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 Worms**»**

по курсу

«Конструирование программного обеспечения»

Выполнил:
студент группы № 9413
Заикин С.Ю.

Проверил:
доцент кафедры ВПМ
Столчнев В. К.

Рязань, 2023 г.

Оглавление

Введение

Необходимо разработать игровое приложение Worms. Приложение должно иметь 2 реализации: консольную (используя консоль) и оконную (используя WPF). Необходимо использовать шаблон проектирования MVC (Модель-Вид-Контроллер) и два дополнительных шаблона проектирования. В приложении обязательно должна использоваться многопоточность и синхронизация между потоками. Для одного из классов приложения должны быть разработаны полноценные модульные тесты, обеспечивающие проверку корректности его работы.

Разработка производится с помощью языка программирования C# в среде разработки Visual Studio. Результатом работы является разработанная игра «Worms».

1 Анализ задачи

Задача курсового проекта – разработка игрового приложения Worms.

Worms представляет собой пошаговую игру. В ней каждый игрок владеет армией червяков и инвентарем с оружием и вспомогательными предметами, которыми могут пользоваться червяки. Игроки по очереди выполняют ограниченный по времени ход, во время которого они должны попытаться нанести наибольший урон червякам противника, либо обезопасить своих червяков. Ход заканчивается тогда, когда текущий червяк получил урон или закончилось время на ход. После окончания своего хода игрок не может управлять червяком. Червяки могут двигаться вправо и влево, прыгать вперед и назад. Они погибают при здоровье, меньшем или равном 0, и при попадании в воду. Перед гибелью червяки производят Взрыв.

Целью игры является уничтожение всех червяков противника. Игра продолжается до тех пор, пока на поле имеются червяки по крайней мере 2-х разных коалиций или пока общий игровой таймер не дойдет до 0.

После окончания игры должно выводиться окно с результатом. Если установлен рекорд, запрашивается имя игрока.

В приложении должно быть меню, состоящее из пунктов: игра, рекорды, справка, выход.

1.1 Разработка иерархии классов

Игровое приложение разрабатывается на основе шаблона проектирования MVC (модель-представление-контроллер). Визуальное представление шаблона MVC отображено на рисунке 1.

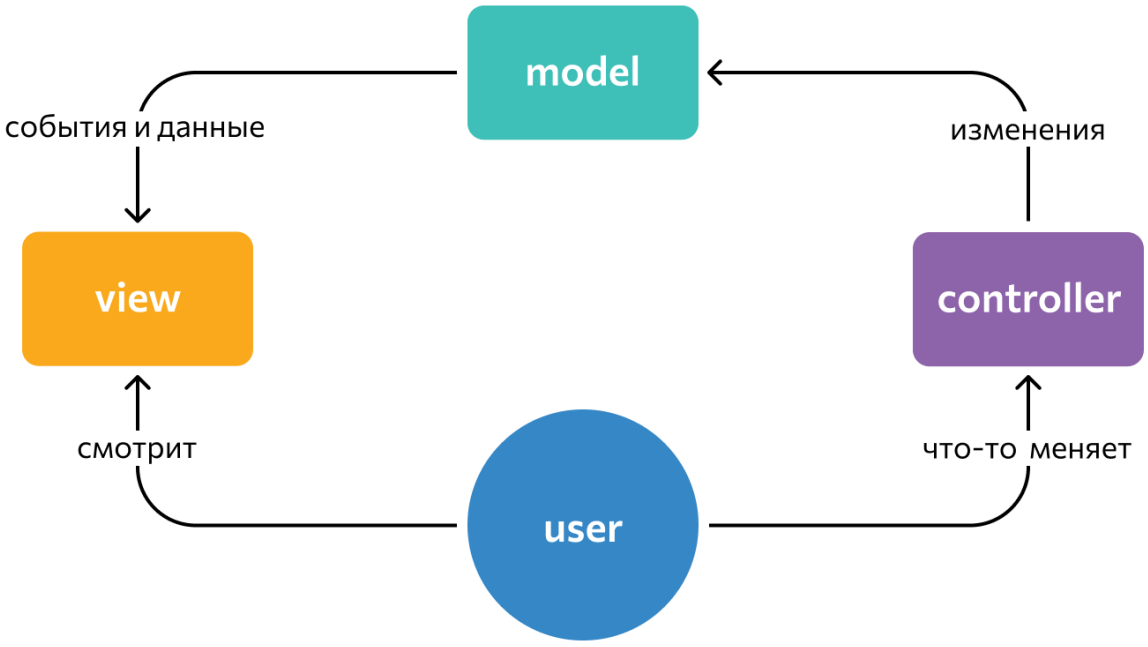

Рисунок 1 – Визуальное представление шаблона MVC

Концепция MVC разделяет данные, представление и обработку действий пользователя на компоненты:

- Модель – предоставляет собой объектную модель некой предметной области, включает в себя данные и методы работы с этими данными, реагирует на запросы из контроллера, возвращая данные и/или изменяя своё состояние. При этом модель не содержит в себе информации о способах визуализации данных или форматах их представления, а также не взаимодействует с пользователем напрямую.
- Представление – отвечает за отображение информации (визуализацию). Одни и те же данные могут представляться различными способами и в различных форматах.
- Контроллер – обеспечивает связь между пользователем и системой, использует модель и представление для реализации необходимой реакции на действия пользователя.

Обощенно приложение состоит из следующих частей, базовые архитектуры для которых представлены на рисунке 2:

- меню;
- рекорды;
- справка;
- игра;
- конец игры.

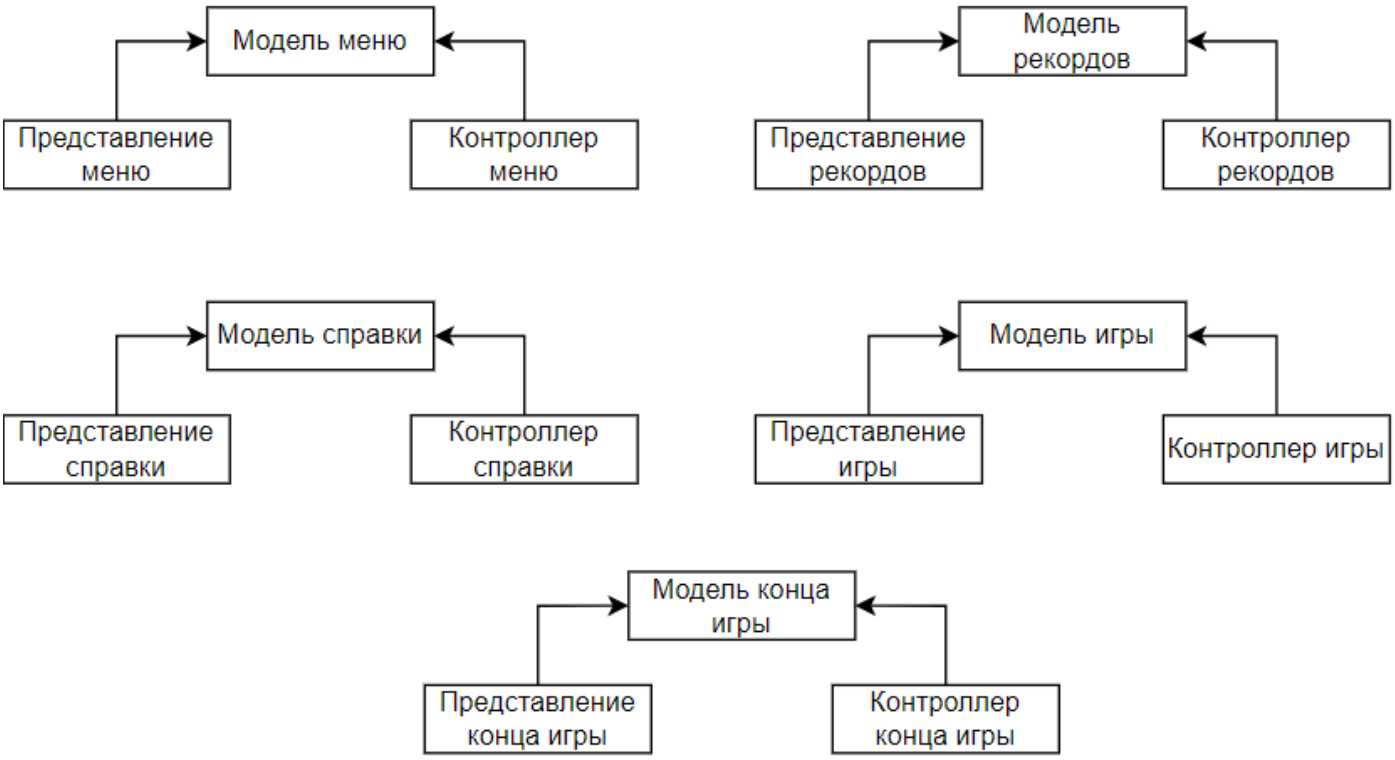

Рисунок 2 – Базовые архитектуры для основных частей приложения

1.1.1 Выделение сущностей

Из предметной области игры Worms можно выделить следующие сущности:

- Червяк – объект, умеющий пользоваться оружием, передвигаться и прыгать, которым управляет игрок.
- Оружие – объект, который используется червяками для нанесения урона другим червякам и/или для разрушения карты.
- Физический объект – объект, на который распространяется физика и логика игры.
- Инвентарь – объект, хранящий список оружия.
- Снаряд – физический объект, создаваемый оружием.

Для реализации шаблона MVC выделяются следующие сущности:

- Модель – состояние объекта, которое может изменяться.
- Представление – отображение модели.
- Контроллер – управление моделью на основе действий пользователя.

1.1.2 Зависимости между классами. Диаграмма классов

Диаграмма классов представлений приложения изображена на рисунке 3.

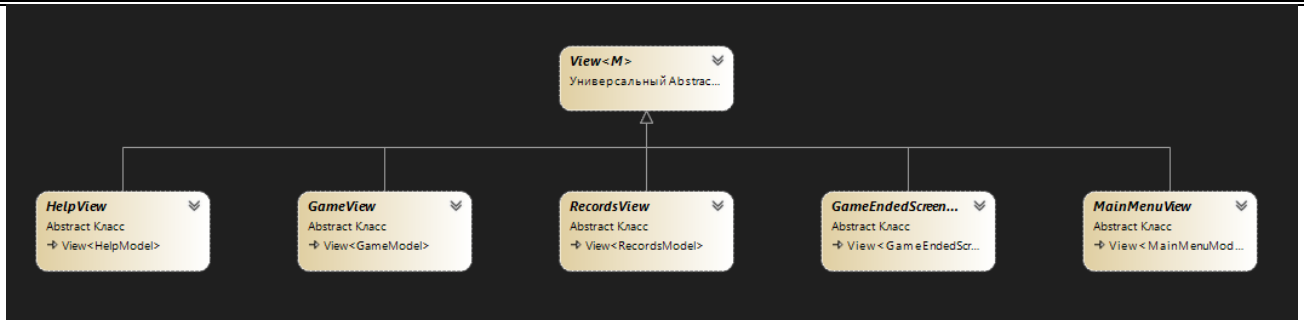

Рисунок 3 - Диаграмма классов представлений приложений

Диаграмма классов моделей приложения изображена на рисунке 4.

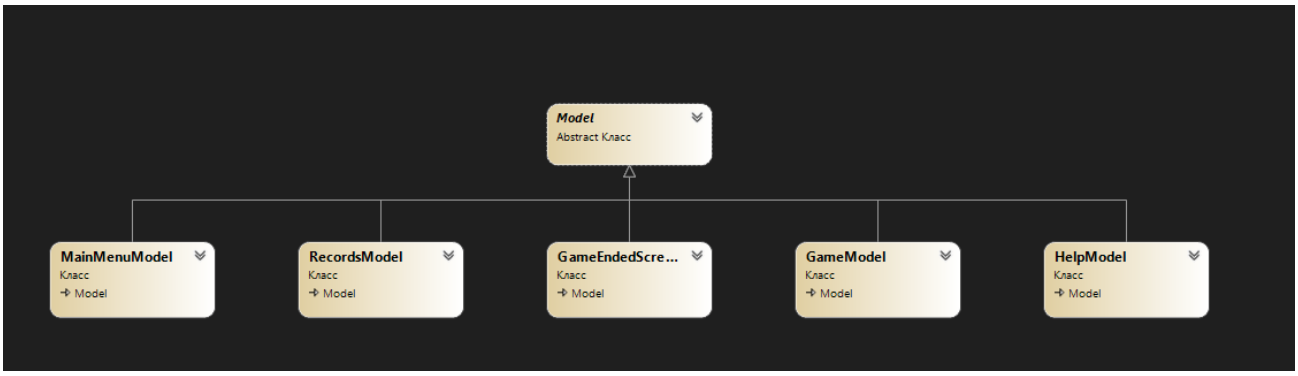

Рисунок 4 – Диаграмма классов моделей приложения

Диаграмма классов контроллеров приложения изображена на рисунке 5.

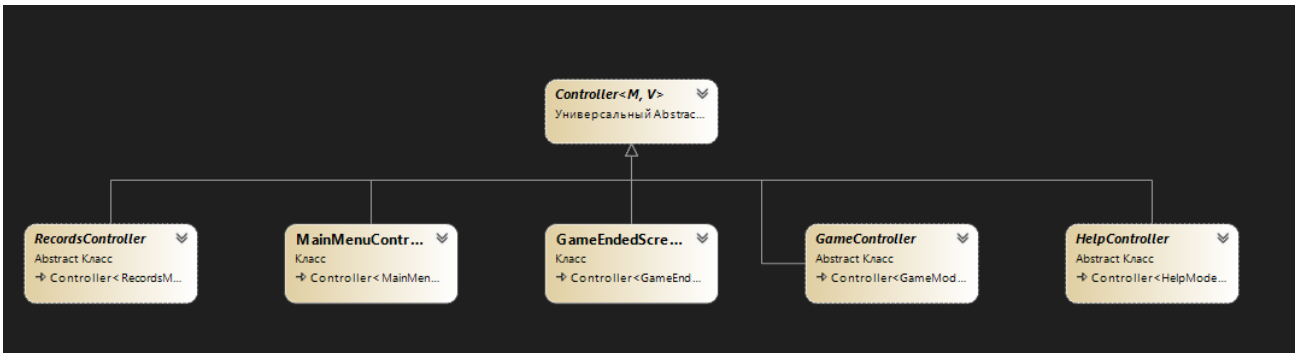

Рисунок 5 – Диаграмма классов контроллеров приложения

Диаграмма классов физических объектов изображена на рисунке 6.

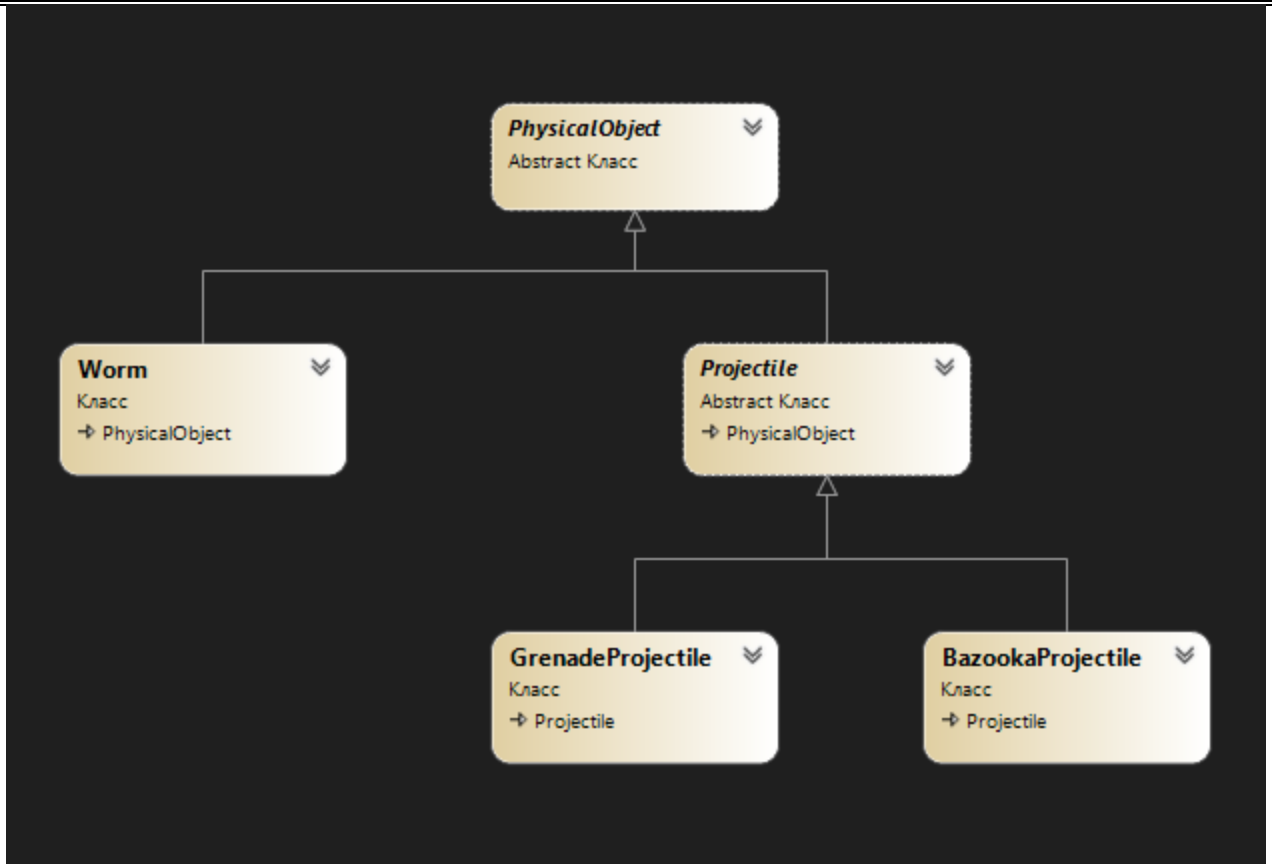

Рисунок 6 - Диаграмма классов физических объектов

Диаграмма классов оружия изображена на рисунке 7.

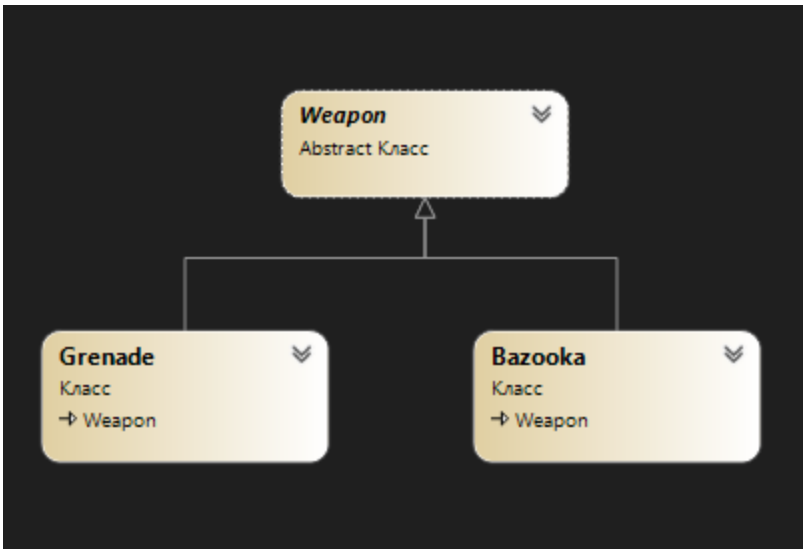

Рисунок 7 – Диаграмма классов оружия

Диаграмма классов представлений объектов изображена на рисунке 8.

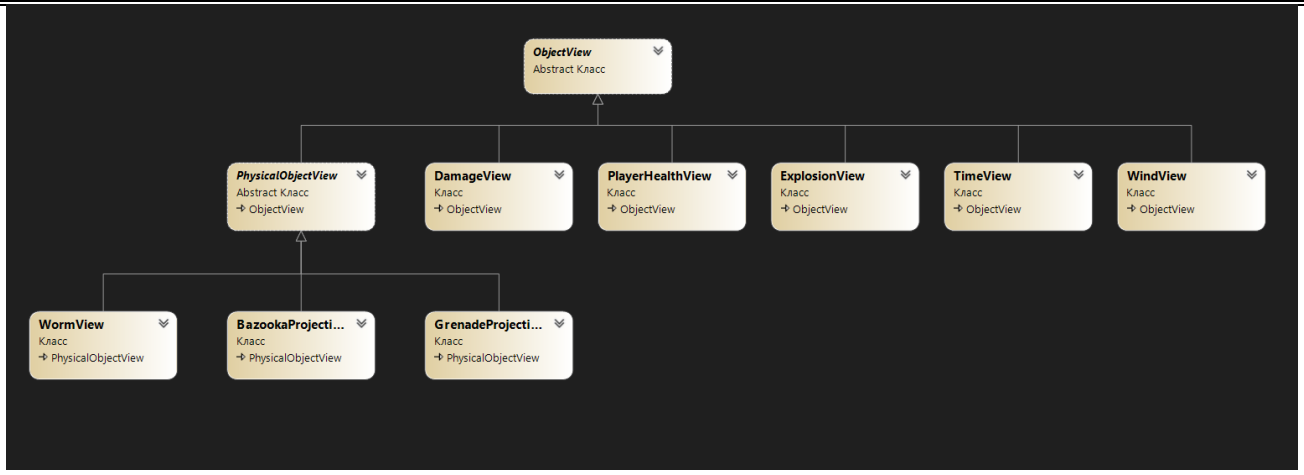

Рисунок 8 – Диаграмма классов представлений объектов

1.2 Алгоритмы

1.2.1 Алгоритм успешного передвижения червяка

Логика передвижения червяка, при котором червяку не помешают препятствия, представлена на диаграмме последовательности успешного передвижения червяка (рисунок 9).

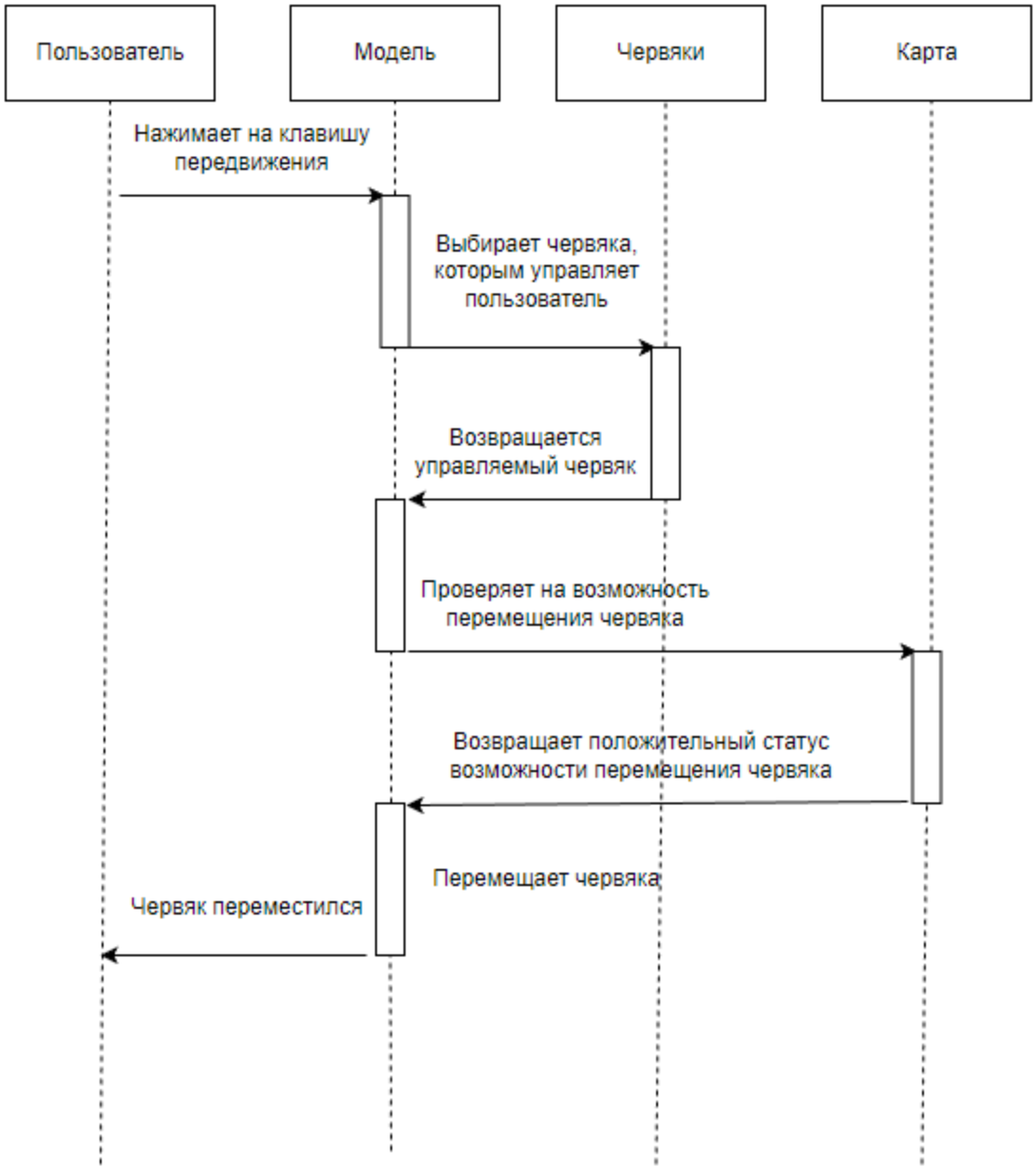

Рисунок 9 – Диаграмма последовательности успешного передвижения червяка

Как только пользователь нажмет клавишу A или D для передвижения червяком, модель выберет червяка, которым управляет игрок и попытается его переместить. Если червяку ничего не мешает, то он успешно переместится.

1.2.2 Алгоритм неудачного передвижения червяка

Однако если червяк не может переместиться, то он останется на месте – данная логика представлена на диаграмме последовательности неудачного передвижения червяка (рисунок 10).

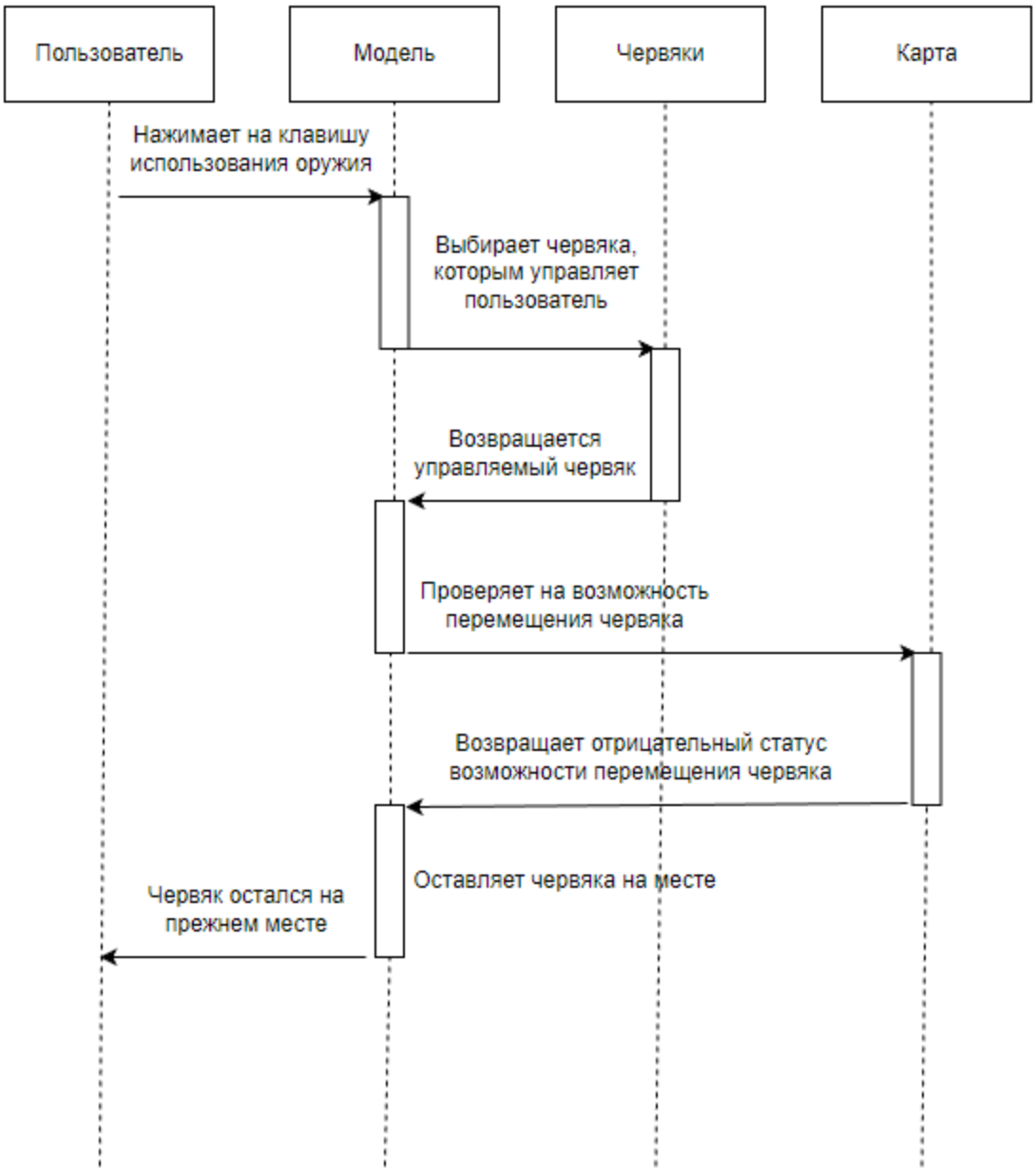

Рисунок 10 – Диаграмма последовательности неудачного передвижения червяка

1.2.3 Алгоритм столкновения червяка с поверхностью

Логика столкновения червяка с поверхностью представлена на диаграмме последовательности столкновения червяка с поверхностью (рисунок 11).

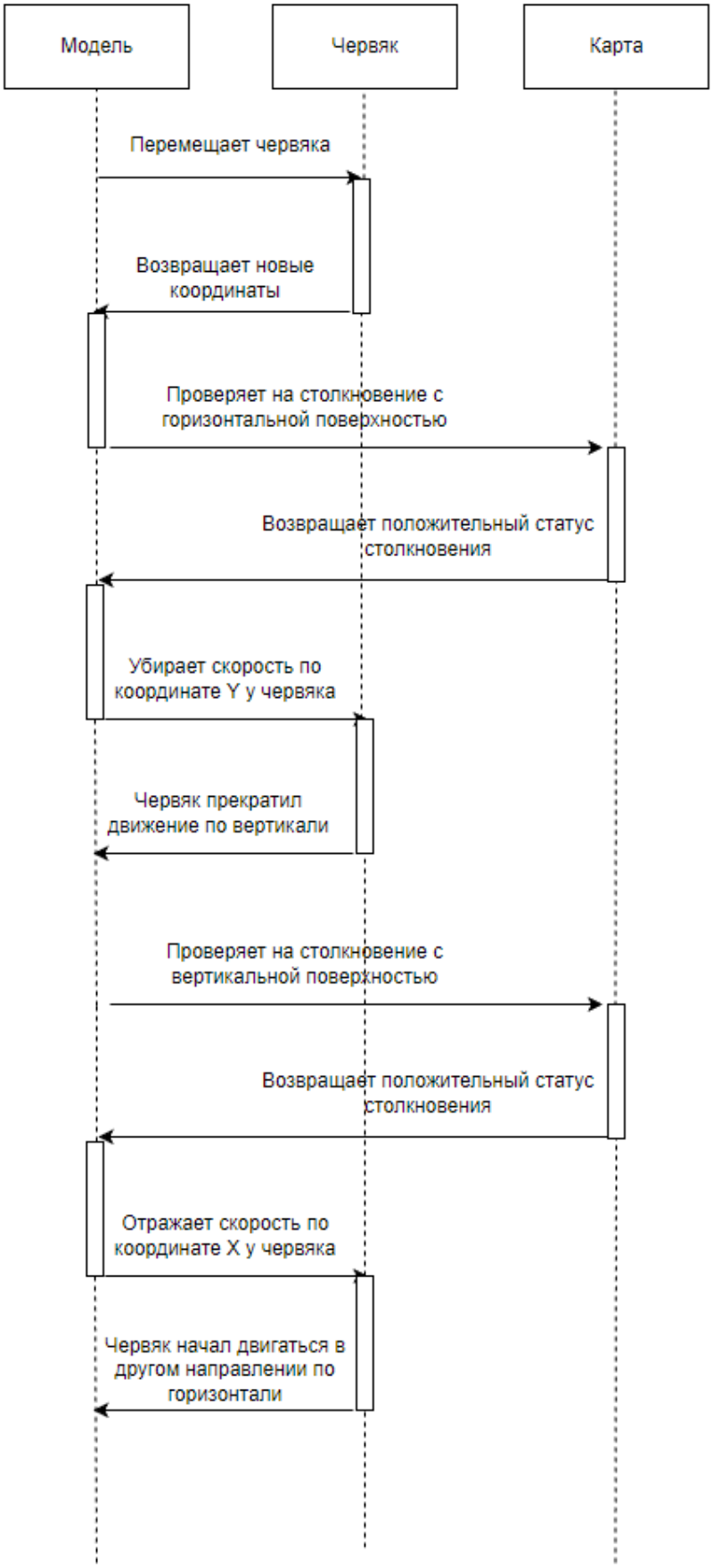

Рисунок 11 – Диаграмма последовательности столкновения червяка с поверхностью

Модель перемещает червяка в зависимости от его скорости по X и по Y. После этого она проверяет червяка на столкновение с горизонтальной поверхностью. Если он столкнулся с горизонтальной поверхностью, то прекращает его движение по вертикали. После этого она проверяет червяка на столкновение с вертикальной

поверхностью. Если червяк столкнулся с вертикальной поверхностью, то она отражает его скорость по X, из-за чего он начинает двигаться в обратном направлении по горизонтали.

1.2.4 Алгоритм столкновения снаряда базуки с червяком

Логика столкновения снаряда базуки с червяком представлена на диаграмме последовательности столкновения снаряда базуки с червяком (рисунок 12).

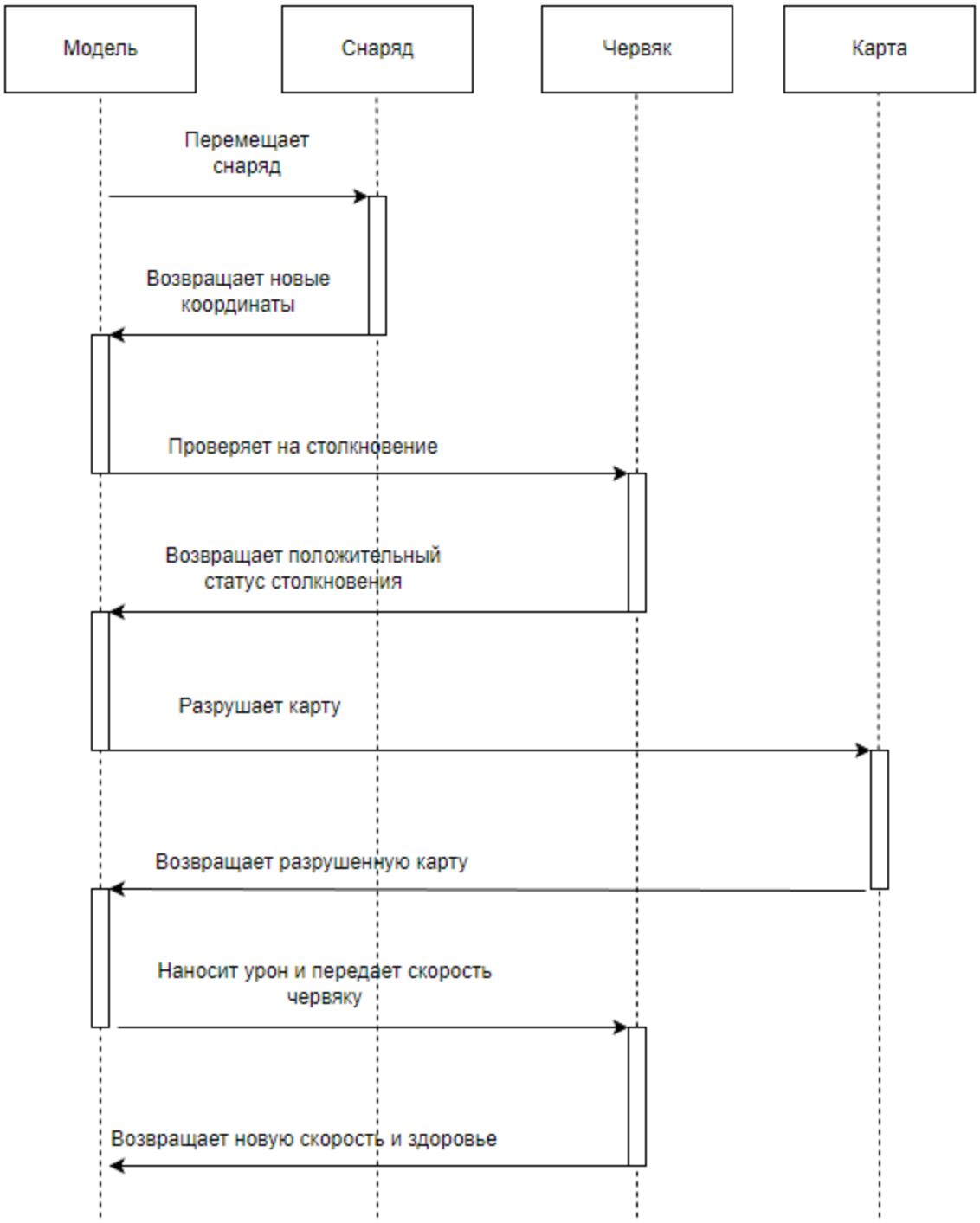

Рисунок 12 – Диаграмма последовательности столкновения снаряда базуки с червяком

Модель перемещает снаряд в зависимости от его скорости по X и по Y. После этого она проверяет снаряд базуки на столкновение с червяком. Если он столкнулся с червяком, то модель разрушает карту, а также наносит червяку урон и передает ему скорость по X и по Y.

1.3 Разработка интерфейса программы

Интерфейс главного меню программы состоит из 4 пунктов: играть, рекорды, справка, выход (рисунок 13).

Рисунок 13 – Интерфейс главного меню

Интерфейс справки содержит текстовую информацию (рисунок 14).

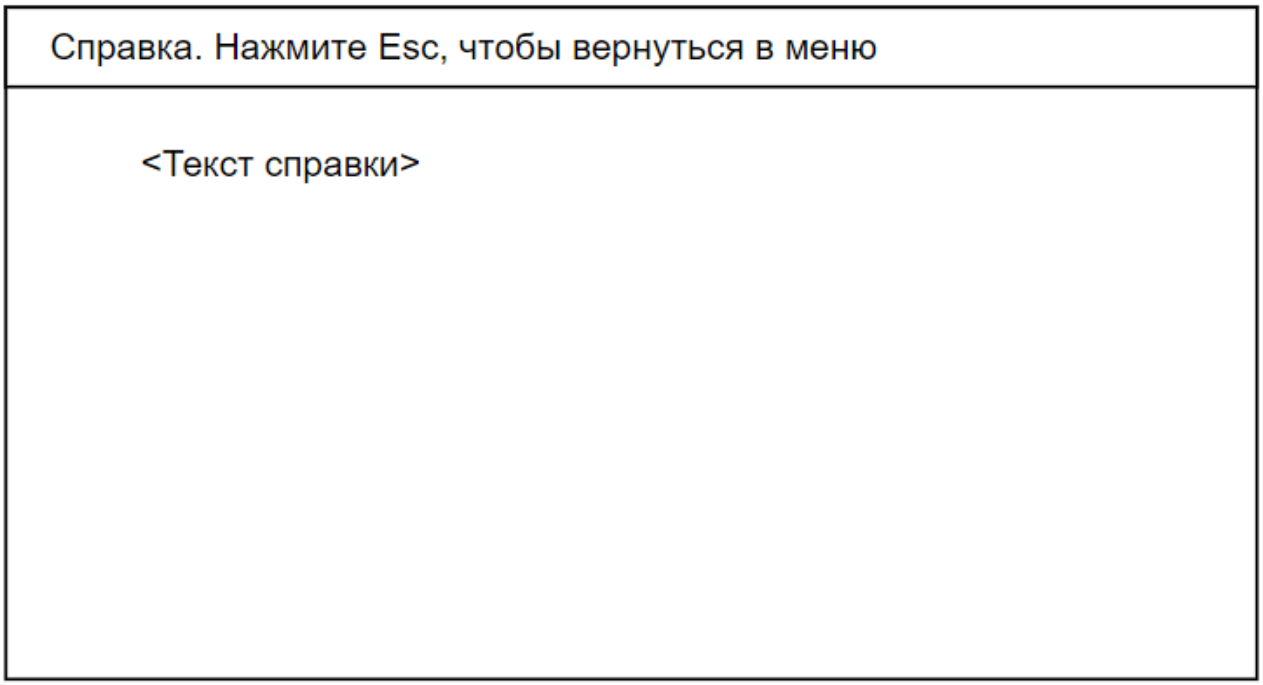

Рисунок 14 – Интерфейс справки

Интерфейс таблицы рекордов содержит номер в списке, имя игрока и количество набранных очков (рисунок 15).

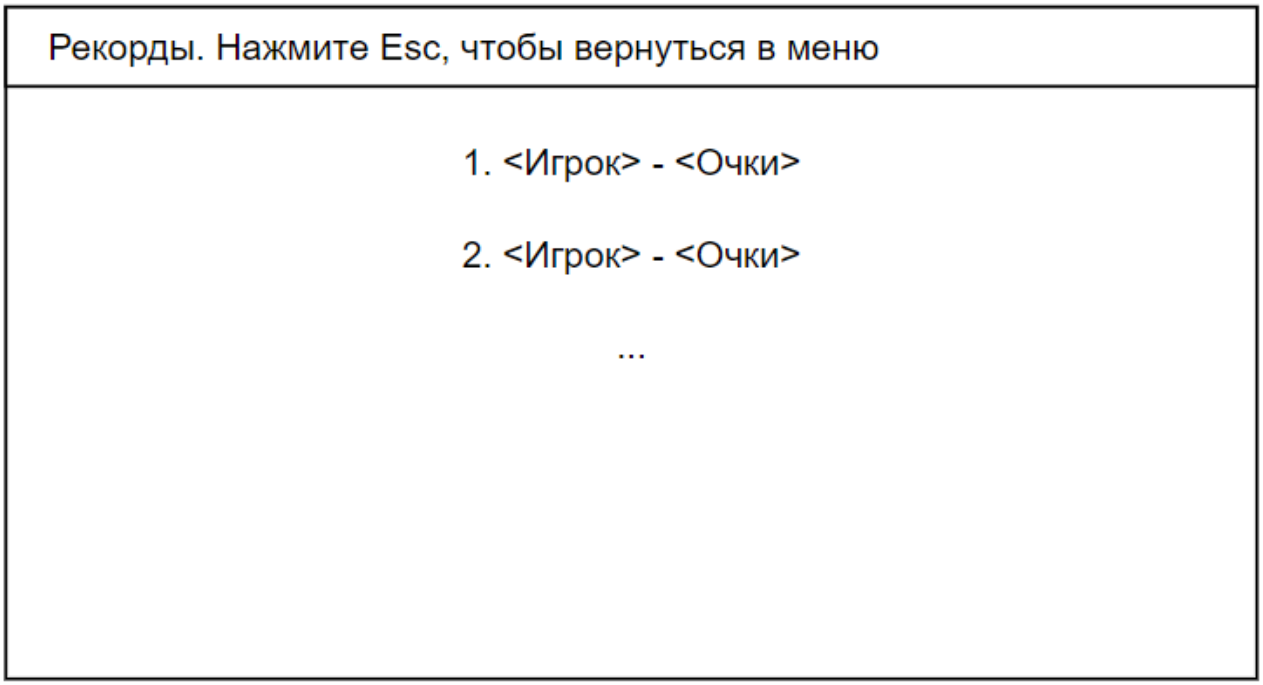

Рисунок 15 – Интерфейс таблицы рекордов

Интерфейс игры представлен на рисунке 16. На рисунке представлены следующие обозначения:

- прямоугольник – червяк;
- X – таймер игры;
- N – таймер хода;
- H – здоровье червяка;
- img – иконка выбранного оружия;
- K – количество оружия;
- стрелка над прямоугольником – указатель на текущего червяка;
- стрелка сверху справа – направление ветра.

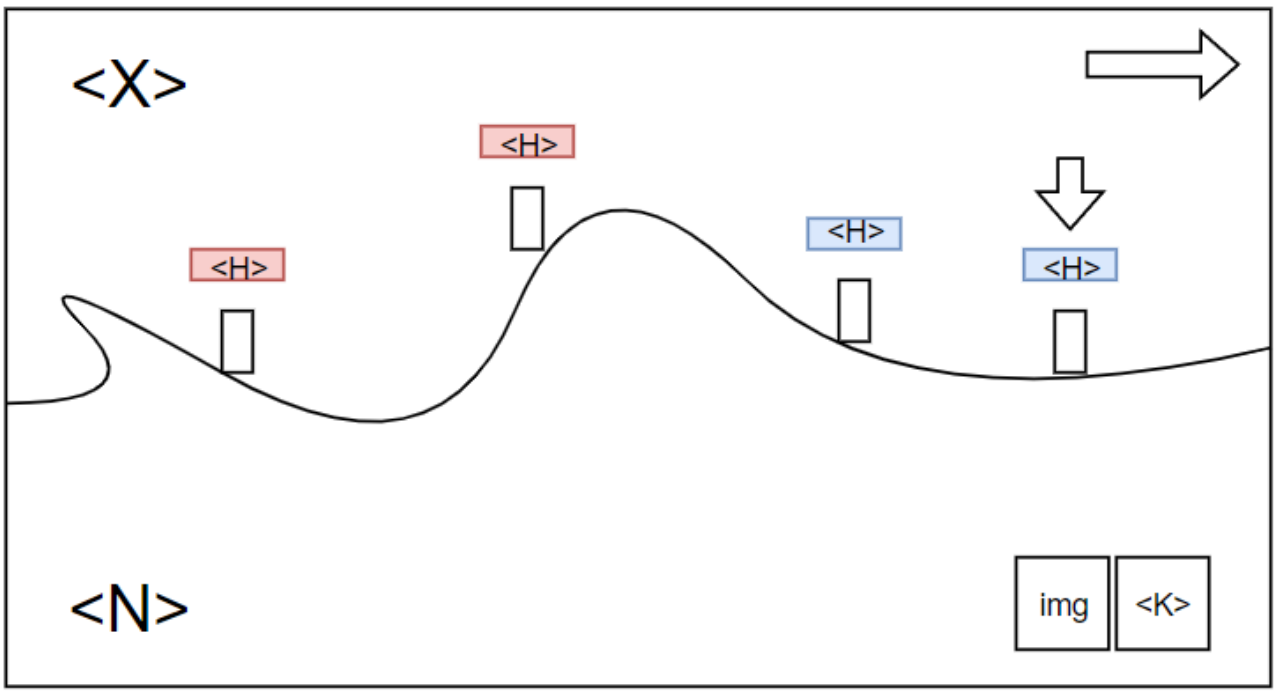

Рисунок 16 – Интерфейс игры

Интерфейс окна с результатом игры представлен на рисунке 17. В случае достижения рекорда выигравший игрок должен ввести свое имя.

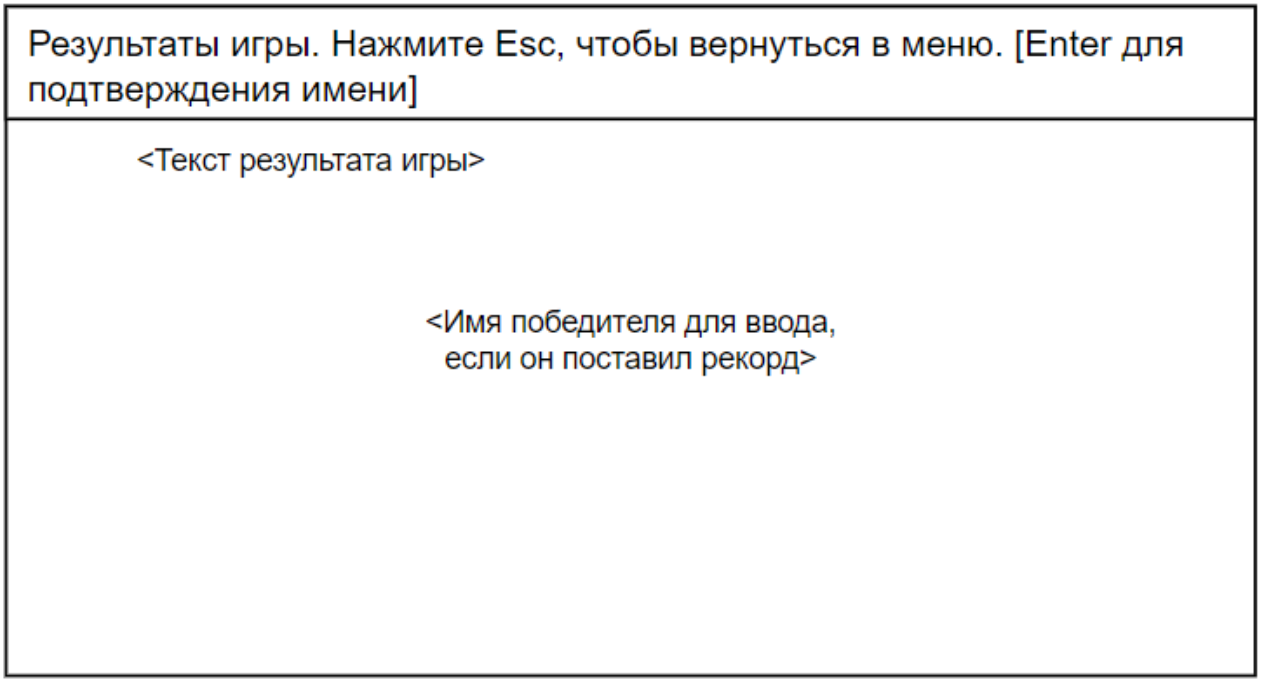

Рисунок 17 – Интерфейс окна с результатом игры

2 Написание программы

2.1.1 Описание разработанных процедур и функций

Таблица 1 – Описание разработанных процедур и функций

Класс/Абстрактный класс	Метод	Описание	Параметры
WormsGame			
AbstractControllersFactory	`public abstract MainMenuController CreateMenuController()`	Создает контроллер меню	-
	`public abstract GameController CreateGameController()`	Создает контроллер игры	-
	`public abstract RecordsController CreateRecordsController()`	Создает контроллер рекордов	-
	`public abstract HelpController CreateHelpController()`	Создает контроллер справки	-
	`public abstract GameEndedScreenController CreateGameEndedScreenController()`	Создает контроллер экрана конца игры	-
WormsApplication	`private void Init()`	Инициализациру ет приложение	-
	`private void InitMainMenuController()`	Инициализирует котроллер главного меню	-
	`private void InitGameController()`	Инициализирует котроллер игры	-
	`private void InitRecordsController()`	Инициализирует котроллер рекордов	-
	`private void InitHelpController()`	Инициализирует котроллер справки	-

	`private void InitGameEndedScreenController()`	Инициализирует котроллер экрана конца игры	-
	`public virtual void Start()`	Запускает приложение	-
WormsGame. BaseMVC			
Controller<M, V>	`public virtual void Start()`	Запускает контроллер	-
	`public virtual void Stop()`	Останавливает контроллер	-
View<M>	`public abstract void Draw()`	Рисует представление	-
	`public virtual void Start()`	Запускает представление	-
	`public virtual void Stop()`	Останавливает представление	-
WormsGame.Game.Controller			
GameController	`private void SelectWorm(Worm parNextWorm)`	Выбирает червяка для управления	`parNextWorm - червяк`
	`public void BackToMenu()`	Возвращает в меню	-
	`public override void Start()`	Запускает контроллер	-
	`public override void Stop()`	Останавливает контроллер	-
WormsGame.Game.Model			
GameInitializer	`public static void Init()`	Инициализирует игру	-
	`public static void InitPlayers()`	Инициализирует игроков	-
	`private static Inventory InitInventory()`	Инициализирует инвентарь	-
	`private static List<Worm> InitWorms(Player parPlayer)`	Инициализирует червяков	`parPlayer - игрок`
	`private static int GetYWorm(int parX)`	Получает Y червяка, при котором он будет стоять на поверхности карты в координате X	`parX – координата`
	`public static void Start()`	Запускает игру	-
	`public static void Stop()`	Останавливает игру	-
	`public static List<Worm> GetAllWorms()`	Получает всех червяков	-
	`public static void AddObject(PhysicalObject parPhysicalObject)`	Добавляет новый физический объект в список всех физических объектов	`parPhysicalObject - физический объект`
	`public static void RemoveObject(PhysicalObject`	Удаляет физический	`parPhysicalObject - физический`

GameModel	parPhysicalObject)	объект из списка всех физических объектов	объект
	public static void AddDamagedWorm(Worm parWorm)	Добавляет червяка в список червяков, которым нанесли урон	parWorm - червяк
	public static void RemoveDamagedWorm(Worm parWorm)	Удаляет червяка из списка червяков, которым нанесли урон	parWorm - червяк
	private static void RemovePlayersWithoutWorms()	Удаляет игроков без червяков	-
	private static bool IsGameEnd()	Проверяет закончилась ли игра	-
	public static void EndGame(Player parPlayer)	Вызывает событие окончания игры	parPlayer – победивший игрок
	private static void SetNextPlayerIndex()	Устанавливает индекс следующего червяка	-
	private static void PrepareWormForTurn()	Подготавливает червяка для хода	-
	private static void StartNextTurn()	Запускает следующий ход	-
	public static void SetRandomWindStrength()	Установливает случайную силу ветра	-
	private static void PrepareNextTurn()	Подготавливает следующий ход	-
	public static void WaitingForChangingTurn()	Запускает ожидание смены хода	-
	public static Worm GetCurrentWorm()	Получает текущего управляемого червяка	-
	public static void TurnCompletion()	Завершает ход	-
	public static void RemoveControl()	Убирает контроль над текущим червяком	-
GameTimers	public static void InitTimers()	Инициализация таймеров	-
	public static void CountDown(object source, ElapsedEventArgs e)	Вызывается при отсчете таймера хода	-
	private static void CountDownGame(object sender, ElapsedEventArgs e)	Вызывается при отсчете таймера игры	-
	public static void ResetTurnTimer()	Сбрасывает таймер хода	-

	`public static void StopTimers()`	Останавливает таймеры	-
MapReader	`public static void ReadMapFromFile(string parFilename)`	Читает карту из файла	`parFilename` – название файла
PhysicalObject	`public int GetCenterX()`	Получает координату X центра объекта	-
	`public int GetCenterY()`	Получает координату Y центра объекта	-
	`public bool IsOnGround()`	Проверяет находится ли на поверхности объект	-
	`public bool IsCollapse(int parX, int parY)`	Проверяет столкнулся ли с картой объект	`parX` – координата X, `parY` – координата Y
	`public void Update()`	Обновляет состояние объекта	-
	`public abstract void OnCollapse()`	Обрабатывает столкновения с картой	-
	`public abstract void OnGoingOutOfBounds()`	Обрабатывает выхода за границы карты	-
Player	`public Worm GetCurrentWorm()`	Получает управляемого червяка	-
	`public void NextWorm()`	Выбирает следующего червяка для управления	-
Worm	`private void SetRandomState()`	Установливает случайное состояние червяка	-
	`public override void OnCollapse()`	Обрабатывает столкновения с картой	-
	`public override void OnGoingOutOfBounds()`	Обрабатывает выхода за границы карты	-
	`private bool IsWormControlling()`	Провяеряет управляется ли червяк	-
	`public void Kill()`	Убивает червяка без взрыва, используется при выходе червяка за границы экрана	-
	`public void KillWithExplosion()`	Убивает червяка со взрывом	-
	`private int FindObstacleHeight(int parX)`	Находит высоту	`parX` – координата X

		препятствия в координате X	
	`public void MoveRight()`	Червяк делает шаг вправо	-
	`private void TryRightMove()`	Червяк пытается передвинуть червяка вправо	-
	`public void MoveLeft()`	Червяк делает шаг влево	-
	`private void TryLeftMove()`	Червяк пытается передвинуть червяка влево	-
	`public void JumpForward()`	Червяк выполняет прыжок вперед	-
	`public void BackFlip()`	Червяк выполняет прыжок назад	-
	`public void UseWeapon()`	Червяк использует оружие	-
WormsGame.Game.Model.Weapons			
Bazooka	`public override void Use(Worm parWorm)`	Использует базуку	parWorm - червяк
BazookaProjectile	`public override void OnCollapse()`	Обрабатывает столкновения с картой	-
ExplosionMaker	`public static void MakeExplosion(int parX, int parY, int parRadius, int parTransmittedSpeed, int parDamage)`	Создает взрыв	parX - X центра взрыва, parY - Y центра взрыва, parRadius - радиус взрыва, parTransmittedSpeed - передаваемая скорость, parDamage - урон
	`private static void MapDestruction(int parX, int parY, int parRadius)`	Разрушает карту	parX - X центра взрыва, parY - Y центра взрыва, parRadius - радиус взрыва
	`private static void WormsInfluence(int parX, int parY, int parRadius, int parTransmittedSpeed, int parDamage)`	Проявляет воздействие на червяков	parX - X центра взрыва, parY - Y центра взрыва, parRadius - радиус взрыва, parTransmittedSpeed - передаваемая скорость, parDamage - урон
	`private static double GetDistance(int parX1, int parY1, int parX2, int parY2)`	Получает расстояние между тчоками	`parX1 – координата X1, parY1 – координата Y1, parX2 – координата X2, parY2 – координата Y2`
	`public static bool IsInExplosion(int parXPoint, int parYPoint, int parXExplosion, int parYExplosion, int parRadius)`	Проверяет находится ли точка в радиусе взрыва	`parXPoint - X точки, parYPoint - Y точки, parXExplosion - X центра взрыва, parYExplosion - Y центра взрыва,`

			parRadius – радиус взрыва
Grenade	public override void Use(Worm parWorm)	Использует гранату	parWorm – червяк
GrenadeProjectile	public void CountDown(object source, System.Timers.ElapsedEventArgs e)	Обрабатывает отсчет таймера	-
	public override void OnCollapse()	Обрабатывает столкновения с картой	-
Projectile	public override void OnGoingOutOfBounds()	Обрабатывает выход за границы игровой карты	
Weapon	public void RiseAngle()	Поднимает угол оружия	-
	public void DownAngle()	Опускает угол оружия	-
	public void PowerUp()	Добавляет заряд оружия	-
	public void SetInitialState()	Устанавливает начальное состояние	-
	public abstract void Use(Worm parWorm)	Использует оружие	parWorm – червяк
WormsGame.GameEndedScreen.Controller			
GameEndedScreenController	public void SetGameResult(Player parPlayer, int parScore)	Устанавливает результат игры	parPlayer - игрок, parScore - очки
	public void BackToMenu()	Возвращает в меню	-
WormsGame.GameEndedScreen.Model			
GameEndedScreenModel	public void SetIsRecord()	Установает свойство рекорд ли	-
	public string GetEndText()	Получает текст конца игры	-
	public bool SaveRecord()	Сохраняет рекорд	-
WormsGame.GameEndedScreen.View			
GameEndedScreenView	private void Redraw()	Перерисовать представление	-
WormsGame.Help.Controller			
HelpController	ublic void BackToMenu()	Возвращает в меню	-
WormsGame.Help.Model			
HelpModel	public string GetHelpText()	Получает текст справки	-
	public void NeedRedraw()	Вызывает событие о необходимости перерисовки	-
WormsGame.Help.View			
HelpView	private void Redraw()	Перерисовывает представление справки	-
WormsGame.MainMenu			
Menu	public MenuItem	Получает пункт	parIndex – пункт

	`GetMenuItemByIndex(int parIndex)`	меню по индексу	`меню`
WormsGame.MainMenu.Controller			
MainMenuController	`public override void Start()`	Запускает контроллер главного меню	-
	`public override void Stop()`	Останавливает контроллер главного меню	-
	`private void MenuClickHandler(MenuItem parSelectedMenuItem)`	Обрабатывает нажатия на выбранный пункт меню	`parSelectedMenuItem – выбранный пункт меню`
WormsGame.MainMenu.Model			
MainMenuModel	`public void NextMenuItem()`	Выбирает следующий пункт меню	-
	`public void PreviousMenuItem()`	Выбирает предыдущий пункт меню	-
	`public void FocusMenuItem(int parIndex)`	Выделяет пункт меню	`parIndex – пункт меню`
	`public void Enter()`	Нажимает на выбранный пункт меню	-
	`public void NeedRedraw()`	Вызывает событие о необходимости перерисовки	-
MenuItemTextFormer	`public static string GetTitle(MenuItem parMenuItem)`	Получает текст для пункта меню	`parMenuItem – пункт меню`
WormsGame.MainMenu.View			
MainMenuView	`private void Redraw()`	Перерисовывает представление главного меню	-
WormsGame.Records			
RecordsRepository	`public bool IsRecord(int parScore)`	Проверяет является ли рекордом данное количество очков	`parScore - очки`
	`public void SaveRecord(GameRecord parRecord)`	Сохраняет рекорд	`parRecord - рекорд`
	`public void SaveRecords(List<GameRecord> parRecords)`	Сохраняет рекорды	`parRecords - рекорды`
	`public List<GameRecord> GetAllRecords()`	Получает все рекорды	-
	`private void SortRecords(List<GameRecord> parRecords)`	Сортирует рекорды	`parRecords - рекорды`
WormsGame.Records.Controller			
RecordsController	`public override void Start()`	Запускает контроллер рекордов	-
	`public void BackToMenu()`	Возвращает в меню	-
WormsGame.Records.Model			

GameRecord	`public override string ToString()`	Получает строку, представляющую объект	-
RecordsModel	`public void NeedRedraw()`	Вызывает событие о необходимости перерисовки	-
	`public List<GameRecord> GetRecords()`	Получает рекорды	-
	`public void UpdateRecords()`	Обновляет рекорды	-
WormsWPF			
WPFControllersFactory	`public override GameController CreateGameController()`	Создает контроллер игры	-
	`public override GameEndedScreenController CreateGameEndedScreenController()`	Создает контроллер экрана конца игры	-
	`public override HelpController CreateHelpController()`	Создает контроллер справки	-
	`public override MainMenuController CreateMenuController()`	Создает контроллер меню	-
	`public override RecordsController CreateRecordsController()`	Создает контроллер рекордов	-
WPFWormsApplication	`private void ConfigureWindow()`	Конфигурирует окно	-
	`public override void Start()`	Запускает приложение	-
WormsWPF.Game.Controller			
WPFGameController	`private void OnClose(object? sender, EventArgs e)`	Обработывает закрытие окна	-
	`public void KeyDownHandler(object sender, KeyEventArgs e)`	Обработывает нажатие клавиши клавиатуры	-
	`public void Window_KeyUp(object sender, KeyEventArgs e)`	Обработывает отпускание клавиши клавиатуры	-
	`public override void Start()`	Запускает WPF контроллер игры	-
	`public override void Stop()`	Останавливает WPF контроллер игры	-
WormsWPF.Game.View			
PhysicalObjectViewFormer	`public static PhysicalObjectView GetShape(PhysicalObject parPhysicalObject)`	Получает представление для физического объекта	`parPhysicalObject` - физический объект
WPFGameView	`private void ViewInit()`	Инициализирует представление	-
	`private void SetWindow()`	Настроивает окно	-
	`private void SetHandlers()`	Присваивает обработчики	-

	`private void UnSetHandlers()`	Убирает обработчики	-
	`private void ChangeGameTime(int parTime)`	Изменяет время у представления таймера игры	`parTime` - время
	`private void ChangeTurnTime(int parTime)`	Изменяет время у представления таймера хода	`parTime` - время
	`private void DeletePhysicalObjectView(PhysicalObject parPhycicalObject)`	Удаляет представление физического объекта из списка представлений	`parPhycicalObject` - физический объект
WormsWPF.Game.View.ObjectsView			
BazookaProjectileView	`public override void Draw(Canvas parCanvas)`	Рисует снаряд базуки	`parCanvas` - канвас
DamageView	`public override void Draw(Canvas parCanvas)`	Рисует представление нанесенного по червяку урона	`parCanvas` - канвас
ExplosionView	`public override void Draw(Canvas parCanvas)`	Рисует представление взрыва	`parCanvas` - канвас
GrenadeProjectileView	`public override void Draw(Canvas parCanvas)`	Рисует представление гранаты	`parCanvas` - канвас
ObjectView	`public abstract void Draw(Canvas parCanvas)`	Рисует представление объекта	`parCanvas` - канвас
PlayerHealthView	`public override void Draw(Canvas parCanvas)`	Рисует представление здоровья команды игрока	`parCanvas` - канвас
TimeView	`public override void Draw(Canvas parCanvas)`	Рисует представление таймера	`parCanvas` - канвас
WindView	`public override void Draw(Canvas parCanvas)`	Рисует представление ветра	`parCanvas` - канвас
WormView	`public override void Draw(Canvas parCanvas)`	Рисует представление червяка	`parCanvas` - канвас
	`private void DrawWormShape(Canvas parCanvas)`	Рисует тело червяка	`parCanvas` - канвас
	`private void DrawWormHealth(Canvas parCanvas)`	Рисует здоровье червяка	`parCanvas` - канвас
	`private void DrawWormWeapon(Canvas parCanvas)`	Рисует оружие червяка	`parCanvas` - канвас
	`private void DrawWeaponPower(Canvas parCanvas)`	Рисует заряд оружия	`parCanvas` - канвас
	`private void DrawWeaponIcon(Canvas parCanvas)`	Рисует иконку оружия	`parCanvas` - канвас
	`private static void DrawWeaponCount(Canvas parCanvas)`	Рисует количество оружия	`parCanvas` - канвас

	`private void DrawAim(Canvas parCanvas)`	Рисует прицел	`parCanvas` – канвас
	`private void DrawWormPointer(Canvas parCanvas)`	Рисует указатель на червяка	`parCanvas` – канвас
WormsWPF.GameEndedScreen.Controller			
WPFGameEndedScreenController	`private void KeyDownHandler(object sender, KeyEventArgs e)`	Обрабатывает нажатие клавиши клавиатуры	-
	`public override void Start()`	Запускает WPF контроллер экрана конца игры	-
	`public override void Stop()`	Останавливает WPF контроллер экрана конца игры	-
WormsWPF.GameEndedScreen.View			
WPFGameEndedScreenView	`private void SetupCanvas()`	Настраивает канвас	-
	`public override void Draw()`	Рисует WPF представление экрана конца игры	-
	`public override void Start()`	Запускает WPF представление экрана конца игры	-
	`public override void Stop()`	Останавливает WPF представление экрана конца игры	-
WormsWPF.Help.Controller			
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	-
WormsWPF.Help.View			
WPFHelpView	`private void SetupCanvas()`	Настраивает канвас	-
	`public override void Draw()`	Рисует WPF представление справки	-
	`public override void Start()`	Запускает WPF представление справки	-
	`public override void Stop()`	Останавливает WPF представление справки	-
WormsWPF.MainMenu.Controller			

WPFMainMenuController	`private void KeyDownHandler(object sender, KeyEventArgs e)`	Обрабатывает нажатие клавиши клавиатуры	-
	`public override void Start()`	Запускает WPF контроллер главного меню	-
	`public override void Stop()`	Останавливает WPF контроллер главного меню	-
WormsWPF.MainMenu.View			
WPFMainMenuView	`private void SetupCanvas()`	Настраивает канвас	-
	`public override void Draw()`	Рисует WPF представление главного меню	-
	`public override void Start()`	Запускает WPF представление главного меню	-
	`public override void Stop()`	Останавливает WPF представление главного меню	-
WormsWPF.Records.Controller			
WPFRecordsController	`private void KeyDownHandler(object sender, KeyEventArgs e)`	Обрабатывает нажатие клавиши клавиатуры	-
	`public override void Start()`	Запускает WPF контроллер рекордов	-
	`public override void Stop()`	Останавливает WPF контроллер рекордов	-
WormsWPF.Records.View			
WPFRecordsView	`private void SetupCanvas()`	Настраивает канвас	-
	`public override void Draw()`	Рисует WPF представление рекордов	-
	`public override void Start()`	Запускает WPF представление рекордов	-
	`public override void Stop()`	Останавливает WPF представление рекордов	-
WormsConsole			
ConsoleControllersFactory	`public override GameController CreateGameController()`	Создает контроллер игры	-
	`public override GameEndedScreenController CreateGameEndedScreenController()`	Создает контроллер экрана конца игры	-
	`public override HelpController CreateHelpController()`	Создает контроллер справки	-
	`public override MainMenuController`	Создает	-

	`CreateMenuController()`	контроллер главного меню	
	`public override RecordsController CreateRecordsController()`	Создает контроллер рекордов	-
ConsoleDrawer	`public static void DrawRectangle(int parX, int parY, int parWidth, int parHeight, ConsoleColor color)`	Рисует прямоугольник	`parX` - координата X, `parY` - координата Y, `parWidth` - ширина прямоугольника, `parHeight` - высота прямоугольника, `color` - цвет
ConsoleWormsApplication	`private void ConfigureWindow()`	Конфигурирует окно	-
FastConsoleWorker	`public static void SetPixel(int parX, int parY, short attribute)`	Установливает пиксель	`parX` - координата X, `parY` - координата Y, `attribute` - цвет
	`public static void Init(short parWidth, short parHeight)`	Инициализирует Помощник по работе с быстрым выводом в консоль	`parWidth` - ширина, `parHeight` - высота
	`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)`	Рисует прямоуголник в буфере	`parX` - координата X, `parY` - координата Y, `parWidth` - ширина, `parHeight` - выста, `parColor` - цвет
	`public static void SetFontSize(short parXSize, short parYSize)`	Устанавливает размер шрифта	`parXSize` - размер по X, `parYSize` - размер по Y
WormsConsole.Game.Controller			
ConsoleGameController	`public void ReadKeysStart()`	Запускат считывание нажатых клавиш в консоли	-
	`public void ReadKeysStop()`	Останавливает считывание клавиш	-
	`public override void Start()`	Запускает консольный контроллер игры	-
	`public override void Stop()`	Останвливает консольный контроллер игры	-
WormsConsole.Game.View			
ConsoleGameView	`public void ViewInit()`	Инициализирует	-

		консольное представление игры	
	`private void DeletePhysicalObjectView(PhysicalObject parPhycicalObject)`	Удаляет представление физического объекта из списка представлений	`parPhycicalObject` – физический объект
	`private void AddNewView(PhysicalObject parPhysicalObject)`	Добавление представления физического объекта в список представлений	`parPhycicalObject` – физический объект
	`private void DrawMap()`	Рисует карту	-
	`private void StartDrawing()`	Запускает отрисовку консольного представления игры	-
	`public override void Draw()`	Рисует консольное представление игры	-
	`public override void Start()`	Запускает консольное представление игры	-
	`public override void Stop()`	Останавливает консольное представление игры	-
PhysicalObjectViewFormer	`public static PhysicalObjectView GetShape(PhysicalObject parPhysicalObject)`	Получает представление для физического объекта	`parPhysicalObject` – физический объект
WormsConsole.GameEndedScreen.Controller			
ConsoleGameEndedScreenController	`public void ReadKeysStart()`	Запускает считывание нажатых клавиш в консоли конца игры	-
	`public void ReadKeysStop()`	Останавливает считывание клавиш в консоли конца игры	-
	`public override void Start()`	Запускает консольный контроллер экрана конца игры	-
	`public override void Stop()`	Останавливает консольный контроллер экрана конца	-

		игры	
WormsConsole.GameEndedScreen.View			
ConsoleGameEndedScreenView	`public void SetupWindow()`	Настраивает окно	-
	`private void InitialDraw()`	Начальная отрисовка при запуске	-
	`public override void Draw()`	Рисует введенное имя победителя	-
	`public override void Start()`	Запускает консольное представление экрана конца игры	-
	`public override void Stop()`	Останавливает консольное представление экрана конца игры	-
WormsConsole.Help.Controller			
ConsoleHelpController	`public void ReadKeysStart()`	Запускает считывание нажатых клавиш в консоли справки	-
	`public void ReadKeysStop()`	Останавливает считывание клавиш в консоли справки	-
	`public override void Start()`	Запускает консольный контроллер справки	-
	`public override void Stop()`	Останавливает консольный контроллер справки	-
WormsConsole.Help.View			
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	-
WormsConsole.MainMenu.Controller			
ConsoleMainMenuController	`public void ReadKeysStart()`	Запускает считывание нажатых клавиш в консоли главного меню	-

	`public void ReadKeysStop()`	Останавливает считывание клавиш в консоли главного меню	-
	`public override void Start()`	Запускает консольный контроллер главного меню	-
	`public override void Stop()`	Останавливает консольный контроллер главного меню	-
WormsConsole.MainMenu.View	`public void SetupWindow()`	Настраивает окно	-
	`public override void Draw()`	Рисует консольное представление главного меню	-
	`public override void Start()`	Запускает консольное представление главного меню	-
	`public override void Stop()`	Останавливает консольное представление главного меню	-
WormsConsole.Records.Controller			
ConsoleRecordsController	`public void ReadKeysStart()`	Запускает считывание нажатых клавиш в консоли рекордов	-
	`public void ReadKeysStop()`	Останавливает считывание клавиш в консоли рекордов	-
	`public override void Start()`	Запускает консольный контроллер рекордов	-
	`public override void Stop()`	Останавливает консольный контроллер рекордов	-
WormsConsole.Records.View			
ConsoleRecordsView	`public override void Draw()`	Рисует консольное представление рекордов	-
	`public override void Start()`	Запускает консольное представление рекордов	-
	`public override void Stop()`	Останавливает	-

		консольное представление рекордов	

2.2 Разработка программы

Описание последовательности разработки:

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

2.2.1 Описание классов, перечислений и интерфейсов проекта

Пространство имен	**Название класса / интерфейса / перечисления**	**Назначение**
WormsGame	AbstractControllersFactory	Абстрактный класс фабрики контроллеров
WormsGame	WormsApplication	Класс приложения
WormsGame.BaseMVC	Controller	Абстрактный контроллер MVC
WormsGame.BaseMVC	Model	Абстрактная модель MVC
WormsGame.BaseMVC	View	Абстрактное представлени MVC
WormsGame.Controller	GameController	Абстрактный контроллер игры
WormsGame.Base	GameInitializer	Класс инициализации игры
WormsGame.Base	GameModel	Класс модели игры
WormsGame.Base	GameTimers	Класс игрового таймера
WormsGame.Base	Inventory	Класс инвентаря
WormsGame.Base	MapReader	Класс считывателя карты
WormsGame.Base	PhysicalObject	Абстрактный класс физического объекта
WormsGame.Base	Player	Класс игрока
WormsGame.Base	PlayersColor	Класс перечислений цвета игроков
WormsGame.Base	State	Класс перечислений состояний червяка
WormsGame.Base	Worm	Класс червяка
WormsGame.Base.Weapons	Bazooka	Класс базуки
WormsGame.Base.Weapons	BazookaProjectile	Класс снаряда базуки
WormsGame.Base.Weapons	ExplosionMaker	Класс создателя

		взрывов
WormsGame.Base.Weapons	Grenade	Класс гранаты
WormsGame.Base.Weapons	GrenadeProjectile	Класс снаряда игры
WormsGame.Base.Weapons	Projectile	Класс снаряда
WormsGame.Base.Weapons	Weapon	Класс оружия
WormsGame.GameEndedScreen .Controller	GameEndedScreenController	Класс контроллера экрана конца игры
WormsGame.GameEndedScreen .Model	GameEndedScreenModel	Класс модели экрана конца игры
WormsGame.GameEndedScreen .View	GameEndedScreenView	Абстрактный класс представления экрана конца игры
WormsGame.Help.Controller	HelpController	Абстрактный класс контроллера справки
WormsGame.Help.Model	HelpModel	Класс модели справки
WormsGame.Help.View	HelpView	Абстрактный класс представления справки
WormsGame.MainMenu .Controller	MainMenuController	Класс контроллера главного меню
WormsGame.MainMenu.Model	MainMenuModel	Класс модели главного меню
WormsGame.MainMenu.Model	MenuItemTextFormer	Класс формирователя текста для пунктов меню
WormsGame.MainMenu.View	MainMenuView	Абстрактный класс представления главного меню
WormsGame.MainMenu	Menu	Класс главного меню
WormsGame.MainMenu	MenuItem	Класс перечислений пунктов главного меню
WormsGame.Records .Controller	RecordsController	Абстрактный класс контроллера рекордов
WormsGame.Records.Model	GameRecord	Класс игрового рекорда
WormsGame.Records.Model	RecordsModel	Класс модели рекордов
WormsGame.Records.View	RecordsView	Абстрактный класс представления рекордов
WormsGame.Records	RecordsRepository	Класс репозитория рекордов
WormsWPF	WPFControllersFactory	Класс фабрики WPF контроллеров
WormsWPF	WPFWormsApplication	Класс WPF приложения
WormsWPF.WPFController	WPFGameController	Класс WPF контроллера игры

WPFWorms.WPFView	PhysicalObjectViewFormer	Класс формирователя представления для физичеких объектов
WormsWPF	WPFGameView	Класс WPF представления игры
WPFWorms.WPFView .ObjectsView	BazookaProjectileView	Класс представления снаряда базуки
WPFWorms.WPFView .ObjectsView	DamageView	Класс представления нанесенного по червяку урона
WPFWorms.WPFView .ObjectsView	ExplosionView	Класс представления взрыва
WPFWorms.WPFView .ObjectsView	GrenadeProjectileView	Класс представления гранаты
WPFWorms.WPFView .ObjectsView	ObjectView	Абстрактный класс представления объекта
WPFWorms.WPFView .ObjectsView	PhysicalObjectView	Абстрактный класс представления физического объекта
WPFWorms.WPFView .ObjectsView	PlayerHealthView	Класс представления здоровья команды игрока
WPFWorms.WPFView .ObjectsView	TimeView	Класс представления таймера
WPFWorms.WPFView .ObjectsView	WindView	Класс представления ветра
WPFWorms.WPFView .ObjectsView	WormView	Класс представления червяка
WormsWPF.GameEndedScreen .Controller	WPFGameEndedScreen Controller	Класс WPF контроллера экрана конца игры
WormsWPF.GameEndedScreen .Model	WPFGameEndedScreenView	Класс WPF представления экрана конца игры
WormsWPF.Help.Controller	WPFHelpController	Класс WPF контроллера справки
WormsWPF.Help.View	WPFHelpView	Класс WPF представления справки
WormsWPF.MainMenu .Controller	WPFMainMenuController	Класс WPF контроллера главного меню
WormsWPF.MainMenu.View	WPFMainMenuView	Класс WPF представления главного меню
WormsWPF.Records.Controller	WPFRecordsController	Класс WPF контроллера рекордов
WormsWPF.Records.View	WPFRecordsView	Класс WPF представления

		рекордов
WormsConsole	ConsoleControllersFactory	Класс фабрики консольных контроллеров
WormsConsole	ConsoleDrawer	Класс консольного рисовальщика
WormsConsole	ConsoleUtils	Класс средств для осуществления работы помощника для работы с быстрым выводом консоли
WormsConsole	ConsoleWormsApplication	Класс консольного приложения
WormsConsole	FastConsoleWorker	Класс помощника по работе с быстрым выводом в консоль
WormsConsole.ControllerConsole	ConsoleGameController	Класс консольного контроллера игры
WormsConsole.ViewConsole	ConsoleGameView	Класс консольного представления игры
WormsConsole.ViewConsole	PhysicalObjectViewFormer	Класс формирователя представлений физических объектов
WormsConsole.ViewConsole .ViewObjects	BazookaProjectileView	Класс представления снаряда базуки
WormsConsole.ViewConsole .ViewObjects	GameTimeView	Класс представления игрового таймера
WormsConsole.ViewConsole .ViewObjects	GrenadeProjectileView	Класс представления гранаты
WormsConsole.ViewConsole .ViewObjects	ObjectView	Абстрактный класс представления объекта
WormsConsole.ViewConsole .ViewObjects	PhysicalObjectView	Абстрактный класс представления физического объекта
WormsConsole.ViewConsole .ViewObjects	TurnTimeView	Класс представления таймера хода
WormsConsole.ViewConsole .ViewObjects	WormView	Класс представления червяка
WormsConsole.GameEndedScreen .Controller	ConsoleGameEndedScreen Controller	Класс консольного контроллера экрана конца игры
WormsConsole.GameEndedScreen .View	ConsoleGameEndedScreenView	Класс консольного представления экрана конца игры
WormsConsole.Help.Controller	ConsoleHelpController	Класс консольного контроллера справки
WormsConsole.Help.View	ConsoleHelpView	Класс представления справки
WormsConsole.MainMenu	ConsoleMainMenuController	Класс консольного

.Controller		контроллера главного меню
WormsConsole.MainMenu.View	ConsoleMainMenuView	Класс консольного представления главного меню
WormsConsole.Records .Controller	ConsoleRecordsController	Класс консольного контроллера рекордов
WormsConsole.Records.View	ConsoleRecordsView	Класс консольного представления рекордов

2.3 Описание шаблонов проектирования, которые использовались при написании программы

2.3.1 Абстрактная фабрика (Abstract factory)

Абстрактная фабрика – это порождающий паттерн проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов. Шаблон реализуется созданием абстрактного класса, который представляет собой интерфейс для создания компонентов системы. Затем пишутся классы, реализующие этот интерфейс.

Фабрика контроллеров реализуется в классе AbstractControllersFactory, фабрика WPF контроллеров – в классе WPFControllersFactory, фабрика консольных контроллеров – в классе ConsoleControllersFactory. Шаблон используется в классе WormsApplication.

2.3.2 Одиночка (Singleton)

Одиночка – порождающий шаблон проектирования, гарантирующий, что в однопоточном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру.

Шаблон реализуется в классе RecordsRepository, используется в классе RecordsModel.

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

2.4.1 Вынесение констант

Программный код до вынесения константы:

```
    /// <summary>
    /// Убить червяка со взрывом
    /// </summary>
    public void KillWithExplosion()
    {
      Player.Worms.Remove(this);
      GameModel.RemoveObject(this);
      ExplosionMaker.MakeExplosion(GetCenterX(), GetCenterY(), 50, 10, 25);
    }
```

Программный код после вынесения константы:

```
    /// <summary>
    /// Радиус взрыва червяка
    /// </summary>
    private const int WORM_EXPLOSION_RADIUS = 50;
    /// <summary>
    /// Передаваемая скорость взрывом червяка
    /// </summary>
    private const int WORM_EXPLOSION_TRANSMITTED_SPEED = 10;
    /// <summary>
    /// Урон взрыва червяка
```

```
    /// </summary>
    private const int WORM_EXPLOSION_DAMAGE = 25;

    /// <summary>
    /// Убить червяка со взрывом
    /// </summary>
    public void KillWithExplosion()
    {
      Player.Worms.Remove(this);
      GameModel.RemoveObject(this);
      ExplosionMaker.MakeExplosion(GetCenterX(), GetCenterY(), WORM_EXPLOSION_RADIUS,
WORM_EXPLOSION_TRANSMITTED_SPEED, WORM_EXPLOSION_DAMAGE);
    }
```

2.4.2 Переименование метода

Программный код до переименования метода:

```
/// <summary>
    /// Удалить игроков без червяков
    /// </summary>
    private static void RemovePlayers()
    {
      int i = 0;
      while (i < Players.Count)
      {
        if (Players[i].Worms.Count == 0)
        {
          Players.RemoveAt(i);
        }
        else
        {
          i++;
        }
      }
    }
```

Программный код после переименования метода:

```
    /// <summary>
    /// Удалить игроков без червяков
    /// </summary>
    private static void RemovePlayersWithoutWorms()
    {
      int i = 0;
      while (i < Players.Count)
      {
        if (Players[i].Worms.Count == 0)
        {
          Players.RemoveAt(i);
        }
        else
        {
          i++;
        }
      }
    }
```

2.4.3 Вынесение кода в отдельный метод

Программный код до вынесения кода в отдельный метод:

```
    /// <summary>
    /// Приготовить следующий ход
    /// </summary>
    private static void PrepareNextTurn()
    {
      int i = 0;
      while (i < Players.Count)
      {
        if (Players[i].Worms.Count == 0)
        {
          Players.RemoveAt(i);
        }
        else
        {
          i++;
        }
      }

      if (IsGameEnd())
      {
```

```
        return;
      }

      Thread.Sleep(2000);

      WindStrength = _random.Next(WIND_STRENGTH_MIN, WIND_STRENGTH_MAX + 1);

      _currentPlayerIndex++;

      if (_currentPlayerIndex >= Players.Count)
      {
        _currentPlayerIndex = 0;
      }

      Players[_currentPlayerIndex].NextWorm();
      Players[_currentPlayerIndex].GetCurrentWorm().IsUsedWeapon = false;
      _isSelectedWormControlling = true;

      WormControlChangedEvent?.Invoke(GetCurrentWorm());
      GameTimers.TurnTimer.Enabled = true;
      GameTimers.ResetTurnTimer();

    }
```

Программный код после вынесения кода в отдельный метод:

```
/// <summary>
    /// Приготовить следующий ход
    /// </summary>
    private static void PrepareNextTurn()
    {
      RemovePlayersWithoutWorms();
      if (IsGameEnd())
      {
        return;
      }
      Thread.Sleep(2000);
      SetRandomWindStrength();
      SetNextPlayerIndex();
      PrepareWormForTurn();
      StartNextTurn();
    }

    /// <summary>
    /// Удалить игроков без червяков
    /// </summary>
    private static void RemovePlayersWithoutWorms()
    {
      int i = 0;
      while (i < Players.Count)
      {
        if (Players[i].Worms.Count == 0)
        {
          Players.RemoveAt(i);
        }
        else
        {
          i++;
        }
      }
    }

    /// <summary>
    /// Установить индекс следующего червяка
    /// </summary>
    private static void SetNextPlayerIndex()
    {
      _currentPlayerIndex++;

      if (_currentPlayerIndex >= Players.Count)
      {
         currentPlayerIndex = 0;
      }
    }

    /// <summary>
    /// Приготовить червяка для хода
    /// </summary>
    private static void PrepareWormForTurn()
    {
      Players[ currentPlayerIndex].NextWorm();
      Players[_currentPlayerIndex].GetCurrentWorm().IsUsedWeapon = false;
      _isSelectedWormControlling = true;
```

```
    }

    /// <summary>
    /// Запустить следующий ход
    /// </summary>
    private static void StartNextTurn()
    {
      WormControlChangedEvent?.Invoke(GetCurrentWorm());
      GameTimers.TurnTimer.Enabled = true;
      GameTimers.ResetTurnTimer();
    }

    /// <summary>
    /// Установить случайную силу ветра
    /// </summary>
    public static void SetRandomWindStrength()
    {
      WindStrength = _random.Next(WIND_STRENGTH_MIN, WIND_STRENGTH_MAX + 1);
    }
```

2.5 Разработка тестов

2.5.1 Test Cases

№	Название	Предусловия	Шаги	Ожидаемый результат
1	Тест на успешное движение червяка влево	Карта инициализирована, червяк находится на земле	Вызывается метод передвижения червяка влево	Координата X червяка уменьшается на один пиксель
2	Тест на успешное движение червяка вправо	Карта инициализирована, червяк находится на земле	Вызывается метод передвижения червяка вправо	Координата X червяка увеличивается на один пиксель
3	Тест на успешное движение червяка вправо через низкое препятствие	Карта инициализирована с низким препятствием, червяк находится на земле	Вызывается метод передвижения червяка вправо через низкое препятствие	Координата X червяка увеличивается на один пиксель, Координата Y увеличивается на высоту препятствия
4	Тест на успешное движение червяка влево через низкое препятствие	Карта инициализирована с низким препятствием, червяк находится на земле	Вызывается метод передвижения червяка влево через низкое препятствие	Координата X червяка уменьшается на один пиксель, Координата Y увеличивается на высоту препятствия
5	Тест на движение червяка вправо через высокое препятствие	Карта инициализирована с высоким препятствием, червяк находится на земле	Вызывается метод передвижения червяка вправо через высокое препятствие	Координаты X,Y червяка не меняются
6	Тест на движение червяка влево через высокое препятствие	Карта инициализирована с высоким препятствием, червяк находится на земле	Вызывается метод передвижения червяка влево через высокое препятствие	Координата X,Y червяка не меняются
7	Тест на состояние	Карта	Вызывается метод	Метод проверки

	червяка на земле	инициализирована, червяк находится на земле	проверки стоит ли червяк на земле	возвращает true
8	Тест на состояние червяка, находящегося не на земле	Карта инициализирована, червяк находится не на земле	Вызывается метод проверки стоит ли червяк на земле	Метод проверки возвращает false
9	Тест на прыжок червяка вперед, когда он повернут влево	Карта инициализирована, червяк находится на земле, червяк повернут влево	Вызывается метод прыжка червяка вперед	Скорость по X и по Y червяка равны скоростям, которую придает прыжок вперед, когда червяк повернут влево
10	Тест на прыжок червяка вперед, когда он повернут вправо	Карта инициализирована, червяк находится на земле, червяк повернут вправо	Вызывается метод прыжка червяка вперед	Скорость по X и по Y червяка равны скоростям, которую придает прыжок вперед, когда червяк повернут вправо
11	Тест на прыжок червяка вперед, когда он не на земле	Карта инициализирована, червяк находится не на земле	Вызывается метод прыжка червяка вперед	Скорость по X и по Y не изменились – равны 0
12	Тест на прыжок червяка назад, когда он повернут влево	Карта инициализирована, червяк находится на земле, червяк повернут влево	Вызывается метод прыжка червяка назад	Скорость по X и по Y червяка равны скоростям, которую придает прыжок назад, когда червяк повернут влево
13	Тест на прыжок червяка назад, когда он повернут вправо	Карта инициализирована, червяк находится на земле, червяк повернут вправо	Вызывается метод прыжка червяка назад	Скорость по X и по Y червяка равны скоростям, которую придает прыжок назад, когда червяк повернут вправо
14	Тест на прыжок червяка назад, когда он находится не на земле	Карта инициализирована, червяк находится не на земле	Вызывается метод прыжка червяка назад	Скорость по X и по Y не изменились – равны 0
15	Тест на состояние червяка, столкнувшегося с землей	Карта инициализирована, червяк находится на земле	Вызывается метод проверки столкнулся ли червяк с землей	Метод проверки возвращает true
16	Тест на состояние червяка, не столкнувшегося с землей	Карта инициализирована, червяк находится не на земле	Вызывается метод проверки столкнулся ли червяк с землей	Метод проверки возвращает false
17	Тест на увеличение скорости червяка по Y, когда он в воздухе	Карта инициализирована, червяк находится не на земле	Вызывается метод обновления состояния червяка	Скорость червяка по Y увеличилась на 1
18	Тест на отсутствие увеличения скорости червяка по Y, когда он на земле	Червяк находится на земле	Вызывается метод обновления состояния червяка	Скорость червяка по Y не изменилась

19	Тест на остановку червяка на земле, когда его скорость превышает расстояние до земли	Карта инициализирована, червяк находится не на земле	Вызывается метод обновления состояния червяка	Червяк находится на земле
20	Тест на перемещение червяка посредством скорости	Карта инициализирована	Вызывается метод обновления состояния червяка	Червяк переместился на значения скоростей по X и по Y

2.5.2 Модульные тесты

Модульное тестирование или юнит-тестирование — это процесс проверки программного кода, при котором проверяют работоспособность отдельных компонентов или модулей разработанной программы. Цель модульного тестирования состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования. Модульное тестирование проводится по принципу «белого ящика», то есть основывается на знании внутренней структуры программы, и часто включает те или иные методы анализа покрытия кода.

Ниже представлен пример тестирования успешного движения червяка вправо. Инициализируется карта с землей, задаются стартовые значения координат X и Y червяка, а также ожидаемое значение координаты X. Создается червяк, для которого вызывается метод движения влево. С помощью метода Assert.AreEqual проверяется совпадение ожидаемого и реального значения координат X и Y.

```
/// <summary>
/// Координата Y земли
/// </summary>
public const int GROUND_Y = 50;
/// <summary>
/// Высота червяка
/// </summary>
public const int WORM_HEIGHT = 18;

/// <summary>
/// Тестирование успешного движения червяка влево
/// </summary>
[TestMethod]
public void TestSuccessWormMoveLeft()
{
  InitializeDefaultTestMap();
  int startX = 50;
  int expectedX = 49;
  int startY = GROUND_Y - WORM_HEIGHT;

  Worm worm = new Worm(null, startX, startY, WORM_WIDTH, WORM_HEIGHT, 100);
  worm.MoveLeft();
  Assert.AreEqual(expectedX, worm.X);
  Assert.AreEqual(startY, worm.Y);
}
```

3 Результат работы программы

3.1 Графическая версия

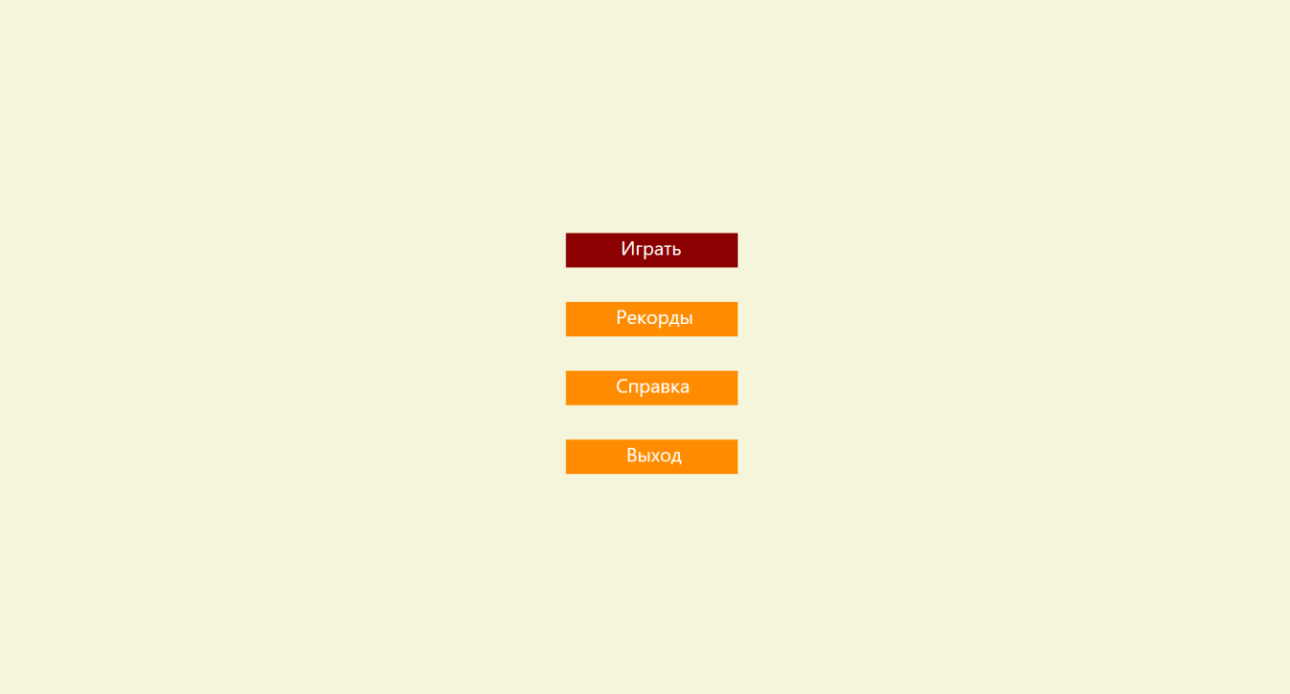

Рисунок 18 – Главное меню графической версии приложения

Рисунок 19 – Таблица рекордов графической версии приложения

Справка по игре Worms. Нажмите Esc для возврата в меню

Worms представляет собой пошаговую игру. В ней каждый игрок владеет армией червяков и инвентарем с оружием и вспомогательными предметами, которыми могут пользоваться червяки. Игроки по очереди выполняют ограниченный по времени ход, во время которого они должны попытаться нанести наибольший урон червякам противника, либо обезопасить своих червяков. Целью игры является уничтожение всех червяков противника.
Игра продолжается до тех пор, пока на поле имеются червяки по крайней мере 2-х разных коалиций, либо до тех пор, пока игровой таймер не дойдет до 0.
Управление: A, D - ходьба червяком влево, вправо. W, S - поднять, опустить прицел оружия. R, F - прыжок назад, вперед. Клавиша 1 - экипировать базуку, Клавиша 2 - экипировать гранату. Удерживание пробела - заряд экипированного оружия, отпускание пробела - выстрел экипированным оружием оружием. Escape - выйти в меню.

Рисунок 20 – Справка графической версии приложения

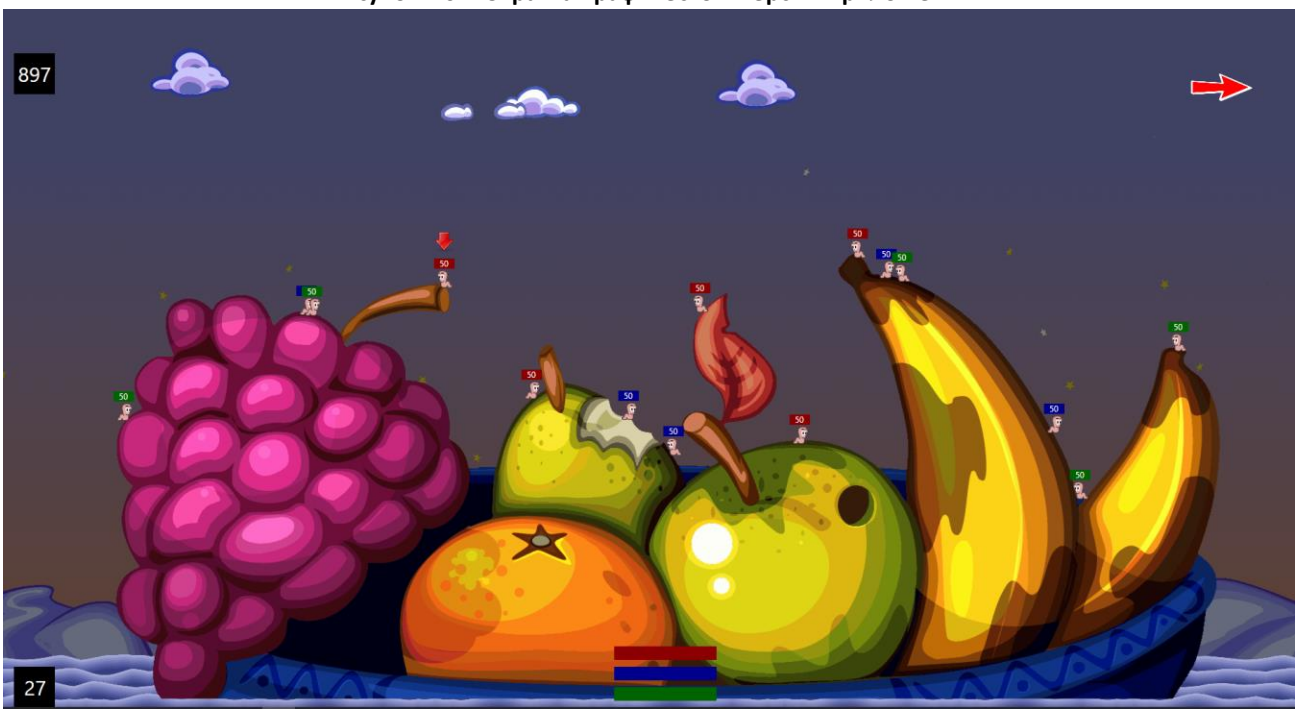

Рисунок 21 – Игра графической версии приложения

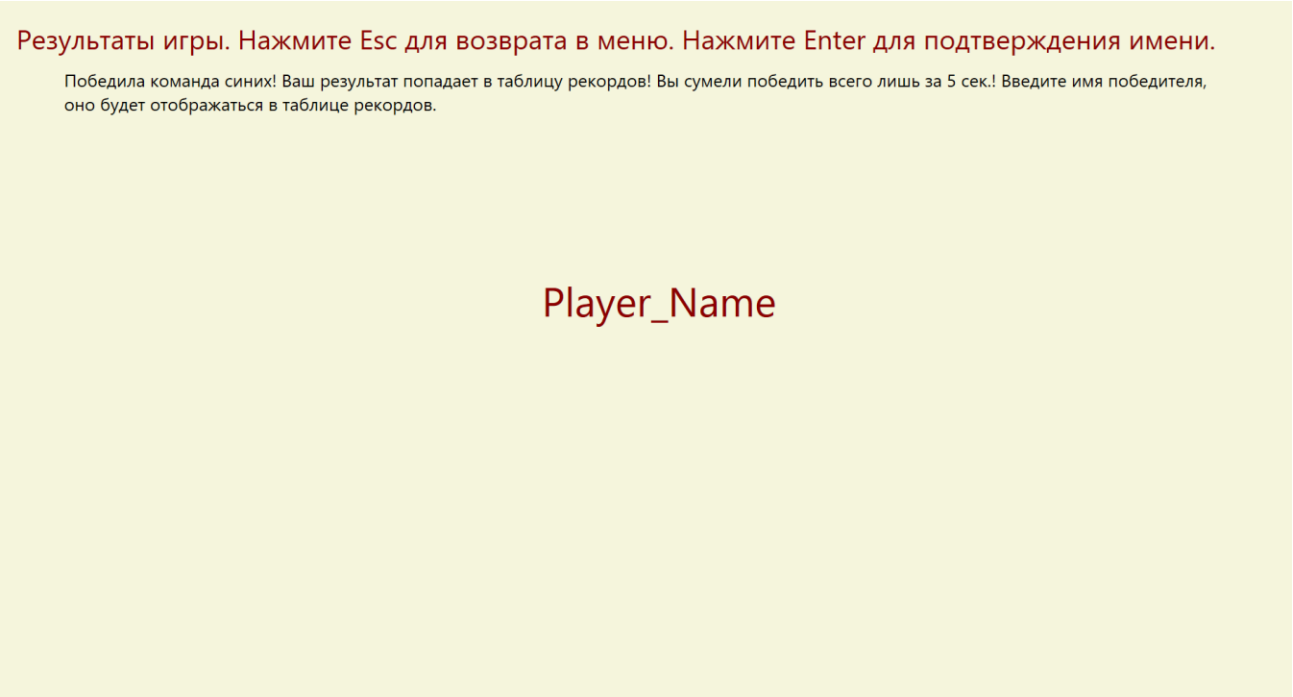

Рисунок 22 – Окно с результатом игры графической версии приложения

3.2 Консольная версия

Рисунок 23 – Главное меню консольной версии приложения

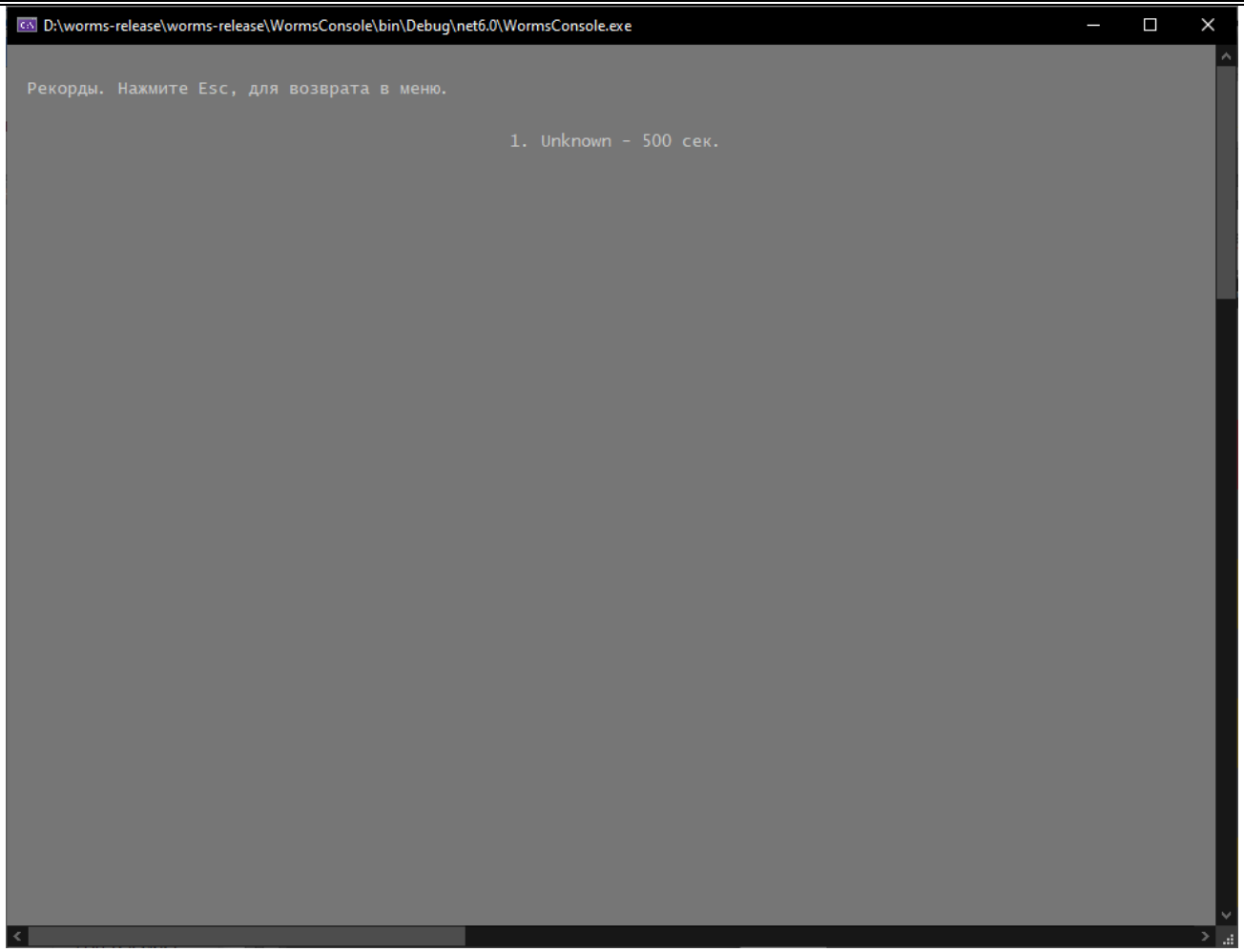

Рисунок 24 – Таблица рекордов консольной версии приложения

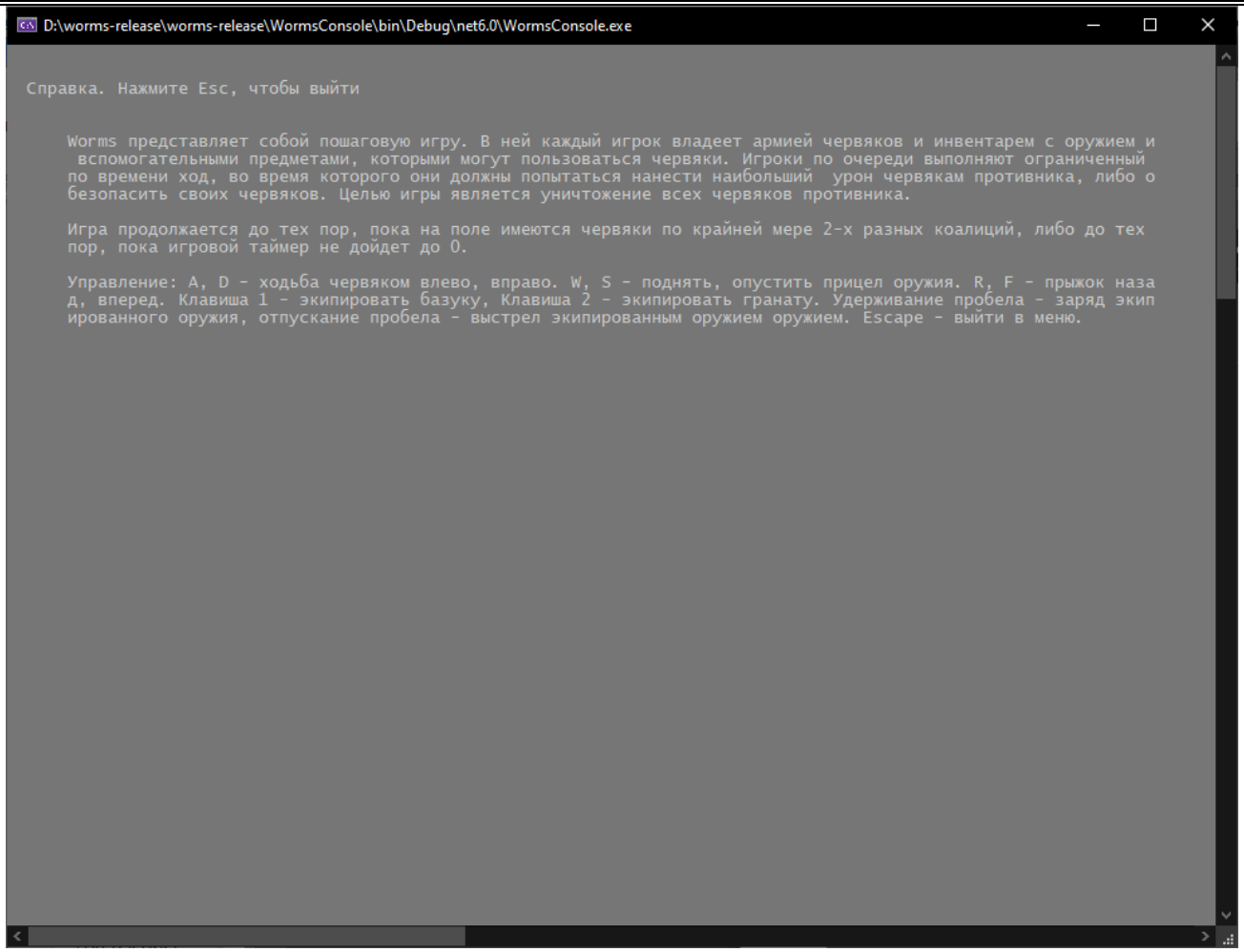

Рисунок 25 – Справка консольной версии приложения

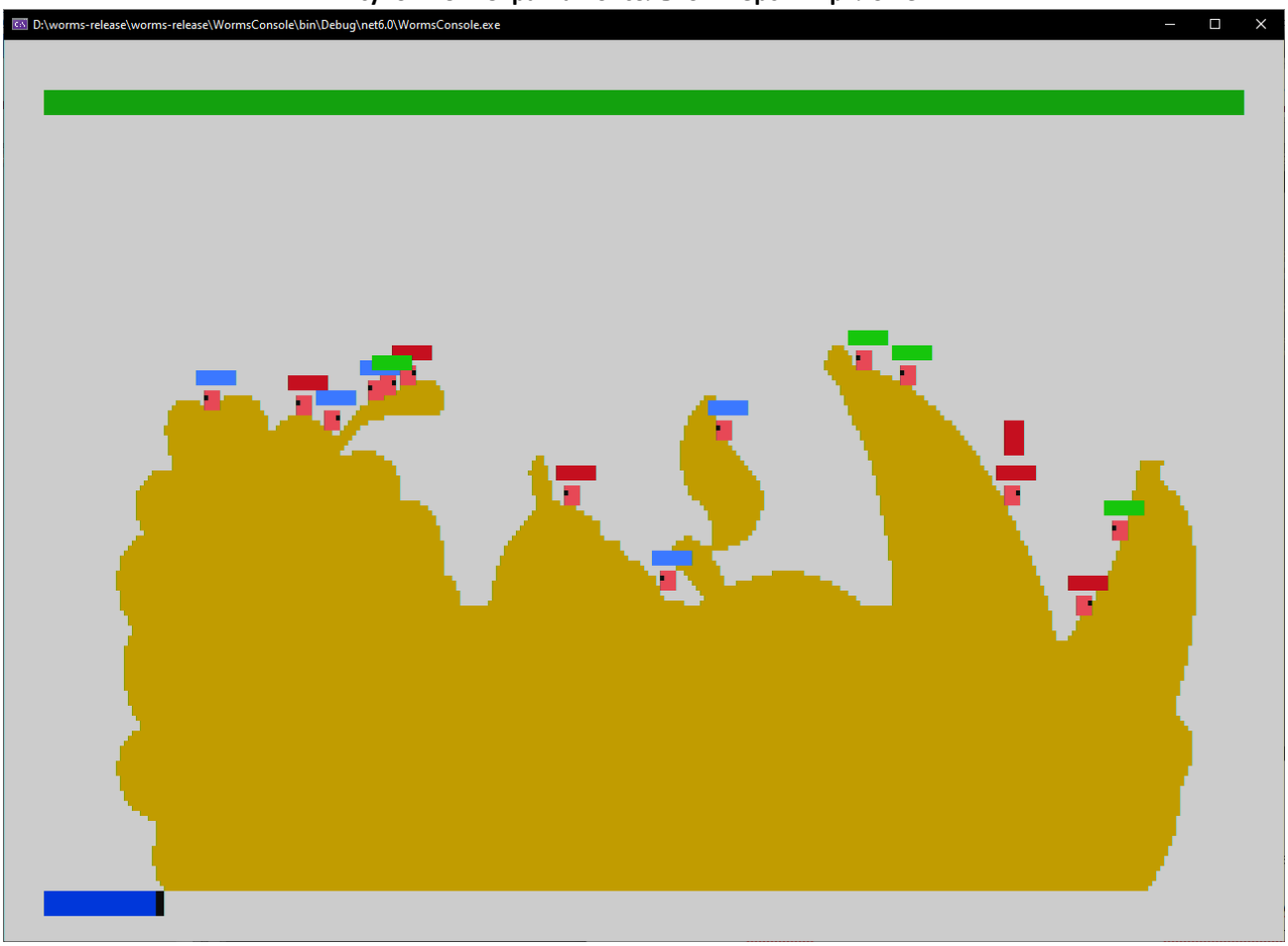

Рисунок 26 – Игра консольной версии приложения

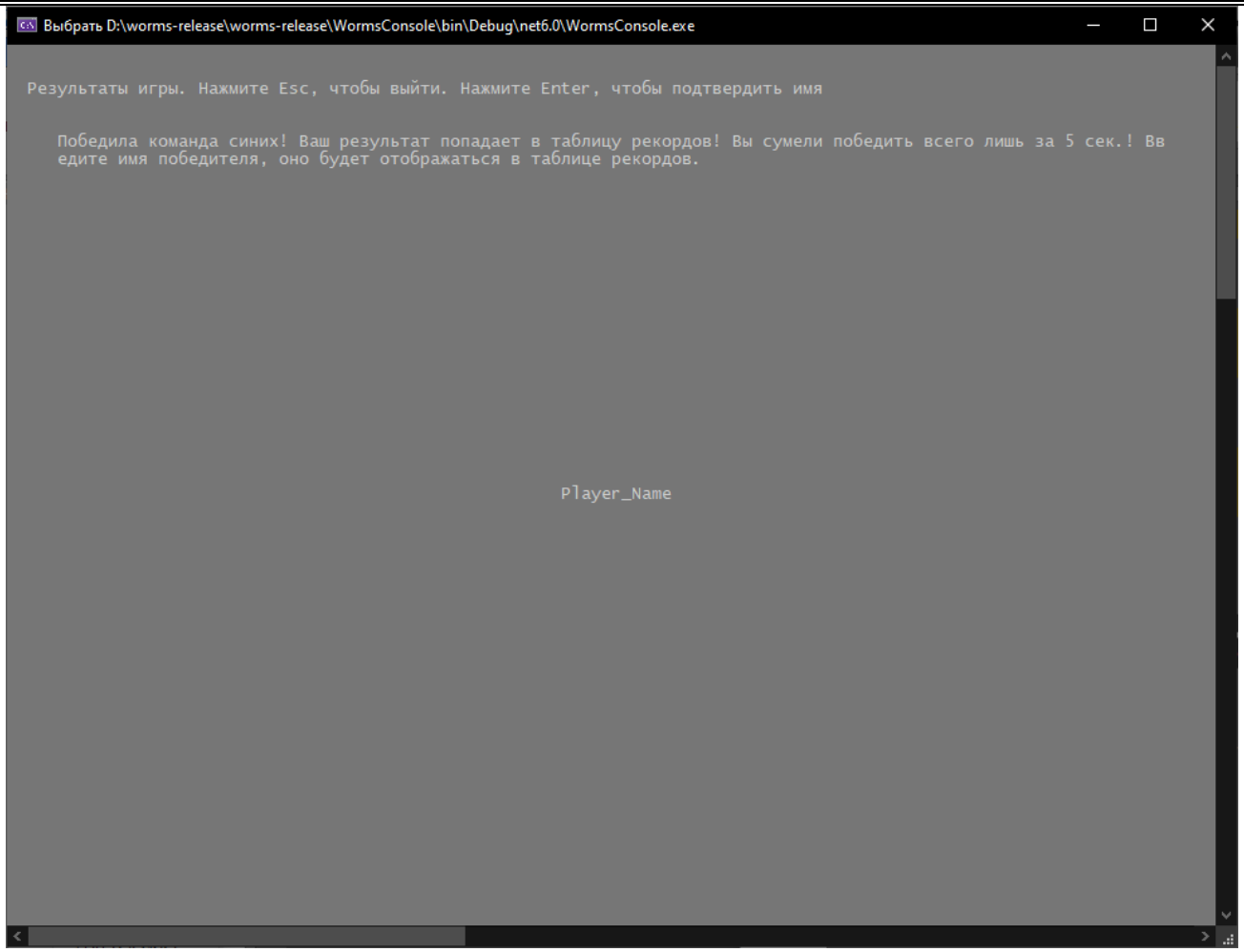

Рисунок 27 – Окно с результатом игры консольной версии приложения

Заключение

В результате выполнения курсовой работы было разработано игровое приложение Worms с 2 реализациями: консольная (используя консоль) и оконая (используя WPF). Были использованы шаблоны проектирования MVC (Модель-Вид-Контроллер) и два дополнительных шаблона проектирования: Abstract factory (Абстрактная фабрика) и Singleton (Одиночка). В приложении использовалась многопоточность и синхронизация между потоками. Для класса Worm разработаны полноценные модульные тесты, обеспечивающие проверку корректности его работы.

Приложения