

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Пензенский государственный университет»

Кафедра «Вычислительная техника»

### ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по курсу «Программирование»

на тему «Разработка программы сложной структуры методом нисходящего  
программирования. Игра Лабиринт»

28.05.2026  
Отлично  
Генералова А.А.

Выполнил:

студент группы 25BBB2

Перетрухин О. С.

Принял:

к.т.н., доцент Генералова А.А.

**ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет Вычислительной техники**

**Кафедра "Вычислительная техника"**

**"УТВЕРЖДАЮ"**

Зав. кафедрой ВТ

 М.А. Митрохин

« 19 » февраля 2016 г

**ЗАДАНИЕ**

**на курсовое проектирование по курсу**

**Программирование**

Студенту Перетрухину Олегу Сергеевичу Группа 25ВВВ2

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

Исходные данные (технические требования) на проектирование

*Игра "Лабиринт". Компьютер генерирует лабиринт для игры. Пользователь пытается пройти лабиринт от начала до конца. Файл сохраняет таблицу ресурсов.*

**Обязательные требования к программе:**

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

## Объем работы по курсу

### 1. Расчетная часть

Разработка программы.

### 2. Графическая часть

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

### 3. Экспериментальная часть

Отладка программы.

## Срок выполнения проекта по разделам

В соответствии с графиком.

Дата выдачи задания "19" февраля 2026

Дата защиты проекта "28" мая 2026

Руководитель Т. Гензюбе А.П.

Задание получил "19" февраля 2026 г.

Студент Перегрухин Олег Сергеевич

## Содержание

|                                      |    |
|--------------------------------------|----|
| 1 Введение.....                      | 5  |
| 2 Постановка задачи.....             | 6  |
| 3 Выбор решения.....                 | 7  |
| 4 Описание разработки программы..... | 7  |
| 5 Отладка и тестирование .....       | 11 |
| 6 Описание программы.....            | 12 |
| 7 Руководство пользователя.....      | 25 |
| Заключение. ....                     | 30 |
| Список используемых источников.....  | 31 |
| Приложение А .....                   | 32 |
| Приложение В.....                    | 38 |

## **1 Введение**

Игра «Лабиринт» — классическая логико-головоломочная игра, известная с глубокой древности. Цель игрока проста и интуитивно понятна: пройти от входа до выхода в запутанной системе коридоров и тупиков. Несмотря на простую формулировку, такие игры тренируют пространственное мышление, память и навыки планирования маршрута, а также являются удобным учебным проектом, в котором сочетаются работа с двумерными структурами данных, генерация случайных конфигураций, обработка пользовательского ввода и сохранение результатов в файл.

Целью данной работы является разработка игры «Лабиринт» на языке C в среде Microsoft Visual Studio 2022. Программа должна автоматически генерировать новый лабиринт при каждом запуске, обеспечивать управление игроком с клавиатуры, отображать игровое поле в простом и наглядном двумерном представлении, измерять время прохождения и сохранять результаты в таблицу рекордов. Для демонстрации навыков низкоуровневого программирования два ключевых блока программы должны быть реализованы при помощи ассемблерных вставок.

## 2 Постановка задачи

Требуется разработать однопользовательскую консольную игру «Лабиринт» на языке C, состоящую из двух логических частей: главного меню и игрового поля.

В главном меню должны располагаться четыре пункта: начать новую игру, открыть таблицу рекордов, сбросить статистику и выйти из программы. Выбор осуществляется вводом цифры с клавиатуры. Также необходимо продумать, как программа будет вести себя при некорректном вводе пользователя, чтобы не происходило аварийное завершение.

После выбора начала игры программа должна запросить имя игрока, после чего сгенерировать новый, заведомо разрешимый лабиринт фиксированного размера (21 столбец на 15 строк) и вывести его в окно консоли. Лабиринт состоит из стен (символ «#») и проходов (символ пробела); игрок обозначается символом «Р», выход — символом «Е». Игрок начинает в верхнем левом углу, выход располагается в правом нижнем углу.

Управление производится клавишами W, A, S, D (вверх, влево, вниз, вправо). При каждом нажатии программа должна выполнить две проверки: проверить, не находится ли в желаемой ячейке стена (если стена — движение отменяется), и проверить, не достиг ли игрок выхода. Эти две проверки являются наиболее частыми операциями в игровом цикле, поэтому именно их разумно реализовать на языке ассемблера для демонстрации повышения скорости выполнения за счёт работы на низком уровне.

После завершения прохождения программа должна записать имя игрока и время прохождения в файл records.txt. В таблице рекордов отображаются пять лучших результатов, отсортированных по возрастанию времени. Также необходимо предусмотреть кнопку сброса всей статистики.

Программа должна разрабатываться в среде Microsoft Visual Studio 2022 в конфигурации Win32 (x86), так как именно в этой конфигурации поддерживается инлайн-ассемблер компилятора Microsoft C/C++ через директиву `__asm { }`.

### 3 Выбор решения

Разработанная программа представляет собой реализацию классической игры «Лабиринт» на языке C. Для её создания были применены следующие технологии и компоненты:

Язык программирования C и среда Visual Studio 2022 — программа написана на стандартном C с использованием API консоли Windows. Этот выбор обусловлен требованием задания, а также тем, что C позволяет напрямую вставлять ассемблерный код в исходный текст программы при помощи директивы `__asm` в конфигурации Win32 (x86).

Консольный 2D-вывод через Windows Console API — для отрисовки лабиринта используются функции `SetConsoleCursorPosition`, `SetConsoleTextAttribute` и `SetConsoleCursorInfo` из заголовочного файла `windows.h`. Такой подход позволяет сделать визуализацию простой и одновременно настоящей двумерной: каждая ячейка лабиринта — это одна символьная позиция в окне консоли, для разных типов ячеек используются разные цвета. Этот вариант гарантированно работает в Visual Studio 2022 без подключения сторонних библиотек.

Ассемблерные вставки на x86 (директива `__asm`) — для оптимизации двух наиболее часто выполняемых проверок (проверка столкновения со стеной и проверка достижения выхода) используются ассемблерные вставки. Это позволяет, во-первых, продемонстрировать навыки низкоуровневого программирования, и во-вторых, выполнять эти операции в полностью контролируемом разработчиком наборе машинных инструкций. Для корректной компиляции таких вставок проект должен быть настроен в конфигурации Win32 (x86), поскольку в x64 компилятор MSVC не поддерживает инлайн-ассемблер.

Алгоритм генерации лабиринта — для построения связного, заведомо разрешимого лабиринта применён алгоритм `recursive backtracker` (рекурсивный возврат).

Основные модули программы:

- Maze.c — единственный исходный файл программы, содержащий всю логику: главное меню, генерацию лабиринта, отрисовку, обработку пользовательского ввода, работу с файлом рекордов, а также две ассемблерные вставки CheckWallAsm и CheckExitAsm.

- records.txt — текстовый файл, в котором сохраняется информация о прохождении: имя игрока и время в секундах. Данные дописываются в конец файла, сортируются при отображении и выводятся в виде таблицы лучших пяти результатов.

## 4 Описание разработки программы

Разработанная программа состоит из нескольких логических блоков, которые разрабатывались в следующем порядке:

- ядро генерации и отрисовки лабиринта;
- блок управления игроком и игровой цикл;
- ассемблерные вставки `CheckWallAsm` и `CheckExitAsm`;
- работа с файлом рекордов;
- главное меню.

Первым этапом стала разработка алгоритма генерации лабиринта. Был выбран алгоритм `recursive backtracker` как наиболее простой в реализации и одновременно гарантирующий связность поля. Лабиринт хранится в статическом двумерном массиве `maze[HEIGHT][WIDTH]`. Изначально все ячейки заполняются стенами, затем рекурсивная функция `carveMaze` случайным образом «прорубает» проходы шагом 2 от стартовой клетки (1, 1) во все четыре направления, перебираемые в случайном порядке.

Вторым этапом стала разработка функции `drawMaze`, отвечающей за вывод лабиринта в окно консоли. Для исключения мерцания при каждой перерисовке курсор перемещается в верхний левый угол функцией `SetConsoleCursorPosition`, а сами символы вывода окрашиваются в разные цвета функцией `SetConsoleTextAttribute`: серый для стен, зелёный для игрока, красный для выхода.

Третьим этапом стала разработка двух ассемблерных вставок. Функция `CheckWallAsm` принимает указатель на начало массива лабиринта, координаты проверяемой ячейки (x, y) и ширину строки. Вставка вычисляет линейный индекс ячейки как  $y * \text{width} + x$  при помощи регистровых операций, считывает символ и сравнивает его со значением '#'. Функция `CheckExitAsm` выполняет ещё более простую операцию — последовательное сравнение пары координат игрока и пары координат выхода.

Четвёртым этапом стала реализация работы с файлом records.txt. Запись производится дописыванием в конец файла (режим «а»), чтение и сортировка — при отображении таблицы рекордов. Сортировка реализована методом пузырька, поскольку количество записей в учебной задаче невелико.

Завершающим этапом стало написание главного меню и склейка всех компонентов в единый игровой цикл с измерением времени прохождения через стандартную функцию time.

## 5 Отладка и тестирование

В качестве среды разработки была выбрана Microsoft Visual Studio 2022[5]. Программа обладает всеми средствами, необходимыми при разработке и отладке программы на языке C[2], включая встроенный отладчик с возможностью пошагового исполнения, окно регистров и окно дизассемблера, что особенно важно при отладке ассемблерных вставок.

Тестирование проводилось как в ходе разработки, так и после её завершения. В ходе тестирования были выявлены следующие классы проблем: переполнение стека при генерации лабиринта (решено увеличением размера стека по умолчанию и небольшим размером поля), мерцание изображения при перерисовке (решено отрисовкой через `SetConsoleCursorPosition` без вызова `system("cls")` в каждом кадре), некорректное чтение пользовательского ввода при использовании `scanf` совместно с `_getch` (решено сбросом буфера через цикл с `_kbhit`), а также ошибка компиляции ассемблерных вставок в конфигурации x64 (решено выбором конфигурации Win32).

Отдельно тестировалась корректность ассемблерных вставок[1]. Для `CheckWallAsm` проводилась проверка возвращаемого значения для всех четырёх соседних клеток вокруг игрока, а также для крайних клеток у границы лабиринта. Для `CheckExitAsm` проводилась проверка возвращаемого значения при совпадении и при различии координат. Все проверки прошли успешно.

## 6 Описание программы

Программа состоит из одного исходного файла Maze.c и одного файла данных records.txt. Логически программу можно разделить на две части: главное меню и игровое поле.

### 6.1 Главное меню

#### 6.1.1 Описание

Главное меню представляет собой консольное окно [6], в котором выводится заголовок «MAZE GAME» и четыре пункта меню: «1 — Start new game», «2 — Records table», «3 — Reset records», «4 — Exit». Выбор пункта осуществляется вводом соответствующей цифры с клавиатуры и нажатием Enter. После выполнения выбранного действия программа возвращается в главное меню ([рисунок 1](#)).

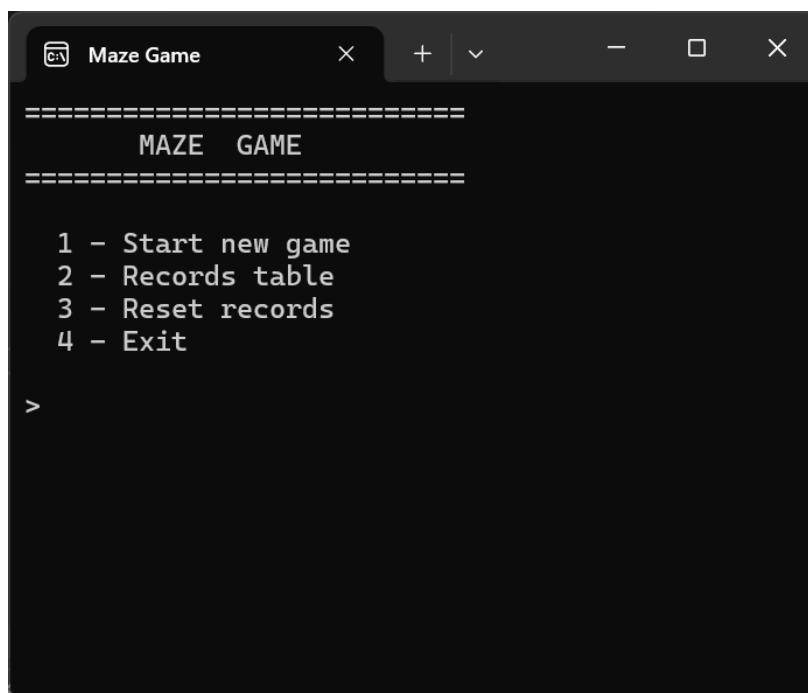


Рисунок 1 – Главное меню программы

При выборе пункта «Records table» программа открывает таблицу рекордов с пятью лучшими результатами, отсортированными по возрастанию времени прохождения. При выборе пункта «Reset records» файл records.txt удаляется, и таблица рекордов очищается. При выборе пункта «Exit» программа завершает работу.

## 6.1.2 Схема программы главного меню

Ниже представлены описания основных функций, отвечающих за работу главного меню, и их схемы.

### 6.1.2.1 Функция main

Функция main является точкой входа программы[3]. Она устанавливает заголовок окна консоли, инициализирует генератор случайных чисел, после чего запускает бесконечный цикл главного меню. В цикле выводятся пункты меню, считывается выбор пользователя и вызывается соответствующая функция: playGame, showRecords, resetRecords либо производится выход. Схема функции представлена на [рисунке 2](#).

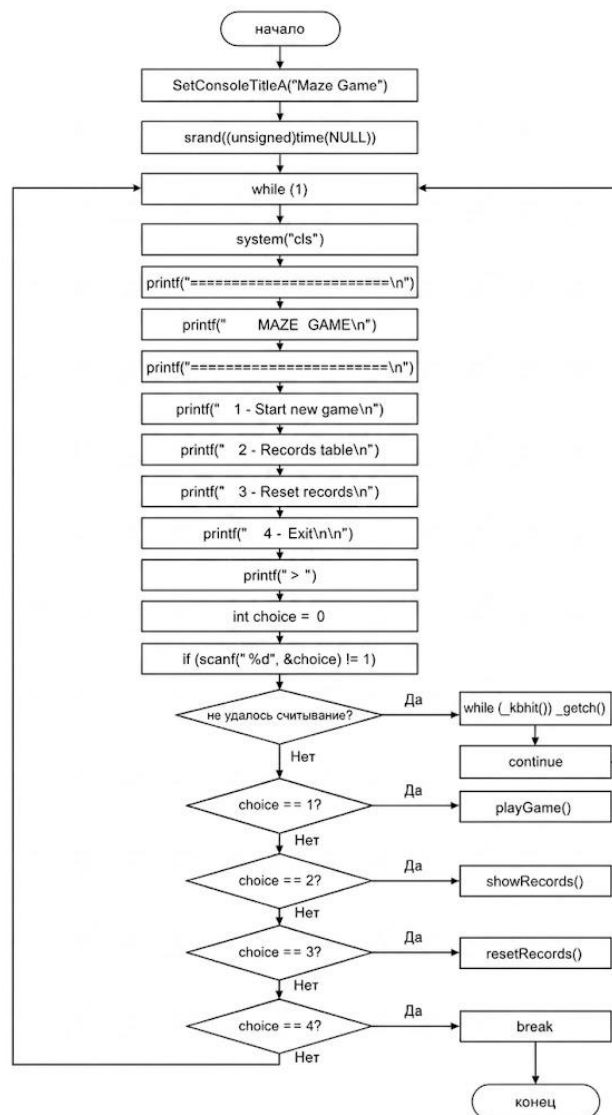


Рисунок 2 – Схема функции main

### 6.1.2.2 Функция showRecords

Функция showRecords отвечает за отображение таблицы рекордов. Она вызывает функцию loadRecords для чтения всех записей из файла records.txt, сортирует их по возрастанию времени прохождения методом пузырьковой сортировки, после чего выводит первые пять записей в виде нумерованного списка. Если файл рекордов отсутствует или пуст, выводится сообщение об отсутствии записей. Схема функции представлена на [рисунке 3](#).

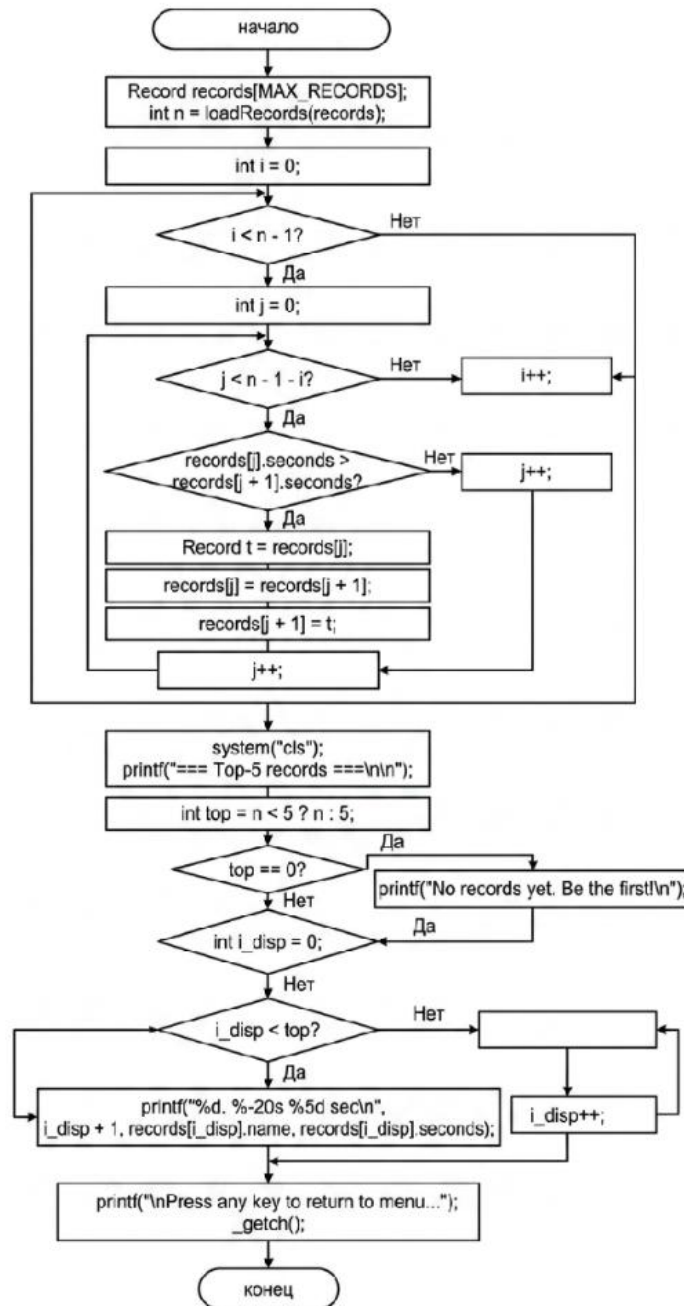


Рисунок 3 – Схема функции showRecords

### 6.1.2.3 Функция resetRecords

Функция resetRecords удаляет файл records.txt с помощью стандартной функции remove, после чего выводит сообщение об успешном сбросе статистики и ожидает нажатия клавиши для возврата в главное меню. Схема функции представлена на [рисунке 4](#).

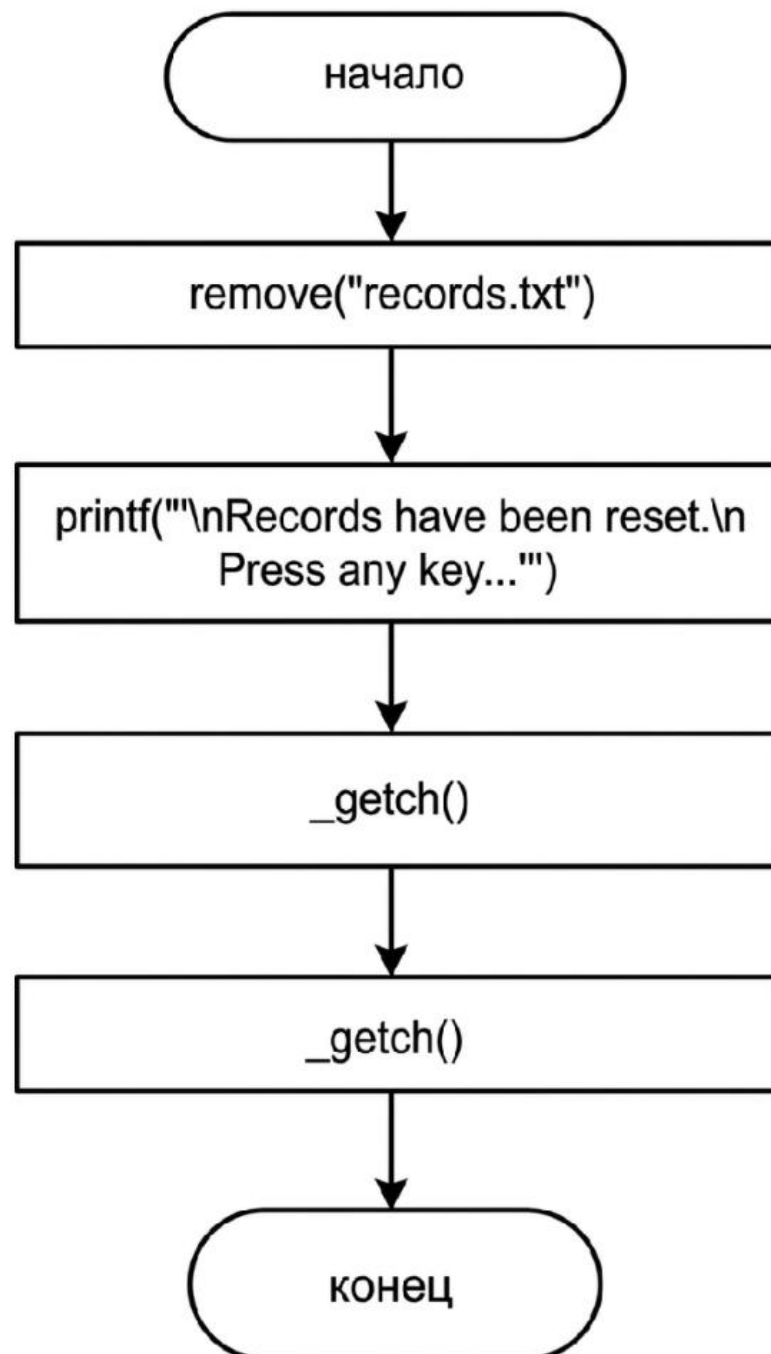


Рисунок 4 – Схема функции resetRecords

#### 6.1.2.4 Функция loadRecords

Функция loadRecords открывает файл records.txt в режиме чтения. Если файл отсутствует, возвращается ноль. В противном случае в цикле производится чтение пар «имя — время» при помощи fscanf, и каждая прочитанная запись помещается в массив structure Record. Функция возвращает количество прочитанных записей. Схема функции представлена на [рисунке 5](#).

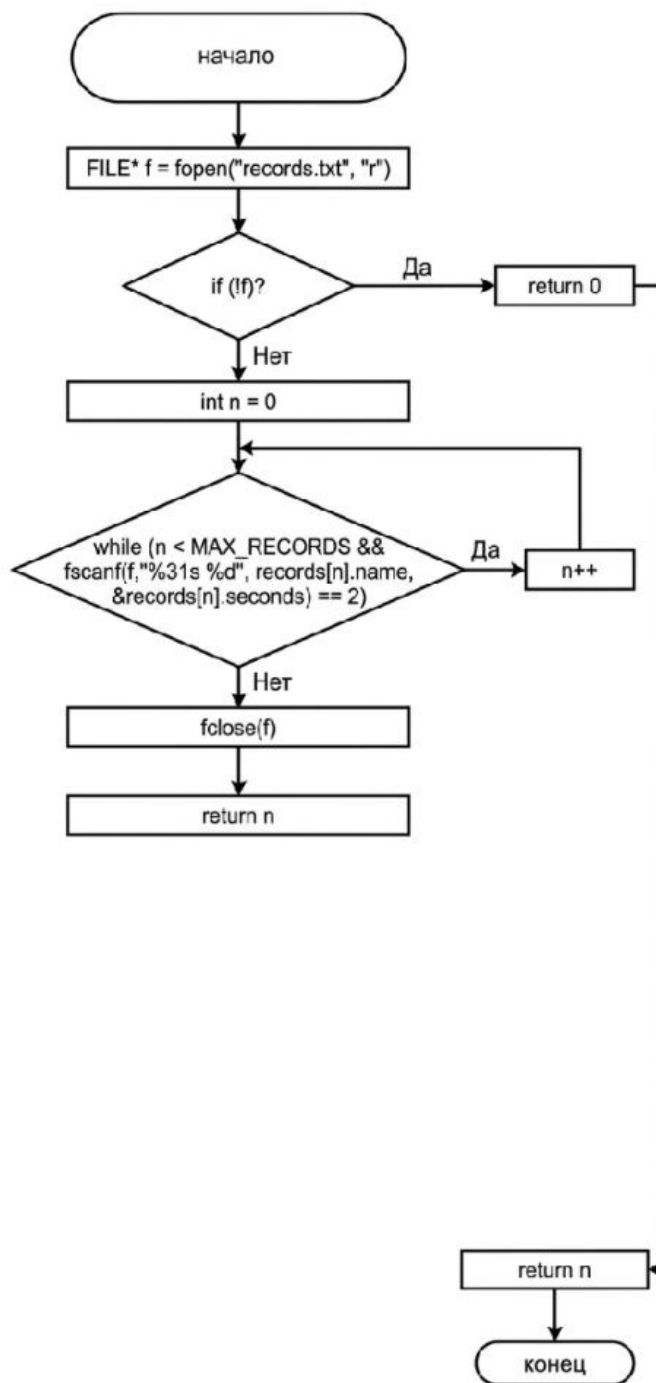


Рисунок 5 – Схема функции loadRecords

## 6.2 Игровое поле

### 6.2.1 Описание

На игровом поле в окне консоли отрисовывается двумерный лабиринт размером 21 на 15 ячеек. Стены отображаются серым символом «#», проходы — пробелами, игрок — зелёной буквой «Р», выход — красной буквой «Е». Под полем выводится строка с текущим временем прохождения и подсказкой по управлению. Внешний вид игрового поля представлен на [рисунке 6](#).

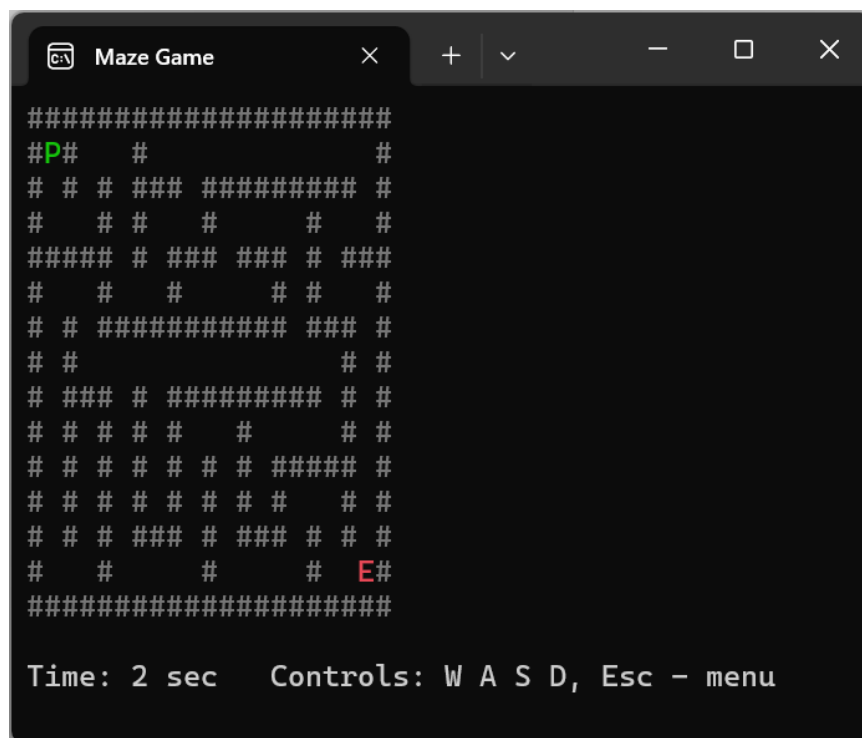


Рисунок 6 – Игровое поле

Управление осуществляется клавишами W (вверх), A (влево), S (вниз), D (вправо). При нажатии клавиши Esc производится возврат в главное меню без сохранения результата. При каждом нажатии клавиши движения программа вызывает ассемблерную вставку `CheckWallAsm` для проверки наличия стены в желаемой ячейке; если ячейка свободна, координаты игрока обновляются. После каждого перемещения вызывается ассемблерная вставка `CheckExitAsm` для проверки достижения выхода; при положительном результате игра завершается, выводится сообщение о победе, и время прохождения сохраняется в файл `records.txt`.

## 6.2.2 Схема программы игрового поля

Ниже представлены описания основных функций, отвечающих за работу игрового поля, и их схемы.

### 6.2.2.1 Функция playGame

Функция playGame отвечает за полный игровой цикл одного прохождения. Она запрашивает у пользователя имя, генерирует новый лабиринт вызовом generateMaze [4], фиксирует время начала игры, после чего входит в основной цикл. В цикле производится отрисовка лабиринта, чтение нажатой клавиши, вычисление новых координат игрока, проверка стены через CheckWallAsm, обновление координат, повторная отрисовка и проверка достижения выхода через CheckExitAsm. При достижении выхода вычисляется время прохождения, выводится поздравительное сообщение, и результат сохраняется вызовом saveRecord. Схема функции представлена на [рисунке 7](#).

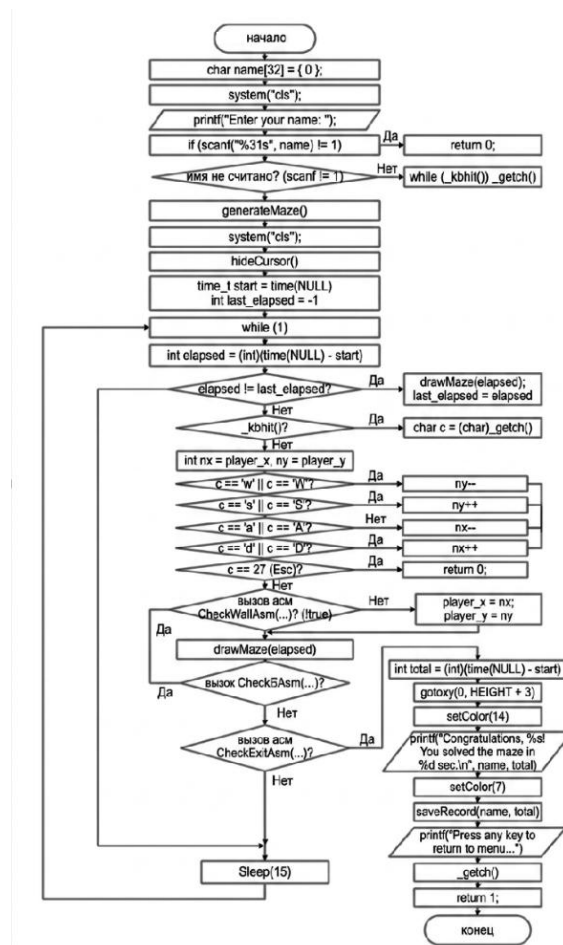


Рисунок 7 – Схема функции playGame

### 6.2.2.2 Функция generateMaze

Функция generateMaze отвечает за создание нового лабиринта. Она заполняет весь массив maze символами стены, помечает стартовую клетку (1, 1) как проход и вызывает рекурсивную функцию carveMaze, которая методом recursive backtracker «прорубает» в массиве сеть проходов. После завершения генерации функция устанавливает начальные координаты игрока и координаты выхода в правом нижнем углу. Схема функции представлена на [рисунке 8](#).

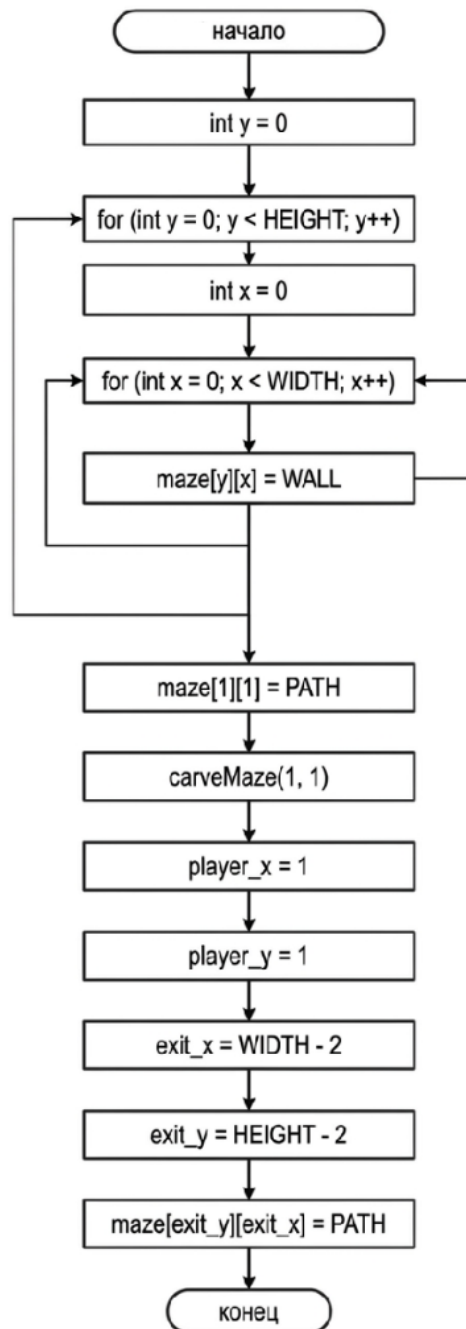


Рисунок 8 – Схема функции generateMaze

### 6.2.2.3 Функция carveMaze

Функция carveMaze является рекурсивной и реализует алгоритм recursive backtracker. Она формирует массив из четырёх направлений, перемешивает его случайным образом, после чего для каждого направления проверяет, лежит ли клетка на расстоянии двух шагов внутри лабиринта и является ли она стеной. Если оба условия выполнены, функция «прорубает» стену между текущей клеткой и клеткой через одну, пометая обе клетки как проход, и рекурсивно вызывает себя из новой клетки. Схема функции представлена на [рисунке 9](#).

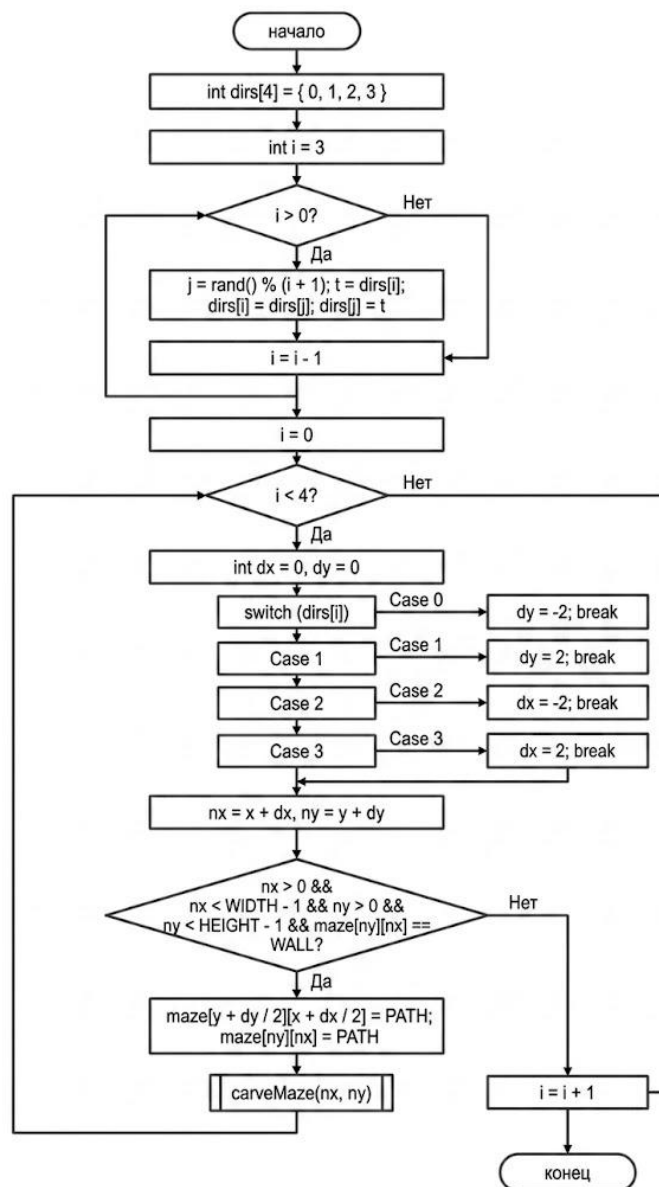


Рисунок 9 – Схема функции carveMaze

#### 6.2.2.4 Функция drawMaze

Функция drawMaze отвечает за отрисовку текущего состояния игрового поля в окне консоли. Она перемещает курсор в верхний левый угол, после чего в двойном цикле по строкам и столбцам массива maze выводит каждую ячейку с соответствующим цветом: зелёным для игрока, красным для выхода, серым для стены и стандартным для прохода. После лабиринта выводится текущее время прохождения и подсказка по управлению. Схема функции представлена на [рисунке 10](#).

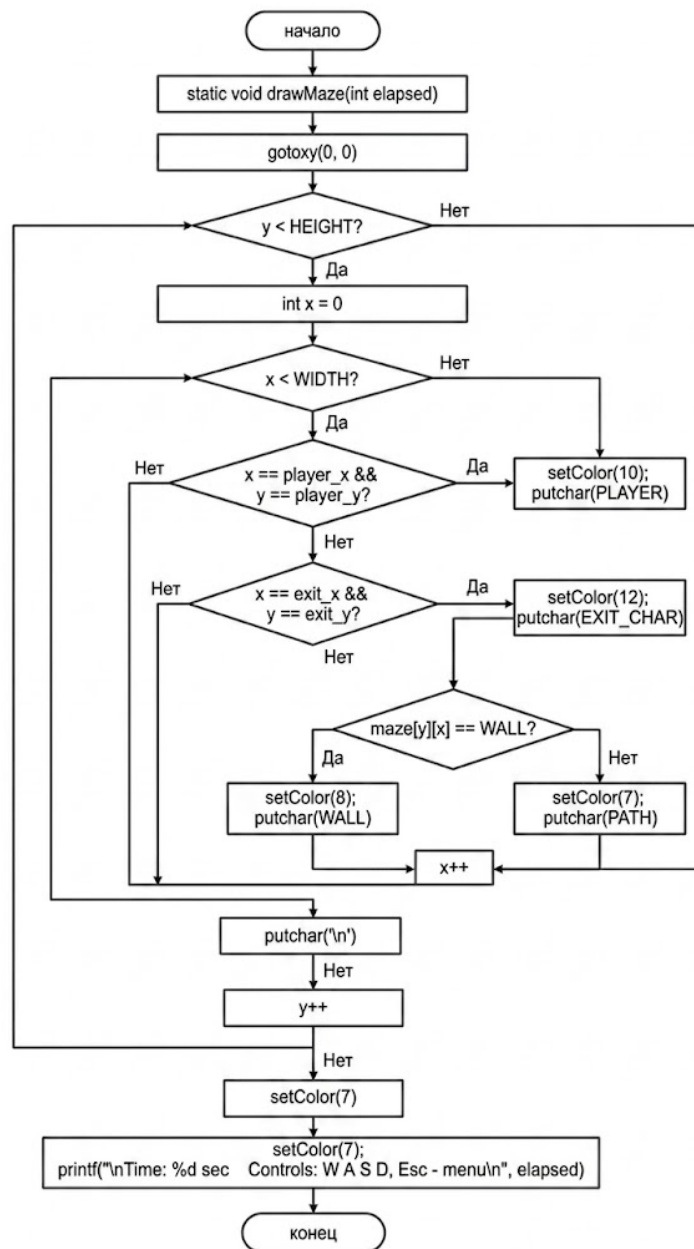


Рисунок 10 – Схема функции drawMaze

### 6.2.2.5 Функция CheckWallAsm

Функция CheckWallAsm является первой из двух ассемблерных вставок и отвечает за быструю проверку столкновения со стеной. Она принимает четыре параметра: указатель на массив лабиринта, координаты проверяемой ячейки (x, y) и ширину строки массива. Внутри блока \_\_asm выполняются следующие действия: в регистр ESI загружается базовый адрес массива, в регистре EAX вычисляется линейный индекс ячейки по формуле  $y * \text{width} + x$  при помощи инструкций imul и add, после чего к ESI прибавляется этот индекс, и из полученного адреса в AL загружается символ. Этот символ сравнивается со значением «#»; если совпадает, в локальную переменную result записывается 1, иначе 0. Схема функции представлена на [рисунке 11](#).

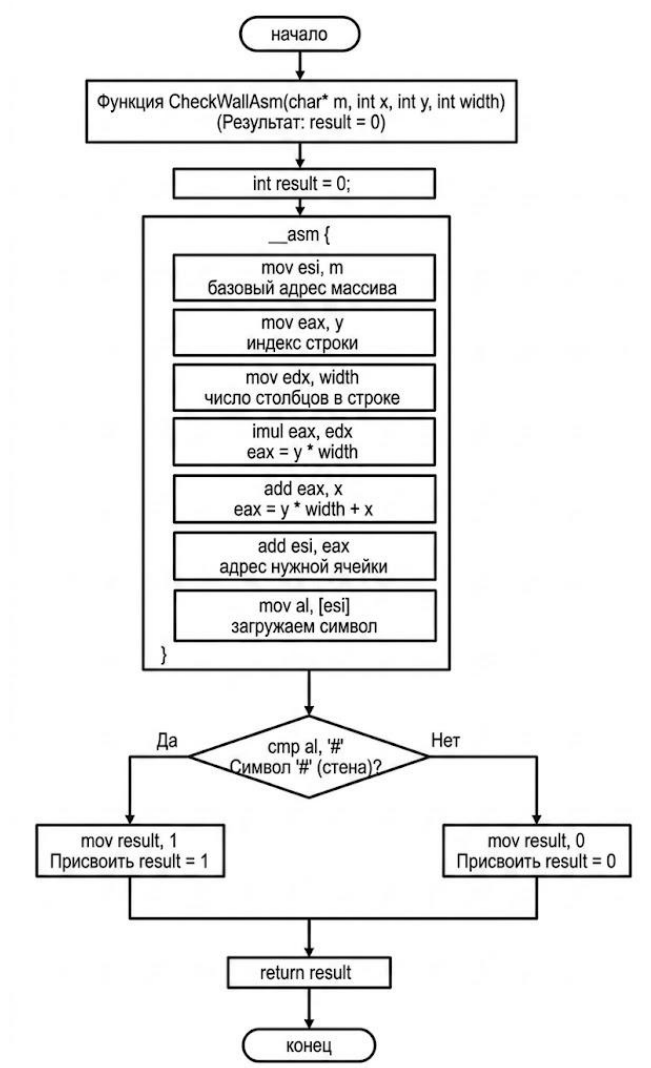


Рисунок 11 – Схема функции CheckWallAsm

### 6.2.2.6 Функция CheckExitAsm

Функция CheckExitAsm является второй ассемблерной вставкой и отвечает за быструю проверку достижения выхода. Она принимает четыре параметра: координаты игрока (px, py) и координаты выхода (ex, ey). Внутри блока \_\_asm производится последовательное сравнение px с ex и py с ey: если оба сравнения совпали, в локальную переменную result записывается 1, иначе 0. Несмотря на простоту, эта проверка выполняется на каждом шаге игрока, и реализация её на ассемблере исключает возможность лишних обращений к памяти и гарантирует минимально возможное число машинных инструкций. Схема функции представлена на [рисунке 12](#).

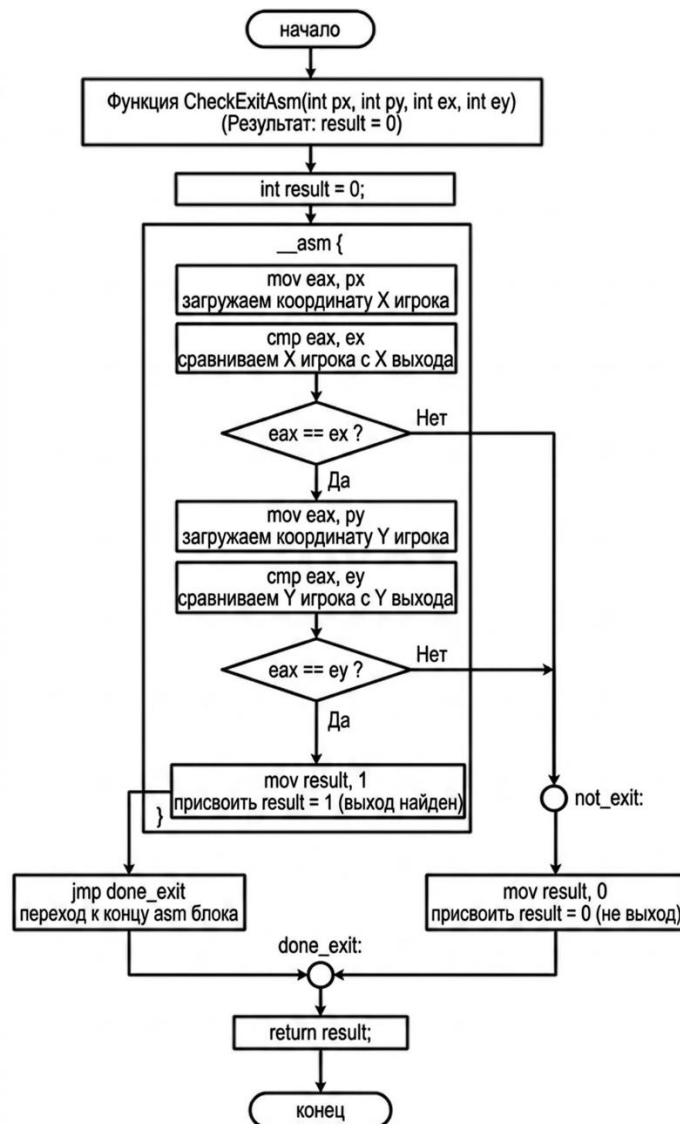


Рисунок 12 – Схема функции CheckExitAsm

### 6.2.2.7 Функция saveRecord

Функция saveRecord открывает файл records.txt в режиме дозаписи и добавляет в конец файла строку вида «имя\_игрока время\_в\_секундах». Если файл не существует, он создаётся. После записи файл закрывается. Схема функции представлена на [рисунке 13](#).

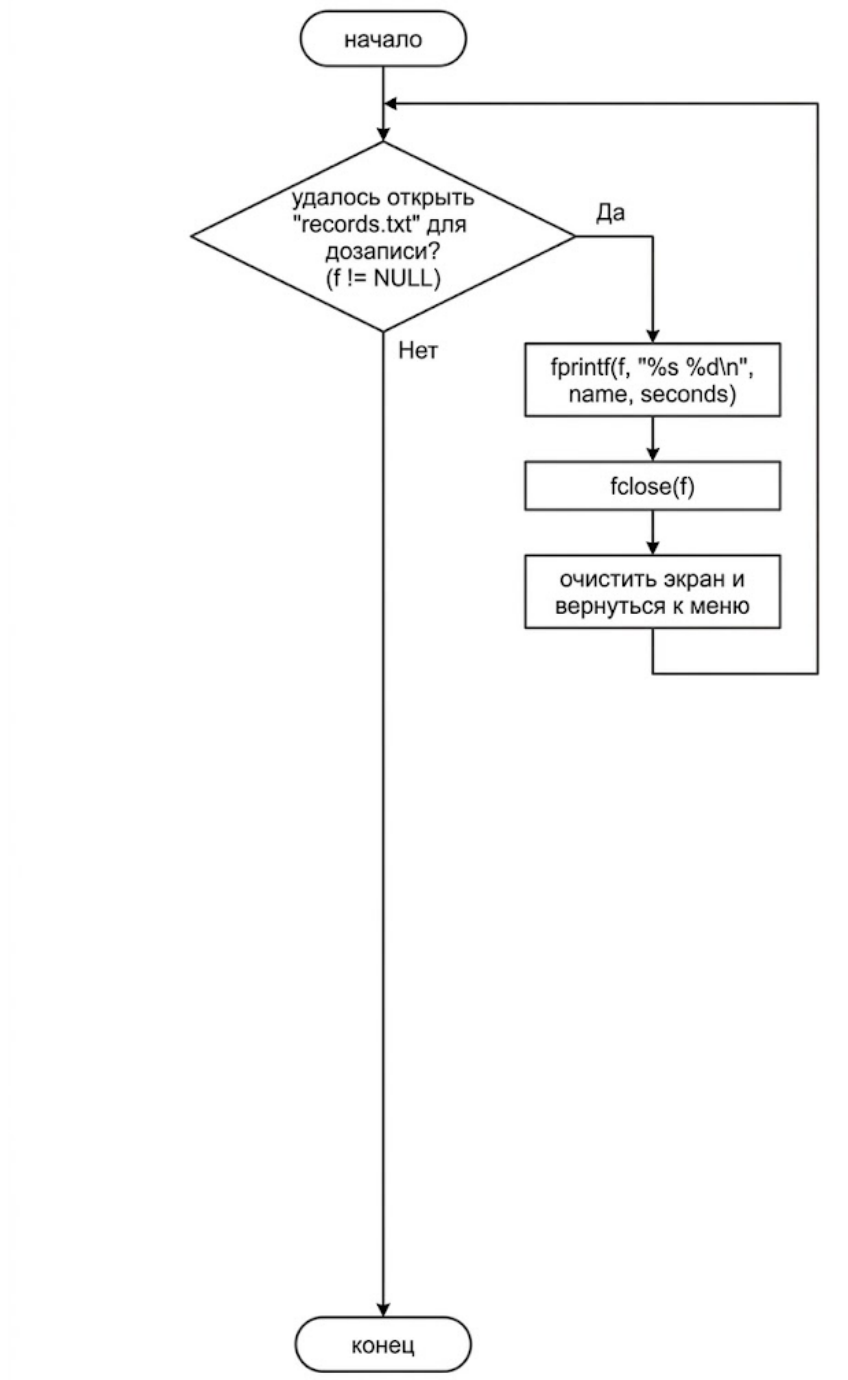


Рисунок 13 – Схема функции saveRecord

Все функции работают исправно.

## 7 Руководство пользователя

### 7.1. Запуск игры

Для запуска игры необходимо дважды нажать по исполняемому файлу Maze.exe, который расположен в папке Debug собранного проекта Visual Studio. Файл Maze.exe изображён на [рисунке 14](#).



Рисунок 14 – Файл Maze.exe

### 7.2. Главное меню

Сразу после запуска перед пользователем открывается главное меню ([рисунк 15](#)).

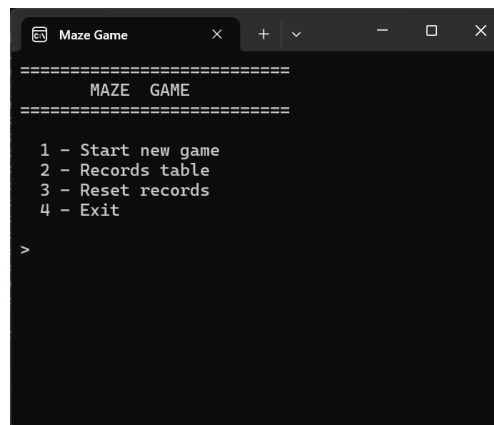


Рисунок 15 – Главное меню

Главное меню содержит четыре пункта:

Start new game — запускает новую игру с автоматически сгенерированным лабиринтом.

Records table — открывает таблицу рекордов с пятью лучшими результатами.

Reset records — полностью очищает таблицу рекордов (удаляет файл records.txt).

Exit — завершает работу программы.

Чтобы выбрать пункт, нужно ввести соответствующую цифру (от 1 до 4) и нажать клавишу Enter.

### 7.2.1 Таблица рекордов

При выборе пункта «Records table» отображается список из пяти лучших результатов прохождения, отсортированных по возрастанию времени ([рисунок 16](#)). Каждая запись содержит порядковый номер, имя игрока и время в секундах. Если рекордов ещё нет, выводится сообщение «No records yet. Be the first!».

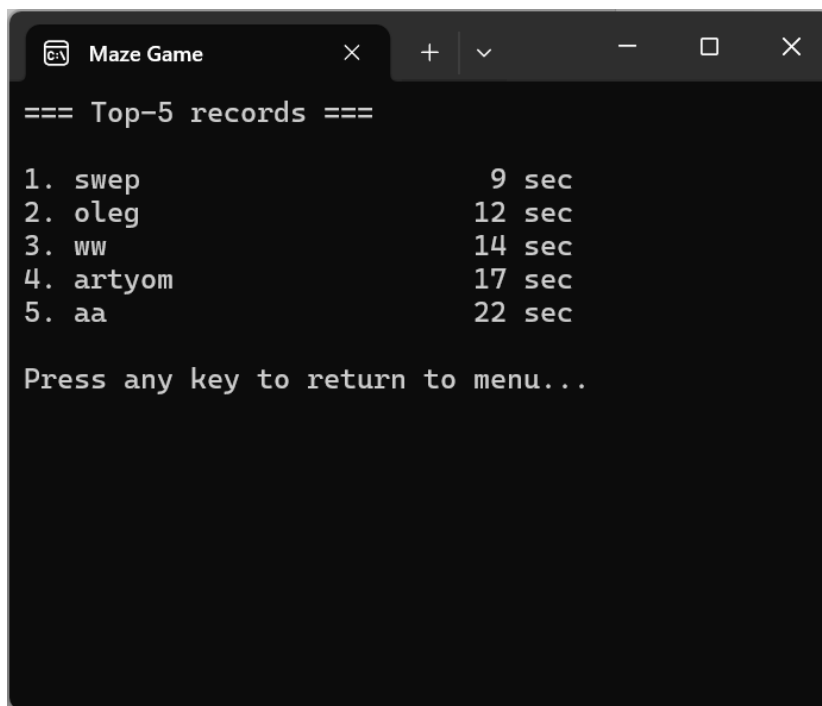


Рисунок 16 – Таблица рекордов

### 7.2.2 Сброс статистики

Чтобы полностью очистить таблицу рекордов, в главном меню нужно выбрать пункт 3 — «Reset records». Файл records.txt будет удалён, и при следующем открытии таблицы рекордов будет выведено сообщение об отсутствии записей.

## 7.3. Игровой процесс

### 7.3.1 Начало игры

После выбора пункта «Start new game» программа предложит ввести имя игрока ([рисунок 17](#)). После ввода имени и нажатия Enter будет сгенерирован новый лабиринт, и откроется игровое окно ([рисунок 18](#)).

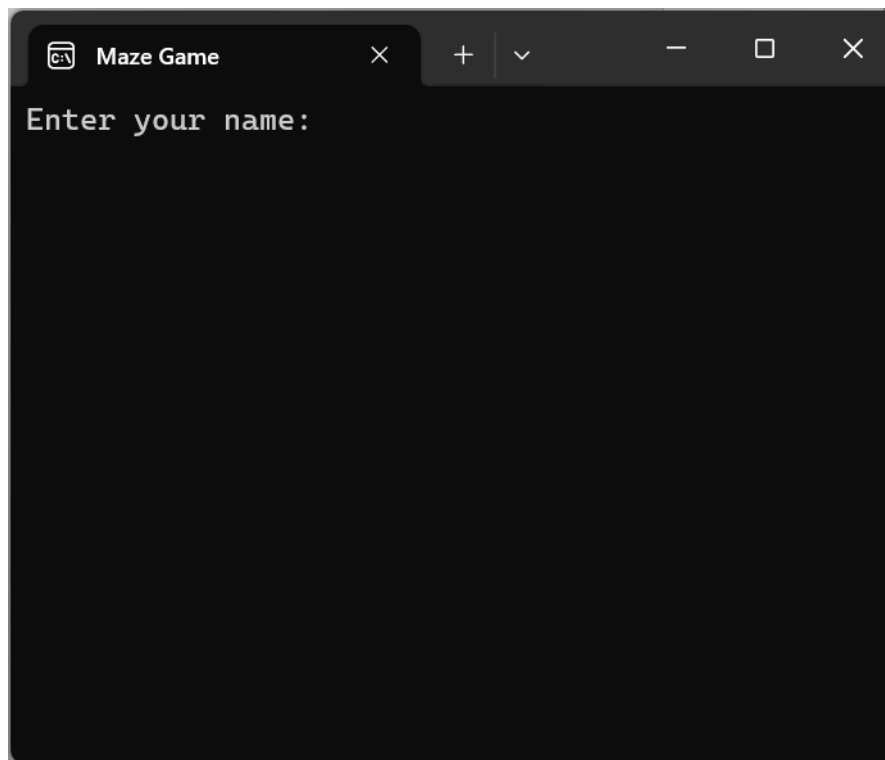


Рисунок 17 – Запрос имени игрока

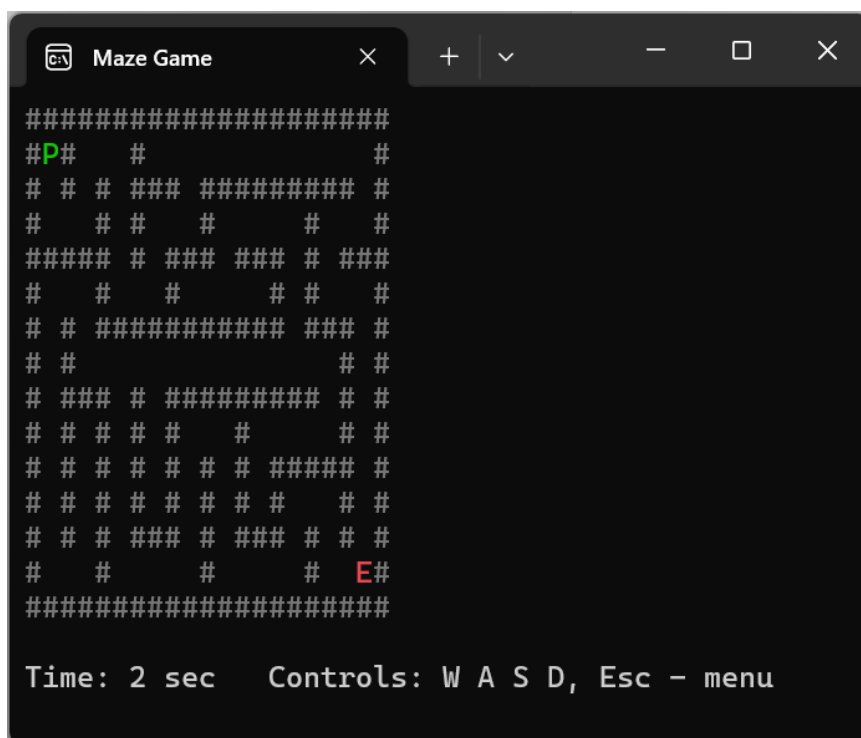


Рисунок 18 – Игровое поле в начале игры

В игровом окне отображаются: лабиринт с игроком (зелёная буква «Р») в верхнем левом углу и выходом (красная буква «Е») в правом нижнем углу, а также строка с таймером и подсказкой по управлению.

### **7.3.2 Управление**

Управление игроком осуществляется четырьмя клавишами на клавиатуре:

W — переместить игрока на одну клетку вверх;

S — переместить игрока на одну клетку вниз;

A — переместить игрока на одну клетку влево;

D — переместить игрока на одну клетку вправо.

Если в желаемой клетке находится стена, движение не происходит — это поведение обеспечивается ассемблерной вставкой `CheckWallAsm`. Для возврата в главное меню без сохранения результата нужно нажать клавишу `Esc`.

### **7.3.3 Цель игры**

Цель игры — переместить игрока (символ «P») из верхнего левого угла в правый нижний угол, где расположен выход (символ «E»). При каждом перемещении ассемблерная вставка `CheckExitAsm` проверяет совпадение координат игрока и выхода; при совпадении игра считается выигранной

### **7.3.4 Элементы игрового интерфейса**

Помимо собственно игрового поля, на экране отображается строка с текущим временем прохождения в секундах и подсказкой по управлению. Таймер обновляется один раз в секунду и отсчитывает время от момента начала игры до момента достижения выхода либо нажатия `Esc`.

## **7.4. Завершение игры**

### **7.4.1 Победа**

При достижении игроком выхода ассемблерная вставка `CheckExitAsm` возвращает 1, таймер останавливается, и под лабиринтом выводится поздравительное сообщение с указанием имени игрока и времени прохождения ([рисунок 19](#)). Результат автоматически записывается в файл `records.txt`. Для возврата в главное меню нужно нажать любую клавишу.

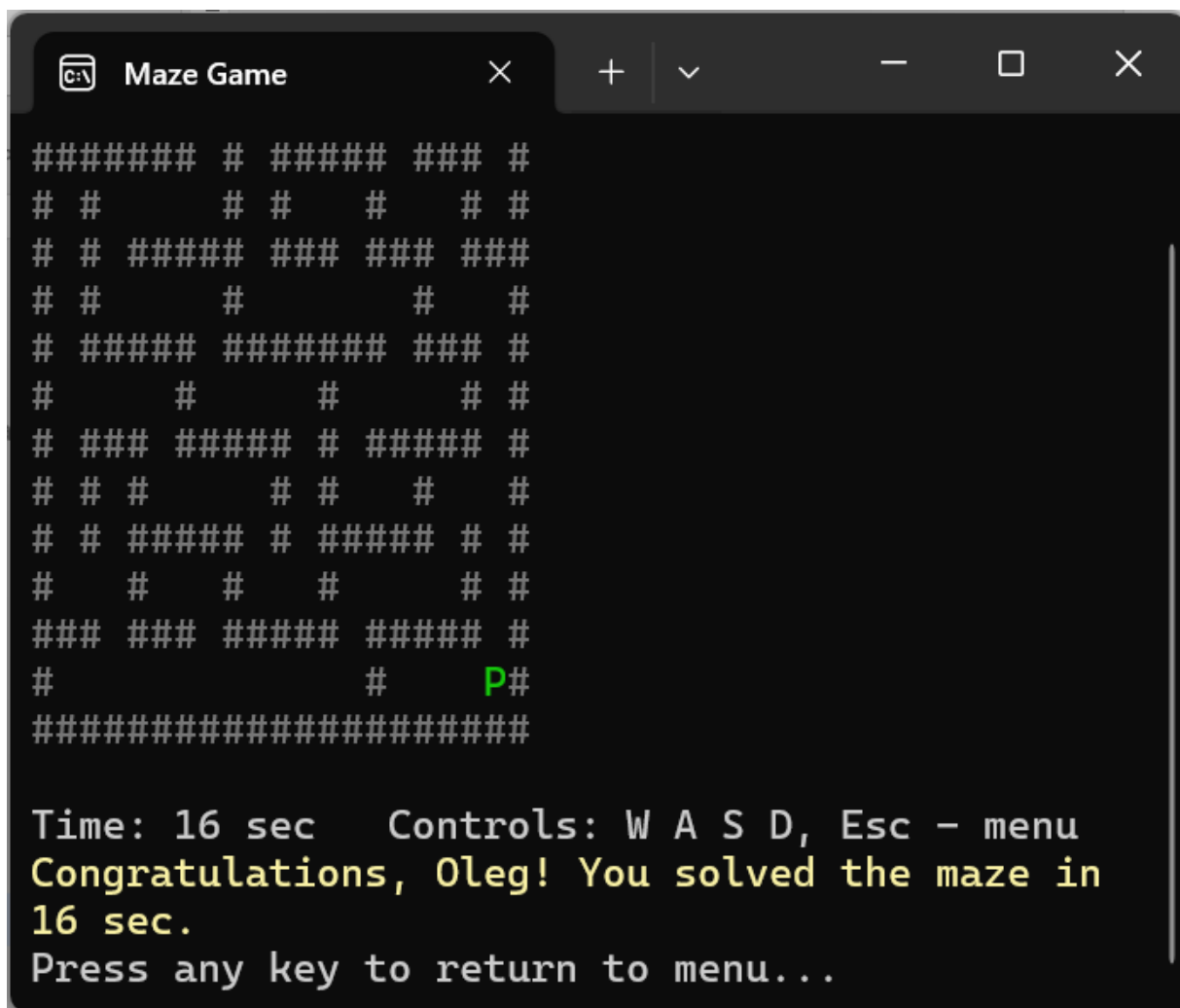


Рисунок 19 – Окно победы

#### 7.4.2 Прерывание игры

В любой момент игры можно нажать клавишу Esc для немедленного возврата в главное меню. В этом случае результат текущего прохождения не сохраняется. Также можно закрыть окно консоли стандартным способом — это эквивалентно завершению программы.

## Заключение

В ходе выполнения курсовой работы была разработана игра «Лабиринт» на языке программирования C в среде Microsoft Visual Studio 2022. В процессе разработки были освоены следующие технологии и приёмы:

- работа с консольным API Windows (SetConsoleCursorPosition, SetConsoleTextAttribute, SetConsoleCursorInfo) для создания простой и наглядной двумерной графики;
- реализация алгоритма генерации связного лабиринта методом recursive backtracker;
- использование инлайн-ассемблерных вставок `__asm` для оптимизации двух наиболее часто выполняемых проверок — столкновения со стеной (CheckWallAsm) и достижения выхода (CheckExitAsm);
- работа с файлами для сохранения и чтения таблицы рекордов;
- организация интерактивного игрового цикла с обработкой пользовательского ввода и измерением времени.

Программа полностью соответствует всем поставленным требованиям: компьютер автоматически генерирует лабиринт, пользователь проходит его от начала до конца, результаты сохраняются в таблицу рекордов, и два ключевых блока программы реализованы при помощи ассемблерных вставок. Программа компилируется и запускается в Microsoft Visual Studio 2022 в конфигурации Win32 и может быть легко расширена добавлением уровней сложности, новых элементов интерфейса либо более сложной графики на основе библиотеки GDI.

### Список используемых источников

1. Ирвин, К. Язык ассемблера для процессоров Intel / К. Ирвин. — 7-е изд. — Москва : Вильямс, 2018. — 912 с. — ISBN 978-5-8459-1903-9.
2. Керниган, Б. Язык программирования C / Б. Керниган, Д. Ритчи. — 2-е изд. — Москва : Вильямс, 2017. — 304 с. — ISBN 978-5-8459-0891-9.
3. Прата, С. Язык программирования C. Лекции и упражнения / С. Прата. — 6-е изд. — Москва : Вильямс, 2015. — 928 с. — ISBN 978-5-8459-1950-3.
4. Maze generation algorithm — Recursive backtracker : [сайт]. — URL: [https://en.wikipedia.org/wiki/Maze\\_generation\\_algorithm](https://en.wikipedia.org/wiki/Maze_generation_algorithm) (дата обращения: 15.05.2025).
5. Microsoft Docs : Inline Assembler (C/C++) : [сайт]. — URL: <https://learn.microsoft.com/cpp/assembler/inline/inline-assembler> (дата обращения: 15.05.2025).
6. Microsoft Docs : Windows Console API : [сайт]. — URL: <https://learn.microsoft.com/windows/console/> (дата обращения: 15.05.2025).

**Приложение А**  
**Листинг программы**  
**Файл Maze.c**

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include <conio.h>

#define WIDTH 21
#define HEIGHT 15
#define WALL '#'
#define PATH ' '
#define PLAYER 'P'
#define EXIT_CHAR 'E'
#define MAX_RECORDS 100

typedef struct {
    char name[32];
    int seconds;
} Record;

static char maze[HEIGHT][WIDTH];
static int player_x, player_y;
static int exit_x, exit_y;

/* === Ассемблерная вставка №1 ===
Быстрая проверка столкновения со стеной.
Принимает указатель на массив, координаты (x, y) и ширину строки.
Возвращает 1, если ячейка содержит символ '#', иначе 0. */
int CheckWallAsm(char* m, int x, int y, int width) {
    int result = 0;
    __asm {
        mov esi, m; базовый адрес массива
        mov eax, y; индекс строки
        mov edx, width; число столбцов в строке
        imul eax, edx; eax = y * width
        add eax, x; eax = y * width + x
        add esi, eax; адрес нужной ячейки
        mov al, [esi]; загружаем символ
        cmp al, '#'; сравниваем со стеной
        jne not_wall
        mov result, 1
        jmp done_wall
    not_wall:
        mov result, 0
    }
```

```

        done_wall :
    }
    return result;
}

/* === Ассемблерная вставка №2 ===
Быстрая проверка достижения выхода игроком.
Принимает координаты игрока (px, py) и выхода (ex, ey).
Возвращает 1, если координаты совпадают, иначе 0. */
int CheckExitAsm(int px, int py, int ex, int ey) {
    int result = 0;
    __asm {
        mov     eax, px
        cmp     eax, ex
        jne     not_exit
        mov     eax, py
        cmp     eax, ey
        jne     not_exit
        mov     result, 1
        jmp     done_exit
    not_exit :
        mov     result, 0
    done_exit :
    }
    return result;
}

static void carveMaze(int x, int y) {
    int dirs[4] = { 0, 1, 2, 3 };
    for (int i = 3; i > 0; i--) {
        int j = rand() % (i + 1);
        int t = dirs[i]; dirs[i] = dirs[j]; dirs[j] = t;
    }
    for (int i = 0; i < 4; i++) {
        int dx = 0, dy = 0;
        switch (dirs[i]) {
            case 0: dy = -2; break;
            case 1: dy = 2; break;
            case 2: dx = -2; break;
            case 3: dx = 2; break;
        }
        int nx = x + dx, ny = y + dy;
        if (nx > 0 && nx < WIDTH - 1 &&
            ny > 0 && ny < HEIGHT - 1 &&
            maze[ny][nx] == WALL) {
            maze[y + dy / 2][x + dx / 2] = PATH;
            maze[ny][nx] = PATH;
            carveMaze(nx, ny);
        }
    }
}

```

```

static void generateMaze(void) {
    for (int y = 0; y < HEIGHT; y++)
        for (int x = 0; x < WIDTH; x++)
            maze[y][x] = WALL;
    maze[1][1] = PATH;
    carveMaze(1, 1);
    player_x = 1; player_y = 1;
    exit_x = WIDTH - 2; exit_y = HEIGHT - 2;
    maze[exit_y][exit_x] = PATH;
}

static void setColor(int color) {
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
(WORD)color);
}

static void gotoxy(int x, int y) {
    COORD c; c.X = (SHORT)x; c.Y = (SHORT)y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), c);
}

static void hideCursor(void) {
    CONSOLE_CURSOR_INFO ci = { 1, FALSE };
    SetConsoleCursorInfo(GetStdHandle(STD_OUTPUT_HANDLE), &ci);
}

static void drawMaze(int elapsed) {
    gotoxy(0, 0);
    for (int y = 0; y < HEIGHT; y++) {
        for (int x = 0; x < WIDTH; x++) {
            if (x == player_x && y == player_y) {
                setColor(10); putchar(PLAYER);
            }
            else if (x == exit_x && y == exit_y) {
                setColor(12); putchar(EXIT_CHAR);
            }
            else if (maze[y][x] == WALL) {
                setColor(8); putchar(WALL);
            }
            else {
                setColor(7); putchar(PATH);
            }
        }
        putchar('\n');
    }
    setColor(7);
    printf("\nTime: %d sec      Controls: W A S D, Esc - menu\n",
elapsed);
}

```

```

static void saveRecord(const char* name, int seconds) {
    FILE* f = fopen("records.txt", "a");
    if (f) {
        fprintf(f, "%s %d\n", name, seconds);
        fclose(f);
    }
}

static int loadRecords(Record records[]) {
    FILE* f = fopen("records.txt", "r");
    if (!f) return 0;
    int n = 0;
    while (n < MAX_RECORDS &&
        fscanf(f, "%31s %d", records[n].name, &records[n].seconds)
== 2) {
        n++;
    }
    fclose(f);
    return n;
}

static void showRecords(void) {
    Record records[MAX_RECORDS];
    int n = loadRecords(records);
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - 1 - i; j++)
            if (records[j].seconds > records[j + 1].seconds) {
                Record t = records[j];
                records[j] = records[j + 1];
                records[j + 1] = t;
            }
    system("cls");
    printf("=== Top-5 records ===\n\n");
    int top = n < 5 ? n : 5;
    if (top == 0) {
        printf("No records yet. Be the first!\n");
    }
    else {
        for (int i = 0; i < top; i++)
            printf("%d. %-20s %5d sec\n",
                i + 1, records[i].name, records[i].seconds);
    }
    printf("\nPress any key to return to menu...");
    _getch();
}

static void resetRecords(void) {
    remove("records.txt");
    printf("\nRecords have been reset.\nPress any key...");
    _getch();
}

```

```

static int playGame(void) {
    char name[32] = { 0 };
    system("cls");
    printf("Enter your name: ");
    if (scanf("%31s", name) != 1) return 0;
    while (_kbhit()) _getch();

    generateMaze();
    system("cls");
    hideCursor();
    time_t start = time(NULL);
    int last_elapsed = -1;

    while (1) {
        int elapsed = (int)(time(NULL) - start);
        if (elapsed != last_elapsed) {
            drawMaze(elapsed);
            last_elapsed = elapsed;
        }
        if (_kbhit()) {
            char c = (char)_getch();
            int nx = player_x, ny = player_y;
            if (c == 'w' || c == 'W') ny--;
            else if (c == 's' || c == 'S') ny++;
            else if (c == 'a' || c == 'A') nx--;
            else if (c == 'd' || c == 'D') nx++;
            else if (c == 27) return 0; /* Esc */

            /* Проверка стены через ассемблерную вставку */
            if (!CheckWallAsm((char*)maze, nx, ny, WIDTH)) {
                player_x = nx;
                player_y = ny;
            }
            drawMaze(elapsed);

            /* Проверка достижения выхода через ассемблерную вставку */
            if (CheckExitAsm(player_x, player_y, exit_x, exit_y)) {
                int total = (int)(time(NULL) - start);
                gotoxy(0, HEIGHT + 3);
                setColor(14);
                printf("Congratulations, %s! You solved the maze in
%d sec.\n",
                    name, total);
                setColor(7);
                saveRecord(name, total);
                printf("Press any key to return to menu...");
                _getch();
                return 1;
            }
        }
    }
}

```

```

        }
        Sleep(15);
    }
}

int main(void) {
    SetConsoleTitleA("Maze Game");
    srand((unsigned)time(NULL));
    while (1) {
        system("cls");
        printf("=====\n");
        printf("        MAZE  GAME\n");
        printf("=====\n\n");
        printf(" 1 - Start new game\n");
        printf(" 2 - Records table\n");
        printf(" 3 - Reset records\n");
        printf(" 4 - Exit\n\n");
        printf("> ");
        int choice = 0;
        if (scanf("%d", &choice) != 1) {
            while (_kbhit()) _getch();
            continue;
        }
        if (choice == 1) playGame();
        else if (choice == 2) showRecords();
        else if (choice == 3) resetRecords();
        else if (choice == 4) break;
    }
    return 0;
}

```

## Приложение В

### Результаты работы программы

```
Maze Game
=====
MAZE  GAME
=====

1 - Start new game
2 - Records table
3 - Reset records
4 - Exit

>
```

В 1 – Главное меню программы

```
Maze Game
#####
#P#  #  #
# # # ### ##### #
#  # #  #  #  #
##### # ### ### # ###
#  # #  #  #  #
# # ##### ### #
# #  #  #  #  #
# ### # ##### # #
# # # #  #  #  #
# # # # # # # ##### #
# # # # # # #  #  #
# # # ### # ### # # #
#  #  #  #  # E#
#####

Time: 2 sec   Controls: W A S D, Esc - menu
```

В 2 – Игровое поле

```
Maze Game
##### # ##### ### #
# #      # #      # #
# # ##### ### ### ###
# #      #      # #
# ##### ##### ### #
#      #      #      #
# ### ##### # ##### #
# # #      # #      #
# # ##### # ##### #
# #      #      # #
### ### ##### ##### #
#      #      # P#
#####

Time: 16 sec   Controls: W A S D, Esc - menu
Congratulations, Oleg! You solved the maze in
16 sec.
Press any key to return to menu...
```

В 3 – Окно победы

```
6.asm
Файл  Изменить  Просмотр  Aa  ...

ww 14
aa 22
artyom 17
oleg 12
swep 9
abddn 306
Oleg 16

Строка 1, столбец 1  55 символов  Обычн  100%  Win  UTF-8
```

В 4 – Файл scores.txt