Министерство образования Российской Федерации
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 игры “Лабиринт”»

Выполнил:
студент группы 24ВВВ2
Макаров Я.С.

Принял:
к/н., доцент Юрова О.В.

Пенза, 2025

Содержание

Введение..... 5

Реферат

Отчет XX страниц, XX рисунка, XX таблицы, XX источника.

ГРАФ, ТЕОРИЯ ГРАФОВ, ПОИСК В ГЛУБИНУ (DFS), ГЕНЕРАЦИЯ ЛАБИРИНТА, ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ, ИГРОВОЙ ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

Цель исследования – разработка программного комплекса, реализующего процедурную генерацию лабиринта методом рекурсивного бэк-трекинга (randomized DFS), а также обеспечивающего его автоматическое прохождение и интерактивное соревнование пользователя с компьютерным агентом.

В работе рассмотрены алгоритм поиска в глубину, на основе которого реализованы методы генерации случайного связного лабиринта и поиска пути от входа к выходу. Разработана система динамического управления сложностью, изменяющая размерность игрового поля и скорость реакции компьютерного оппонента. Реализован соревновательный режим в реальном времени, система сохранения результатов и демонстрационные модули («пасхалки»). Установлено, что выбранный подход обеспечивает высокую вариативность игрового процесса и стабильную работу с динамическим распределением памяти.

Введение

Лабиринт - это сложная разветвленная структура, представляющая собой математический граф, где коридоры являются ребрами, а перекрестки — узлами. В информатике лабиринты служат классической моделью для исследования алгоритмов поиска путей и процедурной генерации контента, позволяя наглядно демонстрировать работу методов обхода графов.

Лабиринты можно классифицировать по различным критериям:

– Линейные лабиринты: имеют один единственный путь без разветвлений, ведущий от входа к выходу;

– Классические лабиринты: содержат множество разветвлений и тупиков, предоставляя сложный маршрут для поиска выхода (именно этот тип реализуется в данной работе);

– Многоуровневые лабиринты: состоят из нескольких уровней, соединённых переходами, что увеличивает их сложность;

– Трёхмерные лабиринты: представляют собой структуры с несколькими плоскостями, где путь может проходить вверх или вниз, добавляя дополнительное измерение сложности.

Существует множество стратегий для прохождения лабиринтов:

– Правило правой (или левой) руки: движение вдоль стены с правой (или левой) стороны, что гарантирует нахождение выхода в простых односвязных лабиринтах;

– Поиск в ширину (BFS): алгоритм, исследующий все возможные пути равномерно на каждом уровне до нахождения выхода (гарантирует кратчайший путь);

– Поиск в глубину (DFS): алгоритм, исследующий один путь до конца, прежде чем возвращаться и пробовать другой путь; данный метод является основным в работе, так как используется и для генерации, и для решения;

– Алгоритм A (A-star): использует эвристическую функцию для оценки расстояния до выхода, оптимизируя поиск кратчайшего пути.

Эти стратегии варьируются по сложности и эффективности в зависимости от структуры и типа лабиринта.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio Community 2022, язык программирования – C (Си), интерфейс взаимодействия – консольное приложение Windows (Console Application) с использованием символьной графики (ASCII-арт).

Целью данной курсовой работы является разработка программы на языке C, который позволяет эффективно управлять динамической памятью и структурами данных. Именно с помощью языка C в данном курсовом проекте реализуется алгоритм поиска в глубину, осуществляющий процедурную генерацию случайного лабиринта и нахождение пути от старта до финиша в режиме соревнования с компьютером.

1. Постановка задачи и обязательные требования

Лабиринт должен задаваться динамической матрицей ячеек, каждая из которых может представлять собой либо свободный проход, либо стену. Состояние каждой ячейки описывается значением в массиве (0 – проход, 1 – стена), что позволяет формировать сложную структуру переходов и тупиков. Размерность лабиринта должна задаваться пользователем через выбор уровня сложности, что определяет выделение соответствующего объема памяти для игрового поля.

Для генерации лабиринта должен использоваться алгоритм случайного поиска в глубину с использованием стека вызовов (рекурсии). Его работа начинается со стартовой ячейки, которая помечается как проход. На каждом шаге алгоритм случайным образом выбирает одно из возможных направлений движения на две клетки вперед. Если целевая клетка не посещена, стена между текущей и целевой клеткой удаляется, и алгоритм рекурсивно переходит в новую клетку. Если же у текущей ячейки нет непосещенных соседей, происходит возврат назад, пока не будет найдена ячейка с возможными ходами. Процесс продолжается до тех пор, пока все доступные ячейки не будут посещены.

Программа должна предусматривать работу в нескольких режимах: одиночное прохождение, автоматическое решение и соревнование с компьютером. После генерации лабиринта необходимо обеспечить возможность его прохождения:

– Вручную пользователем: управление осуществляется с помощью клавиш клавиатуры (w, a, s, d), что обеспечивает интуитивное взаимодействие с интерфейсом;

– Автоматически: с помощью алгоритма поиска в глубину (DFS), который находит путь от входа до выхода.

Алгоритм поиска в глубину строит маршрут, следуя по ветвям лабиринта до тупика, после чего возвращается к последней развилке. Он прекращает свою работу при достижении целевой ячейки (выхода). Этот метод эффективен для поиска путей в односвязных лабиринтах и проверки их проходимости.

Дополнительно программа должна обеспечивать:

– Сохранение результатов прохождения (время или счет) в текстовый файл;

– Динамическое изменение сложности (размеров лабиринта и скорости бота) без перезапуска программы;

– Визуализацию процесса прохождения в консоли с использованием символьной графики.

2. Теоретическая часть задания

Алгоритм случайного поиска в глубину (англ. Randomized Depth-First Search) позволяет генерировать идеальные лабиринты, представляющие собой случайное дерево путей без циклов. Этот алгоритм основан на принципе рекурсивного обхода сетки, начиная с одной стартовой ячейки и прокладывая путь в случайном направлении, удаляя стены между соседними ячейками.

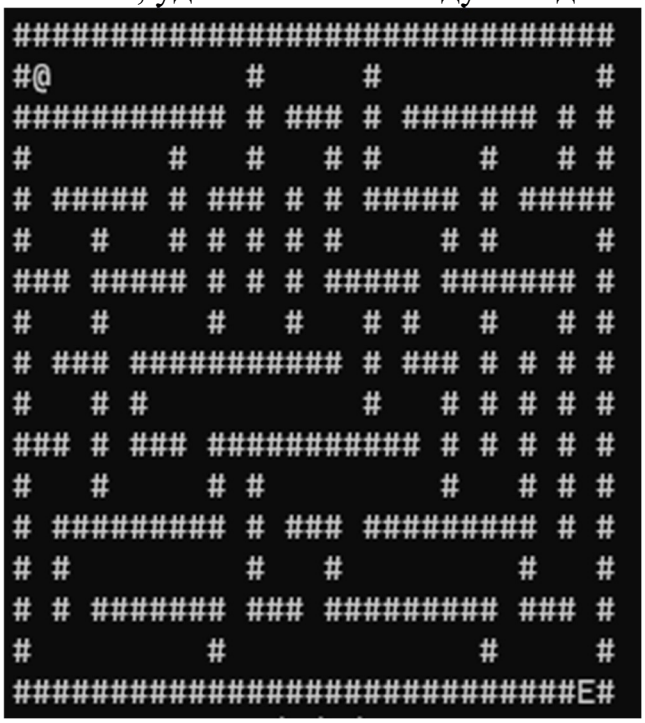

Рисунок 1 — Пример сгенерированного лабиринта

Алгоритм Randomized DFS (рис. 1) позволяет создавать лабиринты с длинными извилистыми коридорами и малым количеством коротких тупиков, что делает их сложными для прохождения.

Отличие Randomized DFS от других методов генерации (например, алгоритма Прима или Краскала) заключается в его простоте реализации через стек и способности создавать длинные, запутанные маршруты. Алгоритм начинается с выбора стартовой ячейки, которая помечается как посещенная. Далее из текущей ячейки случайным образом выбирается один из непосещенных соседей. Если такой сосед найден, стена между ними удаляется, и алгоритм переходит в новую ячейку. Если у текущей ячейки нет непосещенных соседей, происходит возврат назад.

Принцип работы алгоритма генерации:

– Изначально выбирается стартовая ячейка (обычно (1,1)), которая помечается как посещенная (рис. 2). Стек вызовов хранит путь от старта до текущей точки;

– Выбирается случайное направление (вверх, вниз, влево или вправо) и проверяется соседняя ячейка через одну (на расстоянии 2 клеток);

– Если соседняя ячейка еще не была посещена, то стена между текущей и выбранной ячейкой удаляется (становится проходом), а алгоритм

рекурсивно переходит в эту новую ячейку (рис. 3).

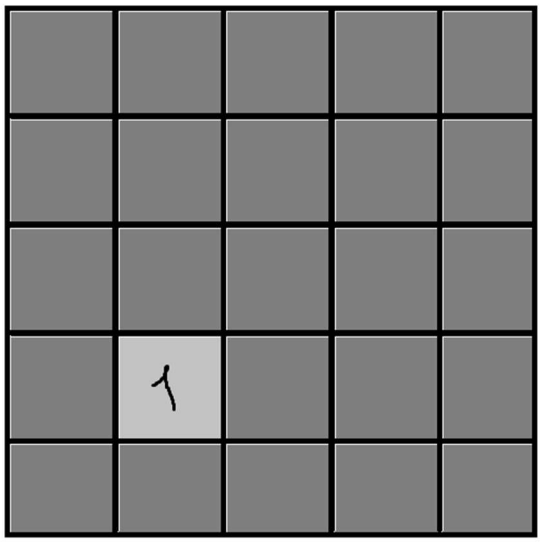

Рисунок 2 — Начальная точка

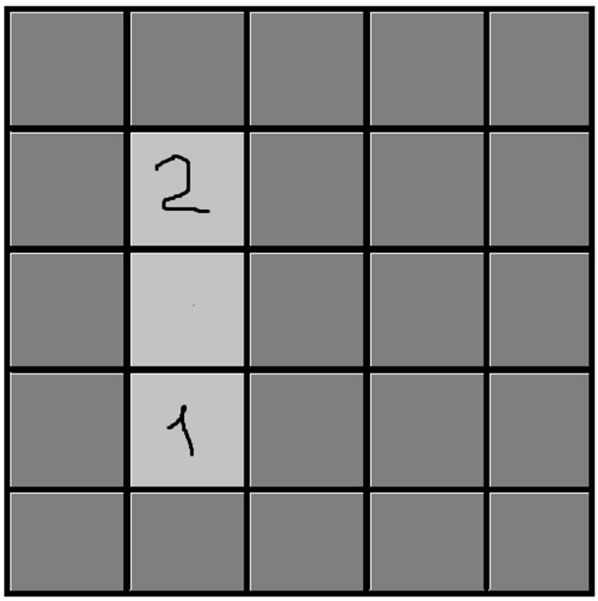

Рисунок 3 — Переход в новую клетку

Этот процесс повторяется, пока алгоритм «не упрётся в тупик» (рис. 4), то есть пока у текущей ячейки есть непосещенные соседи;

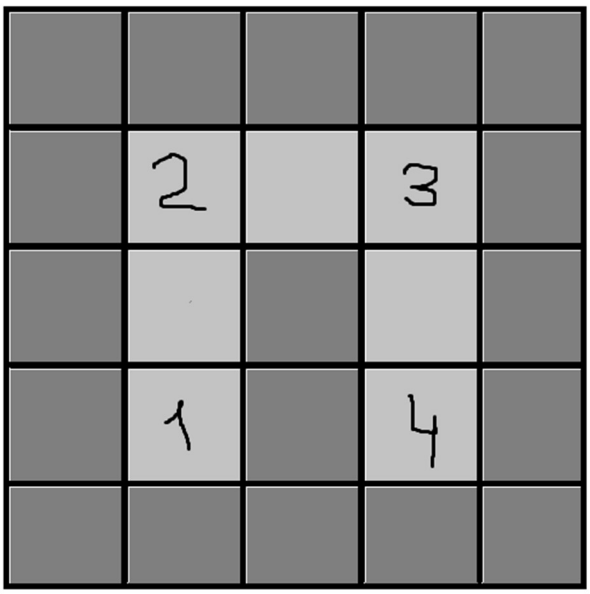

Рисунок 4 — Тупик

– Если у текущей ячейки нет непосещенных соседей (тупик), алгоритм возвращается назад к предыдущей ячейке из стека вызовов. И выбирается другое направление, если оно доступно (рис. 5);

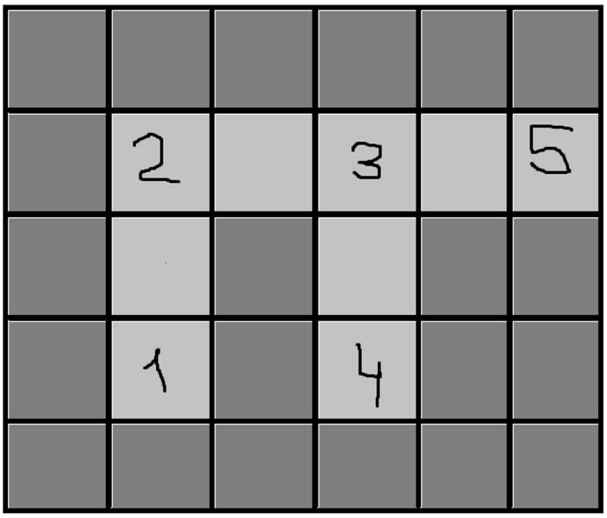

Рисунок 5 — Новое направление

– Процесс повторяется, пока не будут посещены все доступные ячейки и стек вызовов не опустеет.

Алгоритм поиска (или обхода) в глубину (англ. Depth-First Search, DFS) позволяет найти путь от входа до выхода. Он работает по схожему принципу: алгоритм пытается продвинуться как можно дальше по каждому возможному пути, прежде чем вернуться назад.

Отличие поиска в глубину от поиска в ширину заключается в том, что (в случае неориентированного графа, коим является лабиринт) результатом алгоритма DFS является любой найденный путь, который не обязательно является кратчайшим. DFS исследует одну ветвь до самого конца, и только если

она не ведет к выходу, возвращается и проверяет другую. Это принципиально отличается от поиска в ширину (BFS), где одновременно исследуются все возможные пути на текущем расстоянии от старта.

С другой стороны, поиск в глубину проще в реализации (требует меньше памяти при глубоких, но узких графах) и идеально подходит для проверки связности лабиринта или нахождения любого допустимого пути.

Если граф ориентированный (что в нашем случае не так, но теоретически возможно), то поиск в глубину строит дерево путей из начальной вершины во все доступные из нее. В данной работе DFS используется как основной метод решения лабиринта ботом.

3. Описание программы

Программа разработана на языке C (Си) в среде Visual Studio 2022 как консольное приложение. Логика разделена на модули, каждый из которых отвечает за свой функционал. Графический интерфейс реализован посредством символьной псевдографики.

3.1 Основные функции и глобальные переменные программы

В файле main.c расположены глобальные переменные, хранящие состояние игры, и точка входа.

Глобальные переменные:

Переменные для подсчета очков в режиме соревнования:

```
int playerScore = 0;
int botScore = 0;
```

Массивы для хранения путей и состояния посещения клеток:

```
Point path[40000];
int pathLength = 0;

bool** visited = NULL;

Point fullPath[40000];
int fullPathLen = 0;
```

Функция main()

Является главной функцией. Она инициализирует настройки, выделяет память и запускает цикл обработки меню:

```
int main() {
    setlocale(LC_ALL, "Russian");
    srand((unsigned int)time(NULL));
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(hConsole, &cursorInfo);
    cursorInfo.bVisible = FALSE;
    SetConsoleCursorInfo(hConsole, &cursorInfo);

    selectDifficulty();
```

```
        if (!allocMaze() || !allocVisited()) {
            printf("Ошибка выделения памяти.\n");
            return 1;
        }

        int startX = 1, startY = 1;
        int exitX = G_WIDTH - 2, exitY = G_HEIGHT - 1;
        initMaze(startX, startY, exitX, exitY);
        makeFullDfsPath(startX, startY, exitX, exitY);

        int menu_choice, running = 1;

        while (running) {
            system("cls");
            printf("ЛАБИРИНТ\n");
            printf("1 - Одиночная игра\n");
            printf("2 - Игра с подсказками\n");
            printf("3 - Пройти за меня, мне лень\n");
            printf("4 - Секретный режим\n");
            printf("5 - Результаты\n");
            printf("6     -     Соревнование     (игрок     против
компьюетра!!)\n");
            printf("7 - Настройки сложности\n");
            printf("8 - Выход\n");
            printf("Ваш выбор: ");

            if (scanf_s("%d", &menu_choice) != 1) {
                while (getchar() != '\n');
                continue;
            }
            while (getchar() != '\n');

            if (menu_choice == 1 || menu_choice == 2) {
                char name[MAXNAME] = "Player";
                printf("Введите имя: ");
```

```
                fgets(name, MAXNAME, stdin);
                name[strcspn(name, "\n")] = 0;

                int px = startX, py = startY, win = 0;
                clock_t start_t = clock();

                system("cls");

                while (!win) {
                    printMazeSingle(px, py, exitX, exitY);
                    if (menu_choice == 2) showRouteHint(px, py);
                    printf("\nУправление: w, a, s, d (выход -
q)\n");
                    char key = _getch();
                    if (key == 'q' || key == 'Q') break;

                    int nx = px, ny = py;
                    if (key == 'w' || key == 'W') ny--;
                    else if (key == 's' || key == 'S') ny++;
                    else if (key == 'a' || key == 'A') nx--;
                    else if (key == 'd' || key == 'D') nx++;

                    if (manualIsValid(nx, ny)) { px = nx; py = ny;
}

                    if (px == exitX && py == exitY) {
                        system("cls");
                        printf("ТЫ ВЫБРАЛСЯ!\n");
                        win = 1;
                        clock_t end_t = clock();
                        double seconds = (double)(end_t - start_t)
/ CLOCKS_PER_SEC;
                        saveTimeToFile(name,       "пользователь",
seconds);
                        printf("Время: %.2f сек\n", seconds);
                        printf("Нажмите любую клавишу...\n");
                        _getch();
```

```
                }
            }
        }
        else if (menu_choice == 3) {
            makeFullDfsPath(startX, startY, exitX, exitY);
            moveThroughPath(exitX, exitY);
            printf("\nКомпьютер прошел лабиринт.\nНажмите
клавишу...");
            _getch();
        }
        else if (menu_choice == 4) {
            gods_passage(startX, startY, exitX, exitY);
            printf("\nНажмите клавишу...");
            _getch();
        }
        else if (menu_choice == 5) {
            showResultsFromFile();
        }
        else if (menu_choice == 6) {
            competitiveMode();
            startX = 1; startY = 1;
            exitX = G_WIDTH - 2; exitY = G_HEIGHT - 1;
            initMaze(startX, startY, exitX, exitY);
            makeFullDfsPath(startX, startY, exitX, exitY);
        }
        else if (menu_choice == 7) {
            freeMaze();
            freeVisited();
            selectDifficulty();
            if (!allocMaze() || !allocVisited()) return 1;

            startX = 1; startY = 1;
            exitX = G_WIDTH - 2; exitY = G_HEIGHT - 1;

            initMaze(startX, startY, exitX, exitY);
            makeFullDfsPath(startX, startY, exitX, exitY);
```

```
        }
        else if (menu_choice == 8) {
            running = 0;
        }
    }

    freeMaze();
    freeVisited();
    return 0;
}
```

Функция competitiveMode()

Реализует соревновательный режим. В цикле while обрабатывается ввод игрока и таймер бота:

```
void competitiveMode(void) {
    freeMaze();
    freeVisited();
    selectDifficulty();

    if (!allocMaze() || !allocVisited()) {
        printf("Ошибка выделения памяти.\n");
        return;
    }

    int startX = 1, startY = 1;
    int exitX = G_WIDTH - 2, exitY = G_HEIGHT - 1;
    int attempts = 0;
    do {
        initMaze(startX, startY, exitX, exitY);
        makeFullDfsPath(startX, startY, exitX, exitY);
        attempts++;
    } while (fullPathLen < 2 && attempts < 20);

    int px = startX, py = startY;
    int cx = startX, cy = startY;
    int botIndex = 0;
```

```
        clock_t lastBotMove = clock();
        int playerFinished = 0;
        int botFinished = 0;

        system("cls");

        while (!playerFinished && !botFinished) {
            printMazeCompetitive(px, py, cx, cy, exitX, exitY);
            printf("СОРЕВНОВАНИЕ!\n");
            printf("@ - Игрок | & - Бот\n");
            printf("Управление: w, a, s, d (без Enter)\n");
            if (_kbhit()) {
                char key = _getch();
                int nx = px, ny = py;
                if (key == 'w' || key == 'W') ny--;
                else if (key == 's' || key == 'S') ny++;
                else if (key == 'a' || key == 'A') nx--;
                else if (key == 'd' || key == 'D') nx++;

                if (manualIsValid(nx, ny)) { px = nx; py = ny; }
                if (px == exitX && py == exitY) playerFinished = 1;
            }
            clock_t now = clock();
            double ms = (double)(now - lastBotMove) * 1000.0 /
CLOCKS_PER_SEC;
            if (ms >= BOT_DELAY_MS && botIndex < fullPathLen) {
                cx = fullPath[botIndex].x;
                cy = fullPath[botIndex].y;
                botIndex++;
                lastBotMove = now;
                if (cx == exitX && cy == exitY) botFinished = 1;
            }
        }

        printMazeCompetitive(px, py, cx, cy, exitX, exitY);
        printf("\n");
```

```
    if (playerFinished && !botFinished) {
        printf("ПОБЕДА! Ты обогнал компьютер.\n");
        playerScore++;
    }
    else if (!playerFinished && botFinished) {
        printf("ПОРАЖЕНИЕ. Компьютер пришел первым.\n");
        botScore++;
    }
    else {
        printf("НИЧЬЯ! Вы пришли одновременно.\n");
        playerScore++; botScore++;
    }
    saveScore();

    printf("\nНажмите любую клавишу для выхода в меню...\n");
    _getch();
}
```

3.2 Функции модуля генерации (Maze)

Файл maze.c содержит функции для работы с памятью и алгоритмом генерации.

Функция allocMaze()

Выделяет динамическую память под массив лабиринта.

```
int allocMaze(void) {
    maze = (int**)malloc(HEIGHT * sizeof(int*));
    if (!maze) return 0;
    for (int i = 0; i < HEIGHT; i++) {
        maze[i] = (int*)malloc(WIDTH * sizeof(int));
        if (!maze[i]) return 0;
    }
    return 1;
}
```

Функция generateMaze(int x, int y)

Реализует алгоритм Randomized DFS. Использует массив смещений для

прохода через одну клетку (создавая стены).

```
static int dx[] = { 2, -2, 0, 0 };
static int dy[] = { 0, 0, 2, -2 };

static void generateMaze(int x, int y) {
    int dirs[4] = { 0,1,2,3 };
    shuffle(dirs, 4); // Случайное перемешивание направлений
    for (int i = 0; i < 4; i++) {
        int nx = x + dx[dirs[i]];
        int ny = y + dy[dirs[i]];
        // Если соседняя клетка - стена (не посещена)
        if (isInside(nx, ny) && maze[ny][nx] == 1) {
            maze[ny][nx] = 0; // Делаем проход
            // Ломаем стену между текущей и новой клеткой
            maze[y + dy[dirs[i]] / 2][x + dx[dirs[i]] / 2] = 0;
            generateMaze(nx, ny); // Рекурсивный вызов
        }
    }
}
```

Функция initMaze

Заполняет лабиринт стенами перед генерацией:

```
void initMaze(int startX, int startY, int exitX, int exitY) {
    for (int i = 0; i < HEIGHT; i++)
        for (int j = 0; j < WIDTH; j++)
            maze[i][j] = 1; // 1 - стена
    maze[startY][startX] = 0;
    generateMaze(startX, startY);
    maze[exitY][exitX] = 0;
}
```

3.3 Функции модуля настроек (Difficulty)

Файл difficulty.c отвечает за параметры сложности.

Глобальные переменные настроек:

```
int G_WIDTH = 31;
```

```
int G_HEIGHT = 17;
int BOT_DELAY_MS = 900;
```

Функция selectDifficulty()

Изменяет глобальные переменные в зависимости от выбора пользователя.

```
void selectDifficulty(void) {
    int choice;
    printf("Выберите сложность:\n");
    printf("1 - Легко  (маленький лабиринт, медленный бот)\n");
    printf("2 - Норма (средний лабиринт, средний бот)\n");
    printf("3 - Жесть (большой лабиринт, быстрый бот)\n");
    printf("4 - мне повезёт\n");
    printf("Ваш выбор: ");
    if (scanf("%d", &choice) != 1) {
        choice = 1;
    }
    while (getchar() != '\n');

    switch (choice) {
    case 1:
        G_WIDTH = 31; G_HEIGHT = 17; BOT_DELAY_MS = 900;
        break;
    case 2:
        G_WIDTH = 41; G_HEIGHT = 21; BOT_DELAY_MS = 600;
        break;
    case 3:
        G_WIDTH = 61; G_HEIGHT = 31; BOT_DELAY_MS = 250;
        break;
    case 4:
        printLuckyAscii();
        G_WIDTH = 61; G_HEIGHT = 31; BOT_DELAY_MS = 250;
        break;
    default:
        G_WIDTH = 41; G_HEIGHT = 21; BOT_DELAY_MS = 600;
        break;
    }
```

```
    printf("Сложность: %d x %d, задержка бота %d мс\n",
        G_WIDTH, G_HEIGHT, BOT_DELAY_MS);
    Sleep(1200);
}
```

3.4 Функции модуля визуальных эффектов (Gods)

Файл gods.c содержит "секретную" функцию.

Функция gods_passage

Запускает специальный режим с анимацией.

```
    void gods_passage(int startX, int startY, int exitX, int
exitY) {
        // ... Локальная логика поиска пути и отрисовки ...
        // Использует специальные текстовые вставки и задержки
Sleep()
    }
```

3.5 Элементы управления главным меню

Управление осуществляется в функции main через конструкцию switch (аналог кнопок формы).

Пункт 1 и 2**:** Одиночная игра

Запускает цикл ручного управления.

```
            while (getchar() != '\n');

            if (menu_choice == 1 || menu_choice == 2) {
                char name[MAXNAME] = "Player";
                printf("Введите имя: ");
                fgets(name, MAXNAME, stdin);
                name[strcspn(name, "\n")] = 0;

                int px = startX, py = startY, win = 0;
                clock_t start_t = clock();

                system("cls");

                while (!win) {
                    printMazeSingle(px, py, exitX, exitY);
                    if (menu_choice == 2) showRouteHint(px, py);
                    printf("\nУправление: w, a, s, d (выход -
q)\n");
                    char key = _getch();
```

```
                    if (key == 'q' || key == 'Q') break;

                    int nx = px, ny = py;
                    if (key == 'w' || key == 'W') ny--;
                    else if (key == 's' || key == 'S') ny++;
                    else if (key == 'a' || key == 'A') nx--;
                    else if (key == 'd' || key == 'D') nx++;

                    if (manualIsValid(nx, ny)) { px = nx; py = ny;
}

                    if (px == exitX && py == exitY) {
                        system("cls");
                        printf("ТЫ ВЫБРАЛСЯ!\n");
                        win = 1;
                        clock_t end_t = clock();
                        double seconds = (double)(end_t - start_t)
/ CLOCKS_PER_SEC;
                        saveTimeToFile(name,       "пользователь",
seconds);
                        printf("Время: %.2f сек\n", seconds);
                        printf("Нажмите любую клавишу...\n");
                        _getch();
                    }
                }
            }
```

Пункт 3: Автоматическое решение (aka. Пройти за меня, мне лень)

Запускает визуализацию прохода компьютером.

```
    else if (menu_choice == 3) {
        makeFullDfsPath(startX, startY, exitX, exitY);
        moveThroughPath(exitX, exitY);
        printf("\nКомпьютер       прошел       лабиринт.\nНажмите
клавишу...");
        _getch();
    }
```

Пункт 4: Секретный секрет

Запускает секретный режим игры.

```
else if (menu_choice == 4) {
     gods_passage(startX, startY, exitX, exitY);
     printf("\nНажмите клавишу...");
     _getch();
 }
```

Пункт 5: Результаты

Показывает результаты прохождения лабиринта или счёт.

```
else if (menu_choice == 5) {
    showResultsFromFile();
}
```

Пункт 6: Соревнование

Запускает функцию competitiveMode, затем пересоздает лабиринт.

```
        else if (menu_choice == 6) {
            competitiveMode();
            freeMaze(); // Очистка старой памяти
            // ... Пересоздание лабиринта ...
        }
```

Пункт 7: Смена сложности

Вызывает меню настроек и пересоздает динамические массивы.

```
    else if (menu_choice == 7) {
       freeMaze(); freeVisited();
       selectDifficulty(); // Выбор новых параметров
       if (!allocMaze() || !allocVisited()) { /*...*/ }
       // ... Генерация нового лабиринта ...
    }
```

3.6 Структуры данных

Структура Point

Определена в main.c, используется для хранения координат:

```
typedef struct { int x, y; } Point;
```

Массив maze

Определен в maze.c и maze.h. Хранит карту уровня:

```
extern int** maze;
```

3.7 Элементы лабиринта

Визуализация реализована в функциях printMazeSingle и printMazeCompetitive.

```
void printMazeSingle(int px, int py, int ex, int ey) {
    system("cls"); // Очистка экрана
    for (int i = 0; i < HEIGHT; i++) {
        for (int j = 0; j < WIDTH; j++) {
            if (i == py && j == px) printf("@");       // Игрок
            else if (i == ey && j == ex) printf("E");  // Выход
            else if (maze[i][j] == 1) printf("#");     // Стена
            else printf(" ");                          // Проход
        }
        printf("\n");
    }
}
```

В режиме соревнования добавляется символ *&*:

```
            else if (i == cy && j == cx) printf("&");  // Бот
```

4. Руководство пользователя

Программа предназначена для прохождения лабиринта. Она имеет понятный интерфейс и интуитивно понятное управление.

Задается сложность (размер лабиринта и скорость соперника) (рис. 6):

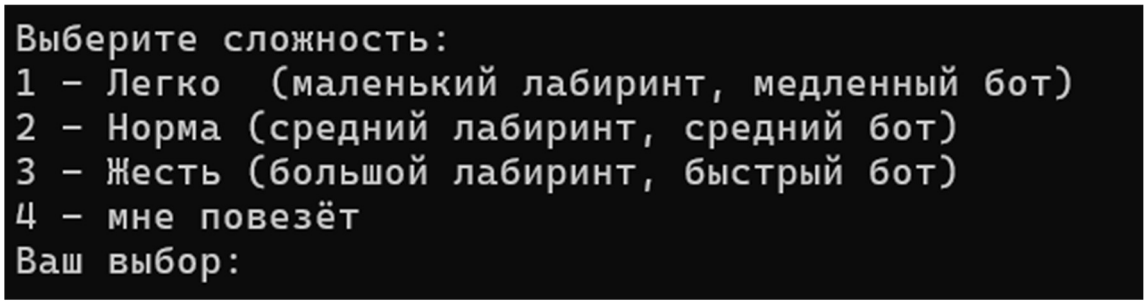

Рисунок 6 — Выбор сложности

Далее перед пользователем отображается меню (рис. 7) программы.

1. Одиночная игра;
2. Игра с подсказками;
3. Пройти за меня мне лень;
4. Секретный режим;
5. Результаты;
6. Соревнование;
7. Настройка сложности;
8. Выход.

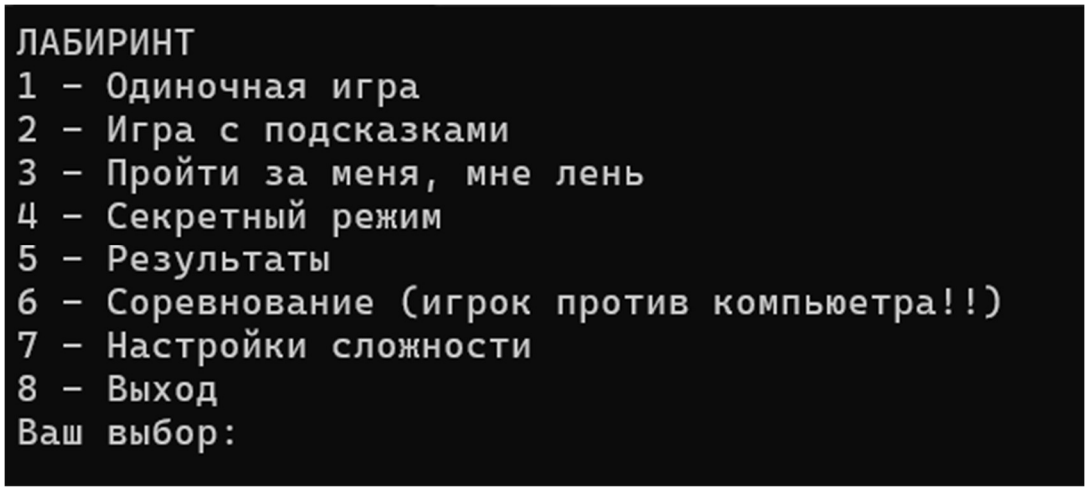

Рисунок 7 — Меню программы

Нажав «1» и подтвердив свой выбор («Enter»), пользователя спросят его имя, потом перед ним появляется лабиринт (рис. 8), который ему предстоит пройти, передвигаясь на клавиши «w», «a», «s», «d» («вверх», «влево», «вниз», «вправо» соответственно).

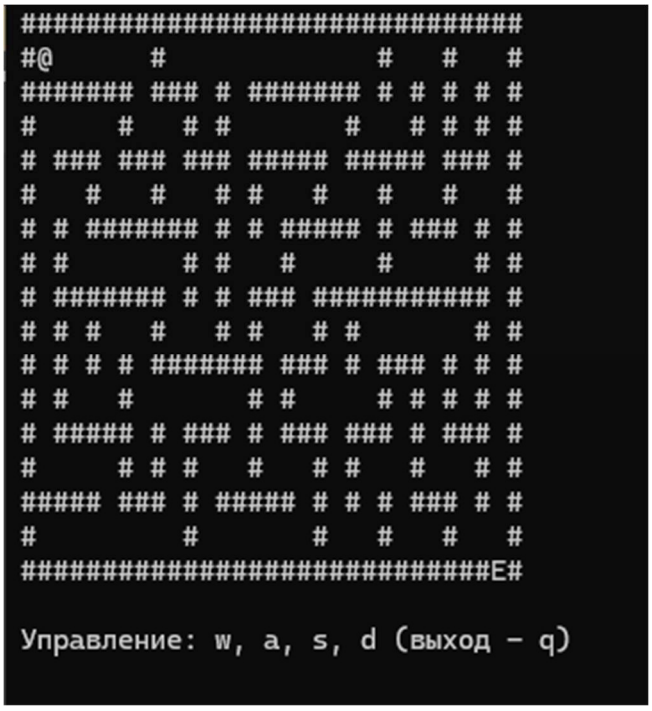

Рисунок 8 — Одиночная игра

Пройдя лабиринт программа поздравит игрока с завершением и напишет его время прохождения (рис. 9). После нажатия любой клавиши, пользователя вернет в меню.

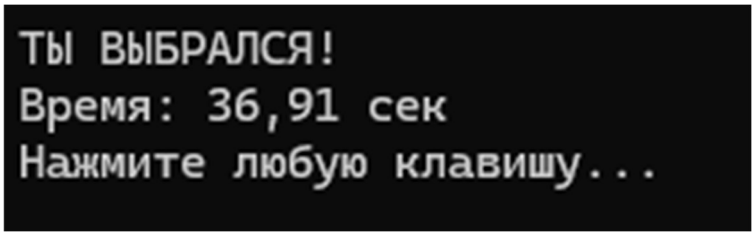

Рисунок 9 — Поздравления

Нажав «2» и подтвердив свой выбор («Enter»), пользователя спросят его имя, потом перед ним появляется лабиринт, который ему предстоит пройти, но с подсказками (рис. 10)!

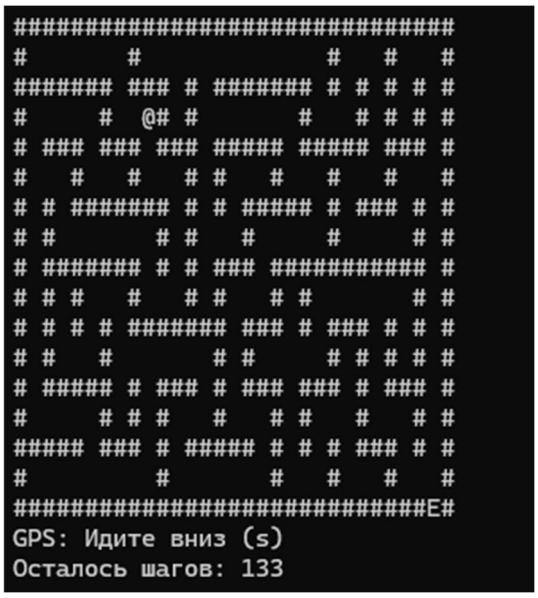

Рисунок 10 — Подсказки

Пройдя лабиринт, пользователя также поздравят и вернут в меню.

Нажав «3» и подтвердив свой выбор («Enter»), за пользователя в реальном времени пройдут (рис. 11) лабиринт. Лень – главный двигатель прогресса!

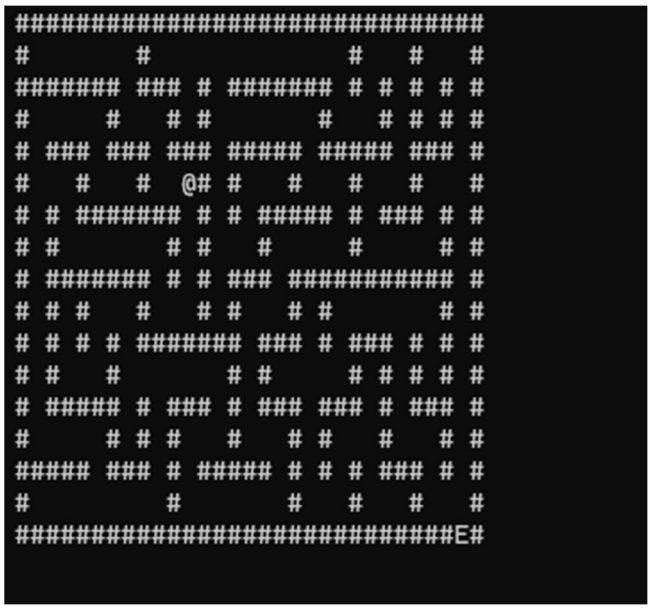

Рисунок 11 — Прохождения за пользователя

Нажав «4» и подтвердив свой выбор («Enter»), перед пользователем появится лабиринт, который будет проходится компьютером, параллельно справа будут появляться слова песни (рис. 12) «Gods of Rock N Roll» в исполнении Оззи Осборна.

Рисунок 12 — Слова песни

А в конце появится изображение гитары с надписью «OZZY FOREVER», как дань уважения музыканту(рис. 13).

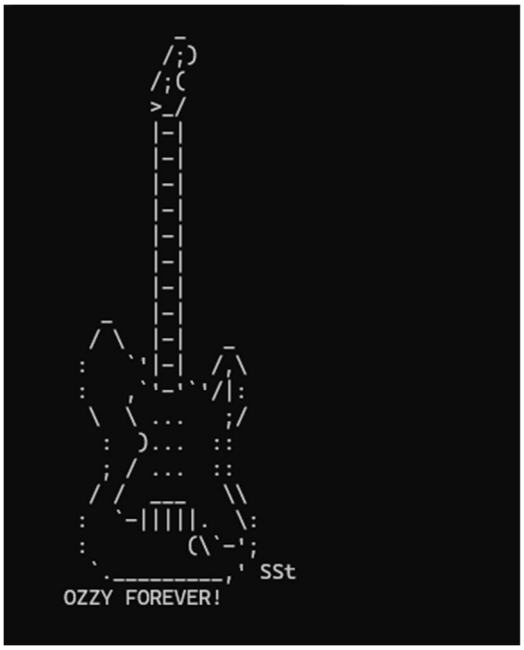

Рисунок 13 — Гитара

Нажав «5» и подтвердив свой выбор («Enter»), пользователю предоставится результаты прохождения лабиринта, который хранятся в файле «result.exe» (рис. 14).

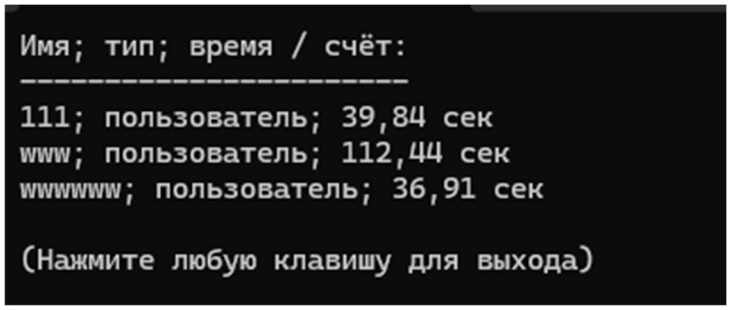

Рисунок 14 — Результаты

Нажав «6» и подтвердив свой выбор («Enter»), у пользователя еще раз спросят выбор сложности, после которого начнется соревнование (рис. 15) игрока с компьютером. По результатам сражения, будет выставлен счет (рис. 16), а также обновится файл с результатом (рис. 17), там появится счёт.

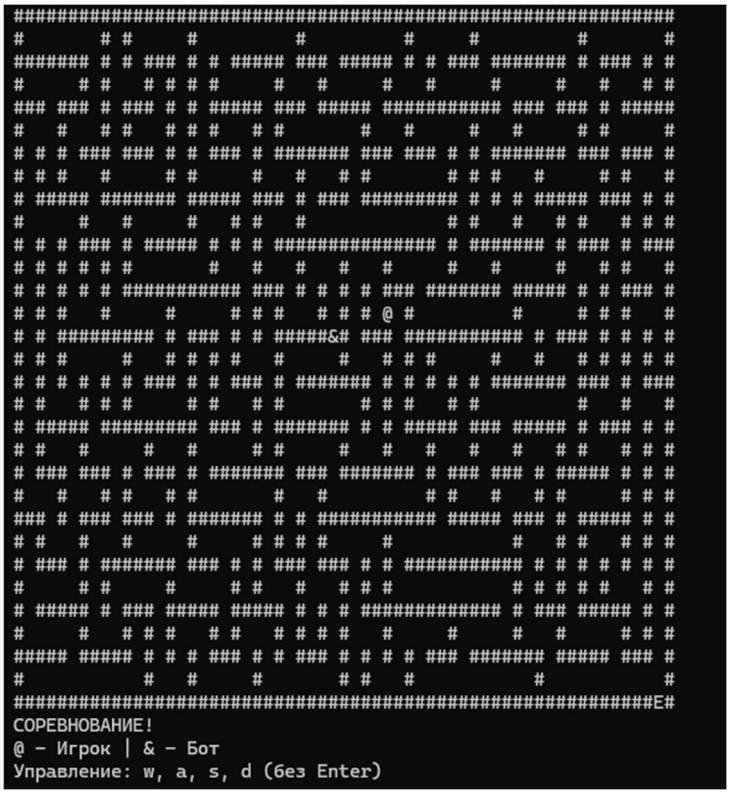

Рисунок 15 — Соревнование

Рисунок 16 — Результат соревнования

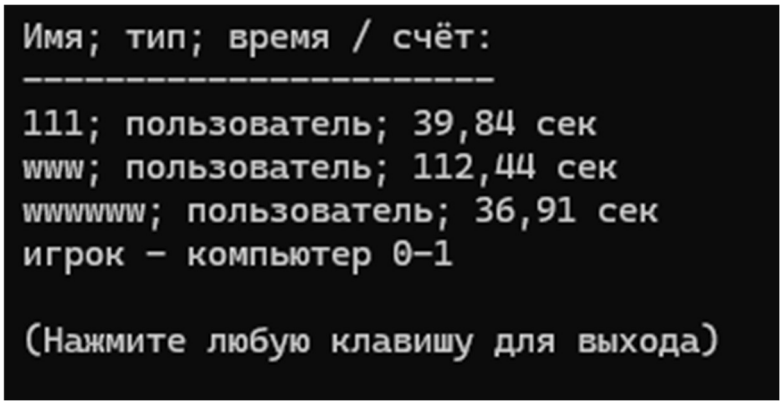

Рисунок 17 — Счет

Нажав «7» и подтвердив свой выбор («Enter»), пользователь может поменять сложность (рис. 18)

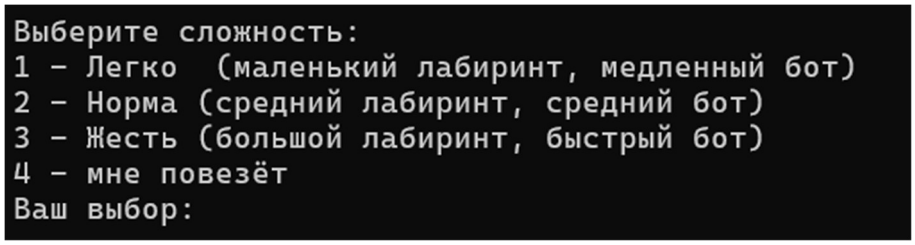

Рисунок 18 — Выбор сложности

Нажав «8» и подтвердив свой выбор («Enter»), программа завершится.

Заключение

В процессе выполнения данной курсовой работы были закреплены навыки процедурного программирования на языке C и освоены методы работы с динамическими структурами данных. Изучен и реализован алгоритм поиска в глубину (Randomized DFS) для генерации идеальных лабиринтов, а также алгоритм DFS для автоматического поиска пути.

Было разработано полноценное консольное приложение, позволяющее пользователю:

- Генерировать уникальные лабиринты различной сложности с использованием алгоритма рандомизированного поиска в глубину;
- Проходить лабиринты в ручном режиме с помощью клавиатуры;
- Использовать систему динамических подсказок, основанную на расчете оптимального маршрута;
- Наблюдать за автоматическим прохождением лабиринта компьютерным агентом;
- Соревноваться с искусственным интеллектом в режиме реального времени, где сложность и скорость бота настраиваются динамически.

В ходе работы были решены задачи эффективного управления памятью при создании и пересоздании игрового поля, реализована система интерактивного меню и визуализации игрового процесса средствами символьной графики. Полученный программный продукт демонстрирует стабильную работу, имеет интуитивно понятный интерфейс и соответствует всем поставленным в задании требованиям.

Список используемых источников

[доделать]

Приложение А.
Листинг программы

A.1 Main.c

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <locale.h>
#include <windows.h>
#include <time.h>
#include <io.h>

#include "difficulty.h"
#include "maze.h"
#include "gods.h"

#define MAXNAME 67

typedef struct { int x, y; } Point;

int playerScore = 0;
int botScore = 0;

Point path[40000];
int pathLength = 0;

bool** visited = NULL;

Point fullPath[40000];
int fullPathLen = 0;

void saveScore(void);

int allocVisited(void) {
    visited = (bool**)malloc(G_HEIGHT * sizeof(bool*));
    if (!visited) return 0;
    for (int i = 0; i < G_HEIGHT; i++) {
        visited[i] = (bool*)malloc(G_WIDTH * sizeof(bool));
        if (!visited[i]) return 0;
    }
    return 1;
```

```
    }

    void freeVisited(void) {
        if (!visited) return;
        for (int i = 0; i < G_HEIGHT; i++)
            free(visited[i]);
        free(visited);
        visited = NULL;
    }

    void clearVisited() {
        for (int i = 0; i < G_HEIGHT; i++)
            for (int j = 0; j < G_WIDTH; j++)
                visited[i][j] = false;
    }

    bool manualIsValid(int x, int y) {
        return x >= 0 && x < G_WIDTH && y >= 0 && y < G_HEIGHT &&
maze[y][x] == 0;
    }

    bool isValid(int x, int y) {
        return x > 0 && x < G_WIDTH - 1 && y > 0 && y < G_HEIGHT - 1
&&
            maze[y][x] == 0 && !visited[y][x];
    }

    bool dfs(int x, int y, int tx, int ty) {
        if (x == tx && y == ty) {
            Point temp; temp.x = x; temp.y = y;
            path[pathLength++] = temp;
            return true;
        }
        if (!isValid(x, y)) return false;
        visited[y][x] = true;
        Point temp; temp.x = x; temp.y = y;
        path[pathLength++] = temp;
        int mx[] = { 1,-1,0,0 }, my[] = { 0,0,1,-1 };
        for (int i = 0; i < 4; i++) {
            int nx = x + mx[i], ny = y + my[i];
            if (dfs(nx, ny, tx, ty)) return true;
        }
        pathLength--;
        return false;
    }
```

```
  void printMazeSingle(int px, int py, int ex, int ey) {
      HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
      COORD pos = { 0, 0 };
      SetConsoleCursorPosition(hConsole, pos);

      for (int i = 0; i < G_HEIGHT; i++) {
          for (int j = 0; j < G_WIDTH; j++) {
              if (i == py && j == px)
                  printf("@");
              else if (i == ey && j == ex)
                  printf("E");
              else if (maze[i][j] == 1)
                  printf("#");
              else
                  printf(" ");
          }
          printf("\n");
      }
  }

  void printMazeCompetitive(int px, int py, int cx, int cy, int ex,
int ey) {
      HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
      COORD pos = { 0, 0 };
      SetConsoleCursorPosition(hConsole, pos);

      for (int i = 0; i < G_HEIGHT; i++) {
          for (int j = 0; j < G_WIDTH; j++) {
              if (i == py && j == px && i == cy && j == cx)
                  printf("*");
              else if (i == py && j == px)
                  printf("@");
              else if (i == cy && j == cx)
                  printf("&");
              else if (i == ey && j == ex)
                  printf("E");
              else if (maze[i][j] == 1)
                  printf("#");
              else
                  printf(" ");
          }
          printf("\n");
      }
```

```
    }

    void moveThroughPath(int exitX, int exitY) {
        system("cls");
        for (int i = 0; i < pathLength; i++) {
            printMazeSingle(path[i].x, path[i].y, exitX, exitY);
            Sleep(100);
        }
    }

    void makeFullDfsPath(int startX, int startY, int exitX, int exitY)
{
        clearVisited();
        pathLength = 0;
        if (dfs(startX, startY, exitX, exitY)) {
            fullPathLen = pathLength;
            for (int i = 0; i < fullPathLen; i++) fullPath[i] =
path[i];
        }
        else {
            fullPathLen = 0;
        }
    }

    void showRouteHint(int px, int py) {
        int i;
        for (i = 0; i < fullPathLen - 1; i++) {
            if (fullPath[i].x == px && fullPath[i].y == py) {
                int nx = fullPath[i + 1].x, ny = fullPath[i + 1].y;
                printf("GPS: Идите ");
                if (ny < py) printf("вверх (w)   \n");
                else if (ny > py) printf("вниз (s)    \n");
                else if (nx < px) printf("влево (a)   \n");
                else if (nx > px) printf("вправо (d)  \n");
                printf("Осталось шагов: %d     \n", fullPathLen - i -
1);
                return;
            }
        }
        printf("GPS: Вы ушли с маршрута!   \n");
        printf("                          \n");
    }

    void saveTimeToFile(const char* name, const char* who, double
seconds) {
```

```
        FILE* f = fopen("results.txt", "a");
        if (f) {
            fprintf(f, "%s; %s; %.2f сек\n", name, who, seconds);
            fclose(f);
        }
    }

    void saveScore(void) {
        FILE* f = fopen("results.txt", "a");
        if (f) {
            fprintf(f, "игрок - компьютер %d-%d\n", playerScore,
botScore);
            fclose(f);
        }
    }

    void showResultsFromFile(void) {
        FILE* f = fopen("results.txt", "r");
        char buf[200];
        system("cls");
        if (!f) printf("Файл результатов не найден\n");
        else {
            printf("Имя; тип; время / счёт:\n");
            printf("----------------------\n");
            while (fgets(buf, sizeof(buf), f)) printf("%s", buf);
            fclose(f);
            printf("\n(Нажмите любую клавишу для выхода)\n");
            _getch();
        }
    }

    void competitiveMode(void) {
        freeMaze();
        freeVisited();
        selectDifficulty();

        if (!allocMaze() || !allocVisited()) {
            printf("Ошибка выделения памяти.\n");
            return;
        }

        int startX = 1, startY = 1;
        int exitX = G_WIDTH - 2, exitY = G_HEIGHT - 1;
        int attempts = 0;
```

```
        do {
            initMaze(startX, startY, exitX, exitY);
            makeFullDfsPath(startX, startY, exitX, exitY);
            attempts++;
        } while (fullPathLen < 2 && attempts < 20);

        int px = startX, py = startY;
        int cx = startX, cy = startY;
        int botIndex = 0;
        clock_t lastBotMove = clock();
        int playerFinished = 0;
        int botFinished = 0;

        system("cls");

        while (!playerFinished && !botFinished) {
            printMazeCompetitive(px, py, cx, cy, exitX, exitY);
            printf("СОРЕВНОВАНИЕ!\n");
            printf("@ - Игрок | & - Бот\n");
            printf("Управление: w, a, s, d (без Enter)\n");
            if (_kbhit()) {
                char key = _getch();
                int nx = px, ny = py;
                if (key == 'w' || key == 'W') ny--;
                else if (key == 's' || key == 'S') ny++;
                else if (key == 'a' || key == 'A') nx--;
                else if (key == 'd' || key == 'D') nx++;

                if (manualIsValid(nx, ny)) { px = nx; py = ny; }
                if (px == exitX && py == exitY) playerFinished = 1;
            }
            clock_t now = clock();
            double ms = (double)(now - lastBotMove) * 1000.0 /
CLOCKS_PER_SEC;
            if (ms >= BOT_DELAY_MS && botIndex < fullPathLen) {
                cx = fullPath[botIndex].x;
                cy = fullPath[botIndex].y;
                botIndex++;
                lastBotMove = now;
                if (cx == exitX && cy == exitY) botFinished = 1;
            }
        }

        printMazeCompetitive(px, py, cx, cy, exitX, exitY);
```

```
    printf("\n");

    if (playerFinished && !botFinished) {
        printf("ПОБЕДА! Ты обогнал компьютер.\n");
        playerScore++;
    }
    else if (!playerFinished && botFinished) {
        printf("ПОРАЖЕНИЕ. Компьютер пришел первым.\n");
        botScore++;
    }
    else {
        printf("НИЧЬЯ! Вы пришли одновременно.\n");
        playerScore++; botScore++;
    }
    saveScore();

    printf("\nНажмите любую клавишу для выхода в меню...\n");
    _getch();
}

int main() {
    setlocale(LC_ALL, "Russian");
    srand((unsigned int)time(NULL));
    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_CURSOR_INFO cursorInfo;
    GetConsoleCursorInfo(hConsole, &cursorInfo);
    cursorInfo.bVisible = FALSE;
    SetConsoleCursorInfo(hConsole, &cursorInfo);

    selectDifficulty();

    if (!allocMaze() || !allocVisited()) {
        printf("Ошибка выделения памяти.\n");
        return 1;
    }

    int startX = 1, startY = 1;
    int exitX = G_WIDTH - 2, exitY = G_HEIGHT - 1;
    initMaze(startX, startY, exitX, exitY);
    makeFullDfsPath(startX, startY, exitX, exitY);

    int menu_choice, running = 1;

    while (running) {
```

```
system("cls");
printf("ЛАБИРИНТ\n");
printf("1 - Одиночная игра\n");
printf("2 - Игра с подсказками\n");
printf("3 - Пройти за меня, мне лень\n");
printf("4 - Секретный режим\n");
printf("5 - Результаты\n");
printf("6 - Соревнование (игрок против компьюетра!!)\n");
printf("7 - Настройки сложности\n");
printf("8 - Выход\n");
printf("Ваш выбор: ");

if (scanf_s("%d", &menu_choice) != 1) {
    while (getchar() != '\n');
    continue;
}
while (getchar() != '\n');

if (menu_choice == 1 || menu_choice == 2) {
    char name[MAXNAME] = "Player";
    printf("Введите имя: ");
    fgets(name, MAXNAME, stdin);
    name[strcspn(name, "\n")] = 0;

    int px = startX, py = startY, win = 0;
    clock_t start_t = clock();

    system("cls");

    while (!win) {
        printMazeSingle(px, py, exitX, exitY);
        if (menu_choice == 2) showRouteHint(px, py);
        printf("\nУправление: w, a, s, d (выход - q)\n");
        char key = _getch();
        if (key == 'q' || key == 'Q') break;

        int nx = px, ny = py;
        if (key == 'w' || key == 'W') ny--;
        else if (key == 's' || key == 'S') ny++;
        else if (key == 'a' || key == 'A') nx--;
        else if (key == 'd' || key == 'D') nx++;

        if (manualIsValid(nx, ny)) { px = nx; py = ny; }
```

```
                    if (px == exitX && py == exitY) {
                        system("cls");
                        printf("ТЫ ВЫБРАЛСЯ!\n");
                        win = 1;
                        clock_t end_t = clock();
                        double seconds = (double)(end_t - start_t) /
CLOCKS_PER_SEC;
                        saveTimeToFile(name,         "пользователь",
seconds);
                        printf("Время: %.2f сек\n", seconds);
                        printf("Нажмите любую клавишу...\n");
                        _getch();
                    }
                }
            }
            else if (menu_choice == 3) {
                makeFullDfsPath(startX, startY, exitX, exitY);
                moveThroughPath(exitX, exitY);
                printf("\nКомпьютер    прошел    лабиринт.\nНажмите
клавишу...");
                _getch();
            }
            else if (menu_choice == 4) {
                gods_passage(startX, startY, exitX, exitY);
                printf("\nНажмите клавишу...");
                _getch();
            }
            else if (menu_choice == 5) {
                showResultsFromFile();
            }
            else if (menu_choice == 6) {
                competitiveMode();
                startX = 1; startY = 1;
                exitX = G_WIDTH - 2; exitY = G_HEIGHT - 1;
                initMaze(startX, startY, exitX, exitY);
                makeFullDfsPath(startX, startY, exitX, exitY);
            }
            else if (menu_choice == 7) {
                freeMaze();
                freeVisited();
                selectDifficulty();
                if (!allocMaze() || !allocVisited()) return 1;

                startX = 1; startY = 1;
                exitX = G_WIDTH - 2; exitY = G_HEIGHT - 1;
```

```
            initMaze(startX, startY, exitX, exitY);
            makeFullDfsPath(startX, startY, exitX, exitY);
        }
        else if (menu_choice == 8) {
            running = 0;
        }
    }

    freeMaze();
    freeVisited();
    return 0;
}
```

A.2 Maze.h

```
#ifndef MAZE_H
#define MAZE_H

#include "difficulty.h"

#define WIDTH  (G_WIDTH)
#define HEIGHT (G_HEIGHT)

extern int** maze;

int  allocMaze(void);
void freeMaze(void);

void initMaze(int startX, int startY, int exitX, int exitY);

#endif
```

A.3 Maze.c

```
#include "maze.h"
#include <stdlib.h>
#include <time.h>

int** maze = NULL;

static int dx[] = { 2, -2, 0, 0 };
static int dy[] = { 0, 0, 2, -2 };

int allocMaze(void) {
    maze = (int**)malloc(HEIGHT * sizeof(int*));
    if (!maze) return 0;
    for (int i = 0; i < HEIGHT; i++) {
        maze[i] = (int*)malloc(WIDTH * sizeof(int));
        if (!maze[i]) return 0;
    }
    return 1;
}

void freeMaze(void) {
    if (!maze) return;
```

```
    for (int i = 0; i < HEIGHT; i++)
        free(maze[i]);
    free(maze);
    maze = NULL;
}

static int isInside(int x, int y) {
    return x > 0 && x < WIDTH - 1 && y > 0 && y < HEIGHT - 1;
}

static void shuffle(int* array, int n) {
    for (int i = n - 1; i > 0; i--) {
        int j = rand() % (i + 1);
        int temp = array[i]; array[i] = array[j]; array[j] = temp;
    }
}

static void generateMaze(int x, int y) {
    int dirs[4] = { 0,1,2,3 };
    shuffle(dirs, 4);
    for (int i = 0; i < 4; i++) {
        int nx = x + dx[dirs[i]], ny = y + dy[dirs[i]];
        if (isInside(nx, ny) && maze[ny][nx] == 1) {
            maze[ny][nx] = 0;
            maze[y + dy[dirs[i]] / 2][x + dx[dirs[i]] / 2] = 0;
            generateMaze(nx, ny);
        }
    }
}

void initMaze(int startX, int startY, int exitX, int exitY) {
    for (int i = 0; i < HEIGHT; i++)
        for (int j = 0; j < WIDTH; j++)
            maze[i][j] = 1;
    maze[startY][startX] = 0;
    generateMaze(startX, startY);
    maze[exitY][exitX] = 0;
}
```

A.4 Difficulty.h

```
#ifndef DIFFICULTY_H
#define DIFFICULTY_H
extern int G_WIDTH;
extern int G_HEIGHT;
extern int BOT_DELAY_MS;

void setEasyDifficulty(void);

void selectDifficulty(void);

#endif
```

A.5 Difficulty.c

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <windows.h>
#include "difficulty.h"

int G_WIDTH = 31;
int G_HEIGHT = 17;
int BOT_DELAY_MS = 900;

void setEasyDifficulty(void) {
    G_WIDTH = 31;
    G_HEIGHT = 17;
    BOT_DELAY_MS = 900;
}

static void printLuckyAscii(void) {
    printf(
        "  ,--.     ,--.\n"
        " (  O )   (  O )\n"
        "  `--'  \\  `--'\n"
        "         \\   _\n"
        "   >-.   /   /|\n"
        "      `-.__.'\n"
    );
    printf("\nнет\n");
    Sleep(2000);
}
```

```
void selectDifficulty(void) {
    int choice;
    printf("Выберите сложность:\n");
    printf("1 - Легко  (маленький лабиринт, медленный бот)\n");
    printf("2 - Норма (средний лабиринт, средний бот)\n");
    printf("3 - Жесть (большой лабиринт, быстрый бот)\n");
    printf("4 - мне повезёт\n");
    printf("Ваш выбор: ");
    if (scanf("%d", &choice) != 1) {
        choice = 1;
    }
    while (getchar() != '\n');

    switch (choice) {
    case 1:
        G_WIDTH = 31; G_HEIGHT = 17; BOT_DELAY_MS = 900;
        break;
    case 2:
        G_WIDTH = 41; G_HEIGHT = 21; BOT_DELAY_MS = 600;
        break;
    case 3:
        G_WIDTH = 61; G_HEIGHT = 31; BOT_DELAY_MS = 250;
        break;
    case 4:
        printLuckyAscii();
        G_WIDTH = 61; G_HEIGHT = 31; BOT_DELAY_MS = 250;
        break;
    default:
        G_WIDTH = 41; G_HEIGHT = 21; BOT_DELAY_MS = 600;
        break;
    }

    printf("Сложность: %d x %d, задержка бота %d мс\n",
        G_WIDTH, G_HEIGHT, BOT_DELAY_MS);
    Sleep(1200);
}
```

A.6 Gods.h

```
#ifndef GODS_H
#define GODS_H

void gods_passage(int startX, int startY, int exitX, int exitY);

#endif
```

A.7 Gods.c

```
#include "gods.h"
#include "maze.h"
#include <windows.h>
#include <stdio.h>
#include <stdbool.h>

typedef struct { int x, y; } Point;

const char* ozzy_lines[] = {
    "Sometimes I feel like I'm so empty",
    "Sometimes I feel like I'm alive",
    "So many voices talking to me",
    "Please God just let my mind survive",
    "I see the moon and tears are falling from my eyes",
    "Will I make it through the night?",
    "I guess tonight will be another nightmare just for me",
    "I found myself another God-shaped hole",
    "Will someone pray for me, pray to save my soul for me",
    "The only gods, the gods of rock'n'roll",
    "Believe me, don't try to understand",
    "Just watch the shadows creeping off the walls",
    "Stay with me, don't want to be alone",
    "I need you as another night time falls",
    "With all the songs and dances",
    "Still looking for the answers",
    "I guess I've lost my self control",
    "With all these crazy voices",
    "I'm running out of choices",
    "I need you now, the gods of rock'n'roll"
};
const int ozzy_count = sizeof(ozzy_lines) / sizeof(ozzy_lines[0]);

static void printOzzyAscii(void) {
```

```
    printf(
        "              _\n"
        "             /;)\n"
        "            /;(\n"
        "            >_/\n"
        "            |-|\n"
        "            |-|\n"
        "            |-|\n"
        "            |-|\n"
        "            |-|\n"
        "            |-|\n"
        "            |-|\n"
        "         _  |-|\n"
        "       / \\  |-|   _\n"
        "      :   `'|-|  /,\\\n"
        "      :   ,`'-'`'/|:\n"
        "       \\  \\ ...   ;/\n"
        "        :  )...  ::\n"
        "        ; / ...  ::\n"
        "       / /  ___    \\\\\\\n"
        "      :  `-|||||.  \\\:\n"
        "      :         (\\`-';\n"
        "       `._________,' SSt\n"
        "    OZZY FOREVER!\n"
    );
}

static bool isValidLocal(int x, int y, bool** visited) {
    return x > 0 && x < WIDTH - 1 &&
        y > 0 && y < HEIGHT - 1 &&
        maze[y][x] == 0 && !visited[y][x];
}

static bool dfsLocal(int x, int y, int tx, int ty,
    Point* path, int* pathLength, bool** visited) {
    if (x == tx && y == ty) {
        Point temp = { x, y };
        path[(*pathLength)++] = temp;
        return true;
    }
    if (!isValidLocal(x, y, visited)) return false;

    visited[y][x] = true;
    Point temp = { x, y };
```

```
    path[(*pathLength)++] = temp;

    int mx[] = { 1,-1,0,0 }, my[] = { 0,0,1,-1 };
    for (int i = 0; i < 4; i++) {
        int nx = x + mx[i], ny = y + my[i];
        if (dfsLocal(nx, ny, tx, ty, path, pathLength, visited))
            return true;
    }
    (*pathLength)--;
    return false;
}

void gods_passage(int startX, int startY, int exitX, int exitY) {
    Point* path = (Point*)malloc(WIDTH * HEIGHT * sizeof(Point));
    if (!path) return;

    bool** visited = (bool**)malloc(HEIGHT * sizeof(bool*));
    if (!visited) { free(path); return; }
    for (int i = 0; i < HEIGHT; i++) {
        visited[i] = (bool*)malloc(WIDTH * sizeof(bool));
        if (!visited[i]) return;
        for (int j = 0; j < WIDTH; j++)
            visited[i][j] = false;
    }

    int pathLength = 0;
    dfsLocal(startX, startY, exitX, exitY, path, &pathLength,
visited);

    for (int step = 0; step < pathLength; step++) {
        system("cls");
        for (int y = 0; y < HEIGHT; y++) {
            for (int x = 0; x < WIDTH; x++) {
                if (y == path[step].y && x == path[step].x)
                    printf("@");
                else if (y == exitY && x == exitX)
                    printf("E");
                else if (maze[y][x] == 1)
                    printf("#");
                else
                    printf(" ");
            }
            if (y == step && step < ozzy_count)
                printf("   %s", ozzy_lines[step]);
```

```
            printf("\n");
        }
        Sleep(900);
    }

    system("cls");
    printOzzyAscii();
    Sleep(3500);

    // очистка
    for (int i = 0; i < HEIGHT; i++)
        free(visited[i]);
    free(visited);
    free(path);
}
```