Министерство науки и высшего образования РФ
Пензенский государственный университет
Кафедра «Вычислительная техника»

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проектированию
по курсу «Логика и основы алгоритмизации в инженерных задачах»
на тему «Разработка игрового агента для иргры “Лабиринт”»

Выполнил:
Студент группы 24ВВВ3
Любченко В.К.

Принял:
Юрова

Пенза 2025

# Содержание

# Реферат

Отчет 88 страниц, 4 рисунка, 2 таблица, 4 источника.

ЛАБИРИНТ, БОТ, ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ЛИ

Цель исследования – разработка интерактивной программы, реализующей соревновательный режим между пользователем и искусственным интеллектом в условиях автоматически генерируемого лабиринта.

В работе рассмотрен алгоритм рекурсивного backtracking для генерации идеальных лабиринтов, а также волновой алгоритм (алгоритм Ли) для нахождения кратчайшего пути искусственным интеллектом. Установлено, что данные алгоритмы обеспечивают высокую эффективность при решении задач генерации проходимых лабиринтов и оптимизации маршрутов в них. Реализована система сохранения и загрузки игрового состояния с использованием JSON-сериализации, а также интуитивно понятный графический интерфейс на основе Windows Forms.

# Введение

**Лабиринт** — это сложная структура, состоящая из запутанных путей и тупиков, которая исторически использовалась в мифологии, архитектуре и развлечениях. В современном мире лабиринты нашли широкое применение в компьютерных играх, искусственном интеллекте, психологических исследованиях и алгоритмических задачах. Особый интерес представляют интерактивные лабиринты, которые сочетают элементы геймдизайна, алгоритмического мышления и человеко-машинного взаимодействия.

Лабиринты можно классифицировать по различным критериям:

**Плоские (двумерные) лабиринты**: классические структуры с одним уровнем сложности;

**Многоуровневые лабиринты**: включают несколько этажей или планов с переходами между ними;

**Динамические лабиринты**: стены и проходы могут изменяться во время прохождения;

**Соревновательные лабиринты**: предполагают одновременное участие нескольких участников (людей или алгоритмов).

Для решения задач навигации в лабиринтах разработаны различные алгоритмы:

**"Правило одной руки"**: простейший метод, гарантирующий выход в связных лабиринтах;

**Поиск в глубину (DFS)**: исследует ветви до тупиков, затем возвращается;

**Поиск в ширину (BFS)**: равномерно исследует все направления на каждом шаге;

**Волновой алгоритм (алгоритм Ли)**: находит кратчайший путь, используя распространение "волны";

**A (A-star)**: эвристический алгоритм, оптимальный для поиска пути в сложных структурах;

**Генетические алгоритмы**: используют принципы эволюции для нахождения оптимальных маршрутов.

В качестве среды разработки была выбрана **Microsoft Visual Studio Community 2022**, язык программирования — **C#**, платформа — **.NET 8.0**, графическая библиотека — **Windows Forms**. Данный выбор обусловлен мощными возможностями среды разработки, широкими графическими возможностями Windows Forms и современной версией платформы .NET, обеспечивающей высокую производительность и кросс-платформенную совместимость.

# 1 Постановка задачи

Разрабатываемая программа представляет собой интерактивную игру-соревнование, в которой пользователь и искусственный интеллект одновременно проходят идентичные лабиринты. Программа должна обеспечивать генерацию случайных лабиринтов, предоставлять средства для ручного управления персонажем игрока, автоматического перемещения бота, отслеживания времени прохождения и определения победителя.

## 1.1 Генерация лабиринта

Лабиринт должен представлять собой двумерную сетку ячеек размером N×N, где N выбирается пользователем (5, 10 или 15 клеток).

Каждая ячейка характеризуется наличием или отсутствием стен на четырех границах (верхней, нижней, левой, правой).

Для генерации лабиринта должен использоваться алгоритм рекурсивного backtracking (вариант алгоритма растущего дерева). Алгоритм начинает работу с начальной ячейки (0,0) и рекурсивно "прорывает" стены в случайном направлении к непосещенной соседней ячейке. Если у текущей ячейки отсутствуют непосещенные соседи, происходит возврат (backtrack) к предыдущей ячейке до тех пор, пока не будут посещены все ячейки сетки.

Лабиринт должен иметь четко определенные стартовую точку (левый верхний угол) и финишную точку (правый нижний угол).

### 1.2 Прохождение лабиринта

Программа должна обеспечивать два параллельных режима прохождения:

**Ручное управление**: пользователь управляет персонажем с помощью клавиш WASD. Движение должно осуществляться с проверкой столкновений со стенами лабиринта.

**Автоматическое прохождение**: бот использует волновой алгоритм (алгоритм Ли) для нахождения кратчайшего пути от старта до финиша. Бот должен перемещаться по найденному маршруту. Программа должна отслеживать время прохождения для каждого участника с точностью до миллисекунд.

### 1.3 Соревновательный режим

Игра должна начинаться одновременно для игрока и бота при запуске уровня.

По достижении финиша каждым участником фиксируется его время.

После завершения игры обоими участниками программа должна:

Определить победителя (участника с меньшим временем прохождения)

Отобразить детальные результаты с временем каждого участника

Предоставить возможность начать новую игру или вернуться в меню

### 1.4 Сохранение и загрузка

Программа должна предоставлять возможность сохранения текущего состояния игры в файл, включая:

Структуру лабиринта (расположение стен)

Позиции игрока и бота

Текущее время игры

Статус завершения для каждого участника

Должна быть реализована возможность загрузки сохраненной игры с полным восстановлением состояния.

### 1.5 Пользовательский интерфейс

Интерфейс должен включать:

Главное меню с выбором сложности (размера лабиринта)

Игровое поле с одновременным отображением двух лабиринтов (для игрока и бота)

Таймер, отображающий текущее время игры

Кнопки управления (выход, сохранение, загрузка)

Форму отображения результатов с определением победителя

Управление должно осуществляться как с помощью мыши (кнопки, меню), так и клавиатуры (управление персонажем).

### 1.6 Технические требования

Программа должна быть реализована на языке C# с использованием платформы .NET 8.0.

Графический интерфейс должен быть разработан с использованием Windows Forms.

Код должен быть структурирован с применением объектно-ориентированного подхода, обеспечивающего модульность и расширяемость.

Алгоритмы должны быть эффективны для лабиринтов размером до 15×15 ячеек.

### 1.7 Критерии оценки работы программы

Корректность генерации лабиринтов (отсутствие изолированных областей, существование пути от старта к финишу).

Точность работы алгоритма поиска пути бота (нахождение кратчайшего маршрута).

Корректность обработки столкновений игрока со стенами.

Точность измерения и сравнения времени прохождения.

Надежность сохранения и загрузки игрового состояния.

Удобство и интуитивность пользовательского интерфейса.

# 2. Теоретическая часть задания

Алгоритм рекурсивного backtracking (возврата)

Алгоритм рекурсивного возврата является одним из наиболее эффективных методов генерации идеальных лабиринтов. Идеальный лабиринт определяется как лабиринт, в котором существует ровно один уникальный путь между любыми двумя точками, что исключает наличие изолированных областей и циклов.

Принцип работы алгоритма:

Инициализация: Все ячейки лабиринта помечаются как непосещенные. Выбирается начальная ячейка (обычно левый верхний угол), которая помечается как посещенная и добавляется в стек текущего пути.

**Рекурсивное исследование:**

Из текущей ячейки проверяются все непосещенные соседи (вверх, вниз, влево, вправо).

Если существуют непосещенные соседи, случайным образом выбирается один из них.

Удаляется стена между текущей ячейкой и выбранным соседом.

Выбранный сосед помечается как посещенный, становится текущей ячейкой и добавляется в стек.

Процесс повторяется для новой текущей ячейки.

**Возврат (backtracking):**

Если у текущей ячейки нет непосещенных соседей, выполняется возврат:

Из стека извлекается предыдущая ячейка.

Она становится текущей ячейкой.

Процесс продолжается с поиска непосещенных соседей для новой текущей ячейки.

**Завершение:**

Алгоритм завершается, когда стек становится пустым, что означает, что все ячейки лабиринта были посещены.

Преимущества алгоритма:

Гарантированно создает идеальный лабиринт.

Относительно простая реализация.

Создает лабиринты с длинными извилистыми коридорами, что увеличивает сложность прохождения.

Время выполнения пропорционально количеству ячеек (O(n)).

Особенности реализации в программе: В разработанной программе используется модифицированная версия алгоритма, сочетающая методы "Kill" и "Hunt". Метод "Kill" отвечает за рекурсивное исследование, а метод "Hunt" обеспечивает поиск непосещенных ячеек при достижении тупика, что гарантирует полный обход всех ячеек сетки.

# 3. Описание основных классов программы

## 3.1. Классы для представления лабиринта

**Класс** Cell **(Пространство имен: AI_maze)**

**Назначение:** Представляет отдельную ячейку лабиринта как базовую структурную единицу.

**Основные свойства:**

int X, int Y - координаты ячейки в сетке лабиринта

bool isVisit - флаг посещения ячейки (используется при генерации лабиринта)

bool topWall, bool downWall, bool leftWall, bool rightWall - наличие стен на каждой из четырех границ

Конструктор: Cell(int x, int y) - инициализирует ячейку с заданными координатами и всеми стенами

Особенности: Класс инкапсулирует состояние одной клетки лабиринта, определяя ее проходимость в разных направлениях.

**Класс Maze (Пространство имен: WinFormsApp1.Class)**

**Назначение**: Представляет весь лабиринт как совокупность ячеек и реализует алгоритмы генерации и отрисовки.

**Основные свойства:**

int Height, int Width - размеры лабиринта

Cell[,] Cells - двумерный массив ячеек

Random rnd - генератор случайных чисел

Pen wallPen - инструмент для рисования стен

Основные методы:

Конструктор Maze(int Height, int Width) - создает лабиринт заданного размера

**Генерация:**

Kill(int startX, int startY) - основной метод генерации по алгоритму "Kill and Hunt"

Hunt() - вспомогательный метод для поиска непосещенных ячеек

Вспомогательные методы:

GetUnVisitNeighbors(Cell cell) - возвращает список непосещенных соседей

GetVisitNeightborn(Cell cell) - возвращает список посещенных соседей

RemoveWall(Cell current, Cell Check) - удаляет стену между двумя ячейками

**Отрисовка:**

Draw(Graphics graphics, int cellSize, Point offset) - рисует весь лабиринт

DrawCell(Graphics graphics, Pen pen, Cell cell, int cellSize, Point offset) - рисует отдельную ячейку

**Алгоритм работы:** Используется модифицированный алгоритм рекурсивного backtracking, который гарантирует создание идеального лабиринта с одним уникальным путем между любыми двумя точками.

### 3.2. Классы для управления персонажами

**Класс Player (Пространство имен: WinFormsApp1.Class)**

**Назначение**: Представляет игрока, управляемого пользователем.

**Основные свойства:**

PictureBox PictureBox - графическое представление игрока

bool IsFacingRight - направление взгляда (для отражения спрайта)

int Speed - скорость перемещения (в пикселях за шаг)

Основные методы:

Конструктор Player() - создает игрока и загружает текстуры

LoadTextures() - загружает изображения персонажа из папки Textures

Move(int deltaX, int deltaY) - перемещает игрока с учетом направления

Flip() - отражает спрайт при смене направления движения

CheckBoundaries(Form form) - проверяет границы формы

**Особенности**: Реализует анимацию движения с поворотом спрайта, загрузку текстур из внешних файлов.

**Класс Bot (Пространство имен: WinFormsApp1.Class)**

Назначение: Представляет автоматического оппонента, использующего алгоритм поиска пути.

**Основные свойства:**

PictureBox PictureBox - графическое представление бота

bool IsMoving - статус движения

int Complexity_Bot - сложность (скорость движения)

Stopwatch stopwatch - секундомер для измерения времени

**Основные методы:**

Конструктор Bot(Maze maze, int cellSize, Point mazeOffset, int Complexity_Bot) - инициализирует бота

InitializeBot() - настраивает графическое представление

FindPath() - находит путь от старта до финиша с помощью волнового алгоритма

StartMoving(), StopMoving() - управление движением

MoveAlongPath() - перемещение по найденному пути

GetElapsedTimeMs(), GetElapsedTime() - получение времени прохождения

**События:**

OnBotFinished - возникает при достижении ботом финиша

**Особенности**: Использует таймер для пошагового движения по предварительно рассчитанному пути.

### 3.3. Классы алгоритмов

**Класс WaveAlgorithm**

**Назначение**: Реализует волновой алгоритм (алгоритм Ли) для поиска кратчайшего пути.

**Основные методы:**

FindPath(Maze maze, Point start, Point finish) - основной метод поиска пути

CanMove(Maze maze, Point from, Point to) - проверка возможности перемещения

ReconstructPath(Point[,] previous, Point finish) - восстановление пути по матрице предшественников

**Алгоритм работы:**

Инициализация матриц расстояний и предшественников

Распространение "волны" от стартовой точки

Реконструкция пути от финиша к старту

**Особенности**: Гарантированно находит кратчайший путь в лабиринте без весов.

### 3.4. Классы управления временем и состоянием

**Класс TimerForGame (Пространство имен: WinFormsApp1.Class)**

**Назначение**: Управление игровым временем.

**Основные свойства:**

Label timeLabel - метка для отображения времени

System.Windows.Forms.Timer gameTimer - таймер обновления

int elapsedSeconds - прошедшее время в секундах

**Основные методы:**

Конструктор TimerForGame(Size ClientSize, Form form) - инициализация с учетом размера формы

CreateTimer() - создание и настройка таймера

StartTimer(), StopTimer() - управление таймером

GetElapsedSeconds() - получение прошедшего времени

GameTimerTick(object sender, EventArgs e) - обработчик тика таймера

**Особенности:** Форматирует время в формате "мм:сс", обновляется каждую секунду.

### 3.5. Классы для сохранения игры

**Класс GameSave (Пространство имен: WinFormsApp1)** - основной класс сохранения:

CellWalls - сериализуемое представление стен ячейки

SerializablePoint - сериализуемая точка

### 3.6. Классы форм

**Класс MainMenu (Пространство имен: WinFormsApp1)**

**Назначение**: Главное меню программы.

Основные компоненты:

**Кнопки** "Играть", "Выход", "Выбор сложности" (Рисунок 1)

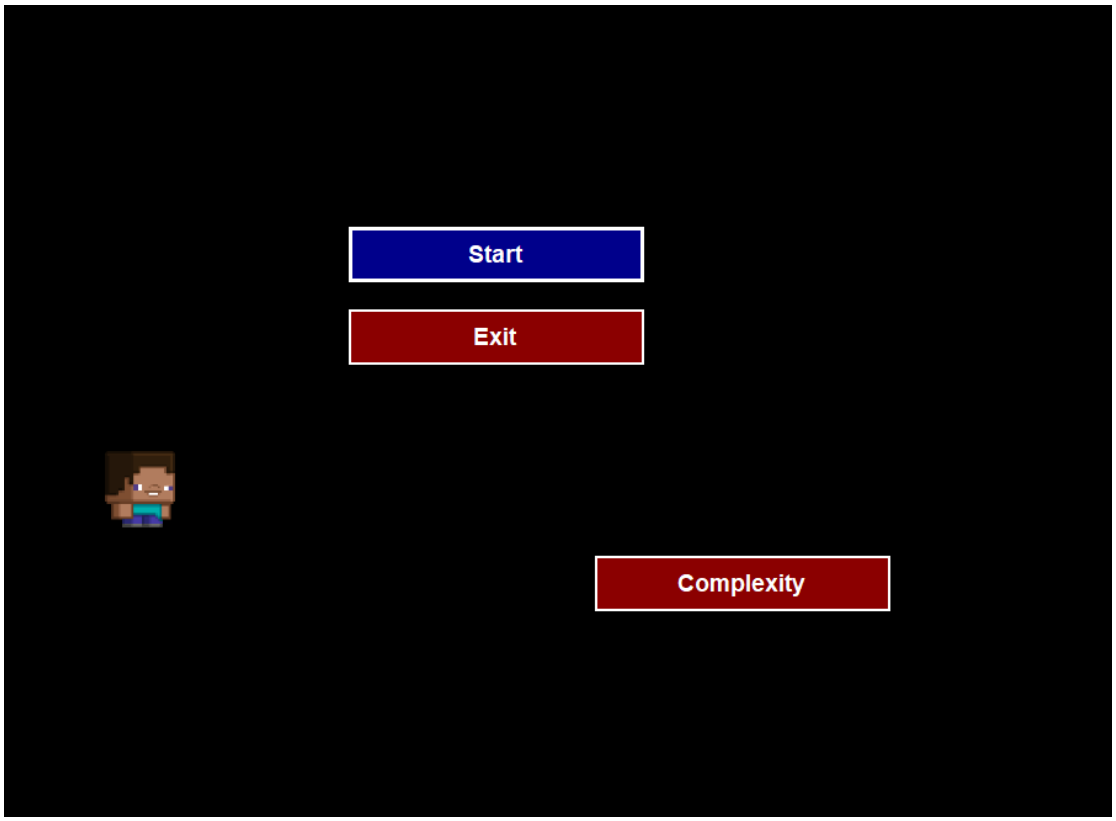



Рисунок 1 - Главное меню

**Панель выбора** сложности с тремя уровнями 5x5, 10x10, 15x15 (Рисунок 2)

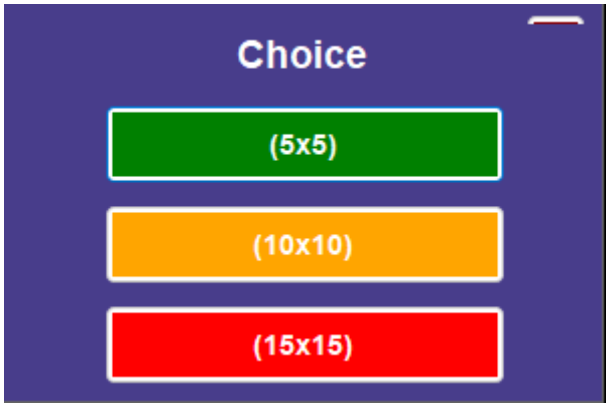



Рисунок 2 - Выбор сложности

Анимированный персонаж в главном меню (Рисунок 1)

**Основные методы:**

Конструктор MainMenu() - инициализация меню

InitializePlayer(), InitializeMenuPanel() - настройка элементов

ShowMenu(), HideMenu() - управление панелью выбора сложности

StartGame() - запуск игры с выбранной сложностью

**Класс GameForm (Пространство имен: WinFormsApp1)**

**Назначение**: Основная игровая форма.

**Основные компоненты:**

Две панели DoubleBufferedPanel для лабиринтов игрока и бота (Рисунок 2)

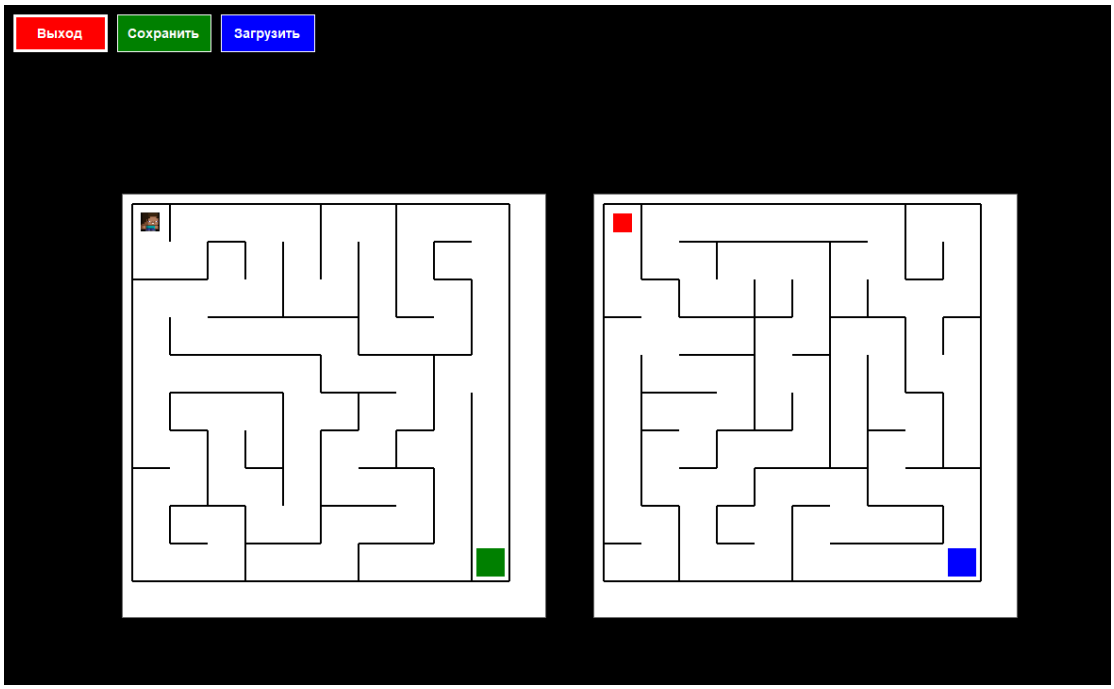



Рисунок 3 - Игровая форма

**Кнопки управления** (Выход, Сохранить, Загрузить)( Рисунок 2)

**Основные методы:**

Конструктор GameForm(int ComplexityMaze) - инициализация игры

InitializeMaze(), InitializePlayer(), InitializeBot() - настройка игровых объектов

GameForm_KeyDown() - обработка управления клавиатурой

WillCollide(), CheckCellCollision() - обработка столкновений

CheckWinCondition() - проверка достижения финиша

SaveGame(), LoadGame() - сохранение и загрузка игры

**Класс ResultForm (Пространство имен: WinFormsApp1)**

**Назначение**: Отображение результатов игры.

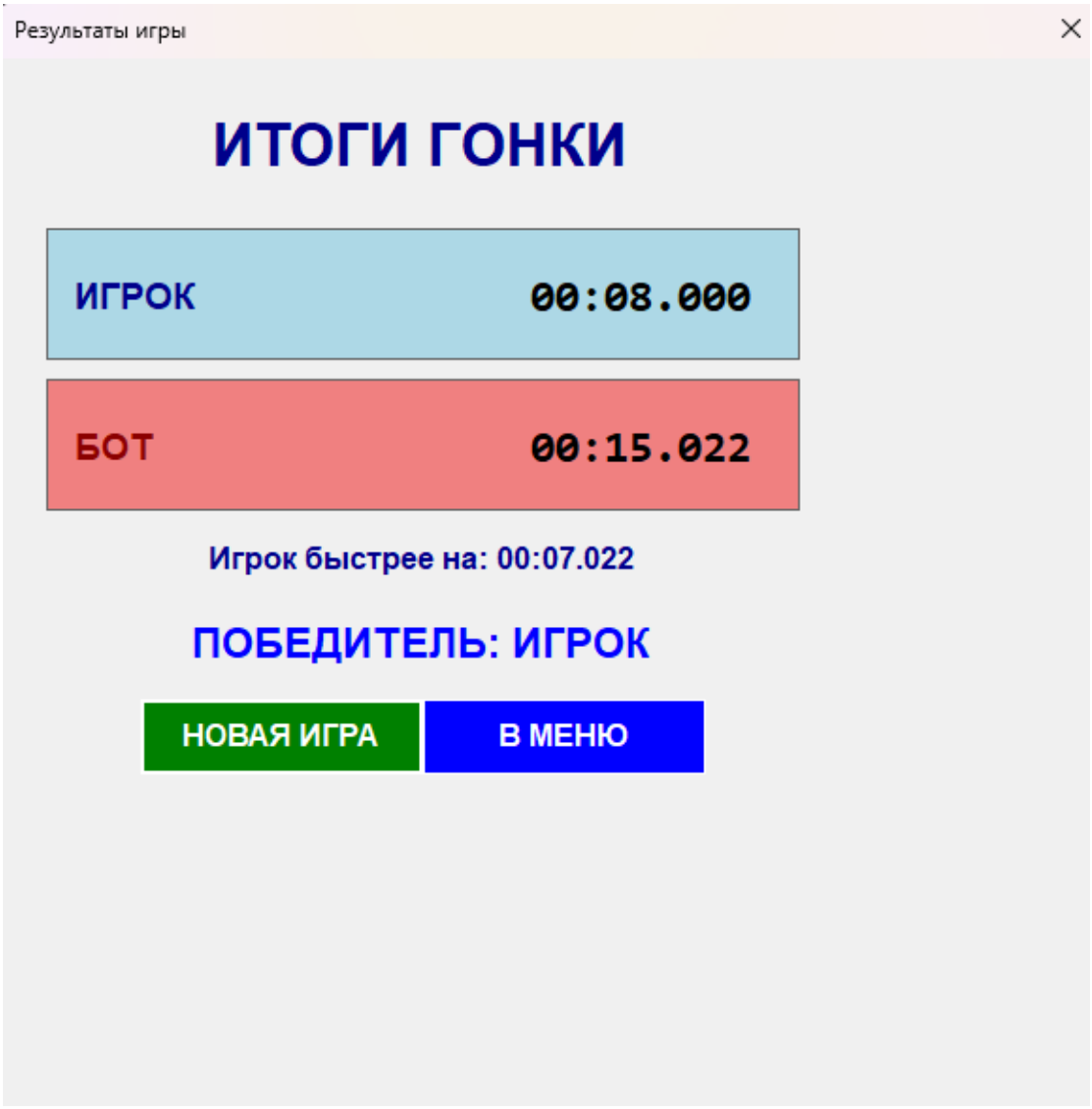



Рисунок 4 - Результат игры

**Основные свойства:**

TimeSpan PlayerTime, TimeSpan BotTime - время участников

string Winner, Color WinnerColor - информация о победителе

**Основные методы:**

Конструктор ResultForm() - инициализация формы

InitializeForm(), SetupControls() - настройка интерфейса

UpdateResults() - обновление данных результатов

**Структура программы**

Программа построена по принципам объектно-ориентированного программирования с четким разделением ответственности:

Модель данных (Cell, Maze, GameSave) - представляют состояние игры

Контроллеры (Player, Bot, WaveAlgorithm) - реализуют логику поведения

Представление (MainMenu, GameForm, ResultForm) - обеспечивают интерфейс пользователя

Служебные классы (TimerForGame) - предоставляют вспомогательные функции

Такая архитектура обеспечивает:

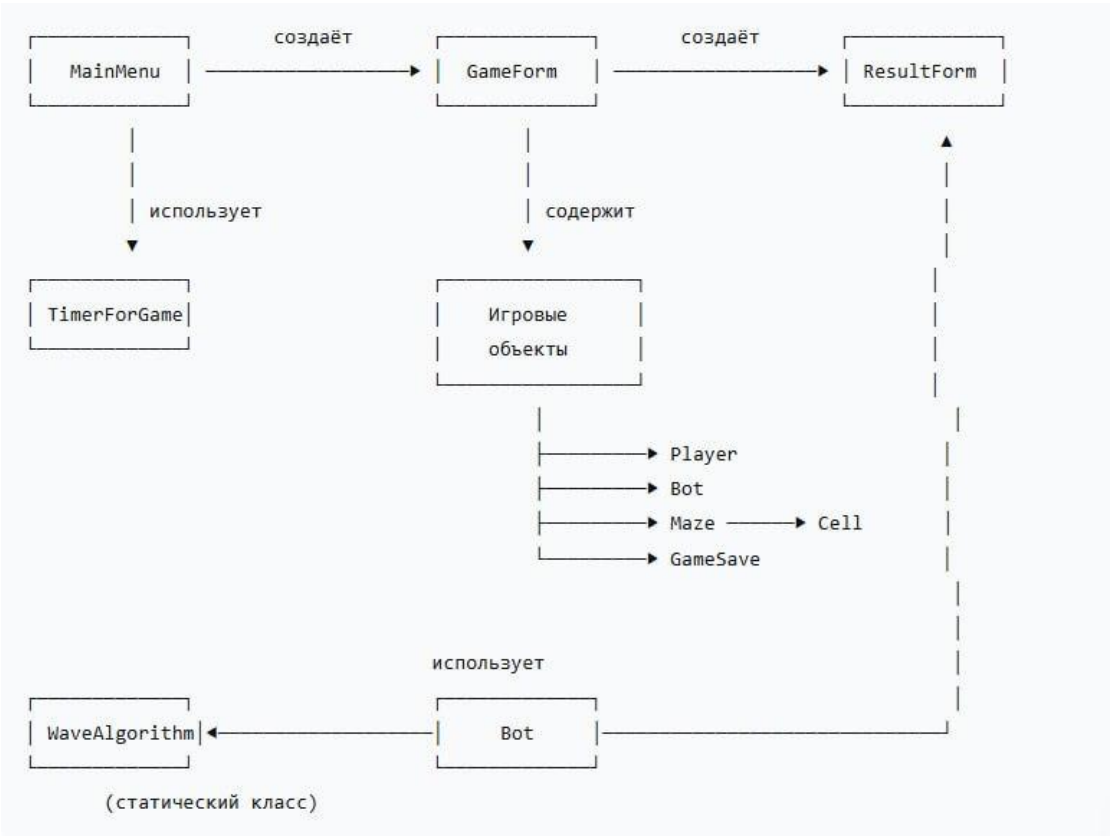
Масштабируемость - легко добавлять новые алгоритмы или режимы игры

Тестируемость - каждый компонент может тестироваться отдельно

Поддержку - четкая структура упрощает отладку и доработку

Переиспользование - многие компоненты могут использоваться в других проектах

## 3.7 Диограмма классов



# 4. Тестирование

Разработанная программа была протестирована в среде разработки **Microsoft Visual Studio Community 2022**. Для обеспечения корректности работы были использованы различные методы тестирования, включая модульное тестирование отдельных компонентов, интеграционное тестирование взаимодействия между классами, а также системное тестирование всего приложения в целом. В процессе разработки применялись следующие техники отладки: пошаговое выполнение кода, установка контрольных точек, мониторинг значений переменных и анализ потока выполнения программы.

Тестирование проводилось как в процессе разработки (для каждой новой функции), так и после завершения основных этапов реализации. Особое внимание уделялось проверке корректности работы алгоритмов генерации лабиринта, поиска пути, обработки столкновений, системы сохранения/загрузки и пользовательского интерфейса.

В таблице 1 представлен план тестирования основных функциональных возможностей программы.

Таблица 1

| № | Название | Предусловие | Ожидаемый езультат |
|---|---|---|---|
| 1 | Запуск Программы | Нет | Запуск MainForm |
| 2 | Выбор сложности | Пользователь в MainForm | Смена параметра сложности с сообщением |
| 3 | Начало новой игры | Пользователь на ResultForm | Запуск MainForm |
| 4 | Управление игроком | Нажатие ионой из кнопок “WASD” | Семена позиции |
| 5 | Определение победителя | Завершение игры против бота | Показ ResultForm |
| 6 | Сохранение игры | Пользователь в GameForm | Сохранения JSON |
| 7 | Загрузка ишры | Пользователь в GameForm | Полная смена GameForm |
| 8 | Обработка Столкновений | Пользователь в GameForm | Сооблюдение коллизии |
| 9 | Генерация лабиринта | Пользователь в GameForm | Лабиринт сгенерирован корректно |

### 4.1 Создание и отображение лабиринта:

**Цель теста:** Проверить корректность генерации лабиринтов разных размеров и их визуального представления.

Предусловия: Программа запущена, открыто главное меню.

**Действия:**

Нажатие кнопки "Выбор сложности"

Выбор уровня "Легкий" (5×5)

Нажатие кнопки "Играть"

Визуальная оценка сгенерированного лабиринта

Возврат в меню, повтор шагов 1-3 для уровней "Средний" (10×10) и "Сложный" (15×15)

**Ожидаемые результаты:**

Для каждого уровня сложности генерируется лабиринт соответствующего размера

Лабиринт имеет четко обозначенные стартовую (левый верхний угол) и финишную (правый нижний угол) точки

Все стены отображаются без разрывов (кроме проходов, созданных алгоритмом генерации)

Лабиринт является идеальным (имеет один уникальный путь между любыми двумя точками)

Визуальное представление одинаково для обоих лабиринтов (игрока и бота)

**Фактические результаты:** Тест пройден успешно. Лабиринты всех размеров генерируются корректно, имеют связную структуру и соответствуют ожидаемым параметрам. Алгоритм рекурсивного backtracking гарантирует создание идеального лабиринта.

### 4.2 Управление персонажем и обработка столкновений

**Цель теста:** Проверить корректность обработки пользовательского ввода и взаимодействие персонажа со стенами лабиринта.

Предусловия: Игровая форма открыта, уровень сложности - "Средний".

**Действия:**

Нажатие клавиши D для движения вправо

Нажатие клавиши S для движения вниз

Попытка движения через стену (в направлении, где стена закрыта)

Движение по всему доступному пути до достижения тупика

Возврат по пройденному пути

**Ожидаемые результаты:**

Персонаж перемещается плавно с заданной скоростью (5 пикселей за шаг)

При изменении направления движения спрайт персонажа отражается соответствующим образом

При столкновении со стеной движение в этом направлении блокируется

Персонаж не может выйти за границы панели лабиринта

Все перемещения корректно отслеживаются системой проверки столкновений

**Фактические результаты:** Тест пройден успешно. Система коллизий работает корректно, персонаж не проходит сквозь стены. Анимация поворота спрайта срабатывает при изменении горизонтального направления движения.

### 4.3 Алгоритмическое прохождение лабиринта ботом

**Цель теста:** Проверить эффективность и корректность работы волнового алгоритма поиска пути.

Предусловия: Игровая форма открыта, уровень сложности - "Средний".

**Действия:**

Наблюдение за автоматическим движением бота

Фиксация времени достижения ботом финиша

Визуальная проверка пути, выбранного ботом

Сравнение пути бота с возможными альтернативными маршрутами

**Ожидаемые результаты:**

Бот находит путь от старта до финиша за конечное время

Пройденный путь **является** кратчайшим (или одним из кратчайших) в данном лабиринте

Движение происходит плавно, с постоянной скоростью, зависящей от выбранной сложности

При достижении финиша срабатывает событие завершения, фиксируется время прохождения

**Фактические результаты:** Тест пройден успешно. Волновой алгоритм гарантированно находит кратчайший путь в лабиринте. Бот движется по оптимальному маршруту, время прохождения соответствует ожидаемому для лабиринта размером 10×10.

### 4.4 Соревновательный режим и определение победителя

**Цель теста:** Проверить корректность работы системы соревнования и определения победителя.

Предусловия: Игровая форма открыта, уровень сложности - "Легкий".

Действия:

Игрок быстро проходит лабиринт (за меньшее время, чем бот)

Ожидание завершения прохождения ботом

Наблюдение за автоматическим переходом к результатам

Анализ отображенных результатов

Повтор теста с ситуацией, когда бот проходит лабиринт быстрее игрока

**Ожидаемые результаты:** При достижении финиша игроком появляется сообщение с его временем

При достижении финиша ботом появляется сообщение с его временем

После финиша обоих участников автоматически открывается форма результатов

На форме корректно отображаются время игрока и бота

Правильно определяется победитель (участник с меньшим временем)

В случае ничьей выводится соответствующее сообщение

Предоставляются опции для начала новой игры или возврата в меню

**Фактические результаты:** Тест пройден успешно. Система корректно определяет победителя в различных сценариях. Форма

результатов наглядно представляет информацию с цветовым кодированием (синий - игрок, красный - бот, серый - ничья).

### 4.5 Система сохранения и загрузки игры

**Цель теста:** Проверить надежность сохранения и восстановления состояния игры.

Предусловия: Игровая форма открыта, игра в процессе.

**Действия:**

Игрок проходит часть лабиринта

Бот проходит часть своего пути

Нажатие кнопки "Сохранить", сохранение в файл "test.save"

Закрытие игровой формы

Возврат в главное меню, начало новой игры

Нажатие кнопки "Загрузить", выбор файла "test.save"

Сравнение восстановленного состояния с исходным

Ожидаемые результаты:

Файл сохранения создается в выбранной директории

При загрузке восстанавливаются: структура лабиринта, позиции игрока и бота, прошедшее время, статус завершения для каждого участника

После загрузки игра продолжается с того же момента

Бот продолжает движение с сохраненной позиции по пересчитанному пути

Все элементы интерфейса отображаются корректно

**Фактические результаты:** Тест пройден успешно. Система сериализации на основе JSON корректно сохраняет и восстанавливает все аспекты игрового состояния. Единственное ограничение - при загрузке необходимо пересчитывать путь для бота,

что реализовано через вызов волнового алгоритма от текущей позиции.

### 4.6 Тестирование пользовательского интерфейса

**Цель теста:** Проверить удобство и интуитивность пользовательского интерфейса.

Предусловия: Программа запущена.

Действия:

Поочередный переход по всем элементам интерфейса

Проверка реакции на различные действия мыши и клавиатуры

Тестирование адаптивности интерфейса при изменении размера окна

Проверка читаемости текста и различимости цветов

**Ожидаемые результаты:** Все кнопки реагируют на наведение и нажатие. Управление с клавиатуры работает корректно (WASD для игрока, Escape для выхода). Интерфейс масштабируется при изменении размера окна

Цветовая схема обеспечивает хорошую контрастность и читаемость

Сообщения об ошибках и информационные сообщения понятны пользователю

**Фактические результаты:** Тест пройден успешно. Пользовательский интерфейс интуитивно понятен, все элементы управления работают корректно. При изменении размера окна происходит перерасчет позиций элементов для сохранения визуальной сбалансированности

### 4.7 Результаты тестирования

Результаты проведенного тестирования представлены в таблице 2.

**Таблица 2 – Результаты тестирования программы**

| № теста | Название теста | Результат |
|---|---|---|
| 1 | Запуск программы | Успешно |
| 2 | Выбор сложности лабиринта | Успешно |
| 3 | Начало новой игры | Успешно |
| 4 | Управление игроком | Успешно |
| 5 | Определение победителя | Успешно |
| 6 | Сохранение игры | Успешно |
| 7 | Загрузка игры | Успешно |
| 8 | Обработка столкновений | Успешно |
| 9 | Генерация лабиринта | Успешно |

Все запланированные тесты были выполнены успешно. Программа демонстрирует стабильную работу, алгоритмы функционируют корректно, пользовательский интерфейс интуитивно понятен. Система сохранения/загрузки обеспечивает надежное хранение игрового состояния. Соревновательный режим

работает как ожидалось, корректно определяя победителя на основе времени прохождения.

Единственным обнаруженным ограничением является необходимость пересчета пути для бота при загрузке сохраненной игры, что было успешно реализовано через повторный вызов волнового алгоритма от текущей позиции бота. В остальном программа соответствует всем предъявленным требованиям и готова к использованию.

# Заключение

В процессе выполнения данной курсовой работы были значительно улучшены навыки разработки программного обеспечения и углублены знания в области алгоритмизации и объектно-ориентированного программирования. Приобретен практический опыт реализации сложных алгоритмов, включая генерацию лабиринтов методом рекурсивного backtracking и поиск кратчайшего пути с использованием волнового алгоритма (алгоритма Ли).

Были успешно освоены современные технологии разработки: язык программирования C#, платформа .NET 8.0, графическая библиотека Windows Forms. Получен ценный опыт работы с сериализацией данных (формат JSON), созданием пользовательских элементов управления, реализацией событийно-ориентированной архитектуры приложения и разработкой интуитивно понятного графического интерфейса.

В рамках курсовой работы была разработана интерактивная игра "Соревнование в лабиринте", предоставляющая пользователю следующие возможности:

Генерацию случайных лабиринтов трех уровней сложности (5×5, 10×10, 15×15)

Ручное управление персонажем с проверкой столкновений со стенами

Автоматическое прохождение лабиринта искусственным интеллектом с использованием волнового алгоритма

Соревновательный режим с одновременным участием игрока и бота

Точное измерение и сравнение времени прохождения для обоих участников

Полноценную систему сохранения и загрузки игрового процесса

Наглядное отображение результатов с определением победителя

Программа демонстрирует корректную работу всех алгоритмов: лабиринты генерируются с гарантией существования пути от старта к финишу, бот всегда находит кратчайший маршрут, система коллизий точно обрабатывает взаимодействие игрока со стенами. Пользовательский интерфейс обеспечивает удобное взаимодействие как с помощью мыши, так и клавиатуры.

Разработанное приложение имеет полноценный функционал, достаточный для комфортного использования и демонстрации принципов работы алгоритмов навигации в лабиринтах. Архитектура программы построена с соблюдением принципов объектно-ориентированного проектирования, что обеспечивает хорошую поддерживаемость и возможность дальнейшего расширения функциональности.

Результаты тестирования подтвердили стабильную работу программы во всех предусмотренных режимах, соответствие техническому заданию и готовность к практическому использованию в качестве учебно-демонстрационного пособия или развлекательного приложения.

# Список литературы


1. **Microsoft Learn: Windows Forms для .NET**
    a. URL: https://learn.microsoft.com/ru-ru/dotnet/desktop/winforms/
    b. Описание: Официальная документация Microsoft по Windows Forms, включающая руководства, примеры кода и API-справочник.
2. **Microsoft Learn: Руководство по C#**
    a. URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/
    b. Описание: Полное руководство по языку программирования C# с примерами и лучшими практиками.
3. **Microsoft Learn: Сериализация JSON в .NET**
    a. URL: https://learn.microsoft.com/ru-ru/dotnet/standard/serialization/system-text-json/how-to
    b. Описание: Документация по работе с JSON сериализацией в .NET, включая System.Text.Json.
4. **GeeksforGeeks: Алгоритмы поиска пути**
    a. URL: https://www.geeksforgeeks.org/search-algorithms/
    b. Описание: Объяснения и реализации различных алгоритмов поиска пути, включая BFS, DFS, A* и волновой алгоритм.

# Приложение А

## WaveAlgorithm.cs:

```
using AI_maze;
using System;
using System.Collections.Generic;
using System.Drawing;
using WinFormsApp1.Class;

namespace WinFormsApp1.Algoritms
{
    public static class WaveAlgorithm
    {

        public static List<Point> FindPath(Maze maze, Point start,
Point finish)
        {
            // Проверяем валидность входных данных
            if (maze == null) return null;
            if (start.X < 0 || start.X >= maze.Width || start.Y <
0 || start.Y >= maze.Height ||
                finish.X < 0 || finish.X >= maze.Width ||
finish.Y < 0 || finish.Y >= maze.Height)
                return null;

            int[,] distance = new int[maze.Width, maze.Height];
// Расстояние от старта
            Point[,] previous = new Point[maze.Width,
maze.Height];

            // Инициализация массивов
            for (int x = 0; x < maze.Width; x++)
            {
                for (int y = 0; y < maze.Height; y++)
                {
                    distance[x, y] = -1;
                    previous[x, y] = new Point(-1, -1);
                }
            }

            Queue<Point> queue = new Queue<Point>();
```

```
                    distance[start.X, start.Y] = 0;
                    queue.Enqueue(start);

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

                    while (queue.Count > 0)
                    {
                        Point current = queue.Dequeue();

                        if (current.Equals(finish))
                            break;

                        foreach (Point dir in directions)
                        {
                            Point neighbor = new Point(current.X + dir.X,
current.Y + dir.Y);

                            if (neighbor.X >= 0 && neighbor.X <
maze.Width &&
                                neighbor.Y >= 0 && neighbor.Y <
maze.Height &&
                                CanMove(maze, current, neighbor) &&
                                distance[neighbor.X, neighbor.Y] == -1)
                            {
                                distance[neighbor.X, neighbor.Y] =
distance[current.X, current.Y] + 1;
                                previous[neighbor.X, neighbor.Y] =
current;
                                queue.Enqueue(neighbor);
                            }
                        }
                    }

                    return ReconstructPath(previous, finish);
                }

                private static bool CanMove(Maze maze, Point from, Point
to)
                {
```

```
                    if (from.X < 0 || from.X >= maze.Width || from.Y < 0
|| from.Y >= maze.Height ||
                        to.X < 0 || to.X >= maze.Width || to.Y < 0 ||
to.Y >= maze.Height)
                        return false;

                    Cell fromCell = maze.Cells[from.X, from.Y];
                    Cell toCell = maze.Cells[to.X, to.Y];

                    int dx = to.X - from.X;
                    int dy = to.Y - from.Y;

                    if (dx == 1) return !fromCell.rightWall
&& !toCell.leftWall;
                    else if (dx == -1) return !fromCell.leftWall
&& !toCell.rightWall;
                    else if (dy == 1) return !fromCell.downWall
&& !toCell.topWall;
                    else if (dy == -1) return !fromCell.topWall
&& !toCell.downWall;

                    return false;
                }

                private static List<Point> ReconstructPath(Point[,]
previous, Point finish)
                {
                    if (previous[finish.X, finish.Y].X == -1) return null;

                    List<Point> path = new List<Point>();
                    Point current = finish;

                    while (current.X != -1 && current.Y != -1)
                    {
                        path.Add(current);
                        current = previous[current.X, current.Y];
                    }

                    path.Reverse();
                    return path.Count > 0 ? path : null;
                }
        }
        }
```

```
}
```

## Bot.cs:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.Windows.Forms;
using WinFormsApp1.Algoritms;
using Timer = System.Windows.Forms.Timer;

namespace WinFormsApp1.Class
{
    public class Bot
    {
        public PictureBox PictureBox { get; private set; }
        public bool IsMoving { get; private set; } = false;

        private Maze maze;
        private Timer movementTimer;
        private List<Point> path;
        private int currentPathIndex = 0;
        private int cellSize;
        private Point mazeOffset;
        private Color botColor = Color.Red;
        public int Complexity_Bot;
        private Stopwatch stopwatch;
        private long elapsedTimeMs = 0;

        public Bot(Maze maze, int cellSize, Point mazeOffset, int
Complexity_Bot =1000)
        {
            this.Complexity_Bot = Complexity_Bot;
            this.maze = maze ?? throw new
ArgumentNullException(nameof(maze));
            this.cellSize = cellSize;
            this.mazeOffset = mazeOffset;
            stopwatch = new Stopwatch();


            InitializeBot();
            FindPath();
            SetupMovementTimer();
        }
```

```
        private void InitializeBot()
        {
            PictureBox = new PictureBox();
            PictureBox.Size = new Size(cellSize / 2, cellSize /
2);
            PictureBox.BackColor = botColor;
            SetPosition(0, 0);
        }

        private void SetPosition(int cellX, int cellY)
        {
            PictureBox.Location = new Point(
                mazeOffset.X + cellX * cellSize + cellSize / 4,
                mazeOffset.Y + cellY * cellSize + cellSize / 4
            );
        }

        private void FindPath()
        {
            path = WaveAlgorithm.FindPath(maze, new Point(0, 0),
                new Point(maze.Width - 1, maze.Height - 1));

            if (path != null && path.Count > 0)
            {
                currentPathIndex = 0;
                SetPosition(path[0].X, path[0].Y);
            }
        }

        private void SetupMovementTimer()
        {
            movementTimer = new Timer();
            movementTimer.Interval = Complexity_Bot;
            movementTimer.Tick += (s, e) => MoveAlongPath();
        }

        public void StartMoving()
        {
            if (path != null && path.Count > 1 && !IsMoving)
            {
                IsMoving = true;
                stopwatch.Start();
```

```
            movementTimer.Start();
        }
    }

    public void StopMoving()
    {
        IsMoving = false;
        movementTimer?.Stop();
        stopwatch.Stop();
        elapsedTimeMs = stopwatch.ElapsedMilliseconds;
    }

    private void MoveAlongPath()
    {
        if (currentPathIndex < path.Count - 1)
        {
            currentPathIndex++;
            Point nextCell = path[currentPathIndex];
            SetPosition(nextCell.X, nextCell.Y);
        }
        else
        {
            StopMoving();
            OnBotFinished?.Invoke(this, EventArgs.Empty);
        }
    }

    public event EventHandler OnBotFinished;

    public void Reset()
    {
        StopMoving();
        currentPathIndex = 0;
        elapsedTimeMs = 0;
        stopwatch.Reset();
        if (path != null && path.Count > 0)
        {
            SetPosition(path[0].X, path[0].Y);
        }
    }

    public long GetElapsedTimeMs()
    {
```

```
                return elapsedTimeMs;
            }

            public TimeSpan GetElapsedTime()
            {
                return TimeSpan.FromMilliseconds(elapsedTimeMs);
            }

            // Метод для отрисовки пути (опционально)
            public void DrawPath(Graphics graphics, Point offset, int
cellSize)
            {
                if (path == null || path.Count < 2) return;

                using (Pen pathPen = new Pen(Color.Yellow, 2))
                {
                    for (int i = 0; i < path.Count - 1; i++)
                    {
                        Point current = path[i];
                        Point next = path[i + 1];

                        int currentX = offset.X + current.X *
cellSize + cellSize / 2;
                        int currentY = offset.Y + current.Y *
cellSize + cellSize / 2;
                        int nextX = offset.X + next.X * cellSize +
cellSize / 2;
                        int nextY = offset.Y + next.Y * cellSize +
cellSize / 2;

                        graphics.DrawLine(pathPen, currentX, currentY,
nextX, nextY);
                    }
                }
            }
        }
    }
```

## Cell.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

namespace AI_maze
{
    public class Cell
    {
        public int X { get; }
        public int Y { get; }
        public bool isVisit = false;

        public bool topWall = true;
        public bool downWall = true;
        public bool leftWall = true;
        public bool rightWall = true;

        public Cell(int x, int y)
        {
            X = x;
            Y = y;
        }

    }
}
```

## GameSave.cs:

```
// GameSave.cs
using System;
using System.Collections.Generic;
using System.Drawing;

namespace WinFormsApp1
{
    [Serializable]
    public class GameSave
    {
        public int Version { get; set; } = 1;
        public int Complexity { get; set; }
        public Point PlayerPosition { get; set; }
        public Point BotPosition { get; set; }
        public int CurrentBotPathIndex { get; set; }
        public bool PlayerFinished { get; set; }
        public bool BotFinished { get; set; }
```

```
        public int PlayerTimeSeconds { get; set; }
        public long BotTimeMs { get; set; }
        public int ElapsedSeconds { get; set; }
        public bool IsBotMoving { get; set; }
        public DateTime SaveDate { get; set; }

        // Сериализуемые стены лабиринта
        public List<CellWalls> MazeWalls { get; set; }

        // Для хранения позиций пути бота
        public List<SerializablePoint> BotPath { get; set; }
    }

    [Serializable]
    public class CellWalls
    {
        public int X { get; set; }
        public int Y { get; set; }
        public bool TopWall { get; set; }
        public bool RightWall { get; set; }
        public bool BottomWall { get; set; }
        public bool LeftWall { get; set; }
    }

    [Serializable]
    public class SerializablePoint
    {
        public int X { get; set; }
        public int Y { get; set; }
    }
}
```

## Maze.cs:

```
using AI_maze;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFormsApp1.Class
{
    public class Maze
    {

        public int Height, Width;

        public Cell[,] Cells { get; private set; }

        Random rnd = new Random();
        public Pen wallPen = new Pen(Color.Black, 2);
        public Maze(int Height, int Width)
        {
            this.Height = Height;
            this.Width = Width;
            Cells = new Cell[Width, Height];

            for (int x = 0; x < Width; x++)
            {
                for (int y = 0; y < Height; y++)
                {
                    Cells[x, y] = new Cell(x, y);
                }
            }
        }

        public void Kill(int startX, int startY)
        {

            Cell cell = Cells[startX, startY];
            cell.isVisit = true;
            List<Cell> listUnvisitNeight =
GetUnVisitNeighbors(cell);
```

```
        if (listUnvisitNeight.Count > 0)
        {
            int rand = rnd.Next(0, listUnvisitNeight.Count);
            Cell randFromListCell = listUnvisitNeight[rand];
            RemoveWall(cell, randFromListCell);
            Kill(randFromListCell.X, randFromListCell.Y);
        }
        else
        {
            Hunt();
        }

    }
    public void Hunt()
    {
        for (int x = 0; x < Width; x++)
        {
            for (int y = 0; y < Height; y++)
            {
                Cell cell = Cells[x, y];
                List<Cell> list = GetVisitNeightborn(cell);
                if (cell.isVisit == false && list.Count > 0)
                {
                    int rand = rnd.Next(0, list.Count);
                    Cell randFromListCell = list[rand];
                    RemoveWall(cell, randFromListCell);
                    Kill(cell.X, cell.Y);
                    return;
                }
            }
        }
    }

    List<Cell> GetVisitNeightborn(Cell cell)
    {
        List<Cell> neighbors = new List<Cell>();

        if (cell.Y > 0)
        {
            Cell topN = Cells[cell.X, cell.Y - 1];
            if (topN.isVisit == true) neighbors.Add(topN);

        }
```

```
    if (cell.Y < Height - 1)
    {
        Cell down = Cells[cell.X, cell.Y + 1];
        if (down.isVisit == true) neighbors.Add(down);
    }
    if (cell.X > 0)
    {
        Cell left = Cells[cell.X - 1, cell.Y];
        if (left.isVisit == true) neighbors.Add(left);
    }
    if (cell.X < Width - 1)
    {
        Cell right = Cells[cell.X + 1, cell.Y];
        if (right.isVisit == true) neighbors.Add(right);
    }

    return neighbors;
}


void RemoveWall(Cell current, Cell Check)
{
    int divX = current.X - Check.X;
    int divY = current.Y - Check.Y;
    if (divX == 1)
    {
        current.leftWall = false;
        Check.rightWall = false;
    }
    else if (divX == -1)
    {
        current.rightWall = false;
        Check.leftWall = false;
    }
    else if (divY == 1)
    {
        current.topWall = false;
        Check.downWall = false;
    }
    else if (divY == -1)
    {
        current.downWall = false;
        Check.topWall = false;
```

```
            }

        }

        private List<Cell> GetUnVisitNeighbors(Cell cell)
        {
            List<Cell> neighbors = new List<Cell>();

            if (cell.Y > 0)
            {
                Cell topN = Cells[cell.X, cell.Y - 1];
                if (topN.isVisit == false) neighbors.Add(topN);

            }
            if (cell.Y < Height - 1)
            {
                Cell down = Cells[cell.X, cell.Y + 1];
                if (down.isVisit == false) neighbors.Add(down);
            }
            if (cell.X > 0)
            {
                Cell left = Cells[cell.X - 1, cell.Y];
                if (left.isVisit == false) neighbors.Add(left);
            }
            if (cell.X < Width - 1)
            {
                Cell right = Cells[cell.X + 1, cell.Y];
                if (right.isVisit == false) neighbors.Add(right);
            }

            return neighbors;
        }

        public void Draw(Graphics graphics, int cellSize, Point
offset)
        {
            graphics.Clear(Color.White);

            using (Pen pen = new Pen(Color.Black, 2))
            {
                for (int x = 0; x < Width; x++)
                {
                    for (int y = 0; y < Height; y++)
```

```
                            {
                                Cell cell = Cells[x, y];
                                DrawCell(graphics, pen, cell, cellSize,
offset);
                            }
                        }
                    }
                }

                void DrawCell(Graphics graphics, Pen pen, Cell cell, int
cellSize, Point offset)
                {
                    int x = cell.X * cellSize + offset.X;
                    int y = cell.Y * cellSize + offset.Y;

                    if (cell.topWall)
                        graphics.DrawLine(pen, x, y, x + cellSize, y);

                    if (cell.downWall)
                        graphics.DrawLine(pen, x, y + cellSize, x +
cellSize, y + cellSize);

                    if (cell.leftWall)
                        graphics.DrawLine(pen, x, y, x, y + cellSize);

                    if (cell.rightWall)
                        graphics.DrawLine(pen, x + cellSize, y, x +
cellSize, y + cellSize);
                }

            }

        }
```

## Player.cs:

```
        using System.Drawing;
        using System.Windows.Forms;

        namespace WinFormsApp1.Class
        {
            public class Player
            {
```

```
        public PictureBox PictureBox { get; private set; }
        public bool IsFacingRight { get; private set; } = true;
        public int Speed { get; set; } = 5;

        private Image originalImage;
        private Image flippedImage;

        public Player()
        {
            PictureBox = new PictureBox();
            PictureBox.Size = new Size(50, 50);
            PictureBox.Location = new Point(100, 100);
            PictureBox.BackColor = Color.Blue;

            LoadTextures();
        }

        private void LoadTextures()
        {
            // Получаем путь к корневой папке проекта
            string projectRoot = GetProjectRootDirectory();
            string texturesPath = Path.Combine(projectRoot,
"Textures");

            string originalPath = Path.Combine(texturesPath,
"minecraft-stev.gif");
            string flippedPath = Path.Combine(texturesPath,
"minecraft-stevFlip.gif");

            if (File.Exists(originalPath) &&
File.Exists(flippedPath))
            {
                try
                {
                    originalImage = Image.FromFile(originalPath);
                    flippedImage = Image.FromFile(flippedPath);
                    PictureBox.Image = originalImage;
                    PictureBox.SizeMode =
PictureBoxSizeMode.StretchImage;
                    PictureBox.BackColor = Color.Transparent;
                }
                catch (Exception ex)
                {
```

```
                    MessageBox.Show($"Ошибка загрузки:
{ex.Message}");
                }
            }
            else
            {
                MessageBox.Show($"Файлы не
найдены:\n{originalPath}\n{flippedPath}");
            }
        }

        private string GetProjectRootDirectory()
        {
            string currentDir = Directory.GetCurrentDirectory();
            DirectoryInfo dir = new DirectoryInfo(currentDir);

            // Поднимаемся до папки WinFormsApp1
            for (int i = 0; i < 3; i++)
            {
                if (dir.Parent != null)
                    dir = dir.Parent;
            }

            return dir.FullName;
        }

        public void Move(int deltaX, int deltaY)
        {
            PictureBox.Left += deltaX;
            PictureBox.Top += deltaY;

            if (deltaX < 0 && IsFacingRight)
            {
                Flip();
            }
            else if (deltaX > 0 && !IsFacingRight)
            {
                Flip();
            }
        }

        public void Flip()
        {
```

```
                    IsFacingRight = !IsFacingRight;

                    if (originalImage != null && flippedImage != null)
                    {
                        PictureBox.Image = IsFacingRight ? originalImage :
flippedImage;
                    }
                }

                public void CheckBoundaries(Form form)
                {
                    if (PictureBox.Top < 0)
                        PictureBox.Top = 0;
                    if (PictureBox.Top > form.ClientSize.Height -
PictureBox.Height)
                        PictureBox.Top = form.ClientSize.Height -
PictureBox.Height;
                    if (PictureBox.Left < 0)
                        PictureBox.Left = 0;
                    if (PictureBox.Left > form.ClientSize.Width -
PictureBox.Width)
                        PictureBox.Left = form.ClientSize.Width -
PictureBox.Width;
                }
            }
        }
```

## Timer.cs:

```
        using System;
        using System.Collections.Generic;
        using System.Linq;
        using System.Text;
        using System.Threading.Tasks;

        namespace WinFormsApp1.Class
        {
            public class TimerForGame
            {
                private Label timeLabel;
                private System.Windows.Forms.Timer gameTimer;
                private int elapsedSeconds = 0;
                Size _clientSize;
                Form _form;
```

```
        public TimerForGame(Size ClientSize, Form form)
        {
            _clientSize= ClientSize;
            _form= form;
        }

        public void StopTimer()
        {
            gameTimer.Stop();
        }

        public int GetElapsedSeconds()
        {
            return elapsedSeconds;
        }

        public void CreateTimer()
        {
            timeLabel = new Label();
            timeLabel.Text = "00:00";
            timeLabel.Size = new Size(80, 30);
            timeLabel.Font = new Font("Consolas", 14,
FontStyle.Bold);
            timeLabel.ForeColor = Color.White;
            timeLabel.BackColor = Color.Black;
            timeLabel.TextAlign = ContentAlignment.MiddleCenter;
            timeLabel.BorderStyle = BorderStyle.FixedSingle;
            timeLabel.Anchor = AnchorStyles.Top |
AnchorStyles.Right;
            timeLabel.Location = new Point(
                _clientSize.Width - timeLabel.Width - 20,
                20
            );
            _form.Controls.Add(timeLabel);

            gameTimer = new System.Windows.Forms.Timer();
            gameTimer.Interval = 1000;
            gameTimer.Tick += GameTimerTick;
            StartTimer();

        }
```

```
        public void GameTimerTick(object sender, EventArgs e)
        {

            elapsedSeconds++;
            TimeSpan time = TimeSpan.FromSeconds(elapsedSeconds);
            timeLabel.Text = string.Format("{0:00}:{1:00}",
(int)time.TotalMinutes, time.Seconds);
        }

        public void StartTimer()
        {

            elapsedSeconds = 0;
            timeLabel.Text = "00:00";
            gameTimer.Start();

        }
    }
}
```

## GameForm.cs:

```
using AI_maze;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;
using System.Threading.Tasks;
using System.Windows.Forms;
using WinFormsApp1.Algoritms;
using WinFormsApp1.Class;

namespace WinFormsApp1
{
    public partial class GameForm : Form
    {
        private bool resultsShown = false;
```

```
        private Player player;
        private Maze maze;
        private Maze mazeForBot;
        private int cellSize = 40;
        private Point finishPoint;
        private Point finishPointForBot;
        private Bot bot;
        private int ComplexityMaze { get; set; }
        private TimerForGame gameTimer;

        private bool playerFinished = false;
        private bool botFinished = false;
        private int playerTimeSeconds = 0;
        private long botTimeMs = 0;

        // Панели для лабиринтов
        private DoubleBufferedPanel playerMazePanel;
        private DoubleBufferedPanel botMazePanel;

        public GameForm(int ComplexityMaze)
        {
            this.ComplexityMaze = ComplexityMaze;

            InitializeComponent();

            InitializePanels(this.ComplexityMaze =
ComplexityMaze);


            InitializeMaze();
            InitializePlayer();
            SpawnButton();
            InitializeBot();

            // Настройка формы
            this.WindowState = FormWindowState.Maximized;
            this.FormBorderStyle = FormBorderStyle.None;
            this.KeyPreview = true;
            this.BackColor = Color.Black;
            this.DoubleBuffered = true;

            // Подписываемся на события
            this.KeyDown += GameForm_KeyDown;
```

```
            CreateTimerObj();
        }

        public void CreateTimerObj()
        {
            gameTimer = new TimerForGame(this.ClientSize, this);
            gameTimer.CreateTimer();
        }

        private void InitializeBot()
        {
            if (mazeForBot == null) return;

            // Создаем бота с лабиринтом для бота
            Point botMazeOffset = new Point(10, 10);
            bot = new Bot(mazeForBot, cellSize, botMazeOffset);

            // Добавляем бота на панель бота
            botMazePanel.Controls.Add(bot.PictureBox);
            bot.PictureBox.BringToFront();


            bot.OnBotFinished += (s, e) =>
            {
                botFinished = true;
                botTimeMs = bot.GetElapsedTimeMs();

                // Отладочное сообщение
                Debug.WriteLine($"Бот дошел до финиша! Время:
{botTimeMs} мс = {botTimeMs / 1000.0} сек");

                // Показываем сообщение, что бот дошел до финиша
                ShowTemporaryMessage($"Бот дошел до
финиша!\nВремя: {TimeSpan.FromMilliseconds(botTimeMs):mm\\:ss\\.fff}");

                // Проверяем, дошел ли уже игрок
                CheckIfBothFinished();
            };
            bot.StartMoving();
        }

        private void CheckIfBothFinished()
        {
```

```
                    // Проверяем, оба ли участника дошли до финиша
                    if (playerFinished && botFinished && !resultsShown)
                    {
                        resultsShown = true;

                        // Ждем немного, чтобы игрок увидел сообщение о
финише бота
                        Task.Delay(1000).ContinueWith(_ =>
                        {
                            this.Invoke(new Action(ShowFinalResults));
                        });
                    }
                }
                private void ShowFinalResults()
                {
                    // Создаем форму с результатами
                    using (ResultForm resultForm = new ResultForm())
                    {
                        resultForm.PlayerTime =
TimeSpan.FromSeconds(playerTimeSeconds);
                        resultForm.BotTime =
TimeSpan.FromMilliseconds(botTimeMs);

                        // Определяем победителя
                        if (playerTimeSeconds < (botTimeMs / 1000.0))
                        {
                            resultForm.Winner = "Игрок";
                            resultForm.WinnerColor = Color.Blue;
                        }
                        else if (playerTimeSeconds > (botTimeMs / 1000.0))
                        {
                            resultForm.Winner = "Бот";
                            resultForm.WinnerColor = Color.Red;
                        }
                        else
                        {
                            resultForm.Winner = "Ничья";
                            resultForm.WinnerColor = Color.Gray;
                        }

                        var dialogResult = resultForm.ShowDialog();

                        if (dialogResult == DialogResult.Retry)
```

```
                {
                    // Начать новую игру
                    this.DialogResult = DialogResult.Retry;
                    this.Close();
                }
                else
                {
                    this.Close();
                }
            }
        }

        private void ShowTemporaryMessage(string message)
        {
            // Создаем форму с сообщением
            Form messageForm = new Form();
            messageForm.Text = "Информация";
            messageForm.Size = new Size(300, 150);
            messageForm.StartPosition =
FormStartPosition.CenterParent;
            messageForm.FormBorderStyle =
FormBorderStyle.FixedDialog;
            messageForm.MaximizeBox = false;
            messageForm.MinimizeBox = false;

            Label messageLabel = new Label();
            messageLabel.Text = message;
            messageLabel.Font = new Font("Arial", 10,
FontStyle.Bold);
            messageLabel.ForeColor = Color.Black;
            messageLabel.Size = new Size(280, 80);
            messageLabel.Location = new Point(10, 20);
            messageLabel.TextAlign =
ContentAlignment.MiddleCenter;
            messageForm.Controls.Add(messageLabel);

            Button okButton = new Button();
            okButton.Text = "OK";
            okButton.Size = new Size(80, 30);
            okButton.Location = new Point(110, 100);
            okButton.Click += (s, e) => messageForm.Close();
            messageForm.Controls.Add(okButton);
```

```
            messageForm.ShowDialog();
        }

        private void ShowResults()
        {
            // Получаем время бота в секундах
            double botTimeSeconds = botTimeMs / 1000.0;

            string winner = "";
            Color winnerColor = Color.Gray;

            // Определяем победителя
            if (playerTimeSeconds < botTimeSeconds)
            {
                winner = "Игрок";
                winnerColor = Color.Blue;
            }
            else if (playerTimeSeconds > botTimeSeconds)
            {
                winner = "Бот";
                winnerColor = Color.Red;
            }
            else
            {
                winner = "Ничья";
                winnerColor = Color.Gray;
            }

            // Создаем форму с результатами
            using (ResultForm resultForm = new ResultForm())
            {
                resultForm.PlayerTime =
TimeSpan.FromSeconds(playerTimeSeconds);
                resultForm.BotTime =
TimeSpan.FromMilliseconds(botTimeMs);
                resultForm.Winner = winner; // Устанавливаем
победителя
                resultForm.WinnerColor = winnerColor; //
Устанавливаем цвет победителя

                resultForm.ShowDialog();
            }
```

```
                // Закрываем игровую форму
                this.Close();
            }

            private void BotMazePanel_Paint(object sender,
PaintEventArgs e)
            {
                // Очищаем панель
                e.Graphics.Clear(Color.White);

                // Отрисовываем лабиринт с небольшим отступом от
краев панели
                Point panelOffset = new Point(10, 10);
                mazeForBot.Draw(e.Graphics, cellSize, panelOffset);

                // Отрисовываем финиш
                DrawFinish(e.Graphics, panelOffset, finishPointForBot,
Color.Blue);

                // Подпись для лабиринта бота
                DrawLabel(e.Graphics, "", panelOffset);

            }

            // При закрытии формы останавливаем бота
            protected override void OnFormClosed(FormClosedEventArgs
e)
            {
                bot?.StopMoving();
                base.OnFormClosed(e);
            }

            private void InitializePanels(int complixity)
            {

                int shiftX, shiftY = 200;
                if (complixity == 15)
                    shiftX = 500;
                else shiftX = 250;
                // Рассчитываем размер панели на основе размера
лабиринта
                int panelSize = ComplexityMaze * cellSize + 50; //
+50 для отступов
```

```
                    // Панель для игрока (левая половина экрана)
                    playerMazePanel = new DoubleBufferedPanel();
                    playerMazePanel.Size = new Size(panelSize, panelSize);
                    playerMazePanel.Location = new Point(
                        (this.ClientSize.Width / 2 - panelSize) / 2 + 150,
                        (this.ClientSize.Height - panelSize) / 2 + shiftY
                    );
                    playerMazePanel.BackColor = Color.White;
                    playerMazePanel.BorderStyle = BorderStyle.FixedSingle;
                    playerMazePanel.Paint += PlayerMazePanel_Paint;
                    this.Controls.Add(playerMazePanel);

                    // Панель для бота (правая половина экрана)
                    botMazePanel = new DoubleBufferedPanel();
                    botMazePanel.Size = new Size(panelSize, panelSize);
                    botMazePanel.Location = new Point(
                        this.ClientSize.Width / 2 +
(this.ClientSize.Width / 2 - panelSize) / 2 + shiftX,
                        (this.ClientSize.Height - panelSize) / 2 + shiftY
                    );
                    botMazePanel.BackColor = Color.White;
                    botMazePanel.BorderStyle = BorderStyle.FixedSingle;
                    botMazePanel.Paint += BotMazePanel_Paint;
                    this.Controls.Add(botMazePanel);
                }

                private void InitializeMaze()
                {
                    maze = new Maze(ComplexityMaze, ComplexityMaze);
                    maze.Kill(0, 0);

                    mazeForBot = new Maze(ComplexityMaze, ComplexityMaze);
                    mazeForBot.Kill(0, 0);

                    // Устанавливаем финиш в правый нижний угол
                    finishPoint = new Point(maze.Width - 1, maze.Height -
1);
                    finishPointForBot = new Point(mazeForBot.Width - 1,
mazeForBot.Height - 1);
                }

                private void InitializePlayer()
```

```
            {
                player = new Player();

                // Размер игрока пропорционально клетке лабиринта
                player.PictureBox.Size = new Size(cellSize / 2,
cellSize / 2);

                // Позиционируем игрока в начале лабиринта на панели
игрока
                player.PictureBox.Location = new Point(
                    playerMazePanel.Location.X + cellSize / 4 + 10,
                    playerMazePanel.Location.Y + cellSize / 4 + 10
                );

                this.Controls.Add(player.PictureBox);
                player.PictureBox.BringToFront();
            }

            private void SpawnButton()
            {
                Button exitButton = new Button();
                exitButton.Text = "Выход";
                exitButton.Size = new Size(100, 40);
                exitButton.Location = new Point(
                    10,10
                );
                exitButton.BackColor = Color.Red;
                exitButton.ForeColor = Color.White;
                exitButton.Font = new Font("Arial", 10,
FontStyle.Bold);
                exitButton.FlatStyle = FlatStyle.Flat;
                exitButton.FlatAppearance.BorderSize = 2;
                exitButton.FlatAppearance.BorderColor = Color.White;
                exitButton.Click += Exit_clic;
                this.Controls.Add(exitButton);
                exitButton.BringToFront();

                Button saveButton = new Button();
                saveButton.Text = "Сохранить";
                saveButton.Size = new Size(100, 40);
                saveButton.Location = new Point(120, 10);
                saveButton.BackColor = Color.Green;
                saveButton.ForeColor = Color.White;
```

```
                    saveButton.Font = new Font("Arial", 10,
FontStyle.Bold);
                    saveButton.FlatStyle = FlatStyle.Flat;
                    saveButton.Click += (s, e) =>
                    {
                        using (SaveFileDialog saveDialog = new
SaveFileDialog())
                        {
                            saveDialog.Filter = "Файлы сохранения
(*.save)|*.save|Все файлы (*.*)|*.*";
                            saveDialog.DefaultExt = "save";
                            saveDialog.Title = "Сохранить игру";

                            if (saveDialog.ShowDialog() ==
DialogResult.OK)
                            {
                                SaveGame(saveDialog.FileName);
                            }
                        }
                    };
                    this.Controls.Add(saveButton);
                    saveButton.BringToFront();

                    Button loadButton = new Button();
                    loadButton.Text = "Загрузить";
                    loadButton.Size = new Size(100, 40);
                    loadButton.Location = new Point(230, 10);
                    loadButton.BackColor = Color.Blue;
                    loadButton.ForeColor = Color.White;
                    loadButton.Font = new Font("Arial", 10,
FontStyle.Bold);
                    loadButton.FlatStyle = FlatStyle.Flat;
                    loadButton.Click += (s, e) =>
                    {
                        using (OpenFileDialog openDialog = new
OpenFileDialog())
                        {
                            openDialog.Filter = "Файлы сохранения
(*.save)|*.save|Все файлы (*.*)|*.*";
                            openDialog.DefaultExt = "save";
                            openDialog.Title = "Загрузить игру";
```

```
                        if (openDialog.ShowDialog() ==
DialogResult.OK)
                        {
                            LoadGame(openDialog.FileName);
                        }
                    }
                };
                this.Controls.Add(loadButton);
                loadButton.BringToFront();

            }

            // Отрисовка лабиринта игрока на его панели
            private void PlayerMazePanel_Paint(object sender,
PaintEventArgs e)
            {
                // Очищаем панель
                e.Graphics.Clear(Color.White);

                // Отрисовываем лабиринт с небольшим отступом от
краев панели
                Point panelOffset = new Point(10, 10);
                maze.Draw(e.Graphics, cellSize, panelOffset);

                // Отрисовываем финиш
                DrawFinish(e.Graphics, panelOffset, finishPoint,
Color.Green);

                // Подпись для лабиринта игрока
                //DrawLabel(e.Graphics, "Игрок", panelOffset);
            }

            //private void BotMazePanel_Paint(object sender,
PaintEventArgs e)
            //{
            //    // Очищаем панель
            //    e.Graphics.Clear(Color.White);

            //    // Отрисовываем лабиринт с небольшим отступом от
краев панели
            //    Point panelOffset = new Point(10, 10);
            //    mazeForBot.Draw(e.Graphics, cellSize, panelOffset);
```

```
                //    // Отрисовываем финиш
                //    DrawFinish(e.Graphics, panelOffset,
finishPointForBot, Color.Blue);

                //    // Подпись для лабиринта бота
                //    //DrawLabel(e.Graphics, "Бот", panelOffset);
                //}

                private void DrawFinish(Graphics graphics, Point offset,
Point finish, Color color)
                {
                    int x = finish.X * cellSize + offset.X + 5;
                    int y = finish.Y * cellSize + offset.Y + 5;
                    int size = cellSize - 10;

                    using (Brush brush = new SolidBrush(color))
                    {
                        graphics.FillRectangle(brush, x, y, size, size);
                    }
                }

                private void DrawLabel(Graphics graphics, string text,
Point offset)
                {
                    Font labelFont = new Font("Arial", 12,
FontStyle.Bold);
                    SizeF textSize = graphics.MeasureString(text,
labelFont);

                    PointF labelPos = new PointF(
                        250, 500
                    );

                    graphics.DrawString(text, labelFont, Brushes.Black,
labelPos);
                }

                private void Exit_clic(object sender, EventArgs e)
                {
                    this.Close();
                }
```

```
        private void GameForm_KeyDown(object sender, KeyEventArgs
e)
        {
            int deltaX = 0, deltaY = 0;

            switch (e.KeyCode)
            {
                case Keys.W: deltaY = -player.Speed; break;
                case Keys.S: deltaY = player.Speed; break;
                case Keys.A: deltaX = -player.Speed; break;
                case Keys.D: deltaX = player.Speed; break;
                case Keys.Escape: this.Close(); break;
            }

            if (!WillCollide(deltaX, deltaY))
            {
                player.Move(deltaX, deltaY);
                CheckWinCondition();
            }

            // Обновляем отрисовку панели игрока
            playerMazePanel.Invalidate();
        }

        private bool WillCollide(int deltaX, int deltaY)
        {
            Rectangle futureBounds = new Rectangle(
                player.PictureBox.Left + deltaX,
                player.PictureBox.Top + deltaY,
                player.PictureBox.Width,
                player.PictureBox.Height
            );

            // Проверяем границы панели игрока
            if (!playerMazePanel.Bounds.Contains(futureBounds))
                return true;

            // Проверяем столкновение с каждой клеткой лабиринта
            for (int x = 0; x < maze.Width; x++)
            {
                for (int y = 0; y < maze.Height; y++)
                {
                    Cell cell = maze.Cells[x, y];
```

```
                            if (CheckCellCollision(cell, futureBounds,
playerMazePanel.Location))
                                return true;
                        }
                    }

                    return false;
                }

                private bool CheckCellCollision(Cell cell, Rectangle
playerBounds, Point panelLocation)
                {
                    // Смещение внутри панели
                    Point panelOffset = new Point(10, 10);
                    int cellX = cell.X * cellSize + panelLocation.X +
panelOffset.X;
                    int cellY = cell.Y * cellSize + panelLocation.Y +
panelOffset.Y;

                    // Создаем прямоугольники для каждой стены
                    if (cell.topWall)
                    {
                        Rectangle topWall = new Rectangle(cellX, cellY,
cellSize, 2);
                        if (playerBounds.IntersectsWith(topWall))
                            return true;
                    }

                    if (cell.downWall)
                    {
                        Rectangle downWall = new Rectangle(cellX, cellY +
cellSize - 2, cellSize, 2);
                        if (playerBounds.IntersectsWith(downWall))
                            return true;
                    }

                    if (cell.leftWall)
                    {
                        Rectangle leftWall = new Rectangle(cellX, cellY,
2, cellSize);
                        if (playerBounds.IntersectsWith(leftWall))
                            return true;
                    }
```

```
                    if (cell.rightWall)
                    {
                        Rectangle rightWall = new Rectangle(cellX +
cellSize - 2, cellY, 2, cellSize);
                        if (playerBounds.IntersectsWith(rightWall))
                            return true;
                    }

                    return false;
                }

                private void CheckWinCondition()
                {
                    // Смещение внутри панели
                    Point panelOffset = new Point(10, 10);

                    // Определяем клетку, в которой находится игрок
                    int playerCellX = (player.PictureBox.Left -
playerMazePanel.Location.X - panelOffset.X + cellSize / 2) / cellSize;
                    int playerCellY = (player.PictureBox.Top -
playerMazePanel.Location.Y - panelOffset.Y + cellSize / 2) / cellSize;

                    // Проверяем, дошел ли игрок до финиша
                    if (playerCellX == finishPoint.X && playerCellY ==
finishPoint.Y && !playerFinished)
                    {
                        playerFinished = true;
                        playerTimeSeconds = gameTimer.GetElapsedSeconds();
                        gameTimer.StopTimer();

                        // Останавливаем движение игрока
                        player.PictureBox.Enabled = false;

                        // Отладочное сообщение
                        Debug.WriteLine($"Игрок дошел до финиша! Время:
{playerTimeSeconds} сек");

                        // Показываем сообщение, что игрок дошел до
финиша
                        ShowTemporaryMessage($"Игрок дошел до
финиша!\nВремя:
{TimeSpan.FromSeconds(playerTimeSeconds):mm\\:ss\\.fff}");
```

```
                // Проверяем, дошел ли уже бот
                CheckIfBothFinished();
            }
        }

        private void GameForm_Load(object sender, EventArgs e)
        {

        }

        private void GameForm_Resize(object sender, EventArgs e)
        {
            // При изменении размера формы пересчитываем позиции
панелей
            if (playerMazePanel != null && botMazePanel != null)
            {
                int panelSize = ComplexityMaze * cellSize + 20;

                playerMazePanel.Location = new Point(
                    (this.ClientSize.Width / 2 - panelSize) / 2,
                    (this.ClientSize.Height - panelSize) / 2
                );

                botMazePanel.Location = new Point(
                    this.ClientSize.Width / 2 +
(this.ClientSize.Width / 2 - panelSize) / 2,
                    (this.ClientSize.Height - panelSize) / 2
                );

                Control exitButton =
this.Controls.OfType<Button>().FirstOrDefault();
                if (exitButton != null)
                {
                    exitButton.Location = new
Point(this.ClientSize.Width - 120, 20);
                }
            }
        }
        public class DoubleBufferedPanel : Panel
        {
            public DoubleBufferedPanel()
            {
```

```
                        this.DoubleBuffered = true;
                        this.SetStyle(ControlStyles.AllPaintingInWmPaint
|
                                      ControlStyles.UserPaint |
                                      ControlStyles.OptimizedDoubleBuffer,
true);
                    }
                }
                private void SaveGame(string filePath)
                {
                    try
                    {
                        var save = new GameSave
                        {
                            Complexity = ComplexityMaze,
                            SaveDate = DateTime.Now,

                            // Сохраняем позицию игрока в клетках
                            PlayerPosition = new Point(
                                (player.PictureBox.Left -
playerMazePanel.Location.X - 10) / cellSize,
                                (player.PictureBox.Top -
playerMazePanel.Location.Y - 10) / cellSize
                            ),

                            // Сохраняем позицию бота в клетках
                            BotPosition = new Point(
                                (bot.PictureBox.Left -
botMazePanel.Location.X - 10) / cellSize,
                                (bot.PictureBox.Top -
botMazePanel.Location.Y - 10) / cellSize
                            ),

                            PlayerFinished = playerFinished,
                            BotFinished = botFinished,
                            PlayerTimeSeconds = playerTimeSeconds,
                            BotTimeMs = botTimeMs,
                            ElapsedSeconds =
gameTimer?.GetElapsedSeconds() ?? 0,
                            IsBotMoving = bot?.IsMoving ?? false
                        };

                        // Сохраняем стены лабиринта как список
```

```
                    save.MazeWalls = new List<CellWalls>();
                    for (int x = 0; x < maze.Width; x++)
                    {
                        for (int y = 0; y < maze.Height; y++)
                        {
                            var cell = maze.Cells[x, y];
                            save.MazeWalls.Add(new CellWalls
                            {
                                X = x,
                                Y = y,
                                TopWall = cell.topWall,
                                RightWall = cell.rightWall,
                                BottomWall = cell.downWall,
                                LeftWall = cell.leftWall
                            });
                        }
                    }

                    // Сериализуем в JSON
                    var json = JsonSerializer.Serialize(save, new
JsonSerializerOptions { WriteIndented = true });
                    File.WriteAllText(filePath, json);

                    MessageBox.Show("Игра сохранена!", "Сохранение",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Ошибка сохранения:
{ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }

            private void LoadGame(string filePath)
            {
                try
                {
                    if (!File.Exists(filePath))
                    {
                        MessageBox.Show("Файл сохранения не найден!",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                        return;
                    }
```

```
                        var json = File.ReadAllText(filePath);
                        var save =
JsonSerializer.Deserialize<GameSave>(json);

                        if (save == null)
                        {
                            MessageBox.Show("Неверный формат файла
сохранения!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                            return;
                        }

                        // Останавливаем и удаляем старого бота полностью
                        if (bot != null)
                        {
                            bot.StopMoving();
                            bot.PictureBox.Dispose();
                            bot = null;
                        }

                        // Восстанавливаем сложность
                        ComplexityMaze = save.Complexity;

                        // Пересоздаем лабиринты
                        InitializeMaze();

                        // Восстанавливаем стены из списка
                        foreach (var cellWalls in save.MazeWalls)
                        {
                            if (cellWalls.X < maze.Width && cellWalls.Y <
maze.Height)
                            {
                                var cell = maze.Cells[cellWalls.X,
cellWalls.Y];
                                cell.topWall = cellWalls.TopWall;
                                cell.rightWall = cellWalls.RightWall;
                                cell.downWall = cellWalls.BottomWall;
                                cell.leftWall = cellWalls.LeftWall;

                                // Копируем в лабиринт бота
                                var botCell =
mazeForBot.Cells[cellWalls.X, cellWalls.Y];
                                botCell.topWall = cellWalls.TopWall;
```

```
                        botCell.rightWall = cellWalls.RightWall;
                        botCell.downWall = cellWalls.BottomWall;
                        botCell.leftWall = cellWalls.LeftWall;
                    }
                }

                // Восстанавливаем позицию игрока
                player.PictureBox.Location = new Point(
                    playerMazePanel.Location.X + 10 +
save.PlayerPosition.X * cellSize + cellSize / 4,
                    playerMazePanel.Location.Y + 10 +
save.PlayerPosition.Y * cellSize + cellSize / 4
                );

                // Восстанавливаем состояние игры
                playerFinished = save.PlayerFinished;
                botFinished = save.BotFinished;
                playerTimeSeconds = save.PlayerTimeSeconds;
                botTimeMs = save.BotTimeMs;
                resultsShown = false;

                // Восстанавливаем таймер
                if (gameTimer != null)
                {
                    gameTimer.StopTimer();
                    // Здесь нужно установить время из сохранения
                    // Добавьте в TimerForGame метод SetTime:
                    // gameTimer.SetTime(save.ElapsedSeconds);
                }

                // Теперь создаем нового бота с правильной
позицией
                // Нужно изменить InitializeBot, чтобы он
принимал начальную позицию
                InitializeBotWithPosition(save.BotPosition.X,
save.BotPosition.Y);

                if (save.IsBotMoving && !botFinished)
                {
                    bot.StartMoving();
                }

                // Обновляем отрисовку
```

```
                    playerMazePanel.Invalidate();
                    botMazePanel.Invalidate();

                    MessageBox.Show($"Игра загружена!\nСохранено:
{save.SaveDate}", "Загрузка",
                        MessageBoxButtons.OK,
MessageBoxIcon.Information);
                }
                catch (Exception ex)
                {
                    MessageBox.Show($"Ошибка загрузки: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
            private void InitializeBotWithPosition(int startCellX,
int startCellY)
            {
                if (mazeForBot == null) return;

                // Создаем бота с лабиринтом для бота
                Point botMazeOffset = new Point(10, 10);
                bot = new Bot(mazeForBot, cellSize, botMazeOffset);

                // Устанавливаем начальную позицию бота
                bot.PictureBox.Location = new Point(
                    botMazePanel.Location.X + botMazeOffset.X +
startCellX * cellSize + cellSize / 4,
                    botMazePanel.Location.Y + botMazeOffset.Y +
startCellY * cellSize + cellSize / 4
                );

                // Добавляем бота на панель бота
                botMazePanel.Controls.Add(bot.PictureBox);
                bot.PictureBox.BringToFront();

                // Пересчитываем путь от текущей позиции до финиша
                RecalculateBotPathFromPosition(startCellX,
startCellY);

                bot.OnBotFinished += (s, e) =>
                {
                    botFinished = true;
                    botTimeMs = bot.GetElapsedTimeMs();
```

```
                        Debug.WriteLine($"Бот дошел до финиша! Время:
{botTimeMs} мс");
                        ShowTemporaryMessage($"Бот дошел до
финиша!\nВремя: {TimeSpan.FromMilliseconds(botTimeMs):mm\\:ss\\.fff}");
                        CheckIfBothFinished();
                    };
                }

                private void RecalculateBotPathFromPosition(int
startCellX, int startCellY)
                {
                    // Пересчитываем путь от сохраненной позиции до
финиша
                    var path = WaveAlgorithm.FindPath(mazeForBot,
                        new Point(startCellX, startCellY),
                        new Point(mazeForBot.Width - 1, mazeForBot.Height
- 1));

                    // Здесь нужно обновить путь бота
                    // Вам нужно добавить метод в класс Bot для установки
нового пути:
                    // bot.SetPath(path);
                }
            }
        }
```

## MainMenu.cs:

```
using WinFormsApp1.Class;

namespace WinFormsApp1
{
    public partial class MainMenu : Form
    {
        private bool isMenuVisible = false;
        private Panel menuPanel;
        private Player player;
        int complexity = 10;
        private TimerForGame gameTimer;

        public MainMenu()
        {
            //инициализация всего
            InitializeComponent();
```

```
            InitializePlayer();
            SpawnButton();
            InitializeMenuPanel();


            this.DoubleBuffered = true;
            this.KeyDown += MainMenuForm_KeyDown;
            this.BackColor = Color.Black;

            CreateTimerObj();
        }

        public void CreateTimerObj()
        {
            gameTimer = new TimerForGame(this.ClientSize, this);
            gameTimer.CreateTimer();
        }

        private void SpawnButton()
        {
            this.WindowState = FormWindowState.Maximized;
            this.FormBorderStyle = FormBorderStyle.None;
            this.KeyPreview = true;
            this.BackColor = Color.Black;

            Button startButton = new Button();
            startButton.Text = "Start";
            startButton.Size = new Size(
                (int)(this.ClientSize.Width * 0.30),
                (int)(this.ClientSize.Height * 0.10)
            );
            startButton.Location = new Point(
                (this.ClientSize.Width - startButton.Width) / 2,
                (int)(this.ClientSize.Height * 0.4)
            );
            startButton.BackColor = Color.DarkBlue;
            startButton.ForeColor = Color.White;
            float fontSize = startButton.Height * 0.3f;
            startButton.Font = new Font("Arial", fontSize,
FontStyle.Bold);
            startButton.FlatStyle = FlatStyle.Flat;
            startButton.FlatAppearance.BorderSize = 2;
            startButton.FlatAppearance.BorderColor = Color.White;
```

```
            startButton.Click += StartGame;
            this.Controls.Add(startButton);

            Button exitButton = new Button();
            exitButton.Text = "Exit";
            exitButton.Size = startButton.Size;
            exitButton.Location = new Point(
                (this.ClientSize.Width - exitButton.Width) / 2,
                (int)(this.ClientSize.Height * 0.55)
            );
            exitButton.BackColor = Color.DarkRed;
            exitButton.ForeColor = Color.White;
            exitButton.Font = new Font("Arial", fontSize,
FontStyle.Bold);
            exitButton.FlatStyle = FlatStyle.Flat;
            exitButton.FlatAppearance.BorderSize = 2;
            exitButton.FlatAppearance.BorderColor = Color.White;

            exitButton.Click += Exit_clic;
            this.Controls.Add(exitButton);

            Button complexityChoice = new Button();
            complexityChoice.Text = "Complexity";
            complexityChoice.Size = startButton.Size;
            complexityChoice.Location = new Point(
                (this.ClientSize.Width - exitButton.Width) / 2 +
200,
                (int)(this.ClientSize.Height * 0.55) + 200
            );
            complexityChoice.BackColor = Color.DarkRed;
            complexityChoice.ForeColor = Color.White;
            complexityChoice.Font = new Font("Arial", fontSize,
FontStyle.Bold);
            complexityChoice.FlatStyle = FlatStyle.Flat;
            complexityChoice.FlatAppearance.BorderSize = 2;
            complexityChoice.FlatAppearance.BorderColor =
Color.White;
            complexityChoice.Click += ComplexityChoice;
            this.Controls.Add(complexityChoice);
        }

        private void InitializeMenuPanel()
        {
```

```
                    // Создаем панель меню
                    menuPanel = new Panel();
                    menuPanel.Size = new Size(300, 200);
                    menuPanel.Location = new Point(
                        (this.ClientSize.Width - 300) / 2,
                        (this.ClientSize.Height - 200) / 2
                    );
                    menuPanel.BackColor = Color.DarkSlateBlue;
                    menuPanel.BorderStyle = BorderStyle.FixedSingle;
                    menuPanel.Visible = false; // Изначально скрыта

                    // Добавляем заголовок
                    Label titleLabel = new Label();
                    titleLabel.Text = "Choice";
                    titleLabel.Font = new Font("Arial", 14,
FontStyle.Bold);
                    titleLabel.ForeColor = Color.White;
                    titleLabel.Size = new Size(280, 30);
                    titleLabel.Location = new Point(10, 10);
                    titleLabel.TextAlign = ContentAlignment.MiddleCenter;
                    menuPanel.Controls.Add(titleLabel);

                    // Кнопка "Легко"
                    Button easyButton = new Button();
                    easyButton.Text = "(5x5)";
                    easyButton.Size = new Size(200, 40);
                    easyButton.Location = new Point(50, 50);
                    easyButton.BackColor = Color.Green;
                    easyButton.ForeColor = Color.White;
                    easyButton.Font = new Font("Arial", 10,
FontStyle.Bold);
                    easyButton.Click += (s, e) =>
                    {
                        complexity = 5;
                        HideMenu();
                        MessageBox.Show("5 X 5!");
                    };
                    menuPanel.Controls.Add(easyButton);

                    // Кнопка "Средне"
                    Button mediumButton = new Button();
                    mediumButton.Text = "(10x10)";
                    mediumButton.Size = new Size(200, 40);
```

```
                    mediumButton.Location = new Point(50, 100);
                    mediumButton.BackColor = Color.Orange;
                    mediumButton.ForeColor = Color.White;
                    mediumButton.Font = new Font("Arial", 10,
FontStyle.Bold);
                    mediumButton.Click += (s, e) =>
                    {
                        complexity = 10;
                        HideMenu();
                        MessageBox.Show("10 X 10!");
                    };
                    menuPanel.Controls.Add(mediumButton);

                    // Кнопка "Сложно"
                    Button hardButton = new Button();
                    hardButton.Text = "(15x15)";
                    hardButton.Size = new Size(200, 40);
                    hardButton.Location = new Point(50, 150);
                    hardButton.BackColor = Color.Red;
                    hardButton.ForeColor = Color.White;
                    hardButton.Font = new Font("Arial", 10,
FontStyle.Bold);
                    hardButton.Click += (s, e) =>
                    {
                        complexity = 15;
                        HideMenu();
                        MessageBox.Show("15 X 15!");
                    };
                    menuPanel.Controls.Add(hardButton);

                    // Кнопка закрытия меню
                    Button closeButton = new Button();
                    closeButton.Text = "?";
                    closeButton.Size = new Size(30, 30);
                    closeButton.Location = new Point(260, 5);
                    closeButton.BackColor = Color.DarkRed;
                    closeButton.ForeColor = Color.White;
                    closeButton.Font = new Font("Arial", 12,
FontStyle.Bold);
                    closeButton.Click += (s, e) => HideMenu();
                    menuPanel.Controls.Add(closeButton);

                    this.Controls.Add(menuPanel);
```

```
        menuPanel.BringToFront();
    }

    private void ShowMenu()
    {
        isMenuVisible = true;
        menuPanel.Visible = true;
        menuPanel.BringToFront();

        // Затемняем фон
        foreach (Control control in this.Controls)
        {
            if (control != menuPanel)
                control.Enabled = false;
        }
    }

    private void HideMenu()
    {
        isMenuVisible = false;
        menuPanel.Visible = false;

        // Восстанавливаем фон
        foreach (Control control in this.Controls)
        {
            control.Enabled = true;
        }
    }

    private void ComplexityChoice(object sender, EventArgs e)
    {
        ShowMenu();
    }

    private void StartGame(object sender, EventArgs eventArgs)
    {
        GameForm gameForm = new GameForm(complexity);
        var result = gameForm.ShowDialog();

        if (result == DialogResult.Retry)
        {
            // Пользователь хочет новую игру
            StartGame(sender, eventArgs);
```

```
            }
            else
            {
                this.Show(); // Возвращаемся в меню
            }
        }
        private void Exit_clic(object srnder, EventArgs e)
        {
            this.Close();
        }

        private void InitializePlayer()
        {
            player = new Player();
            this.Controls.Add(player.PictureBox);
            player.PictureBox.BringToFront(); // Чтобы был поверх
других элементов
            ResizePlayer(); // Первоначальная установка размера

        }
        private void ResizePlayer()
        {
            int playerSize = (int)(this.ClientSize.Width * 0.08);
            player.PictureBox.Size = new Size(playerSize,
playerSize);

            player.PictureBox.Location = new Point(
                (int)(this.ClientSize.Width * 0.1),
                (int)(this.ClientSize.Height * 0.8)
            );
        }

        private void MainMenuForm_KeyDown(object sender,
KeyEventArgs e)
        {
            int deltaX = 0, deltaY = 0;

            switch (e.KeyCode)
            {
                case Keys.W: deltaY = -player.Speed; break;
                case Keys.S: deltaY = player.Speed; break;
                case Keys.A: deltaX = -player.Speed; break;
                case Keys.D: deltaX = player.Speed; break;
```

```
            }

            player.Move(deltaX, deltaY);
            player.CheckBoundaries(this);
        }

        private void MainMenu_Load(object sender, EventArgs e)
        {

        }
    }

}
```

## ResultForm.cs:

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace WinFormsApp1
{
    public partial class ResultForm : Form
    {
        public TimeSpan PlayerTime { get; set; }
        public TimeSpan BotTime { get; set; }
        public string Winner { get; set; } = "";
        public Color WinnerColor { get; set; } = Color.Gray;

        private Label playerTimeLabel;
        private Label botTimeLabel;
        private Label diffLabel;
        private Label winnerLabel;

        public ResultForm()
        {
            InitializeForm();
            SetupControls();
        }

        private void InitializeForm()
        {
            // Настройки формы
            this.Text = "Результаты игры";
```

```
            this.Size = new Size(600, 600);
            this.StartPosition = FormStartPosition.CenterScreen;
            this.BackColor = Color.FromArgb(240, 240, 240);
            this.FormBorderStyle = FormBorderStyle.FixedDialog;
            this.MaximizeBox = false;
            this.MinimizeBox = false;
            this.AutoScaleMode = AutoScaleMode.Font;
        }

        private void SetupControls()
        {
            // Заголовок
            Label titleLabel = new Label();
            titleLabel.Text = "ИТОГИ ГОНКИ";
            titleLabel.Font = new Font("Arial", 24,
FontStyle.Bold);
            titleLabel.ForeColor = Color.DarkBlue;
            titleLabel.Size = new Size(400, 50);
            titleLabel.Location = new Point(25, 20);
            titleLabel.TextAlign = ContentAlignment.MiddleCenter;
            this.Controls.Add(titleLabel);

            // Панель для результатов игрока
            Panel playerPanel = new Panel();
            playerPanel.Size = new Size(400, 70);
            playerPanel.Location = new Point(25, 90);
            playerPanel.BackColor = Color.LightBlue;
            playerPanel.BorderStyle = BorderStyle.FixedSingle;
            this.Controls.Add(playerPanel);

            Label playerTitle = new Label();
            playerTitle.Text = "ИГРОК";
            playerTitle.Font = new Font("Arial", 14,
FontStyle.Bold);
            playerTitle.ForeColor = Color.DarkBlue;
            playerTitle.Size = new Size(100, 30);
            playerTitle.Location = new Point(10, 20);
            playerTitle.TextAlign = ContentAlignment.MiddleLeft;
            playerPanel.Controls.Add(playerTitle);

            playerTimeLabel = new Label();
            playerTimeLabel.Font = new Font("Consolas", 18,
FontStyle.Bold);
```

```
                    playerTimeLabel.ForeColor = Color.Black;
                    playerTimeLabel.Size = new Size(200, 30);
                    playerTimeLabel.Location = new Point(180, 20);
                    playerTimeLabel.TextAlign =
ContentAlignment.MiddleRight;
                    playerPanel.Controls.Add(playerTimeLabel);

                    // Панель для результатов бота
                    Panel botPanel = new Panel();
                    botPanel.Size = new Size(400, 70);
                    botPanel.Location = new Point(25, 170);
                    botPanel.BackColor = Color.LightCoral;
                    botPanel.BorderStyle = BorderStyle.FixedSingle;
                    this.Controls.Add(botPanel);

                    Label botTitle = new Label();
                    botTitle.Text = "БОТ";
                    botTitle.Font = new Font("Arial", 14, FontStyle.Bold);
                    botTitle.ForeColor = Color.DarkRed;
                    botTitle.Size = new Size(100, 30);
                    botTitle.Location = new Point(10, 20);
                    botTitle.TextAlign = ContentAlignment.MiddleLeft;
                    botPanel.Controls.Add(botTitle);

                    botTimeLabel = new Label();
                    botTimeLabel.Font = new Font("Consolas", 18,
FontStyle.Bold);
                    botTimeLabel.ForeColor = Color.Black;
                    botTimeLabel.Size = new Size(200, 30);
                    botTimeLabel.Location = new Point(180, 20);
                    botTimeLabel.TextAlign = ContentAlignment.MiddleRight;
                    botPanel.Controls.Add(botTimeLabel);

                    // Разница во времени
                    diffLabel = new Label();
                    diffLabel.Font = new Font("Arial", 12,
FontStyle.Bold);
                    diffLabel.Size = new Size(400, 30);
                    diffLabel.Location = new Point(25, 250);
                    diffLabel.TextAlign = ContentAlignment.MiddleCenter;
                    this.Controls.Add(diffLabel);

                    // Победитель
```

```
                winnerLabel = new Label();
                winnerLabel.Font = new Font("Arial", 16,
FontStyle.Bold);
                winnerLabel.Size = new Size(400, 40);
                winnerLabel.Location = new Point(25, 290);
                winnerLabel.TextAlign = ContentAlignment.MiddleCenter;
                this.Controls.Add(winnerLabel);

                // Кнопка новой игры
                Button newGameButton = new Button();
                newGameButton.Text = "НОВАЯ ИГРА";
                newGameButton.Size = new Size(150, 40);
                newGameButton.Location = new Point(75, 340);
                newGameButton.BackColor = Color.Green;
                newGameButton.ForeColor = Color.White;
                newGameButton.Font = new Font("Arial", 12,
FontStyle.Bold);
                newGameButton.FlatStyle = FlatStyle.Flat;
                newGameButton.Click += (s, e) =>
                {
                    this.DialogResult = DialogResult.Retry;
                    this.Close();
                };
                this.Controls.Add(newGameButton);

                // Кнопка выхода в меню
                Button menuButton = new Button();
                menuButton.Text = "В МЕНЮ";
                menuButton.Size = new Size(150, 40);
                menuButton.Location = new Point(225, 340);
                menuButton.BackColor = Color.Blue;
                menuButton.ForeColor = Color.White;
                menuButton.Font = new Font("Arial", 12,
FontStyle.Bold);
                menuButton.FlatStyle = FlatStyle.Flat;
                menuButton.Click += (s, e) =>
                {
                    this.DialogResult = DialogResult.OK;
                    this.Close();
                };
                this.Controls.Add(menuButton);

                // Обновляем данные после создания элементов
```

```
            this.Load += (s, e) => UpdateResults();
        }

        private void UpdateResults()
        {
            // Обновляем время игрока
            playerTimeLabel.Text =
PlayerTime.ToString(@"mm\:ss\.fff");

            // Обновляем время бота
            botTimeLabel.Text = BotTime.ToString(@"mm\:ss\.fff");

            // Разница во времени
            TimeSpan difference = PlayerTime - BotTime;
            string diffText = "";
            Color diffColor = Color.Black;

            if (difference.TotalMilliseconds > 0)
            {
                diffText = $"Бот быстрее на:
{difference:mm\\:ss\\.fff}";
                diffColor = Color.DarkRed;
            }
            else if (difference.TotalMilliseconds < 0)
            {
                diffText = $"Игрок быстрее на: {-
difference:mm\\:ss\\.fff}";
                diffColor = Color.DarkBlue;
            }
            else
            {
                diffText = "Время одинаковое!";
                diffColor = Color.DarkGreen;
            }

            diffLabel.Text = diffText;
            diffLabel.ForeColor = diffColor;

            // Победитель
            string winnerText = string.IsNullOrEmpty(Winner) ?
"ПОБЕДИТЕЛЬ НЕ ОПРЕДЕЛЕН" : $"ПОБЕДИТЕЛЬ: {Winner.ToUpper()}";
            winnerLabel.Text = winnerText;
            winnerLabel.ForeColor = WinnerColor;
```

```
                }
            }
        }
```

## Структура проекта: