

Министерство образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по курсу «Программирование»
на тему «Змейка»

Выполнил:
студент группы 24ВВВ3
Буров А.О.

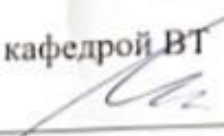
Принял:
к.т.н., доцент Токарев А.Н.

Пенза 2025

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники
Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

 М.А. Митрохин

«__» ____ 20__ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование

Студенту Буров Александр Олегович Группа 24ВВВЗ

Тема проекта: *Разработка программы сложной структуры методом нисходящего программирования.*

Исходные данные (технические требования) на проектирование

Игра "Змейка"

- инициализация игрового поля, змейки
- игровой цикл, проверка столкновений, съедение еды
- завершение игры, при столкновении с границей поля, или с телом змейки.
- подсчет очков и их вывод

Обязательные требования к программе:

1. Многомодульность.
2. Использование сложных типов данных – структур, списков, файлов.
3. Режим работы видеосистемы – текстовый / графический.
4. Устройство ввода информации – клавиатура, мышь.
5. Пользовательский интерфейс должен быть построен на основе меню и панели инструментов.
6. Наличие заставки.
7. Операционная система MS Windows.
8. Язык программирования – Си и Ассемблер.
9. Среда разработки ПО – Microsoft Visual Studio.

1. Расчетная часть

Разработка программы.

2. Графическая часть

Схема данных, схема ресурсов системы, схема работы системы, иерархическая структура программы, схема взаимодействия программы.

3. Экспериментальная часть

Отладка программы.

Срок выполнения проекта по разделам

В соответствии с графиком.

Дата выдачи задания "20" 02 2025

Дата защиты проекта "15" 05. 2025

Руководитель _____

Задание получил "20" Феврале 2025г.

Студент Буров Александр Олегович ЭП

Содержание

Введение	5
1 Постановка задачи	6
2 Выбор решения	7
3 Описание разработки программы	8
4 Отладка и тестирование	9
5 Описание программы.....	10
6 Руководство пользователя	25
Заключение.....	29
Список использованных источников	30
Приложение А Листинги	31
Приложение В Пример Файла статистики.....	50

Введение

Игра "Змейка" представляет собой классическую аркадную игру, которая уже несколько десятилетий остается популярной среди пользователей разных возрастов. Ее простые, но увлекательные правила, где игрок управляет растущей змейкой, собирая пищу и избегая столкновений, делают ее отличным объектом для изучения основ программирования.

В рамках данной курсовой работы была разработана современная версия этой игры с использованием языка программирования C# и технологии WPF. Выбор этих технологий обусловлен их широкими возможностями для создания графических интерфейсов и эффективной обработки пользовательского ввода, что особенно важно для игровых приложений.

Полученный опыт разработки может быть использован для создания более сложных игровых и прикладных приложений, а сама программа служит наглядным примером реализации классического игрового алгоритма с использованием современных инструментов разработки.

1 Постановка задачи

Была поставлена задача разработать классическую игру "Змейка" с использованием технологий WPF и C#. Основной целью являлось создание полностью функционального игрового приложения с интуитивно понятным интерфейсом и классическим геймплеем.

Требовалось разработать игру "Змейка" со следующими функциями:

- управление змейкой с помощью клавиатуры.
- Генерация еды в случайных местах игрового поля.
- Увеличение длины змейки при поедании еды.
- Подсчёт очков и сохранение статистики (рекорд, последний результат, количество сыгранных игр).
- Отображение игрового поля с сеткой и элементами игры.

Дополнительные требования включали реализацию вспомогательных систем: главного меню с возможностью начала игры и просмотра статистики, экрана завершения игры с отображением результатов и вариантами дальнейших действий, а также системы паузы. Важным аспектом было обеспечение стабильной работы приложения на различных аппаратных конфигурациях.

2 Выбор решения

2.1. Выбор технологий

Для реализации игры выбран C# и WPF благодаря:

- богатым графическим возможностям WPF.
- Удобной работе с событиями в C#.
- Поддержке объектно-ориентированного подхода.
- Простоте интеграции дополнительных компонентов.

2.2. Архитектура системы

Используется Canvas размером 20×15 клеток с программной отрисовкой сетки. Каждая клетка имеет четкие границы, а игровые элементы (змейка и еда) визуально выделяются: змейка состоит из прямоугольников с закругленными углами, а еда представлена эллипсом с эффектным оформлением для лучшей видимости.

Реализована точная обработка нажатий клавиш стрелок с плавным изменением направления. Система предотвращает резкие развороты на 180° (кроме начального этапа игры), а клавиша пробела обеспечивает мгновенную паузу/продолжение игры без потери игрового состояния.

Для генерации еды используется внешняя DLL-библиотека, гарантирующая равномерное распределение. Игровой цикл работает на стабильных 60 FPS, обеспечивая плавную анимацию. Механизм проверки столкновений обрабатывает как границы поля, так и тело змейки с минимальными вычислительными затратами.

Статистика хранится в простом текстовом формате (рекорд/последний результат/количество игр) и автоматически обновляется при каждом завершении игры. Реализована надежная система обработки ошибок чтения/записи, предотвращающая потерю данных при нештатных ситуациях.

3 Описание разработки программы

Разработка велась поэтапно, начиная с создания основных классов для змейки и игрового интерфейса. Было реализовано игровое поле размером 20×15 клеток с четкой визуализацией элементов - змейка отображается в виде цепочки прямоугольников, а еда как цветной кружок.

Ключевые функции включают плавное управление стрелками с защитой от резких поворотов, автоматическую генерацию еды с использованием внешней библиотеки, систему подсчета очков с сохранением статистики между сеансами, а также надежную проверку столкновений. Игровой процесс обеспечивает стабильную частоту обновления 60 кадров в секунду.

Архитектурно проект разделен на компоненты интерфейса, игровой логики и алгоритмов генерации, что обеспечивает модульность и возможность дальнейшего расширения функционала.

4 Отладка и тестирование

Процесс отладки осуществлялся в среде Visual Studio 2022 с применением комплексного подхода. Для анализа работы программы активно использовались точки останова, позволяющие детально отслеживать выполнение кода на критических участках. Особое внимание уделялось мониторингу значений ключевых переменных через окно отладки, что помогало выявлять неочевидные ошибки. Дополнительно было реализовано консольное логирование основных игровых событий для последующего анализа.

В ходе разработки удалось выявить и устранить несколько существенных проблем. Одной из наиболее сложных оказалась ошибка генерации еды, когда она периодически появлялась внутри тела змейки. Для решения этой проблемы был переработан алгоритм проверки свободных клеток и оптимизирован процесс выбора позиции для новой еды. Также были обнаружены и исправлены недостатки в системе сохранения статистики, включая добавление обработки исключений при работе с файлами и реализацию проверки целостности данных.

Особые сложности возникли при отладке системы управления. Первоначально наблюдалась заметная задержка реакции на нажатия клавиш, что существенно ухудшало игровой опыт. После тщательного анализа удалось оптимизировать обработку ввода и устранить задержки. Дополнительно была реализована защита от одновременного нажатия противоположных направлений, что сделало управление более предсказуемым.

Тестирование готового продукта проводилось по различным сценариям для обеспечения стабильной работы. Проверялся как стандартный игровой процесс, так и пограничные случаи - движение вдоль стен, достижение максимальной длины змейки. Особое внимание уделялось тестированию системы сохранения статистики, включая проверку корректности работы после перезапуска приложения. Для каждого тестового сценария анализировалась не только правильность работы игровой логики, но и стабильность частоты кадров, а также соответствие поведения программы ожиданиям пользователя.

5 Описание программы

Программа "Змейка" представляет собой классическую аркадную игру, разработанную с использованием технологии WPF. Приложение включает два основных режима работы: интерактивное главное меню и непосредственно игровой процесс. В главном меню пользователю доступны кнопки для запуска новой игры и выхода из приложения, а также информационная панель, отображающая игровую статистику. Система сохраняет три ключевых показателя: максимальный достигнутый результат, счет последней завершенной игры и общее количество сыгранных партий. Все статистические данные автоматически записываются в файл `snake_stats.txt` при каждом изменении и загружаются при последующих запусках программы, обеспечивая сохранение прогресса между игровыми сеансами.

Игровой процесс разворачивается на поле размером 20×15 клеток, где каждая ячейка имеет размер 25×25 пикселей. Визуальное оформление включает стилизованную змейку с четко различимыми головой (ярко-зеленый элемент) и телом (темно-зеленые сегменты), а также эффектно выделяющуюся еду в виде красного круга с градиентной заливкой. Управление реализовано через клавиши-стрелки с интеллектуальной системой обработки ввода, предотвращающей мгновенные развороты на противоположное направление (за исключением начального этапа игры при минимальной длине змейки). Игровая динамика постепенно усложняется за счет плавного увеличения скорости перемещения змейки по мере набора очков, которые начисляются по 10 баллов за каждую съеденную еду.

Особенностью программы является использование внешней динамической библиотеки `FoodCore.dll`, отвечающей за алгоритмы генерации пищи на игровом поле. При завершении игры (в случае столкновения с границами поля или собственным телом) активируется специальный экран, предлагающий игроку выбор между повторным запуском или возвратом в главное меню. Архитектурное решение проекта предусматривает возможности для дальнейшего расширения функционала, включая реализацию дополнительных игровых режимов, систему бонусов, различные уровни сложности и альтернативные визуальные темы оформления. Все элементы интерфейса оптимизированы для комфортного взаимодействия как с помощью клавиатуры, так и через мышь, обеспечивая плавную и стабильную работу на различных аппаратных конфигурациях.

5.1 Схема программы `MainWindow.xaml.cs`

Ниже представлены описание основных функций `MainWindow.xaml.cs` и их схемы.

5.1.1 Функция `public MainWindow`

Функция является конструктором класса главного окна игры и выполняет первоначальную настройку приложения. Она инициализирует графический

интерфейс, задавая размеры игрового поля 20x15 клеток (500x375 пикселей) с фиксированным окном. Схема представлена на рисунке 1.

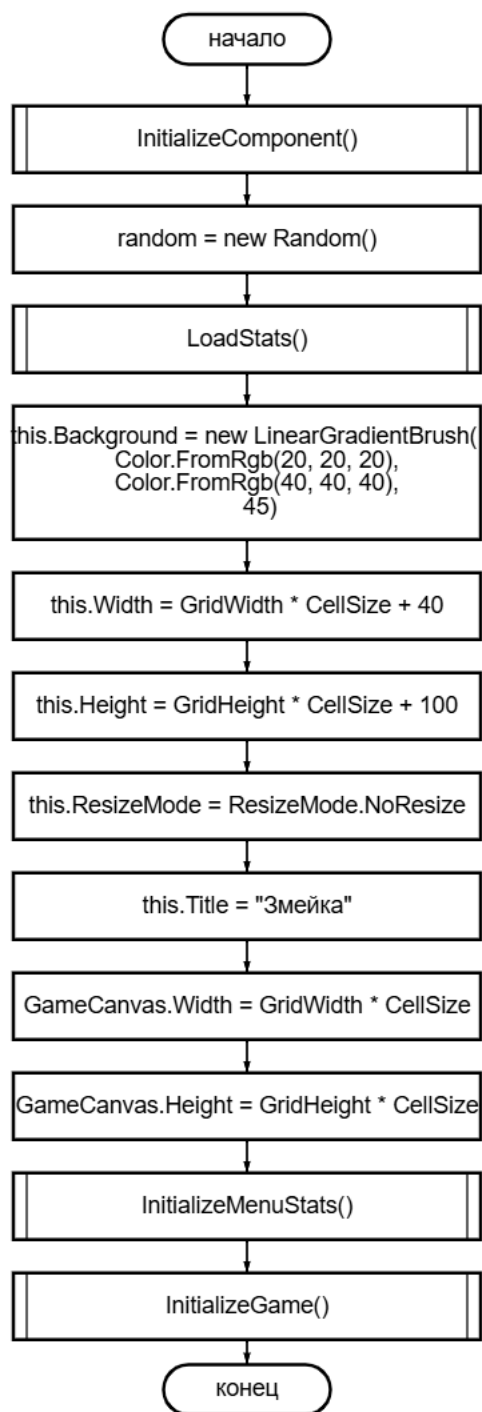


Рисунок 1 – Схема функции public MainWindow

5.1.2 Функция UpdateStats

Функция UpdateStats обновляет игровую статистику, а именно сохраняет текущий счёт как последний результат (lastScore), увеличивает счётчик сыгранных игр (gamesPlayed++) и обновляет рекорд (maxScore), если текущий счёт его превысил. Затем сохраняет статистику в файл (SaveStats()) и обновляет отображение в меню (UpdateMenuStats()). Схема представлена на рисунке 2.

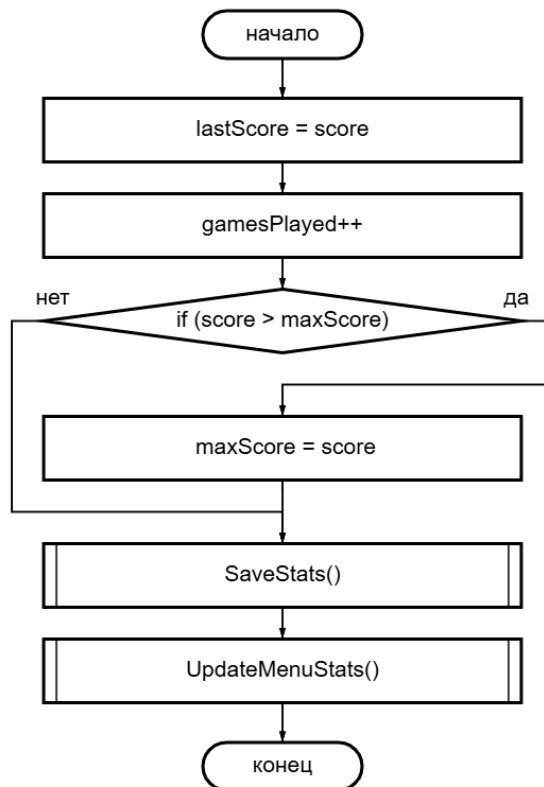


Рисунок 2 - Схема функции UpdateStats

5.1.3 Функция InitializeGame

Функция InitializeGame выполняет подготовку к новой игре, а именно очищает игровое поле, создает змейку в центре, сбрасывает направление движения и счет на 0, генерирует новую еду и показывает главное меню. Схема представлена на рисунке 3.

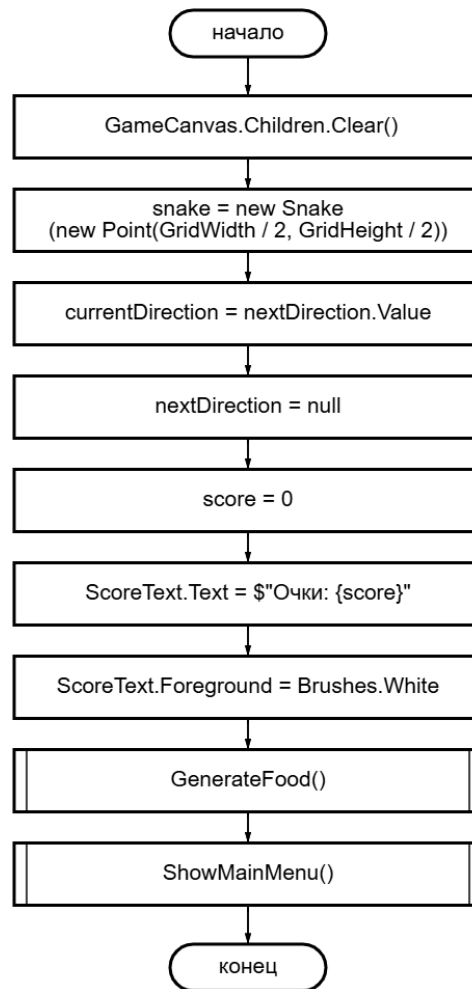


Рисунок 3 - Схема функции InitializeGame

5.1.4 Функция StartGame

Функция StartGame запускает игровой процесс, а именно скрывает меню, отображает игровое поле и счет, создает таймер для обновления игры (200 мс) и рисует игровую сетку. Проверяет наличие змейки перед запуском. Схема представлена на рисунке 4.

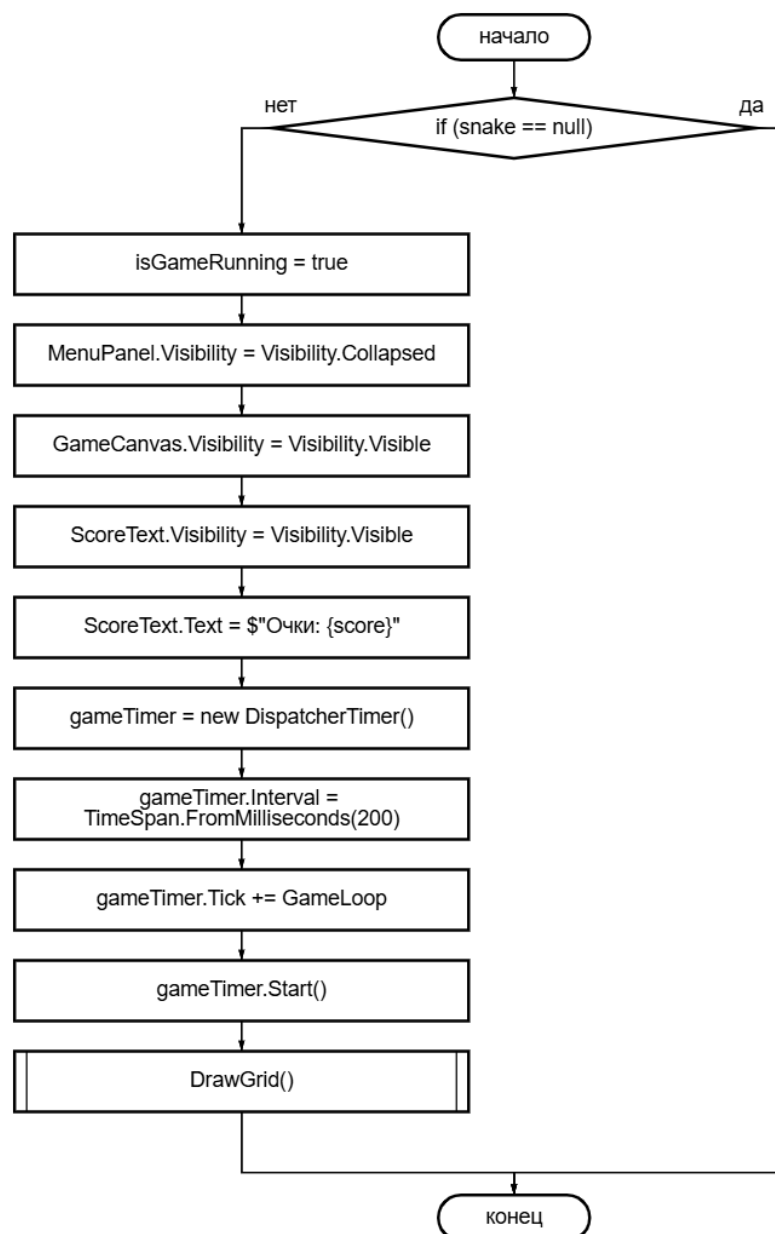


Рисунок 4 - Схема функции StartGame

5.1.5 Функция DrawGrid

Функция DrawGrid отрисовывает игровую сетку на холсте, создавая вертикальные и горизонтальные линии серого цвета с толщиной 0.3 пикселя. Вертикальные линии проводятся через каждый столбец игрового поля, а горизонтальные - через каждую строку, формируя сетку из 20 столбцов и 15 строк. Размер каждой ячейки сетки соответствует 25x25 пикселям, что создает четкое визуальное разделение игрового пространства. Линии добавляются в коллекцию дочерних элементов игрового холста для последующего отображения. Схема представлена на рисунке 5.

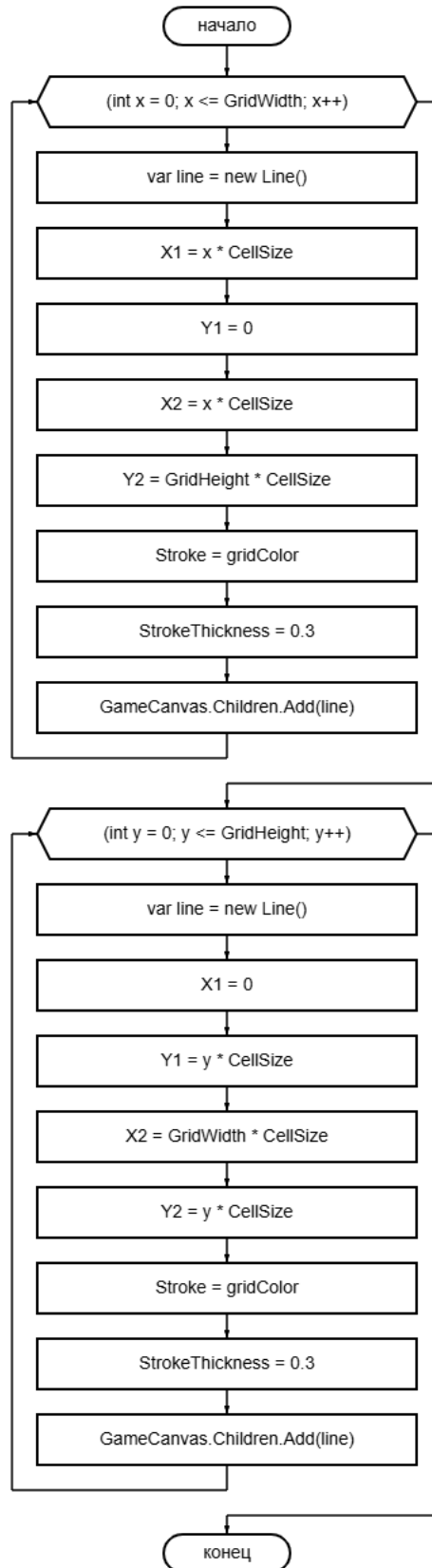


Рисунок 5 - Схема функции DrawGrid

5.1.6 Функция GameLoop

Функция GameLoop является основным игровым циклом, выполняющимся по таймеру. Она обрабатывает движение змейки, проверяет

столкновения и обновляет игровое состояние. Сначала применяет изменение направления движения, если оно было запрошено. Затем вычисляет новую позицию головы змейки и проверяет возможные столкновения с границами поля или телом змейки - при обнаружении завершает игру. Если змейка достигает еды, увеличивает счет на 10 очков, генерирует новую еду и постепенно ускоряет игру. В завершении цикла перерисовывает все игровые элементы на поле. Схема представлена на рисунке 6.

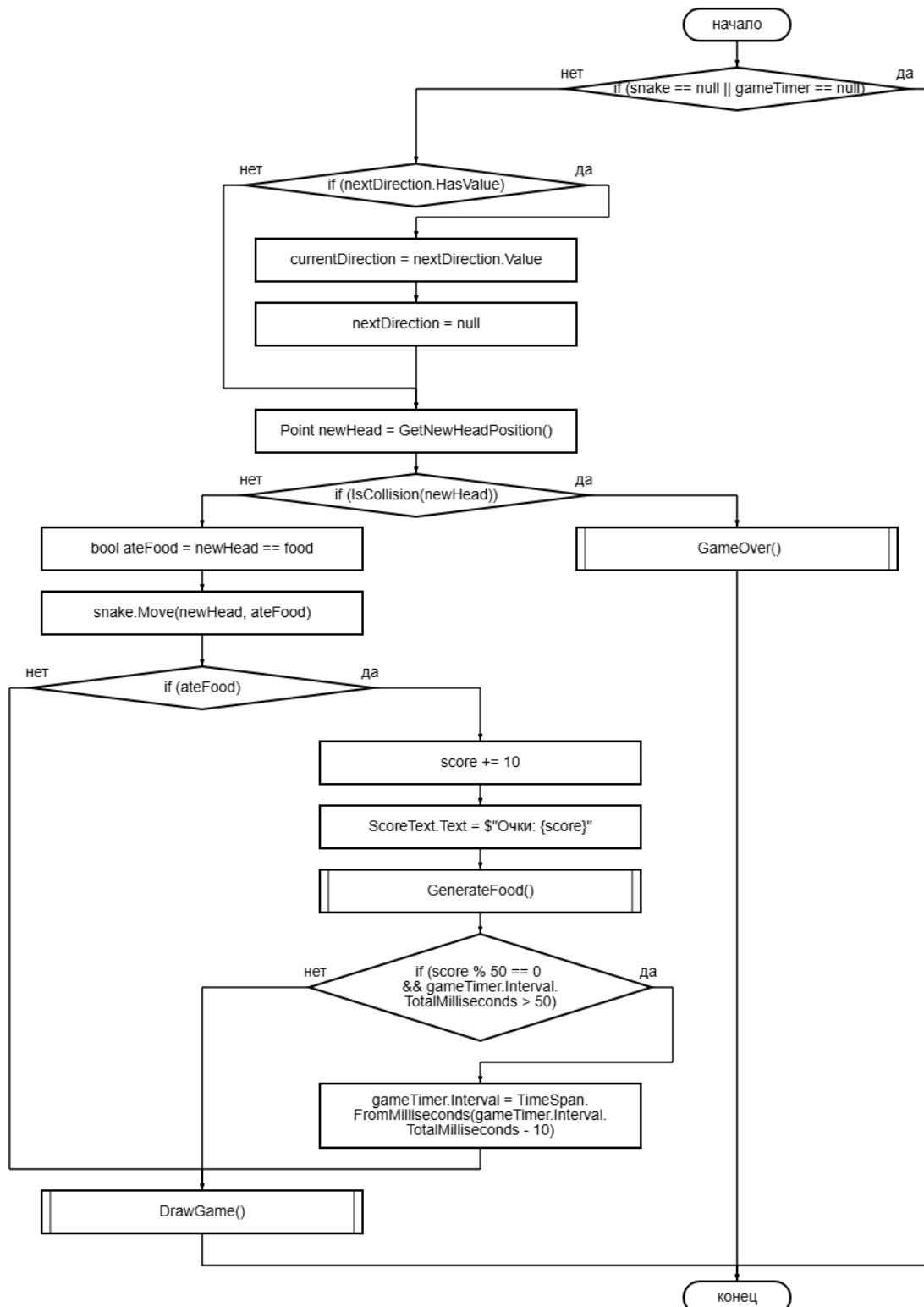


Рисунок 6 - Схема функции GameLoop

5.1.7 Функция IsCollision

Функция IsCollision проверяет возможные столкновения змейки в указанной позиции. Сначала анализирует выход за границы игрового поля (20x15 клеток) — если координаты выходят за допустимые пределы, возвращает true. Затем проверяет каждую часть тела змейки на совпадение с проверяемой позицией — если голова змейки сталкивается с любой частью её тела, также возвращает true. Если столкновений не обнаружено, возвращает false. Эта проверка выполняется перед каждым перемещением змейки для определения конца игры. Схема представлена на рисунке 7.

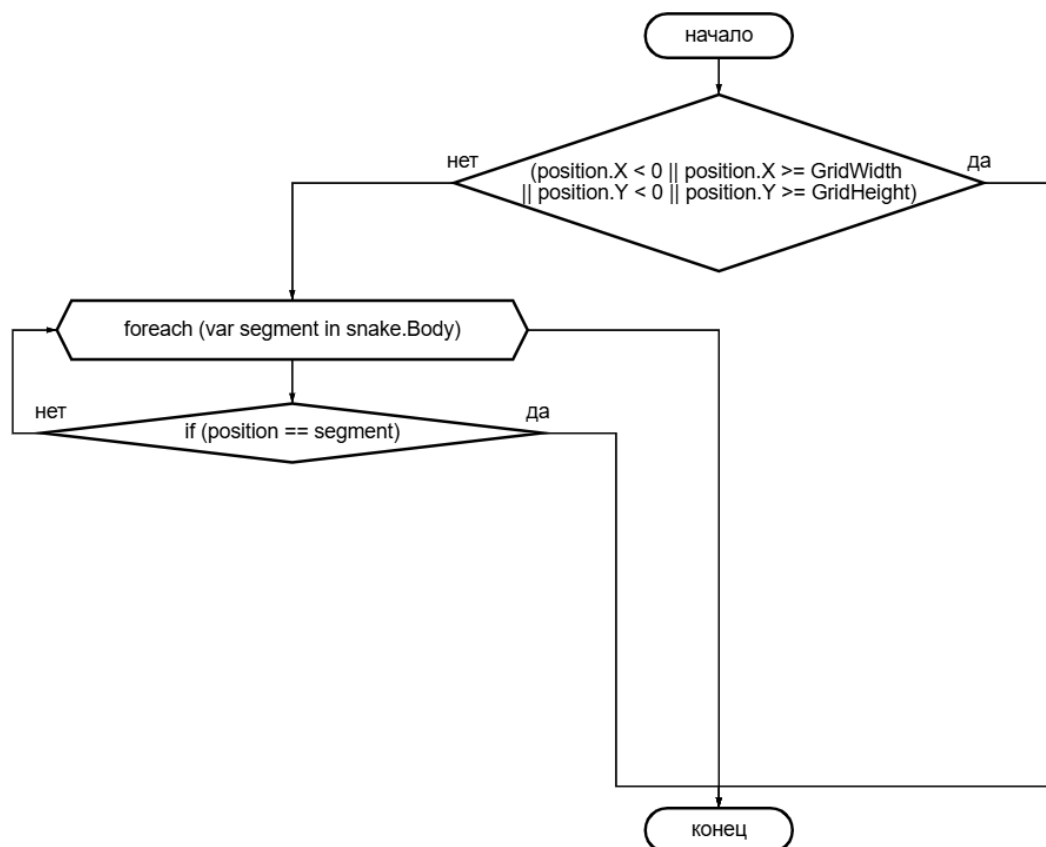


Рисунок 7 - Схема функции IsCollision

5.1.8 Функция GenerateFood

Функция GenerateFood создает еду в случайной свободной позиции на поле, используя внешний генератор. Проверяет, чтобы еда не попала на змейку, повторяя генерацию при необходимости. Схема представлена на рисунке 8.

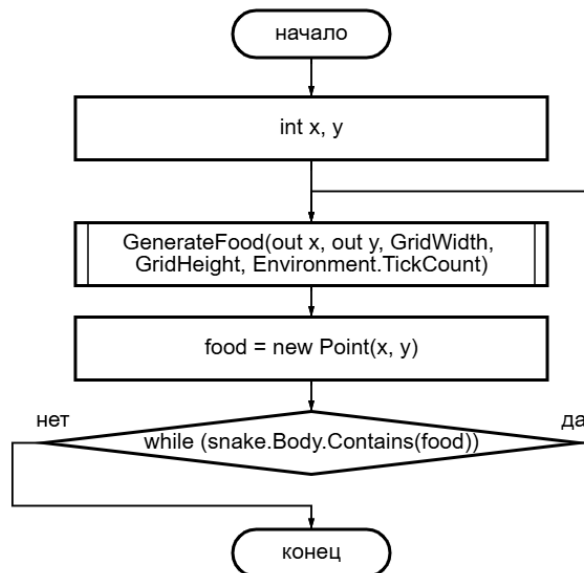


Рисунок 8 - Схема функции GenerateFood

5.1.9 Функция DrawGame

Функция DrawGame обновляет графику на игровом поле. Сначала удаляет предыдущие элементы змейки и еды. Затем рисует новую еду в виде красного круга с темно-красной обводкой в текущей позиции. После этого отрисовывает змейку: голова выделяется ярко-зеленым цветом и увеличенным размером, а тело - темно-зелеными прямоугольниками со скругленными углами. Каждый элемент размещается с учетом размера клетки (25x25 пикселей) и текущих координат. Схема представлена на рисунках 9-а и 9-б.

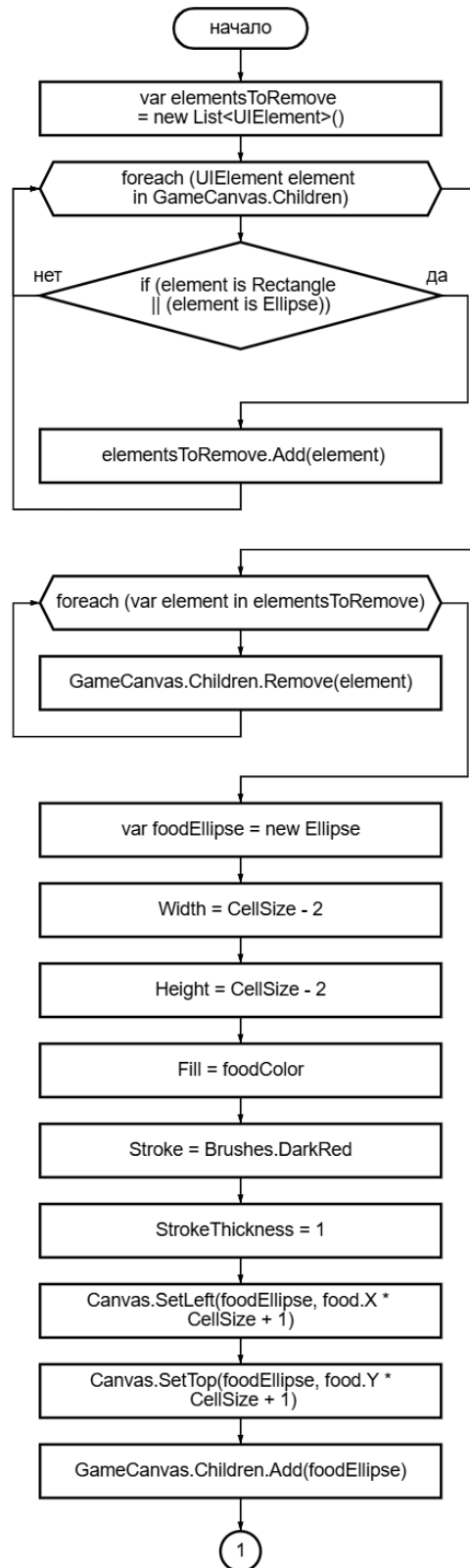


Рисунок 9а - Схема функции DrawGame

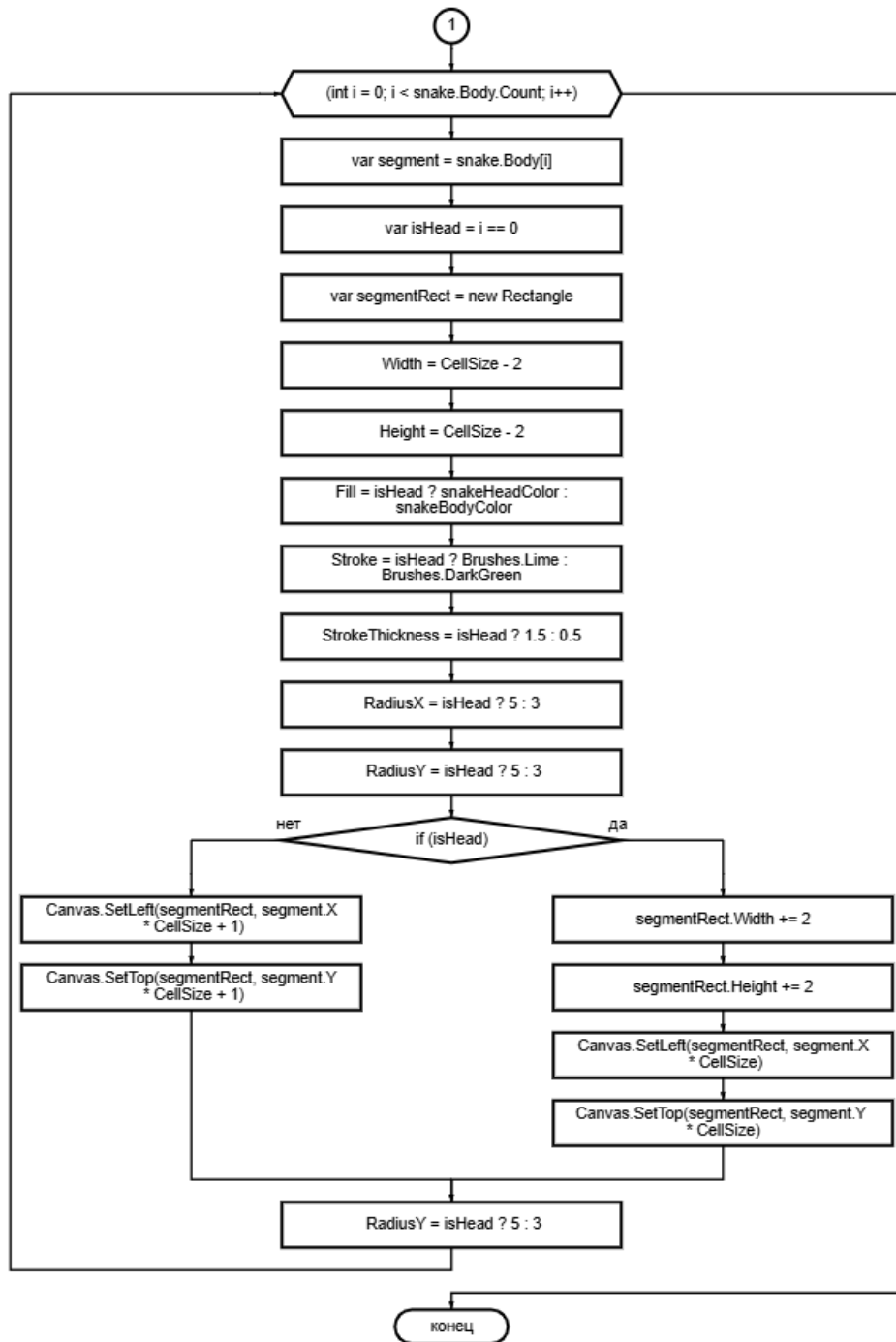


Рисунок 9b – Схема функции DrawGame

5.1.10 Функция GameOver

Функция GameOver обрабатывает завершение игры, а именно останавливает игровой таймер, обновляет статистику и выводит диалоговое окно с результатами. Окно имеет фиксированный размер, отображается по центру главного окна и оформлено в темной цветовой гамме с градиентным фоном. Оно содержит информацию о финальном счете и кнопки для дальнейших действий (повтор игры или выход в меню). Схема представлена на рисунках 10-а и 10-б.

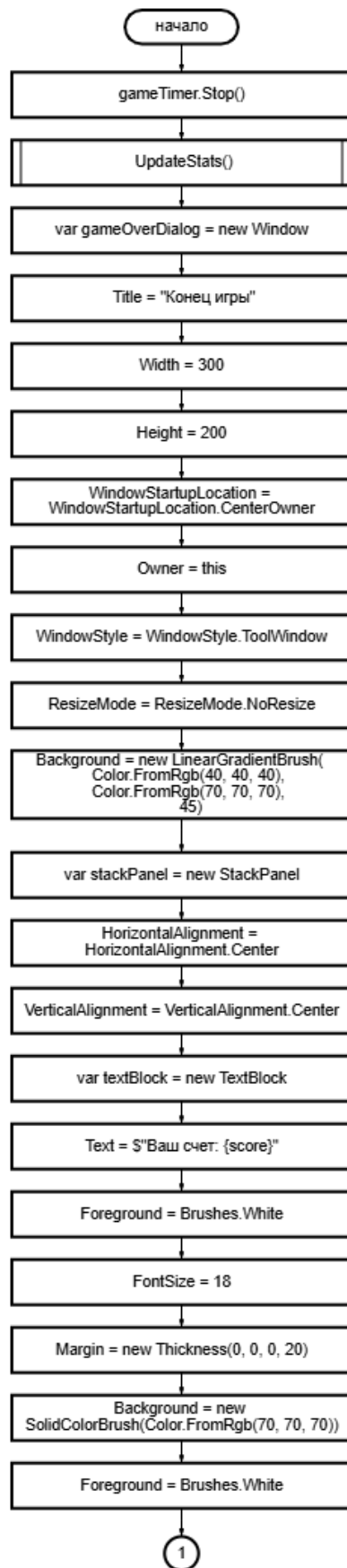


Рисунок 10 - Схема функции GameOver

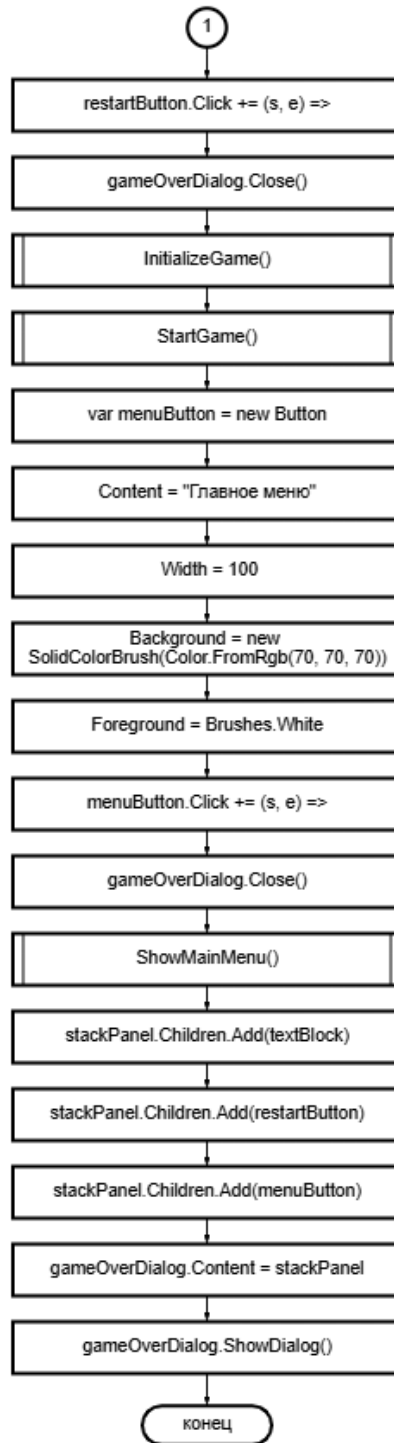


Рисунок 10b – Схема функции GameOver

5.1.11 Функция Window_KeyDown

Функция Window_KeyDown обрабатывает нажатия клавиш во время игры. Она позволяет управлять змейкой с помощью стрелок, запрещая резкие развороты на 180 градусов (когда длина змейки больше 1 сегмента). Клавиша пробела служит для паузы/продолжения игры с соответствующим обновлением

текста счета. Направление движения сохраняется в переменной nextDirection для последующей обработки в игровом цикле. Схема представлена на рисунке 11.

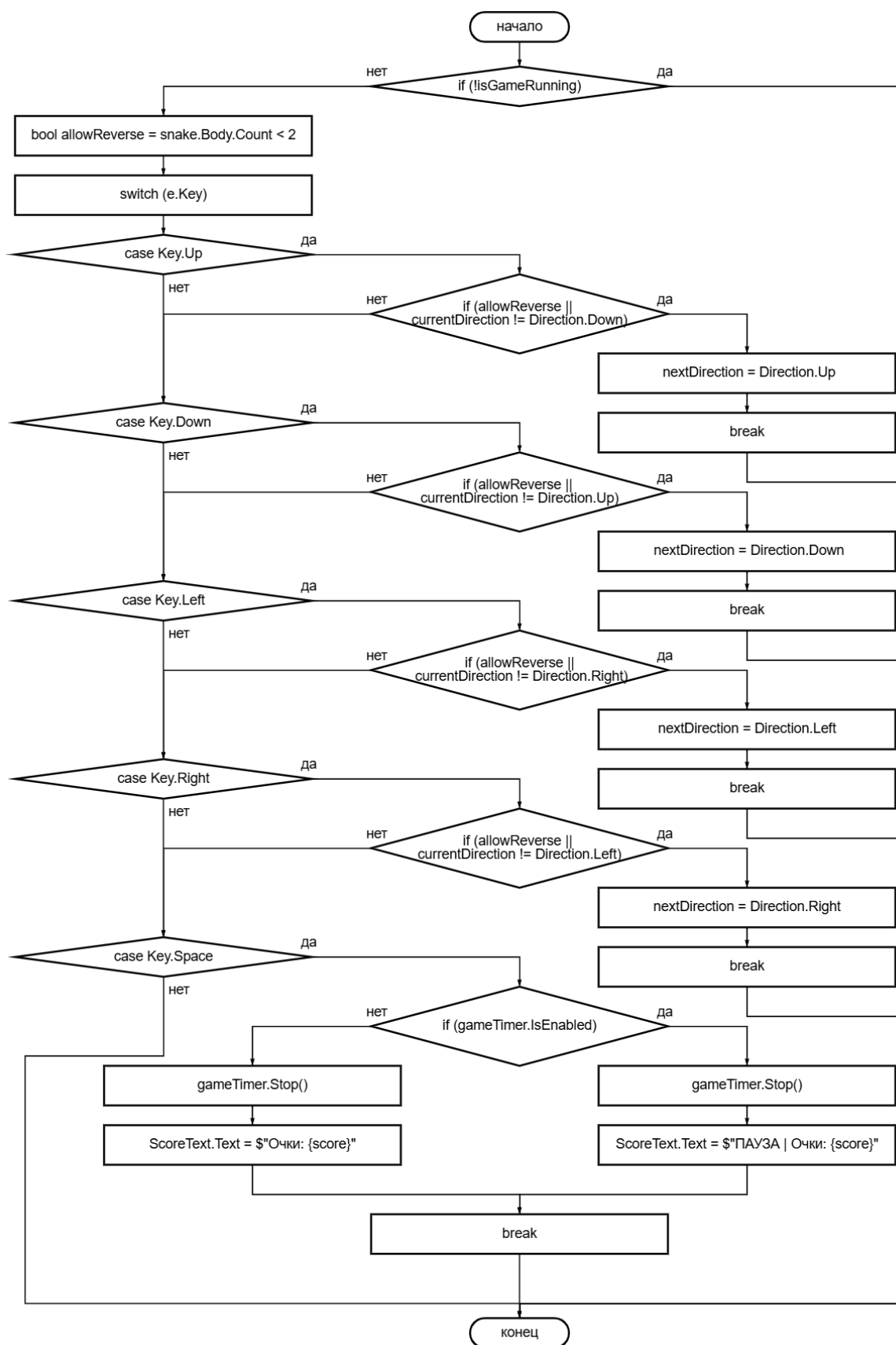


Рисунок 11 – Схема функции Window_KeyDown

5.1.12 Функция Move

Функция Move перемещает змейку, добавляя новую голову в начало списка и при необходимости удаляя хвост (если змейка не съела еду). Параметр grow определяет, нужно ли увеличивать длину змейки. При grow=false последний элемент тела удаляется, создавая эффект движения. Таким образом реализуется плавное перемещение змейки по игровому полю с возможностью роста при поедании еды. Схема представлена на рисунке 12.

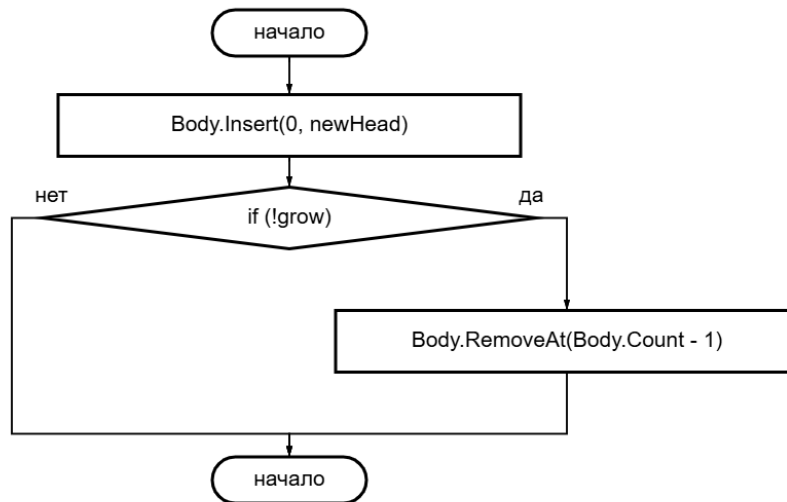


Рисунок 12 – Схема функции Move

6 Руководство пользователя

Для запуска игры дважды кликните по файлу Puzzle.exe (Рисунок 13).

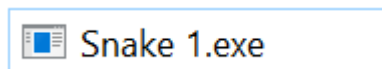


Рисунок 13 - Файл Snake 1

После запуска игры перед вами открывается главное меню, на котором изображена статистика, и находятся две кнопки (Рисунок 14).

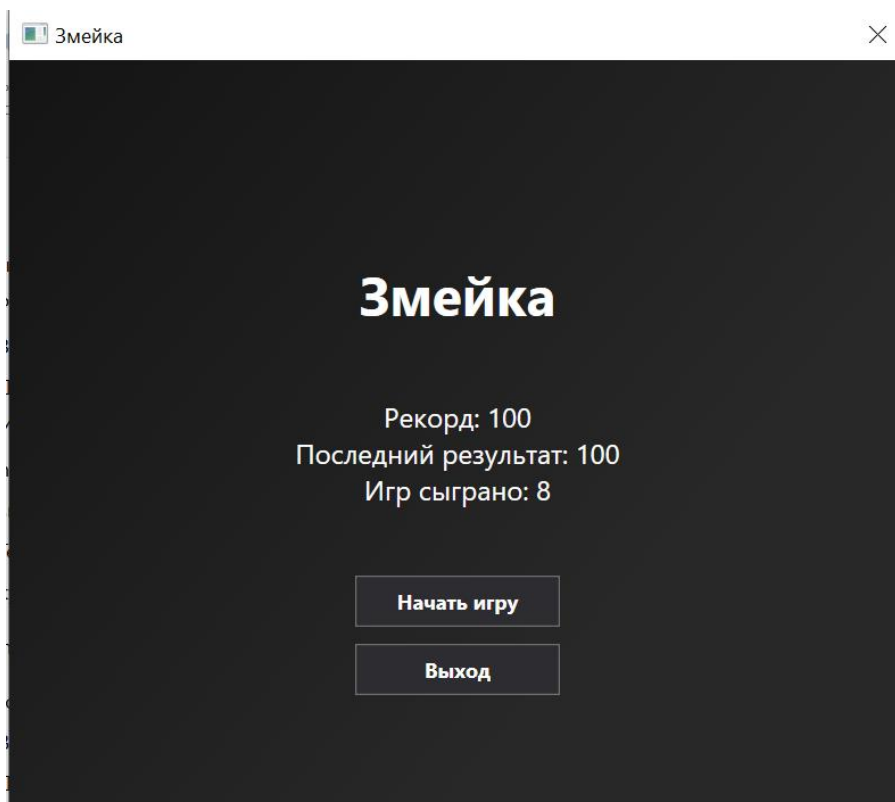


Рисунок 14 - Главное меню

Начать игру – кнопка, запускающая игру (Рисунок 15).

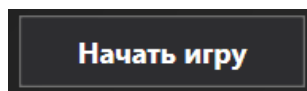


Рисунок 15 - Кнопка "Начать игру"

Выход – кнопка которая завершает работу программы (Рисунок 16).

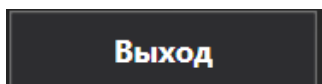


Рисунок 16 - Кнопка "Выход"

На главном меню также отображается статистика игр:

- рекорд.
- Последний результат.

- Общее количество сыгранных игр.

После нажатия на кнопку "Начать игру" открывается игровое поле с движущейся змейкой (Рисунок 17).



Рисунок 17 – Игровое поле

- еда появляется в случайном месте.
- Голова змейки, появившаяся в случайном месте, начинает движение.

Для управления змейкой используются следующие клавиши:

- стрелка вверх - движение вверх.
- Стрелка вниз - движение вниз.
- Стрелка влево - движение влево.
- Стрелка вправо - движение вправо.
- Пробел - постановка игры на паузу/снятие с паузы.
- Escape - выход в главное меню (доступно только во время паузы).

В игре "Змейка" игрок управляет движением змейки клавишами-стрелками, собирает появляющуюся на поле красную еду и старается избегать столкновений с границами поля и собственным телом. Каждая съеденная еда приносит 10 очков и увеличивает длину змейки. Главная цель - набрать максимальное количество очков и установить новый рекорд по длине змейки.

Общее количество очков - отображает количество очков, набранных в реальном времени (Рисунок 18).

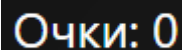
A black rectangular box with the text "Очки: 0" in a yellow, monospace-style font.

Рисунок 18 – Количество очков

Игра завершается автоматически в трех случаях: когда голова змейки сталкивается с границами игрового поля, когда она пересекается с любой частью тела змейки, либо при попытке выполнения недопустимого маневра (в редких исключительных ситуациях). Эти условия обеспечивают четкие правила завершения игрового сеанса.

После завершения игры появляется окно с итоговым счетом (Рисунок 19), и двумя кнопками: "Играть снова" (Рисунок 20) и "Главное меню" (Рисунок 21).

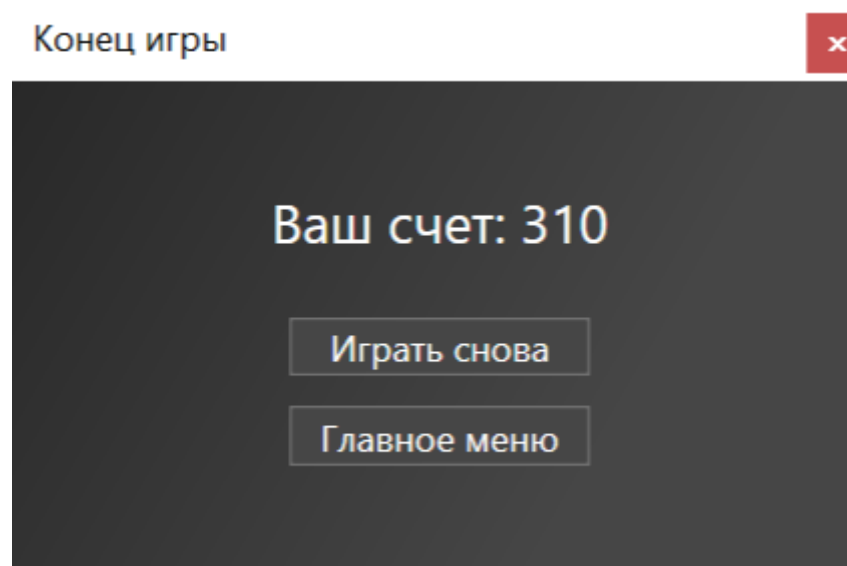


Рисунок 19 – Окно с итоговым счётом

Кнопка "Новая игра" позволяет начать игру заново.

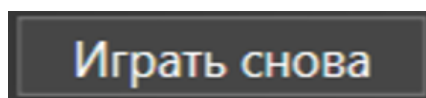


Рисунок 20 - Кнопка "Играть снова"

Кнопка "Меню" позволяет вернуться в главное меню.

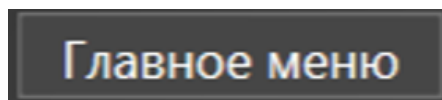


Рисунок 21 - Кнопка "Главное меню"

Статистика игры автоматически обновляется после каждого завершения игровой сессии. Система фиксирует три ключевых параметра: максимальный достигнутый результат (рекорд), количество очков в последней игре и общее число сыгранных партий. Данные сохраняются в файл `snake_stats.txt` сразу при

завершении игры, используя надежный механизм записи с обработкой возможных ошибок ввода-вывода. При следующем запуске приложения статистика автоматически загружается из файла, обеспечивая непрерывность отслеживания игровых достижений. Особое внимание уделено корректному обновлению рекорда - система автоматически сравнивает текущий результат с сохраненным максимумом и при необходимости обновляет его.

Заключение

В процессе разработки игры "Змейка" на платформе WPF были успешно освоены ключевые аспекты работы с графикой и анимацией, что позволило создать плавный и визуально привлекательный игровой процесс. Особое внимание было уделено реализации надежной системы сохранения статистики, которая обеспечивает запись и восстановление результатов игровых сессий через работу с файловой системой.

Управление в игре было тщательно проработано для достижения максимальной отзывчивости и адаптивности к действиям пользователя. Игровая логика подверглась оптимизации, включая эффективные алгоритмы обработки столкновений и генерации игровых объектов, что обеспечивает стабильную работу приложения на различных аппаратных конфигурациях.

Разработанное решение демонстрирует высокую надежность в работе, интуитивно понятное управление и корректное сохранение игрового прогресса между сеансами. Архитектурные особенности проекта заложили прочный фундамент для будущего расширения функциональности, включая потенциальное добавление новых игровых режимов, системы бонусов и препятствий, а также возможностей для сетевого взаимодействия. Данная реализация может служить качественной основой для разработки более сложных игровых проектов с использованием технологий WPF.

Список использованных источников

1. Троелсен, Э. Язык программирования C# 7 и платформы .NET и .NET Core / Э. Троелсен. – Москва: Вильямс, 2018. – 1040 с. – ISBN 978-5-8459-2089-9.
2. MSDN C#: [сайт]. – URL: <https://www.tpointtech.com/msdn-c-sharp> (дата обращения: 13.05.2024).
3. C# Basics: [сайт]. – URL: <https://code-basics.com/ru/languages/csharp> (дата обращения: 13.05.2024).
4. Metanit: Полное руководство по языку программирования C# 13 и платформе .NET 9: [сайт]. – URL: <https://metanit.com> (дата обращения: 14.05.2024).

Приложение А Листинги

Файл MainWindow.xaml.cs:

```
using System;

using System.Collections.Generic;

using System.IO;

using System.Runtime.InteropServices;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Shapes;

using System.Windows.Threading;

namespace SnakeGameWPF

{

    public partial class MainWindow : Window

    {

        [DllImport("FoodCore.dll", CallingConvention =

CallingConvention.StdCall)]

        public static extern void GenerateFood(out int x, out int

y, int gridW, int gridH, int seed);

        private const int CellSize = 25;

        private const int GridWidth = 20;

        private const int GridHeight = 15;

        private const string StatsFileName = "snake_stats.txt";

        private DispatcherTimer? gameTimer;

        private Snake? snake;

        private Point food;

        private Direction currentDirection;

        private Direction? nextDirection;

        private Random random = new Random();
```

```

private int score;

private bool isGameRunning;


private int maxScore = 0;

private int lastScore = 0;

private int gamesPlayed = 0;


private TextBlock recordText = new TextBlock();

private TextBlock lastScoreText = new TextBlock();

private TextBlock gamesPlayedText = new TextBlock();


private readonly SolidColorBrush darkBackground = new
SolidColorBrush(Color.FromRgb(30, 30, 30));

private readonly SolidColorBrush gridColor = new
SolidColorBrush(Color.FromRgb(50, 50, 50));

private readonly SolidColorBrush foodColor = new
SolidColorBrush(Color.FromRgb(255, 80, 80));

private readonly SolidColorBrush snakeBodyColor = new
SolidColorBrush(Color.FromRgb(100, 200, 100));

private readonly SolidColorBrush snakeHeadColor = new
SolidColorBrush(Color.FromRgb(50, 220, 50));

public MainWindow()
{
    InitializeComponent();

    random = new Random();

    LoadStats();


    GameCanvas.Background = darkBackground;

    this.Background = new LinearGradientBrush(
        Color.FromRgb(20, 20, 20),
        Color.FromRgb(40, 40, 40),

```

```

        45);

        this.Width = GridWidth * CellSize + 40;
        this.Height = GridHeight * CellSize + 100;
        this.ResizeMode = ResizeMode.NoResize;
        this.Title = "Змейка";
        GameCanvas.Width = GridWidth * CellSize;
        GameCanvas.Height = GridHeight * CellSize;

        InitializeMenuStats();
        InitializeGame();
    }

    private void InitializeMenuStats()
    {

        var statsPanel = new StackPanel
        {
            Margin = new Thickness(0, 20, 0, 20),
            HorizontalAlignment = HorizontalAlignment.Center
        };

        recordText = new TextBlock
        {
            FontSize = 16,
            Foreground = Brushes.White,
            HorizontalAlignment = HorizontalAlignment.Center
        };

        lastScoreText = new TextBlock
        {
            FontSize = 16,

```

```

        Foreground = Brushes.White,
        HorizontalAlignment = HorizontalAlignment.Center
    };
    gamesPlayedText = new TextBlock
    {
        FontSize = 16,
        Foreground = Brushes.White,
        HorizontalAlignment = HorizontalAlignment.Center,
        Margin = new Thickness(0, 0, 0, 20)
    };
    statsPanel.Children.Add(recordText);
    statsPanel.Children.Add(lastScoreText);
    statsPanel.Children.Add(gamesPlayedText);

    MenuPanel.Children.Insert(1, statsPanel);
}

private void UpdateMenuStats()
{
    if (recordText != null)
        recordText.Text = $"Рекорд: {maxScore}";
    if (lastScoreText != null)
        lastScoreText.Text = $"Последний результат:
{lastScore}";
    if (gamesPlayedText != null)
        gamesPlayedText.Text = $"Игр сыграно:
{gamesPlayed}";
}

private void LoadStats()
{
    try

```

```

        {
            if (File.Exists(StatsFileName))
            {
                string[] stats =
File.ReadAllLines(StatsFileName);

                if (stats.Length >= 3)
                {
                    maxScore = int.Parse(stats[0]);

                    lastScore = int.Parse(stats[1]);

                    gamesPlayed = int.Parse(stats[2]);
                }
            }
        }
    catch
    {
        maxScore = 0;

        lastScore = 0;

        gamesPlayed = 0;
    }
}

private void SaveStats()
{
    try
    {
        File.WriteAllText(StatsFileName,
$"{maxScore}\n{lastScore}\n{gamesPlayed}");
    }
    catch
    {

```

```

        }

    }

    private void UpdateStats()
    {
        lastScore = score;
        gamesPlayed++;
        if (score > maxScore)
        {
            maxScore = score;
        }

        SaveStats();

        UpdateMenuStats();
    }

    private void InitializeGame()
    {
        GameCanvas.Children.Clear();

        snake = new Snake(new Point(GridWidth / 2, GridHeight
/ 2));

        currentDirection = Direction.Right;
        nextDirection = null;
        score = 0;

        ScoreText.Text = $"Очки: {score}";
        ScoreText.Foreground = Brushes.White;

        GenerateFood();

        ShowMainMenu();
    }

    private void ShowMainMenu()
    {
        isGameRunning = false;

        MenuPanel.Visibility = Visibility.Visible;
    }

```

```

        GameCanvas.Visibility = Visibility.Collapsed;
        ScoreText.Visibility = Visibility.Collapsed;
        UpdateMenuStats();
    }

    private void StartGame()
    {
        if (snake == null) return;
        isGameRunning = true;
        MenuPanel.Visibility = Visibility.Collapsed;
        GameCanvas.Visibility = Visibility.Visible;
        ScoreText.Visibility = Visibility.Visible;
        ScoreText.Text = $"Очки: {score}";
        gameTimer = new DispatcherTimer();
        gameTimer.Interval = TimeSpan.FromMilliseconds(200);
        gameTimer.Tick += GameLoop;
        gameTimer.Start();
        DrawGrid();
    }

    private void DrawGrid()
    {
        for (int x = 0; x <= GridWidth; x++)
        {
            var line = new Line()
            {
                X1 = x * CellSize,
                Y1 = 0,
                X2 = x * CellSize,
                Y2 = GridHeight * CellSize,
            }
        }
    }

```

```

        Stroke = gridColor,
        StrokeThickness = 0.3
    };

    GameCanvas.Children.Add(line);
}

for (int y = 0; y <= GridHeight; y++)
{
    var line = new Line()
    {
        X1 = 0,
        Y1 = y * CellSize,
        X2 = GridWidth * CellSize,
        Y2 = y * CellSize,
        Stroke = gridColor,
        StrokeThickness = 0.3
    };

    GameCanvas.Children.Add(line);
}

}

private void GameLoop(object? sender, EventArgs e)
{
    if (snake == null || gameTimer == null) return;

    if (nextDirection.HasValue)
    {
        currentDirection = nextDirection.Value;
        nextDirection = null;
    }
}

```

```

        Point newHead = GetNewHeadPosition();

        if (IsCollision(newHead))
        {
            GameOver();
            return;
        }

        bool ateFood = newHead == food;

        snake.Move(newHead, ateFood);
        if (ateFood)
        {
            score += 10;

            ScoreText.Text = $"Очки: {score}";

            GenerateFood();

            if (score % 50 == 0 &&
gameTimer.Interval.TotalMilliseconds > 50)
            {
                gameTimer.Interval =
TimeSpan.FromMilliseconds(gameTimer.Interval.TotalMilliseconds -
10);
            }
        }

        DrawGame();
    }

    private Point GetNewHeadPosition()
    {
        Point head = snake.Head;

        return currentDirection switch

```

```

        {
            Direction.Up => new Point(head.X, head.Y - 1),
            Direction.Down => new Point(head.X, head.Y + 1),
            Direction.Left => new Point(head.X - 1, head.Y),
            Direction.Right => new Point(head.X + 1, head.Y),
            _ => head
        };
    }

    private bool IsCollision(Point position)
    {
        if (position.X < 0 || position.X >= GridWidth ||
            position.Y < 0 || position.Y >= GridHeight)
            return true;

        foreach (var segment in snake.Body)
        {
            if (position == segment)
                return true;
        }

        return false;
    }

    private void GenerateFood()
    {
        int x, y;
        do
        {
            GenerateFood(out x, out y, GridWidth, GridHeight,
                Environment.TickCount);

            food = new Point(x, y);
        }
    }

```

```

    }

    while (snake.Body.Contains(food));
}

private void DrawGame()
{

    var elementsToRemove = new List<UIElement>();
    foreach (UIElement element in GameCanvas.Children)
    {
        if (element is Rectangle || (element is Ellipse))
        {
            elementsToRemove.Add(element);
        }
    }
    foreach (var element in elementsToRemove)
    {
        GameCanvas.Children.Remove(element);
    }

    var foodEllipse = new Ellipse
    {
        Width = CellSize - 2,
        Height = CellSize - 2,
        Fill = foodColor,
        Stroke = Brushes.DarkRed,
        StrokeThickness = 1
    };

    Canvas.SetLeft(foodEllipse, food.X * CellSize + 1);
    Canvas.SetTop(foodEllipse, food.Y * CellSize + 1);

```

```

GameCanvas.Children.Add(foodEllipse);

for (int i = 0; i < snake.Body.Count; i++)
{
    var segment = snake.Body[i];
    var isHead = i == 0;
    var segmentRect = new Rectangle
    {
        Width = CellSize - 2,
        Height = CellSize - 2,
        Fill = isHead ? snakeHeadColor :
snakeBodyColor,
        Stroke = isHead ? Brushes.Lime :
Brushes.DarkGreen,
        StrokeThickness = isHead ? 1.5 : 0.5,
        RadiusX = isHead ? 5 : 3,
        RadiusY = isHead ? 5 : 3
    };
    if (isHead)
    {
        segmentRect.Width += 2;
        segmentRect.Height += 2;
        Canvas.SetLeft(segmentRect, segment.X *
CellSize);
        Canvas.SetTop(segmentRect, segment.Y *
CellSize);
    }
    else
    {
        Canvas.SetLeft(segmentRect, segment.X *
CellSize + 1);
    }
}

```

```

        Canvas.SetTop(segmentRect, segment.Y *
CellSize + 1);

    }

    GameCanvas.Children.Add(segmentRect);

}

}

private void GameOver()
{
    gameTimer.Stop();

    UpdateStats();

    var gameOverDialog = new Window
    {
        Title = "Конец игры",
        Width = 300,
        Height = 200,
        WindowStartupLocation =
WindowStartupLocation.CenterOwner,

        Owner = this,
        WindowStyle = WindowStyle.ToolWindow,
        ResizeMode = ResizeMode.NoResize,
        Background = new LinearGradientBrush(
            Color.FromRgb(40, 40, 40),
            Color.FromRgb(70, 70, 70),
            45)

    };

    var stackPanel = new StackPanel
    {
        HorizontalAlignment = HorizontalAlignment.Center,
        VerticalAlignment = VerticalAlignment.Center

    };

```

```

var textBlock = new TextBlock
{
    Text = $"Ваш счет: {score}",
    Foreground = Brushes.White,
    FontSize = 18,
    Margin = new Thickness(0, 0, 0, 20),
    HorizontalAlignment = HorizontalAlignment.Center
};

var restartButton = new Button
{
    Content = "Играть снова",
    Width = 100,
    Margin = new Thickness(0, 0, 0, 10),
    Background = new SolidColorBrush(Color.FromRgb(70,
70, 70)),
    Foreground = Brushes.White
};

restartButton.Click += (s, e) =>
{
    gameOverDialog.Close();
    InitializeGame();
    StartGame();
};

var menuButton = new Button
{
    Content = "Главное меню",
    Width = 100,
    Background = new SolidColorBrush(Color.FromRgb(70,
70, 70)),
    Foreground = Brushes.White
}

```

```

};

menuButton.Click += (s, e) =>
{
    gameOverDialog.Close();

    ShowMainMenu();
};

stackPanel.Children.Add(textBlock);
stackPanel.Children.Add(restartButton);
stackPanel.Children.Add(menuButton);
gameOverDialog.Content = stackPanel;
gameOverDialog.ShowDialog();
}

private void Window_KeyDown(object sender, KeyEventArgs e)
{
    if (!isGameRunning) return;

    bool allowReverse = snake.Body.Count < 2;
    switch (e.Key)
    {
        case Key.Up:
            if (allowReverse || currentDirection !=
Direction.Down)
                nextDirection = Direction.Up;
            break;
        case Key.Down:
            if (allowReverse || currentDirection !=
Direction.Up)
                nextDirection = Direction.Down;
            break;
        case Key.Left:

```

```

        if (allowReverse || currentDirection !=
Direction.Right)

            nextDirection = Direction.Left;

        break;

    case Key.Right:

        if (allowReverse || currentDirection !=
Direction.Left)

            nextDirection = Direction.Right;

        break;

    case Key.Space:

        if (gameTimer.IsEnabled)

        {

            gameTimer.Stop();

            ScoreText.Text = $"ПАУЗА | Очки: {score}";

        }

        else

        {

            gameTimer.Start();

            ScoreText.Text = $"Очки: {score}";

        }

        break;

    }

}

private void StartButton_Click(object sender,
RoutedEventArgs e)

{

    InitializeGame();

    StartGame();

}

private void ExitButton_Click(object sender,
RoutedEventArgs e)

```

```

        {
            this.Close();
        }
    }

    public enum Direction { Up, Down, Left, Right }

    public class Snake
    {
        public List<Point> Body { get; }
        public Point Head => Body[0];
        public Snake(Point startPosition)
        {
            Body = new List<Point> { startPosition };
        }

        public void Move(Point newHead, bool grow)
        {
            Body.Insert(0, newHead);

            if (!grow)
            {
                Body.RemoveAt(Body.Count - 1);
            }
        }
    }
}

```

Файл MainWindow.xaml:

```

<Window x:Class="SnakeGameWPF.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

```

```

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"

    xmlns:local="clr-namespace:SnakeGameWPF"

    mc:Ignorable="d"

    Title="Snake Game" Height="450" Width="600"
KeyDown="Window_KeyDown">

    <Grid>

        <Canvas x:Name="GameCanvas" HorizontalAlignment="Center"
VerticalAlignment="Top" Margin="0,50,0,0"/>

        <StackPanel x:Name="MenuPanel"
HorizontalAlignment="Center" VerticalAlignment="Center"
Margin="0,50,0,0">

            <!-- Простая белая надпись "Змейка" -->

            <TextBlock Text="Змейка"

                FontSize="32"

                FontWeight="Bold"

                Foreground="White"

                HorizontalAlignment="Center"

                Margin="0,0,0,20"/>

            <Button x:Name="StartButton" Content="Начать игру"

                Width="120" Height="30"
Click="StartButton_Click"

                Background="#FF2D2D30" Foreground="White"

                FontWeight="Bold" Margin="0,0,0,10"/>

            <Button x:Name="ExitButton" Content="Выход"

                Width="120" Height="30"
Click="ExitButton_Click"

                Background="#FF2D2D30" Foreground="White"

                FontWeight="Bold"/>

        </StackPanel>

```

```

        <TextBlock x:Name="ScoreText" HorizontalAlignment="Left"
VerticalAlignment="Top"

                Margin="10,10,0,0" Foreground="White"
FontSize="16"/>

    </Grid>

</Window>

```

Файл FoodCore.asm:

```

.686
.model flat, stdcall
public GenerateFood
.code
; void __stdcall GenerateFood(int* x, int* y, int gridW, int
gridH, int seed)
GenerateFood proc
    push ebp
    mov ebp, esp
    push ebx
    push esi
    push edi

    mov esi, [ebp+8]    ; x ptr
    mov edi, [ebp+12]   ; y ptr
    mov ebx, [ebp+16]   ; gridW
    mov ecx, [ebp+20]   ; gridH
    mov eax, [ebp+24]   ; seed

    test eax, eax
    jnz seed_ok
    rdtsc
seed_ok:
    imul eax, 1664525
    add eax, 1013904223

    ; Генерация X (0..gridW-1)
    xor edx, edx
    div ebx
    mov [esi], edx

    ; Генерация Y (0..gridH-1)
    imul eax, 1664525
    add eax, 1013904223
    xor edx, edx
    div ecx
    mov [edi], edx

    pop edi
    pop esi
    pop ebx

```

```
    pop ebp
    ret 20
GenerateFood endp
end
```

Приложение В Пример Файла статистики

Файл Snake_stats.txt:

