactory*, *WPFControllersFactory*, *ConsoleControllersFactory*, используется в классе *MainApplication*.

### 2.3.3 Одиночка (Singleton)

Шаблон проектирования «Одиночка» применяется если должен существовать только один экземпляр некоторого класса, и этому экземпляру нужна глобальная точка доступа (возможность обратиться из любой точки программы).

Шаблон реализуется в классе *WindowKeeper*, используется *во всех классах с WPF представлением*.

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

### 2.4.1 Вынесение констант

Данный метод рефакторинга заключается в вынесении чисел в константы с многократным использованием их в программном коде.

До вынесения констант:

```
/// <summary>
/// Перемещает самолет вниз
/// </summary>
public void MoveDown()
{
 if (Y + 50 < GameModel.SCREEN_HEIGHT)
 {
  Y += OFFSET;
  if (Y + 50 > GameModel.SCREEN_HEIGHT)
  {
    Y = GameModel.SCREEN_HEIGHT - 50;
  }
 }
}
/// <summary>
/// Перемещает самолет вправо
/// </summary>
public void MoveRight()
{
 if (X + 30 < GameModel.SCREEN_WIDTH)
 {
  X += OFFSET;
  if (X + 30 > GameModel.SCREEN_WIDTH)
  {
    X = GameModel.SCREEN_WIDTH - 30;
  }
 }
}
```

После вынесения констант:

```
/// <summary>
/// Ширина
/// </summary>
public const int WIDTH = 30;
/// <summary>
/// Высота
/// </summary>
public const int HEIGHT = 50;
/// <summary>
```

```
/// Перемещает самолет вниз
/// </summary>
public void MoveDown()
{
 if (Y + HEIGHT < GameModel.SCREEN_HEIGHT)
 {
  Y += OFFSET;
  if (Y + HEIGHT > GameModel.SCREEN_HEIGHT)
  {
    Y = GameModel.SCREEN_HEIGHT - HEIGHT;
  }
 }
}
/// <summary>
/// Перемещает самолет вправо
/// </summary>
public void MoveRight()
{
 if (X + WIDTH < GameModel.SCREEN_WIDTH)
 {
  X += OFFSET;
  if (X + WIDTH > GameModel.SCREEN_WIDTH)
  {
    X = GameModel.SCREEN_WIDTH - WIDTH;
  }
 }
}
```

### 2.4.2 Выделение метода

Данный метод рефакторинга заключается в выделении частс функциональности из метода в новый метод.

До выделения метода:

```
/// <summary>
/// Инициализирует модель игры
/// </summary>
public void Init()
{
  GameObjects.Clear();
  MachineGun machineGun = new MachineGun(this, 10);

  Airplane = new Airplane(this, 200, 400, machineGun);
  Airplane.LightningUsed += () => LightningUsed?.Invoke();

  new Thread(() =>
  {
    lock (GameObjects)
    {
      GameObjects.Add(Airplane);
    }
  }).Start();
}
```

После выделения метода:

```
/// <summary>
/// Инициализирует модель игры
/// </summary>
public void Init()
{
  GameObjects.Clear();
  MachineGun machineGun = new MachineGun(this, 10);

  Airplane = new Airplane(this, 200, 400, machineGun);
  Airplane.LightningUsed += () => LightningUsed?.Invoke();
  AddGameObject(Airplane);
  StartGenerateEnemies();
}
/// <summary>
/// Добавляет объект
/// </summary>
```

```
/// <param name="parGameObject">Игровой объект</param>
public void AddGameObject(GameObject parGameObject)
{
  new Thread(() =>
  {
    lock (GameObjects)
    {
      GameObjects.Add(parGameObject);
    }
  }).Start();
}
```

### 2.4.3 Выделение локальной переменной

Данный метод рефакторинга заключается в вынесении чисел в локальные переменные, которые используются многократно и только в текущем методе.

До выделения локальной переменной:

```
/// <summary>
/// Рисует молнию
/// </summary>
private void LightningDraw()
{
  new Thread(() =>
  {
    for (int i = 0; i < 3; i++)
    {
      _window.Dispatcher.Invoke(() =>
      {
        _canvas.Background = new SolidColorBrush(Colors.White);
      });
       Thread.Sleep(100);
       _window.Dispatcher.Invoke(() =>
       {
         _canvas.Background = new SolidColorBrush(Colors.Cyan);
       });
       Thread.Sleep(100);
     }
   }).Start();
 }
```

После выделения локальной переменной:

```
/// <summary>
/// Рисует молнию
/// </summary>
private void LightningDraw()
{
  new Thread(() =>
  {
    int millisecondsTimeout = 100;
    int flickCount = 3;
    for (int i = 0; i < flickCount; i++)
    {
      _window.Dispatcher.Invoke(() =>
      {
        _canvas.Background = new SolidColorBrush(Colors.White);
      });
       Thread.Sleep(millisecondsTimeout);
       _window.Dispatcher.Invoke(() =>
       {
         _canvas.Background = new SolidColorBrush(Colors.Cyan);
       });
       Thread.Sleep(millisecondsTimeout);
     }
   }).Start();
 }
```

### 2.4.4 Переименование метода

Данный метод рефакторинга заключается в наиболее точном именовании метода, описывающем назначение метода.

До переименования метода:

```
/// <summary>
/// Самолет получает урон
/// </summary>
/// <param name="parDamage">Урон</param>
public void GetHurt(int parDamage)
{
  if (_invulnerabilityFrames <= 0)
  {
   Health -= parDamage;
   _invulnerabilityFrames = INVULNERABILITY_FRAMES_AMOUNT;
  }
}
```

После переименования метода:

```
/// <summary>
/// Самолет получает урон
/// </summary>
/// <param name="parDamage">Урон</param>
public void TakeDamage(int parDamage)
{
  if (_invulnerabilityFrames <= 0)
  {
   Health -= parDamage;
   _invulnerabilityFrames = INVULNERABILITY_FRAMES_AMOUNT;
  }
}
```

### 2.4.5 Изменение сигнатуры метода

Данный метод рефакторинга заключается в добавлении, изменении или удалении параметра метода.

До изменения сигнатуры метода:

```
/// <summary>
/// Конструктор самолета игрока
/// </summary>
/// <param name="parGame">Модель игры</param>
/// <param name="parX">Координата X</param>
/// <param name="parY">Координата Y</param>
/// <param name="parWeapon">Оружие</param>
/// <param name="parEnergy">Энергия</param>
/// <param name="parHealth">Здоровье</param>
/// <param name="parScore">Игровые очки</param>
public Airplane(
GameModel parGame,
double parX,
double parY,
Weapon parWeapon,
int parEnergy,
int parHealth,
int parScore)
: base (
   parGame,
   parX,
   parY,
   WIDTH,
   HEIGHT,
   0,
   0)
{
      Health = HEALTH_AMOUNT;
      Energy = ENERGY_AMOUNT;
      Weapon = parWeapon;
      Score = 0;
      _invulnerabilityFrames = 0;
```

```
 }
```

После изменения сигнатуры метода:

```
/// <summary>
/// Конструктор самолета игрока
/// </summary>
/// <param name="parGame">Модель игры</param>
/// <param name="parX">Координата X</param>
/// <param name="parY">Координата Y</param>
/// <param name="parWeapon">Оружие</param>
public Airplane(
GameModel parGame,
double parX,
double parY,
Weapon parWeapon)
: base (
   parGame,
   parX,
   parY,
   WIDTH,
   HEIGHT,
   0,
   0)
{
      Health = HEALTH_AMOUNT;
      Energy = ENERGY_AMOUNT;
      Weapon = parWeapon;
      Score = 0;
      _invulnerabilityFrames = 0;
 }
```

## *2.5 Разработка тестов*

### 2.5.1 Test Cases

| № теста | Название | Предусловия | Шаги | Ожидаемый результат |
|---|---|---|---|---|
| 1 | Тест на перемещение самолета вправо, когда он находится у края экрана | Создан самолет вплотную к правому краю экрана | Попытка перемещения самолета вправо | Самолет не переместился |
| 2 | Тест на перемещение самолета вправо, когда он находится рядом с границей экрана на расстоянии, меньшем чем его минимальное передвижение | Создан самолет около правой границы экрана | Попытка перемещения самолета вправо | Самолет переместился вплотную к краю экрана |
| 3 | Тест на свободное перемещение самолета влево | Создан самолет в заданных координатах | Попытка перемещения самолета влево | Самолет переместился на необходимое количество |
| 4 | Тест на перемещение самолета влево, когда он находится у края экрана | Создан самолет вплотную к левому краю экрана | Попытка перемещения самолета влево | Самолет не переместился |
| 5 | Тест на перемещение самолета влево, когда он находится рядом с границей экрана на расстоянии, меньшем чем его минимальное передвижение | Создан самолет около левой границы экрана | Попытка перемещения самолета влево | Самолет переместился вплотную к краю экрана |

| | | | | |
|---|---|---|---|---|
| 6 | Тест на свободное перемещение самолета вверх | Создан самолет в заданных координатах | Попытка перемещения самолета вверх | Самолет переместился на необходимое количество |
| 7 | Тест на перемещение самолета вверх, когда он находится у края экрана | Создан самолет вплотную к верхнему краю экрана | Попытка перемещения самолета вверх | Самолет не переместился |
| 8 | Тест на перемещение самолета вверх, когда он находится рядом с границей экрана на расстоянии, меньшем чем его минимальное передвижение | Создан самолет около верхней границы экрана | Попытка перемещения самолета вверх | Самолет переместился вплотную к краю экрана |
| 9 | Тест на свободное перемещение самолета вниз | Создан самолет в заданных координатах | Попытка перемещения самолета вниз | Самолет переместился на необходимое количество |
| 10 | Тест на перемещение самолета вниз, когда он находится у края экрана | Создан самолет вплотную к нижнему краю экрана | Попытка перемещения самолета вниз | Самолет не переместился |
| 11 | Тест на перемещение самолета вниз, когда он находится рядом с границей экрана на расстоянии, меньшем чем его минимальное передвижение | Создан самолет около нижней границы экрана | Попытка перемещения самолета вниз | Самолет переместился вплотную к краю экрана |
| 12 | Тест на взятие бонуса оружия | Создан самолет и бонусное оружие в одинаковых координатах | Попытка взятия бонуса | Самолет получил бонусное оружие |
| 13 | Тест на взятие бонуса энергии | Создан самолет и бонусная энергия в одинаковых координатах | Попытка взятия бонуса | Самолет получил бонусную энергию |
| 14 | Тест на не взятие бонуса оружия | Создан самолет и бонусное оружие в разных координататах | Попытка взятия бонуса | Самолет не получил бонусное оружие |
| 15 | Тест на не взятие бонуса энергии | Создан самолет и бонусная энергия в разных координататах | Попытка взятия бонуса | Самолет не получил бонусную энергию |
| 16 | Тест на расход энергии молнией | Создан самолет с полной энергией | Попытка использования молнии | Энергия уменьшилась на «стоимость» использования молнии |
| 17 | Тест на использование молнии при недостаточном количестве энергии | Создан самолет с нулевой энергией | Попытка использования молнии | Энергия не уменьшилась |
| 18 | Тест на получение урона от вражеского снаряда | Создан самолет и вражеский снаряд в одинаковых координатах | Попытка получения урона от вражеского снаряда | Здоровье уменьшилось на количество урона снаряда |

| | | | | |
|---|---|---|---|---|
| 19 | Тест на отсутствие получения урона от собственных снарядов | Создан самолет и снаряд оружия самолета в одинаковых координатах | Попытка получения урона от собственного снаряда | Здоровье не уменьшилось |
| 20 | Тест на свободное перемещение самолета вправо | Создан самолет в заданных координатах | Попытка перемещения самолета вправо | Самолет переместился на необходимое количество |

### 2.5.2 Модульные тесты

Модульное тестирование – это тип тестирования программного обеспечения, при котором тестируются отдельные модули или компоненты программного обеспечения. Основная цель - изолировать каждую единицу системы для выявления, анализа и исправления дефектов. Модульный тест - это автоматический тест, который проверяет модуль программы изолированно.

В курсовом проекте были разработаны модульные тесты для класса Airplane (самолет игрока). Ниже представлен тест, проверяющий расход энергии при использовании молнии. `Assert.AreEquals` – это метод класса `Assert`, который выкидывает `AssertFailedException` если два объекта не равны. Тест является успешным, если после использования молнии ожидаемая энергия, рассчитанная как разность энергии самолета и «стоимости» расходования молнии, совпадает с текущей энергией самолета.

```
/// <summary>
/// Координата для свободного перемещения самолета
/// </summary>
private const int COORD_FOR_FREE_MOVE = 200;

/// <summary>
/// Тестирует расход энергии молнией
/// </summary>
[TestMethod]
public void LightningEnergyWastingTest()
{
  GameModel newGame = new GameModel();
  Airplane airplane = new Airplane(newGame, COORD_FOR_FREE_MOVE, COORD_FOR_FREE_MOVE,
  null);
  int expectedEnergy = airplane.Energy - Airplane.LIGHTNING_ENERGY_COST;
  airplane.UseLightning();

  Assert.AreEqual(expectedEnergy, airplane.Energy);
}
```

# 3 Результат работы программы

## *3.1 Графическая версия*

Главное меню графической версии приложения представлено на рисунке 22.


**Рисунок 22 – Главное меню графической версии приложения**

Игра графической версии приложения представлена на рисунке 23.

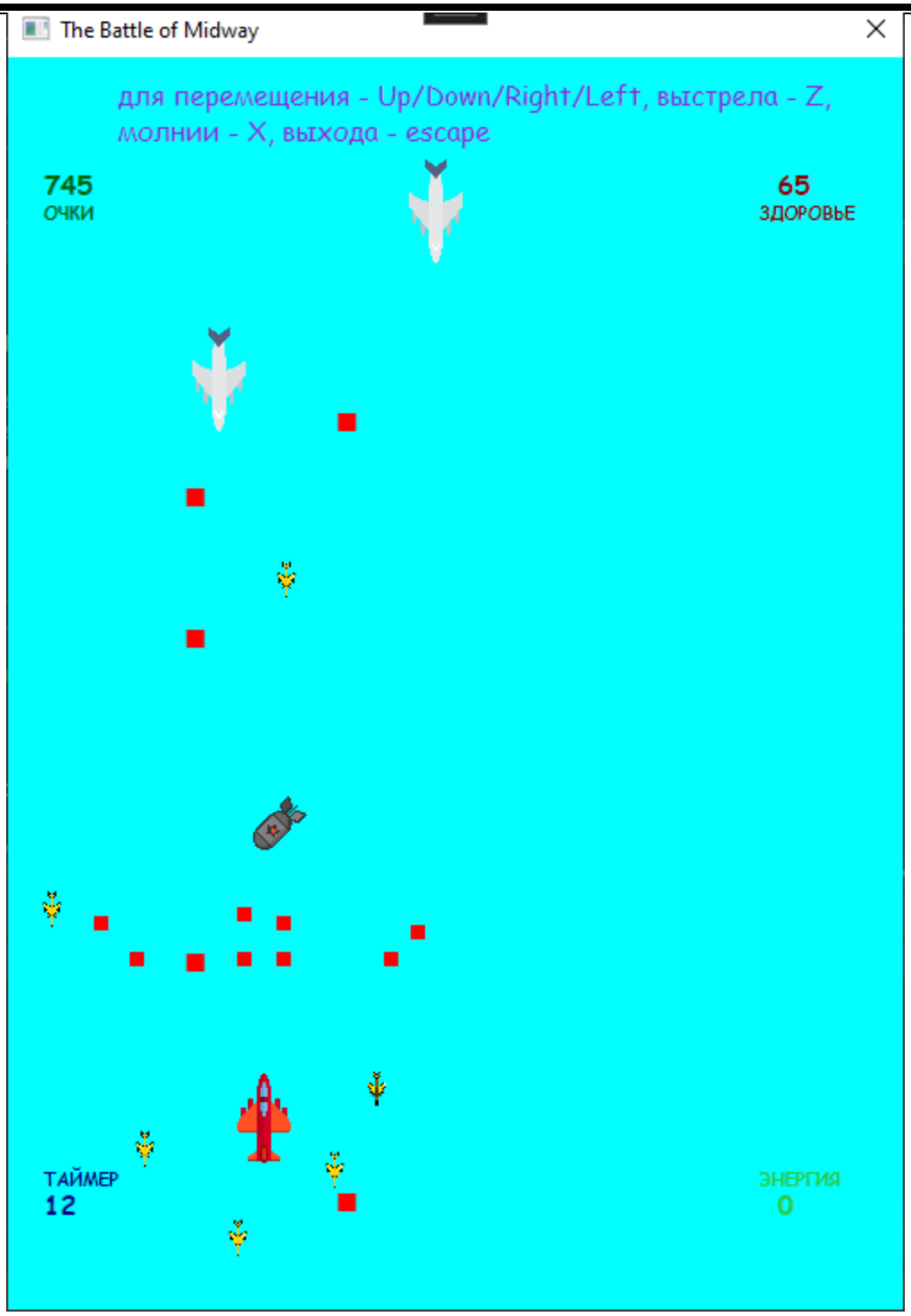


**Рисунок 23 – Игра графической версии приложения**

Окно результата игры графической версии приложения представлено на рисунке 24.

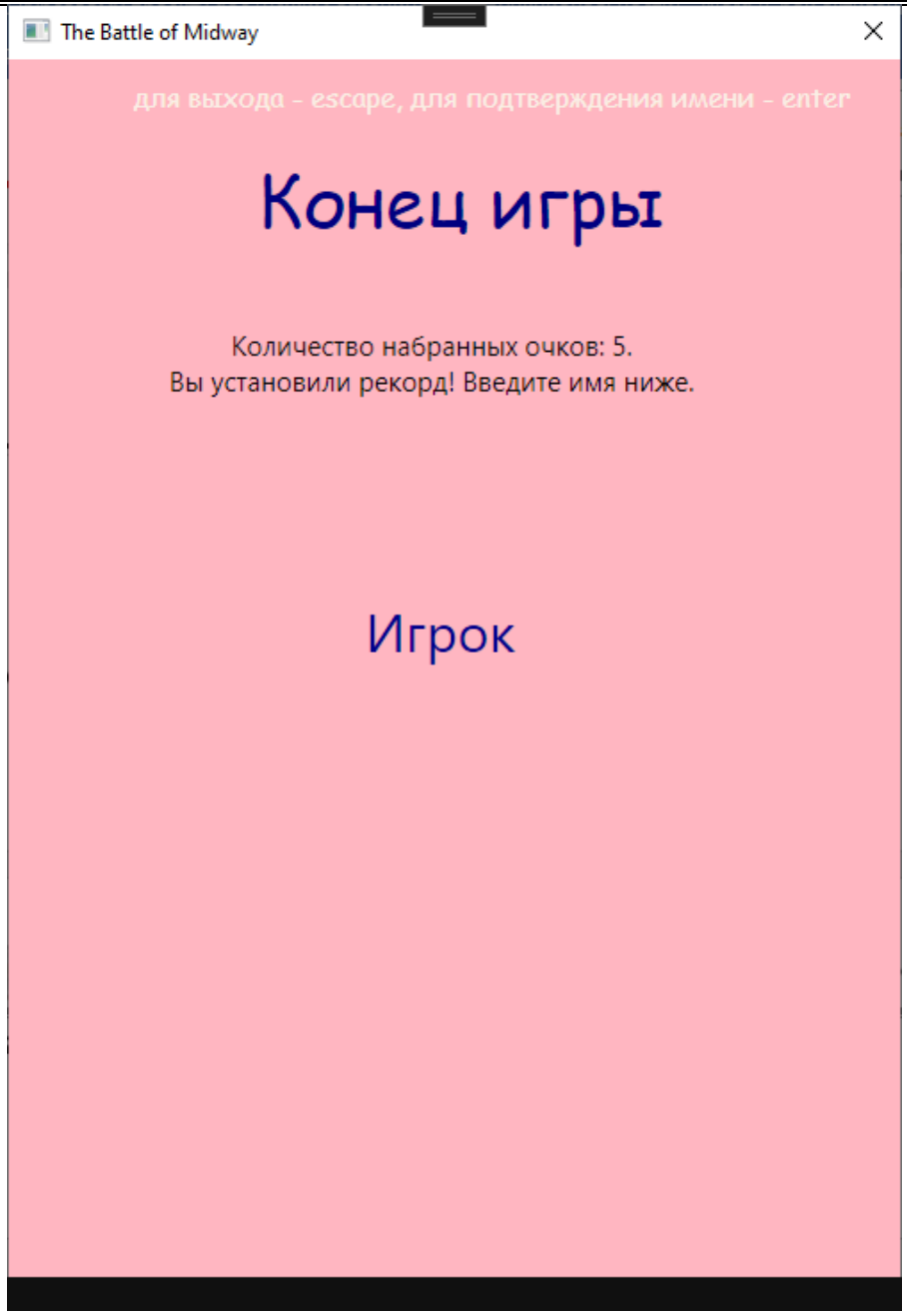


**Рисунок 24 – Окно результата игры графической версии приложения**

Таблица рекордов графической версии приложения представлена на рисунке 25.

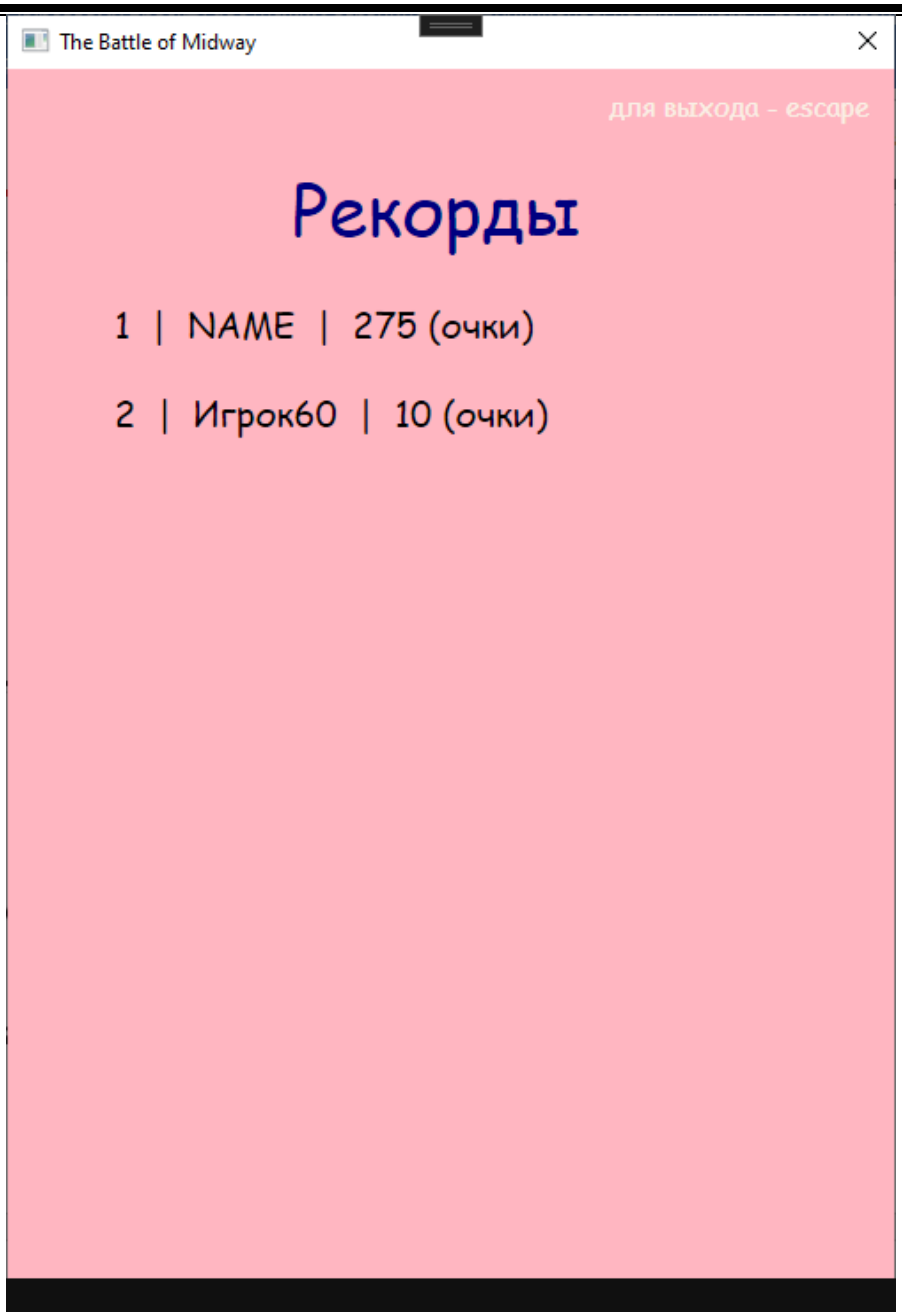


**Рисунок 25 – Таблица рекордов графической версии приложения**

Справка графической версии приложения представлена на рисунке 26.

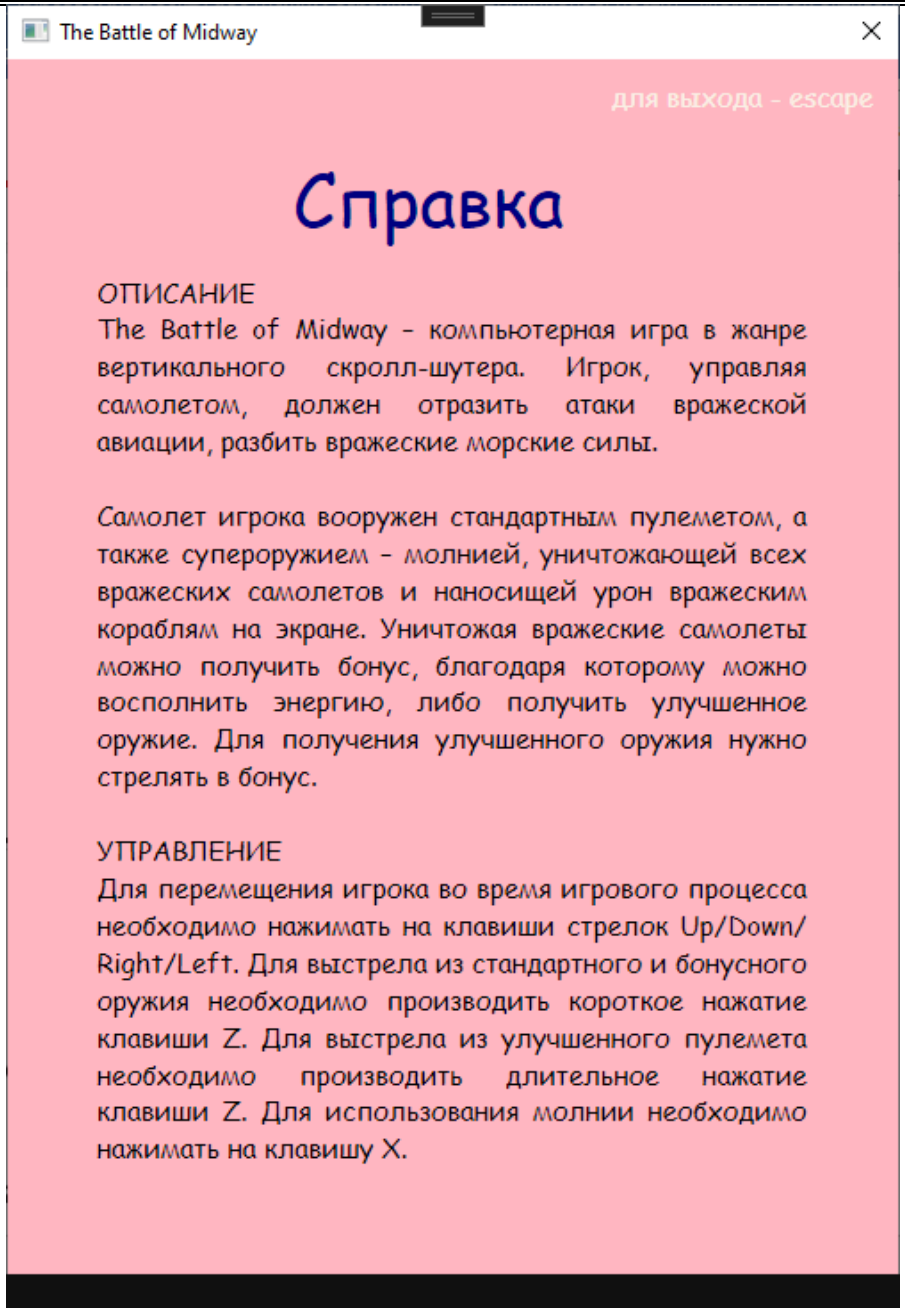


**Рисунок 26 – Справка графической версии приложения**

## *3.2 Консольная версия*

Главное меню консольной версии приложения представлено на рисунке 27.

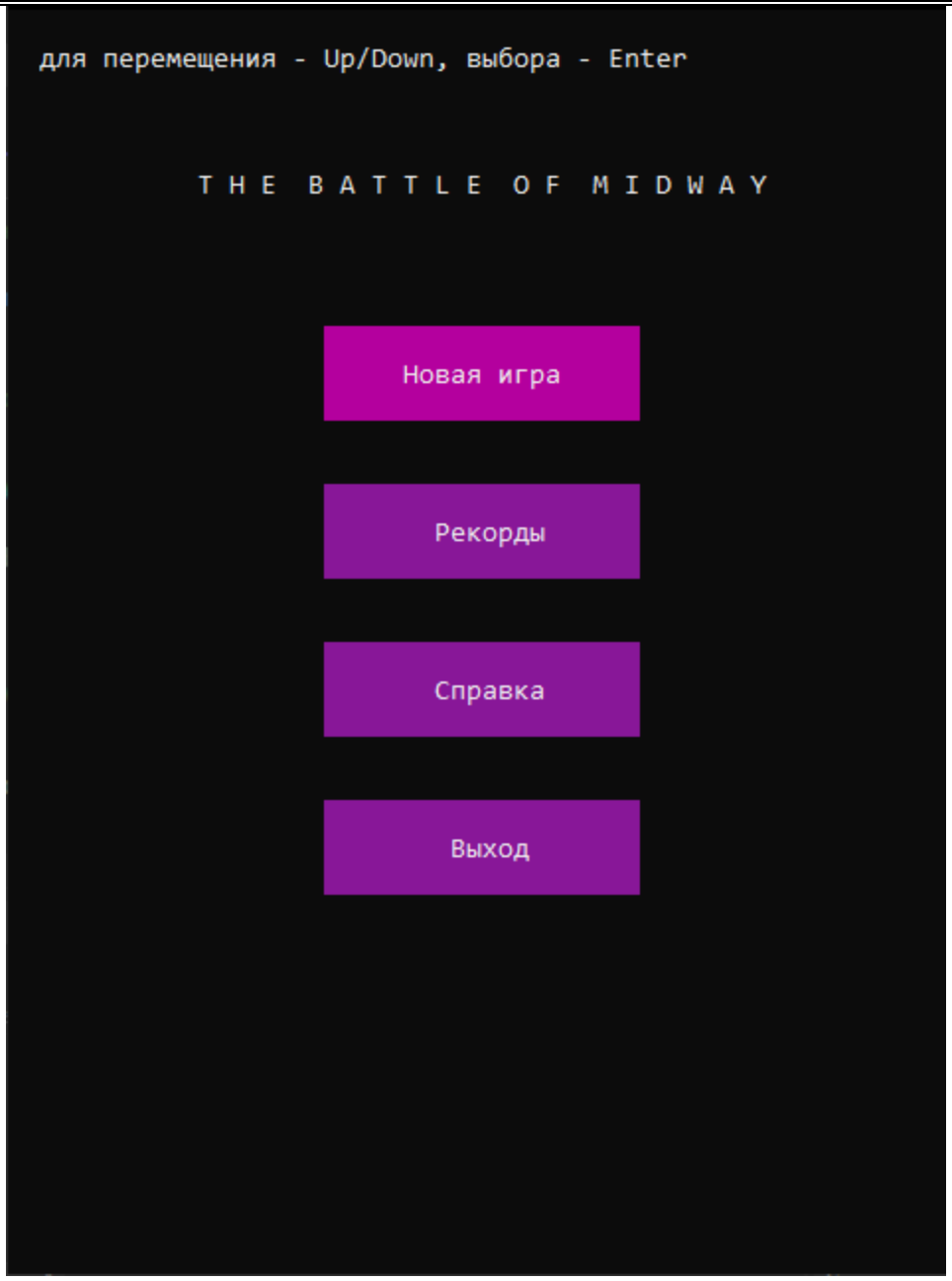


**Рисунок 27 – Главное меню консольной версии приложения**

Игра консольной версии приложения представлена на рисунке 28.

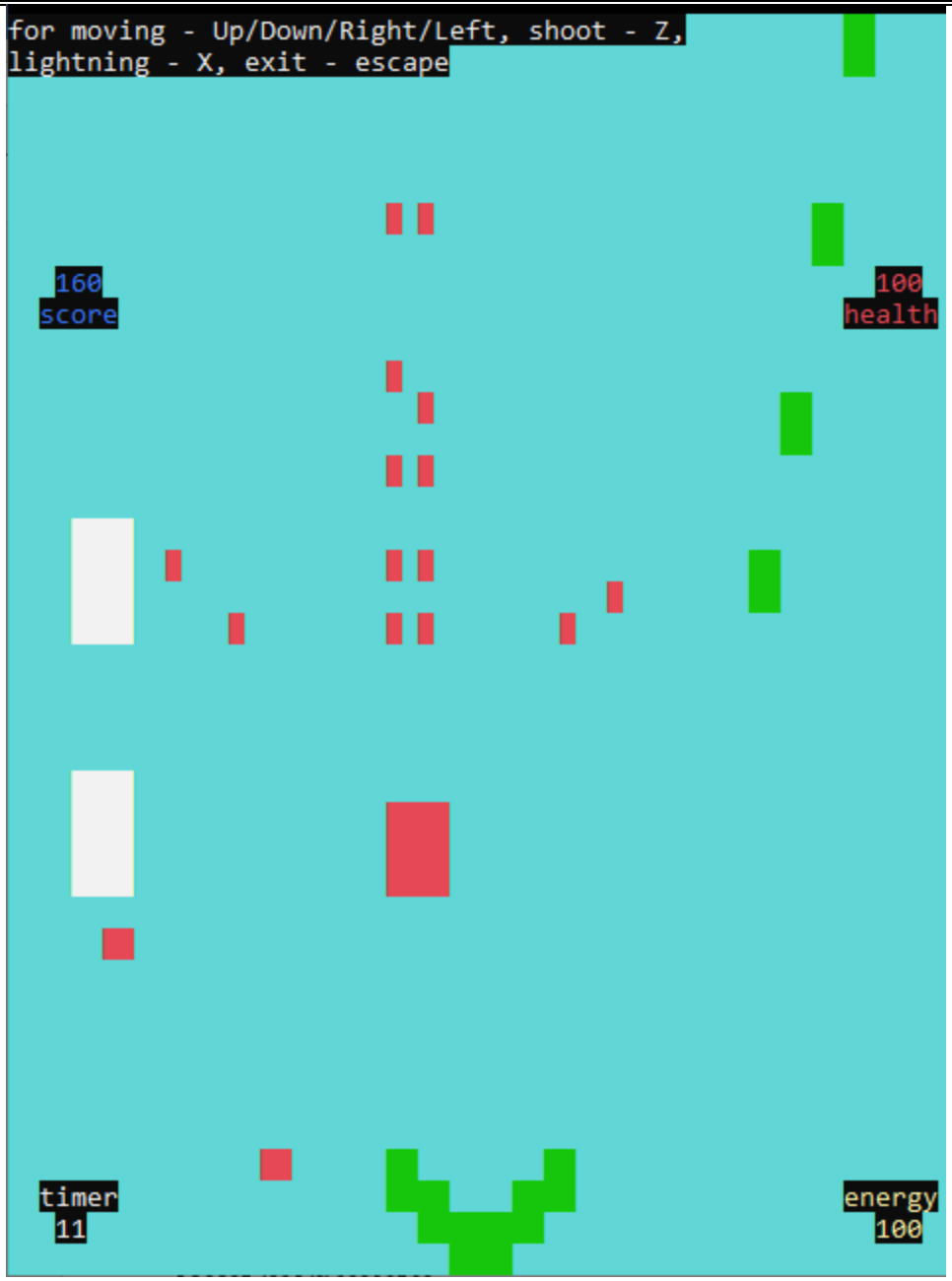


**Рисунок 28 – Игра в консольной версии приложения**

Окно результата игры консольной версии приложения представлено на рисунке 29.

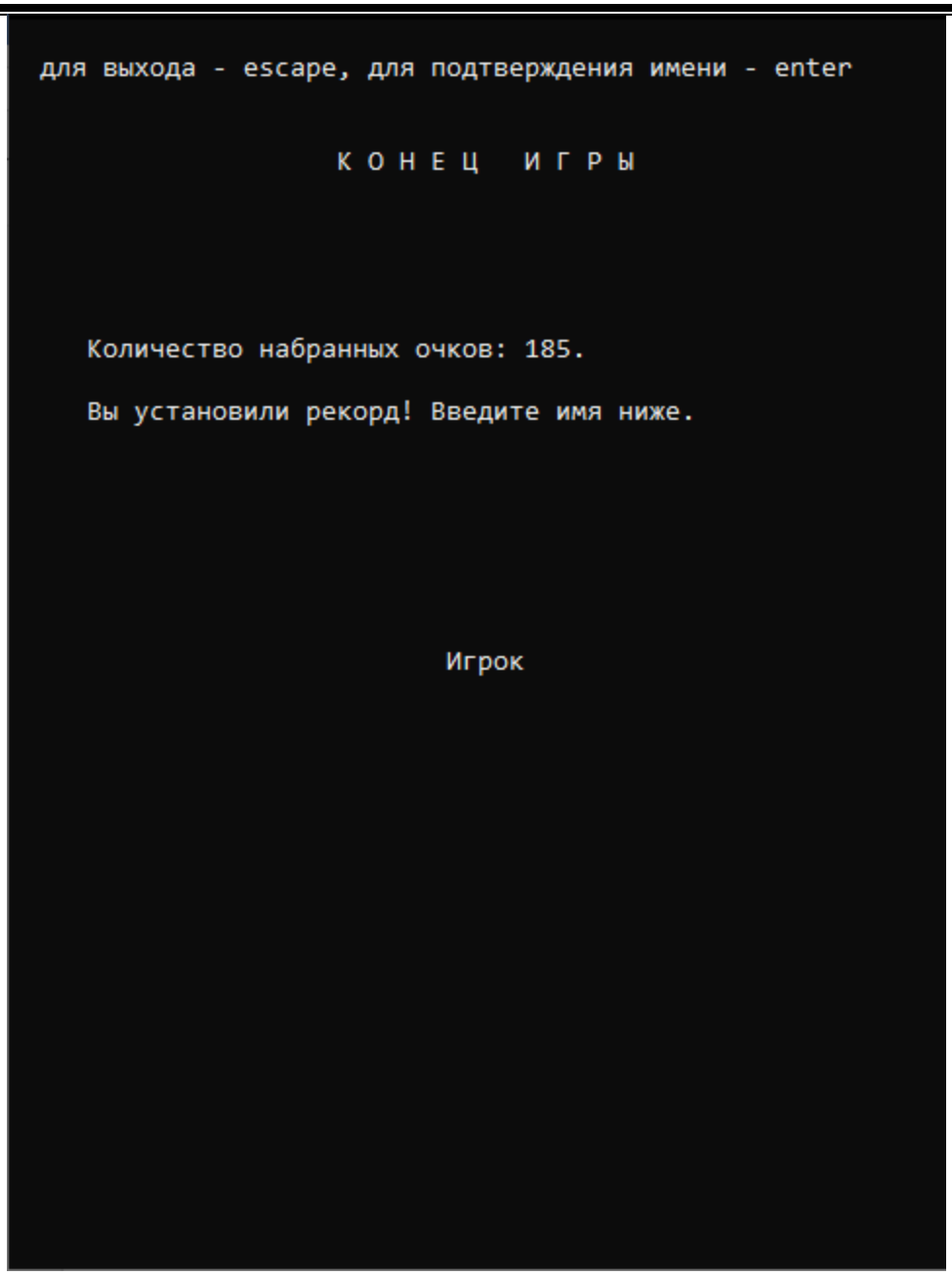


**Рисунок 29 – Окно результата игры консольной версии приложения**

Таблица рекордов консольной версии приложения представлена на рисунке 30.

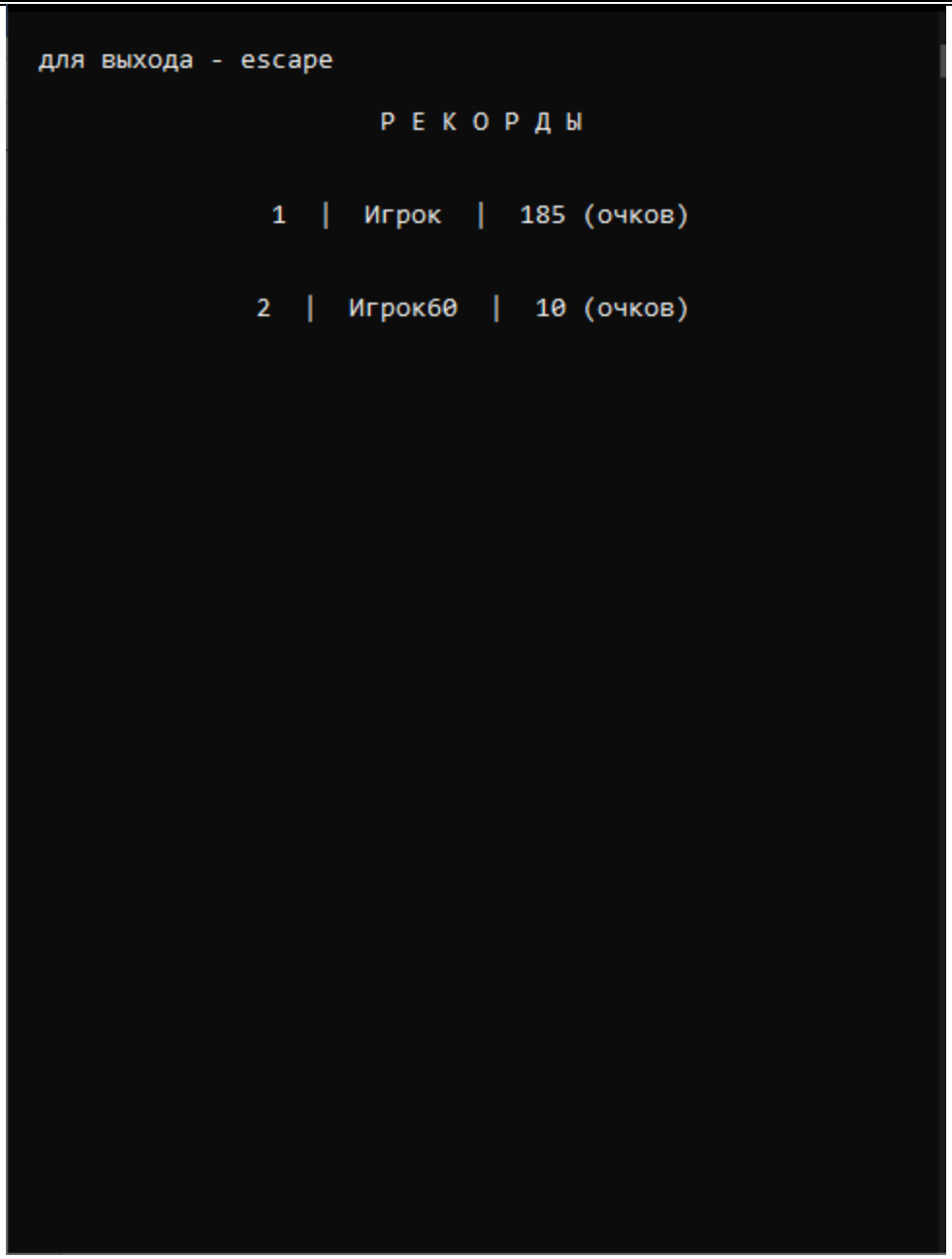


**Рисунок 30 – Таблица рекордов консольной версии приложения**

Справка консольной версии приложения представлена на рисунке 31.

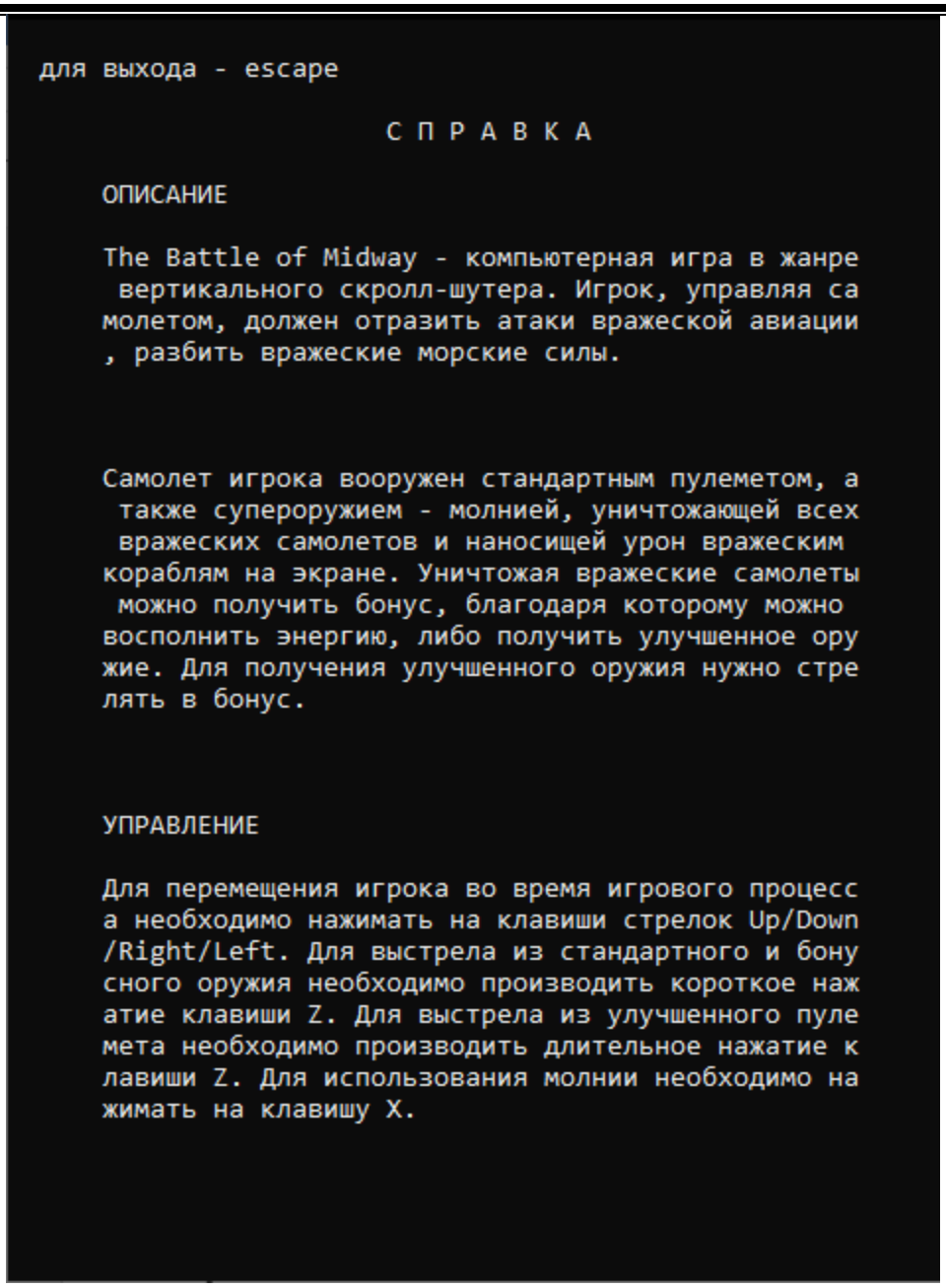


**Рисунок 31 – Справка в консольной версии приложения**

# Заключение

Была разработана игровая программа The Battle of Midway с двумя реализациями интерфейса программы, основанными на разработанных общих классах и библиотеках: WPF и консоль. В курсовом проекте использовался шаблон проектирования Модель-Представление-Контроллер (MVC) и два дополнительных шаблона: Абстрактная фабрика (Abstract factory) и Одиночка (Singleton). В приложении используется многопоточность и синхронизация между потоками. Для класса Airplane разработаны полноценные модульные тесты.

# Приложения