Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Пензенский государственный университет»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по курсу «Программирование»

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

Выполнил: ст. гр. 25ВВВ1

Горбатов А.Ю

Проверил: к.т.н. доцент

Генералова А.А

Пенза, 2026

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

_____ М.А. Митрохин

«_8_» _июня_ 20_16_ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование

Студенту Горбачеву Арсению Юрьевичу Группа 25ВВВ1

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

Исходные данные (технические требования) на проектирование

Игра „Шахматы". Программа осуществляет генерацию шахматной доски и фигур, интерфейс управления фигурами с помощью мыши, а также управление игрой со стороны компьютера. Очередность хода определяется случайно в начале каждой партии.

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 "8" июня 2026

Дата защиты проекта "27" мая 2026

Руководитель Генералова Д. А.

Задание получил "8" июня 2026 г.

Студент Горбатов Арсений Юрьевич

Оглавление

Введение 3

1. **Постановка задачи** 4

2. **Выбор решения** 4

3. **Описание разработки программы** 6

4. **Отладка и тестирование** 7

5. **Описание программы** 8

5.1 Схема функции main 8

5.2 Описание модулей 8

6. **Руководство пользователя** 12

6.1 Заставка 12

6.2 Правила игры 13

6.3 Игровое поле 13

6.4 Завершение игры 13

6.5 Выход из игры 13

Список используемых источников 16

Приложение А. Листинг программы 17

Введение

Разработка шахматного приложения на языке Си представляет собой отличную практику системного программирования: здесь сочетаются работа с графическим интерфейсом, четкая логика правил и оптимизация вычислений. Выбранный проект реализует графическую версию классической игры «Шахматы» с отображением доски и фигур, интерактивным управлением игрока, обработкой специальных правил и игрой против компьютера.

Цель проекта — создать мультимодульную шахматную программу на Windows с использованием Win32/GDI и частичной ассемблерной оптимизацией, демонстрирующую:

1. продуманную архитектуру игры;
2. работу с иерархическими структурами данных;
3. разделение на модули (main.c, gui.c, board.c, moves.c, bot.c, chess.h);
4. проверку легальности ходов и специальные правила (рокировка, взятие на проходе, превращение пешки);
5. реализацию простого ИИ и графического меню;
6. наличие заставки при запуске.

Язык Си выбран как основной инструмент из-за предсказуемости, скорости и полного контроля над памятью. Ассемблерная вставка применяется для ускорения ключевых операций и демонстрирует подход к низкоуровневой оптимизации. Такой проект позволяет отработать навыки проектирования алгоритмов, управления ресурсами и создания реального прикладного программного продукта.

1. Постановка задачи

Цель проекта: разработать графическую мультимодульную версию игры «Шахматы» на языке Си с использованием низкоуровневых модулей на Ассемблере для оптимизации критических участков.

Функционал проекта:

1. Отрисовка шахматной доски 8×8 и фигур в окне Win32/GDI.
2. Управление ходами игрока мышью и поддержка клавиатурных команд для перезапуска/меню.
3. Ход компьютера с простым логичным алгоритмом.
4. Проверка легальности ходов, шаха и мата; критические проверки могут выполняться с ассемблерной оптимизацией.
5. Графическая заставка при запуске программы.
6. Модульная структура проекта с разделением на файлы main.c, gui.c, board.c, moves.c, bot.c и заголовочный файл chess.h.
7. Платформа: Windows, сборка возможна через Microsoft Visual Studio x86.

2. Выбор решения

В процессе разработки проекта была выбрана среда Microsoft Visual Studio 2026. Она предоставляет удобные инструменты для создания, отладки и оптимизации графического шахматного приложения на языке Си.

Visual Studio также подходит для работы с ассемблерной вставкой, которая в этом проекте применяется для ускорения критических проверок шахматных правил и некоторых вычислений. Это демонстрирует практическое применение низкоуровневого программирования в системном приложении.

Проект структурирован с использованием многомодульности: main.c, gui.c, board.c, moves.c, bot.c и заголовочные файлы в include. Такое разделение

логически разделяет интерфейс, логику доски, правила ходов и поведение компьютера, что делает код более поддерживаемым.

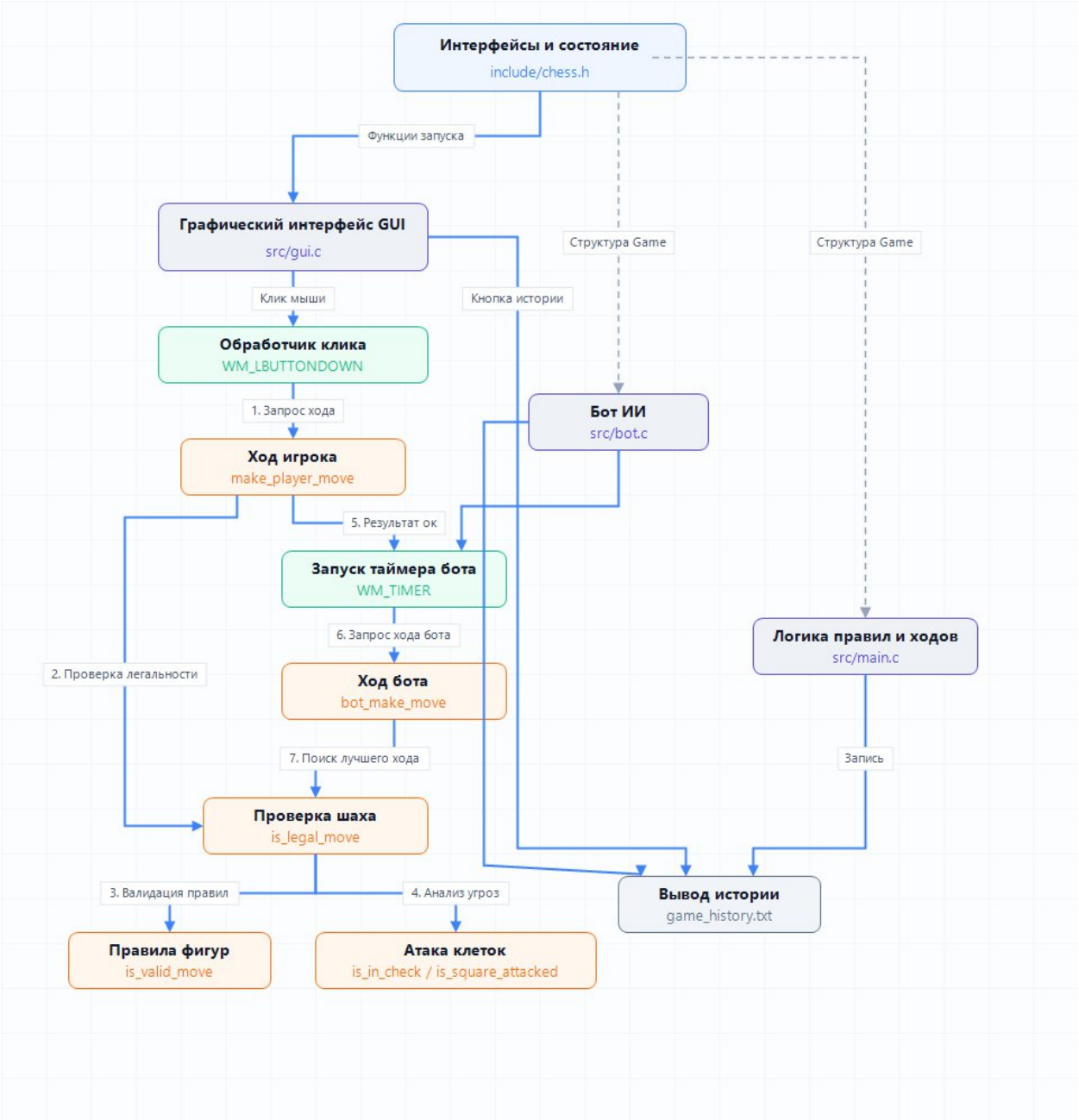

Рисунок 1. Схема многомодульности программы.

Пользовательский интерфейс реализован в графическом окне Win32/GDI. При запуске выводится заставка с надписью и кнопкой «Играть». Управление осуществляется мышью для выбора клетки и фигуры, а также клавиатурными командами для перезапуска и других команд. В приложении реализованы:

1. отрисовка шахматной доски 8×8 и шахматных фигур;
2. проверка легальности ходов;
3. поддержка специальных правил (рокировка, взятие на проходе, превращение пешки);
4. игра против компьютера с простым логическим алгоритмом хода.

3. Описание разработки программы

Для реализации был выбран нисходящий метод разработки, потому что он позволяет сначала определить общую структуру программы, а затем постепенно детализировать её компоненты.

При запуске программы на экран выводится графическая заставка с заголовком и кнопкой «Играть», после чего открывается главное окно с шахматной доской и статусной строкой.

Программа разделена на несколько модулей, что позволяет изменять одну часть, не затрагивая остальные:

1. **main.c** – главный файл программы, который инициализирует и запускает приложение.
2. **gui.c** – модуль интерфейса: отрисовка окна, доски, статуса, обработка кликов мыши и клавиатуры, отображение заставки.
3. **board.c** – модуль данных шахматной доски: хранение позицию, преобразование координат, проверка состояния.
4. **moves.c** – модуль правил ходов: проверка легальности, применение ходов и обработка специальных шахматных правил.
5. **bot.c** – модуль поведения компьютера: алгоритм выбора хода для противника.
6. **chess.h** – заголовочный файл с определениями структур, перечислений и прототипами функций.
7. **asm_util.c** – модуль низкоуровневых функций на ассемблере, например, для преобразования координат и ускоренных проверок.

Ход компьютера реализован простым логичным алгоритмом: сначала учитываются приоритетные ходы, затем случайные свободные варианты. Такой подход делает игру наглядной и удобной для реализации.

4. Отладка и тестирование

Для проекта использовалась среда разработки на Windows с компилятором GCC (MinGW-w64) и утилитами сборки, обеспечивающими стабильную работу с кодом на C и интеграцию с Win32-библиотеками. Отладка проводилась через точки останова, поэтапную трассировку исполнения и анализ значений переменных.

Тестирование выполнялось в процессе разработки и после завершения реализации.

На рисунке показан пример использования точек останова при отладке main-файла.

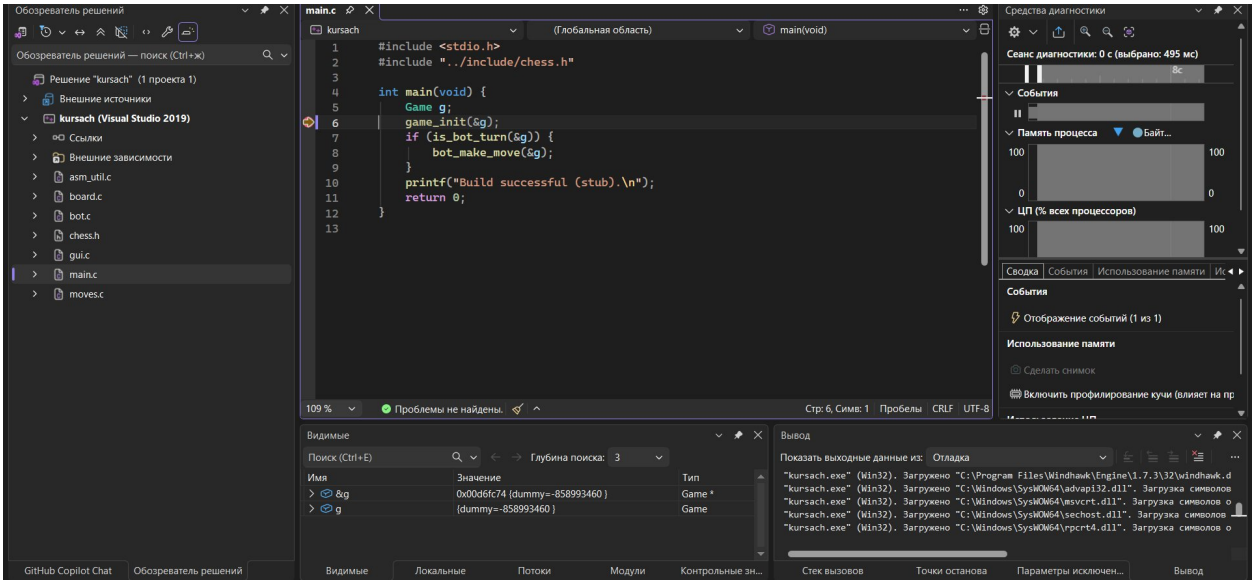

Рисунок 2. Использование точек останова

5. Описание программы

5.1 Схема функции main

На рисунке представлена схема функции main.

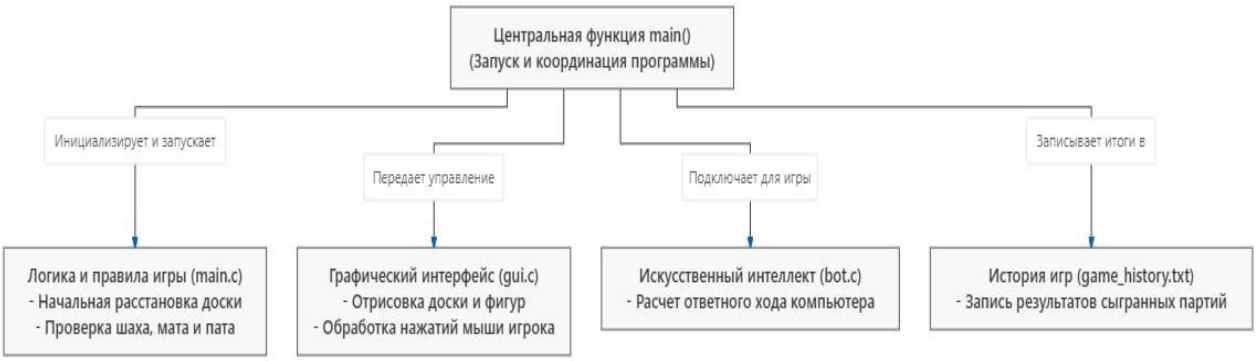

Рисунок 3. Схема функции main

5.2 Описание модулей

1. main.c — главный файл программы, который инициализирует и запускает приложение. Он содержит точку входа (main для консольной версии и WinMain для графической с Win32 API). В нем сосредоточена инициализация новой партии (функция game_init), которая расставляет фигуры на доске и случайным образом определяет цвет игрока. Здесь же реализована вся математическая проверка правил ходов фигур (is_valid_move) – пешки, ладьи, коня, слона, ферзя и короля. Также в файле находятся алгоритмы обнаружения шаха королю (is_in_check), проверки на мат или пат (has_legal_moves) и функции для работы с файлом истории сыгранных матчей (game_history.txt).

2. gui.c — модуль графического интерфейса пользователя. Построен целиком на Win32 API. Создает главное окно приложения, обрабатывает системные сообщения Windows (отрисовку экрана, таймеры, клики мыши). Содержит реализацию стартовой заставки с кнопками начала игры и просмотра архива матчей. Отслеживает нажатия мыши по доске, преобразует экранные пиксели в координаты ячеек, подсвечивает выбранные фигуры и отрисовывает их на экране (через Unicode-шрифты либо внешние растровые

изображения). Модуль также управляет фоновым таймером, который с задержкой запускает ход компьютера, делая игру плавной.

3. board.c — модуль данных шахматной доски. В текущей версии проекта служит структурным файлом-заготовкой для дальнейшего расширения проекта. Его архитектурная роль заключается в изоляции хранения массива доски от логики интерфейса, обеспечении функций конвертации координат (например, перевод текстового обозначения клетки вроде "e2" в индексы строки и столбца матрицы 8x8) и проверки выхода координат за границы доски.

4. moves.c — модуль правил ходов. Также является вспомогательным файлом-заглушкой, созданным для будущей разгрузки главного файла main.c. В нем планируется разместить функции детальной генерации списка возможных ходов для каждой фигуры, а также реализацию сложных шахматных правил, таких как рокировка (длинная и короткая), взятие пешки на проходе и превращение пешки в более сильную фигуру при достижении противоположного края доски.

5. bot.c — модуль поведения компьютера (ИИ). Функция bot_make_move ищет на текущей доске все теоретически доступные для компьютера ходы, фильтрует их, отсекая нелегальные (которые ставят собственного короля под шах), и делает выбор хода. Выбор основан на оценке ценности фигур: пешка оценивается в 10 очков, конь и слон — в 30, ладья — в 50, ферзь — в 90. Бот автоматически понимает, за какой цвет ему играть (белые или черные), в зависимости от того, какой цвет случайным образом выпал человеку при старте.

6. chess.h — глобальный заголовочный файл проекта. Объединяет константы (размер доски 8x8, максимальное число ходов в партии), перечисление исходов игры GameResult (победа белых, победа черных,

ничья), и структуру Game, хранящую состояние текущей партии (двумерный массив board, флаг очереди хода white_turn, флаг стороны игрока player_is_white, историю ходов и их счетчик). Также содержит прототипы всех функций для сборки модулей.

7. asm_util.c — модуль низкоуровневых функций на ассемблере. Содержит функцию asm_abs для быстрого нахождения абсолютного значения (модуля) числа. Используется при расчете расстояний между клетками (по горизонтали, вертикали или диагонали) во время проверки правил перемещения фигур. Оптимизация выполнена с помощью ассемблерной вставки под синтаксис компиляторов GCC/MinGW и MSVC.

Функция asm_abs принимает целое число x (которое может быть положительным или отрицательным) и возвращает его модуль. Вставка состоит из 5 последовательных инструкций процессора (на примере компилятора GCC):

1. movl %[x], %%eax

Загружает значение входной переменной x в регистр общего назначения eax процессора.

2. cltd (в синтаксисе компилятора MSVC используется эквивалентная инструкция cdq)

Эта инструкция расширяет знак из регистра eax в регистр edx.

- Если число в eax положительное или ноль (старший знаковый бит равен 0), то регистр edx полностью заполняется нулями (00000000 00000000 00000000 00000000, что представляет собой число 0).

- Если число в eax отрицательное (старший знаковый бит равен 1), то регистр edx полностью заполняется единицами (11111111 11111111 11111111 11111111, что в дополнительном двоичном коде представляет собой число -1).

3. xorl %%edx, %%eax

Выполняет побитовую операцию Исключающее ИЛИ (XOR) между регистрами edx и eax.

- Если в edx записан 0 (число x изначально было положительным), то операция XOR с нулями оставляет биты в eax нетронутыми.

- Если в edx записано -1 (все биты равны 1, число x изначально было отрицательным), то операция XOR инвертирует каждый бит в регистре eax. Это аналогично логическому отрицанию (операции NOT).

4. subl %%edx, %%eax

Вычитает значение регистра edx из регистра eax.

- Если в edx записан 0 (число положительное), то вычитание нуля ничего не меняет в eax.

- Если в edx записано -1 (число отрицательное), то операция вычитания -1 (eax - (-1)) фактически прибавляет 1 к значению в eax. По правилам дополнительного двоичного кода инверсия всех бит числа с последующим прибавлением единицы представляет собой смену знака числа (перевод отрицательного числа в положительное: -x = NOT(x) + 1).

5. movl %%eax, %[res]

Выгружает полученный итоговый модуль из регистра eax в выходную переменную result.

Главная цель этой ассемблерной оптимизации:

Она вычисляет абсолютное значение числа на уровне процессора без использования инструкций ветвления (инструкций сравнения и условных

переходов типа cmp, jz, jg). При обычном вычислении с условием "if (x < 0) x = -x;" процессор задействует предсказатель переходов. Поскольку знаки разностей координат постоянно меняются случайным образом при переборе шахматных ходов, предсказатель часто ошибается, из-за чего процессор сбрасывает конвейер команд и теряет время. Данная вставка решает задачу за фиксированное число тактов без какого-либо ветвления.

6. Руководство пользователя

Программа предназначена для игры в «Шахматы» на поле 8×8 против компьютера. Управление осуществляется мышью.

6.1 Заставка

При запуске программы отображается заставка.

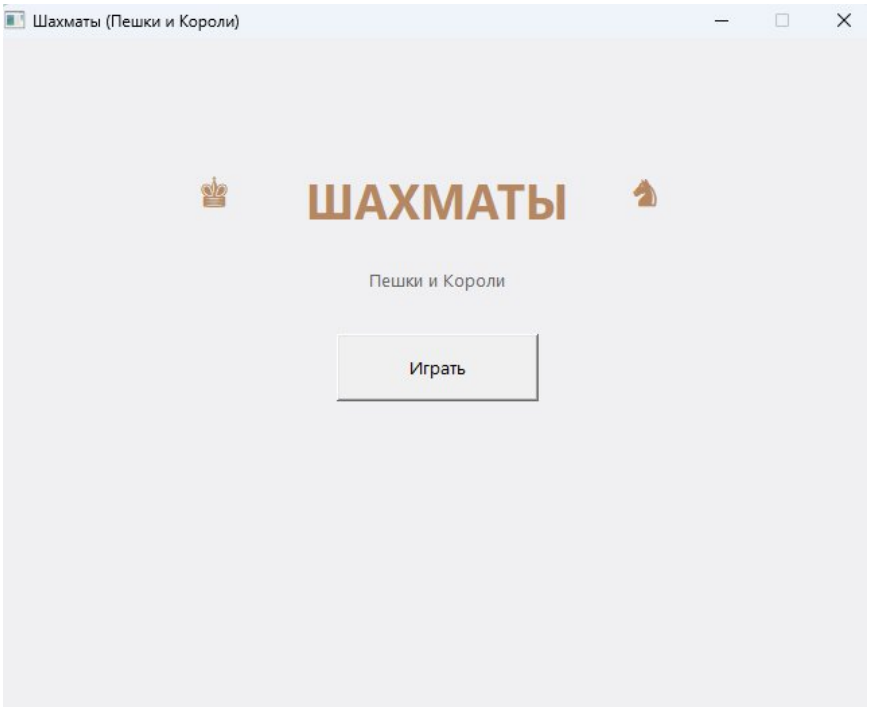

Рисунок 4. Заставка программы

6.2 Правила игры

Обычные правила для стандартных шахмат. Ход определяется случайно.

6.3 Игровое поле

После нажатие на кнопку «Играть» отображается игровое поле.

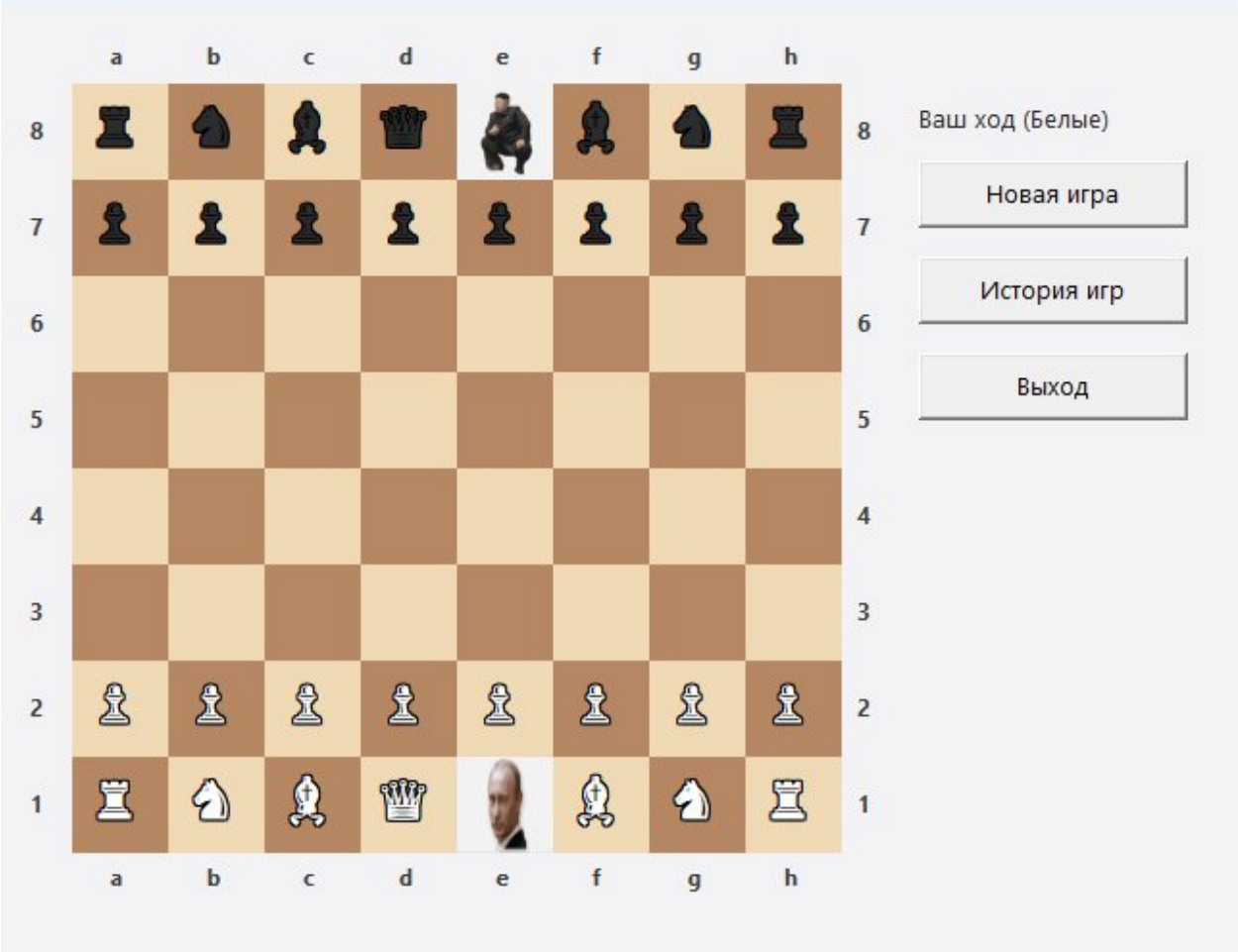

Рисунок 5. Игровое поле

6.4 Завершение игры

Игра заканчивается, когда один из игроков ставит мат, либо же в партии на поле происходит пат.

6.5 Выход из игры

При проигрыше нужно нажать «Новая игра» и партия начнется заново

Заключение

В ходе выполнения работы была создана компьютерная игра «Шахматы» на языке программирования Си с элементами низкоуровневой

оптимизации. Проект показал, как можно применять возможности языка Си для реализации сложной логики игры и взаимодействия с Win32/GUI-средой.

Основные достижения и реализованные функции:

1. Реализована игровая логика:

- представление шахматной доски и фигур;

- проверка допустимости ходов;

- обработка ходов игрока и компьютерного партнёра;

- определение победы, ничьей и отмены хода.

2. Создан графический интерфейс пользователя:

- отрисовка шахматного поля и фигур;

- меню управления игрой;

3. Обеспечена работа с файлами:

- сохранение таблицы рекордов при нормальном завершении и при прерывании;

- загрузка существующего файла рекордов при запуске;

- работа с файловой системой и устойчивое восстановление данных.

4. Проведена отладка и тестирование:

- использование точек останова, пошаговой трассировки и анализа значений переменных;

- тестирование во время разработки и после завершения реализации;

- проверка сценариев запуска, сохранения/загрузки и корректного завершения.

5. Продемонстрировано практическое применение системного программирования:

- организация проекта для Visual Studio;

- соблюдение структуры кода и разделение на модули;

- применение низкоуровневых возможностей C для управления ресурсами.

Выполненная работа продемонстрировала, что язык Си эффективно подходит для создания настольных логических игр с графическим интерфейсом и файловой подсистемой. Программа реализует ключевые функции шахматной игры и обеспечивает стабильность работы при взаимодействии с пользователем.

Список используемых источников

1. Седжвик Р. Язык С. Базовый курс с примерами и задачами. — М.: Диалектика, 2021. — 402 с.
2. Ирвин К. Программирование на ассемблере x86-64. От теории к практике. — СПб.: БХВ-Петербург, 2020. — 312 с.
3. Рихтер Дж. Windows API. Разработка системных приложений на С и Assembler. — М.: Русская редакция, 2019. — 648 с.
4. Торрес Р. Основы машинно-ориентированного программирования. Архитектура и ассемблер. — М.: ДМК Пресс, 2023. — 208 с.
5. Лафоре Р. Структуры данных и алгоритмы на С для начинающих. — СПб.: Питер, 2022. — 304 с.
6. Стивенс Р. Практический курс С и С++. Низкоуровневое программирование. — М.: НОУ ИНТУИТ, 2023. — 275 с.
7. Таненбаум Э. Архитектура вычислительных систем. От процессора до ОС. — 9-е изд. — М.: Вильямс, 2021. — 912 с.

Приложение А. Листинг программы

main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <windows.h>
#include "../include/chess.h"

#ifndef WIN32_GUI
static void print_menu(void) {
    printf("========================================\n");
    printf("       Ш А Х М А Т Ы  /  C H E S S       \n");
    printf("========================================\n");
    printf("                _   _   _   _              \n");
    printf("           | | | | | | | |            \n");
    printf("           |_| |_| |_| |_|              \n");
    printf("           | | | | | | | |            \n");
    printf("           |_______________|            \n");
    printf("           (_______________)             \n");
    printf("========================================\n\n");
    printf("1. Играть (Новая игра)\n");
    printf("2. История игр\n");
    printf("3. Выход\n");
    printf("Выберите пункт: ");
}
#endif

static void initialize_board(Game *g) {
    for (int row = 0; row < BOARD_SIZE; ++row) {
        for (int col = 0; col < BOARD_SIZE; ++col) {
            g->board[row][col] = '.';
        }
    }

    // Черные фигуры (строка 0)
    g->board[0][0] = 'r';
    g->board[0][1] = 'n';
    g->board[0][2] = 'b';
    g->board[0][3] = 'q';
    g->board[0][4] = 'k';
    g->board[0][5] = 'b';
    g->board[0][6] = 'n';
    g->board[0][7] = 'r';
```

```
    // Черные пешки (строка 1)
    for (int col = 0; col < BOARD_SIZE; ++col) {
        g->board[1][col] = 'p';
    }

    // Белые пешки (строка 6)
    for (int col = 0; col < BOARD_SIZE; ++col) {
        g->board[6][col] = 'P';
    }

    // Белые фигуры (строка 7)
    g->board[7][0] = 'R';
    g->board[7][1] = 'N';
    g->board[7][2] = 'B';
    g->board[7][3] = 'Q';
    g->board[7][4] = 'K';
    g->board[7][5] = 'B';
    g->board[7][6] = 'N';
    g->board[7][7] = 'R';

    g->player_is_white = (rand() % 2 == 0);
    g->white_turn = true;
    g->move_count = 0;
}

void game_init(Game *g) {
    if (!g) {
        return;
    }
    initialize_board(g);
}

void print_board(const Game *g) {
    printf("\n  a b c d e f g h\n");
    for (int row = 0; row < BOARD_SIZE; ++row) {
        printf("%d ", BOARD_SIZE - row);
        for (int col = 0; col < BOARD_SIZE; ++col) {
            printf("%c ", g->board[row][col]);
        }
        printf("%d\n", BOARD_SIZE - row);
    }
    printf("  a b c d e f g h\n");
}
```

```
static bool parse_move_text(const char *text, int *from_row, int *from_col, int
*to_row, int *to_col) {
    if (!text || strlen(text) < 4) {
        return false;
    }

    int fcol = text[0] - 'a';
    int frow = '8' - text[1];
    int tcol = text[2] - 'a';
    int trow = '8' - text[3];

    if (fcol < 0 || fcol >= BOARD_SIZE || tcol < 0 || tcol >= BOARD_SIZE || frow <
0 || frow >= BOARD_SIZE || trow < 0 || trow >= BOARD_SIZE) {
        return false;
    }

    *from_row = frow;
    *from_col = fcol;
    *to_row = trow;
    *to_col = tcol;
    return true;
}

bool is_valid_move(const Game *g, int fr, int fc, int tr, int tc) {
    if (!g || fr < 0 || fr >= BOARD_SIZE || fc < 0 || fc >= BOARD_SIZE ||
        tr < 0 || tr >= BOARD_SIZE || tc < 0 || tc >= BOARD_SIZE) {
        return false;
    }
    if (fr == tr && fc == tc) {
        return false;
    }

    char piece = g->board[fr][fc];
    if (piece == '.' || piece == 0) {
        return false;
    }

    char target = g->board[tr][tc];

    // Проверка цветов фигур
    bool is_white = (piece >= 'A' && piece <= 'Z');
    bool is_target_white = (target >= 'A' && target <= 'Z');

    if (target != '.' && is_white == is_target_white) {
        // Нельзя бить фигуру своего цвета
        return false;
```

```
}

char type = is_white ? piece : (piece - 'a' + 'A');

switch (type) {
    case 'P': { // Пешка
        int dir = is_white ? -1 : 1;
        int start_row = is_white ? 6 : 1;

        // Ход на 1 клетку вперед
        if (tc == fc && tr == fr + dir && target == '.') {
            return true;
        }
        // Ход на 2 клетки вперед со стартовой позиции
        if (tc == fc && fr == start_row && tr == fr + 2 * dir && target == '.') {
            if (g->board[fr + dir][fc] == '.') {
                return true;
            }
        }
        // Взятие по диагонали
        if ((tc == fc - 1 || tc == fc + 1) && tr == fr + dir) {
            if (target != '.' && is_white != is_target_white) {
                return true;
            }
        }
        break;
    }
    case 'R': { // Ладья
        if (fr != tr && fc != tc) return false;
        int dr = (tr > fr) - (tr < fr);
        int dc = (tc > fc) - (tc < fc);
        int r = fr + dr;
        int c = fc + dc;
        while (r != tr || c != tc) {
            if (g->board[r][c] != '.') return false;
            r += dr;
            c += dc;
        }
        return true;
    }
    case 'N': { // Конь
        int dr = asm_abs(tr - fr);
        int dc = asm_abs(tc - fc);
        return (dr == 2 && dc == 1) || (dr == 1 && dc == 2);
    }
```

```
        case 'B': { // Слон
            if (asm_abs(tr - fr) != asm_abs(tc - fc)) return false;
            int dr = (tr > fr) - (tr < fr);
            int dc = (tc > fc) - (tc < fc);
            int r = fr + dr;
            int c = fc + dc;
            while (r != tr || c != tc) {
                if (g->board[r][c] != '.') return false;
                r += dr;
                c += dc;
            }
            return true;
        }
        case 'Q': { // Ферзь
            if (fr != tr && fc != tc && asm_abs(tr - fr) != asm_abs(tc - fc)) return false;
            int dr = (tr > fr) - (tr < fr);
            int dc = (tc > fc) - (tc < fc);
            int r = fr + dr;
            int c = fc + dc;
            while (r != tr || c != tc) {
                if (g->board[r][c] != '.') return false;
                r += dr;
                c += dc;
            }
            return true;
        }
        case 'K': { // Король
            int dr = asm_abs(tr - fr);
            int dc = asm_abs(tc - fc);
            return (dr <= 1 && dc <= 1);
        }
    }
    return false;
}

static bool find_king(const Game *g, bool white_king, int *king_row, int
*king_col) {
    char target_king = white_king ? 'K' : 'k';
    for (int r = 0; r < BOARD_SIZE; ++r) {
        for (int c = 0; c < BOARD_SIZE; ++c) {
            if (g->board[r][c] == target_king) {
                *king_row = r;
                *king_col = c;
                return true;
            }
```

```
        }
    }
    return false;
}

bool is_square_attacked(const Game *g, int r, int c, bool by_white) {
    Game temp_game = *g;
    temp_game.board[r][c] = by_white ? 'p' : 'P';

    for (int fr = 0; fr < BOARD_SIZE; ++fr) {
        for (int fc = 0; fc < BOARD_SIZE; ++fc) {
            char piece = temp_game.board[fr][fc];
            if (piece == '.' || piece == 0) continue;
            bool is_piece_white = (piece >= 'A' && piece <= 'Z');
            if (is_piece_white == by_white) {
                if (is_valid_move(&temp_game, fr, fc, r, c)) {
                    return true;
                }
            }
        }
    }
    return false;
}

bool is_in_check(const Game *g, bool white_king) {
    int kr = -1, kc = -1;
    if (find_king(g, white_king, &kr, &kc)) {
        return is_square_attacked(g, kr, kc, !white_king);
    }
    return false;
}

bool is_legal_move(const Game *g, int fr, int fc, int tr, int tc) {
    if (!is_valid_move(g, fr, fc, tr, tc)) {
        return false;
    }

    Game next_state = *g;
    next_state.board[tr][tc] = next_state.board[fr][fc];
    next_state.board[fr][fc] = '.';

    if (next_state.board[tr][tc] == 'P' && tr == 0) {
        next_state.board[tr][tc] = 'Q';
    }
    if (next_state.board[tr][tc] == 'p' && tr == 7) {
```

```
        next_state.board[tr][tc] = 'q';
    }

    bool white_turn = (g->board[fr][fc] >= 'A' && g->board[fr][fc] <= 'Z');
    return !is_in_check(&next_state, white_turn);
}

bool has_legal_moves(const Game *g, bool white_player) {
    for (int fr = 0; fr < BOARD_SIZE; ++fr) {
        for (int fc = 0; fc < BOARD_SIZE; ++fc) {
            char piece = g->board[fr][fc];
            if (piece == '.' || piece == 0) continue;
            bool is_white = (piece >= 'A' && piece <= 'Z');
            if (is_white == white_player) {
                for (int tr = 0; tr < BOARD_SIZE; ++tr) {
                    for (int tc = 0; tc < BOARD_SIZE; ++tc) {
                        if (is_legal_move(g, fr, fc, tr, tc)) {
                            return true;
                        }
                    }
                }
            }
        }
    }
    return false;
}

bool make_player_move(Game *g, const char *move_text) {
    int fr, fc, tr, tc;
    if (!parse_move_text(move_text, &fr, &fc, &tr, &tc)) {
        return false;
    }

    char piece = g->board[fr][fc];
    bool is_white = (piece >= 'A' && piece <= 'Z');
    if (is_white != g->white_turn) {
        return false; // Не ваш ход или не ваша фигура
    }

    if (!is_legal_move(g, fr, fc, tr, tc)) {
        return false; // Недопустимый ход (не по правилам фигуры или оставляет
короля под шахом)
    }

    // Делаем ход
```

```
    g->board[tr][tc] = piece;
    g->board[fr][fc] = '.';

    // Авто-продвижение пешки в ферзя на последней горизонтали
    if (piece == 'P' && tr == 0) {
        g->board[tr][tc] = 'Q';
    }

    if (g->move_count < MAX_MOVES) {
        snprintf(g->moves[g->move_count], MAX_MOVE_TEXT, "%s",
move_text);
    }
    g->move_count++;
    g->white_turn = !g->white_turn;
    return true;
}

GameResult check_game_result(const Game *g) {
    bool active_player_white = g->white_turn;
    bool in_chk = is_in_check(g, active_player_white);
    bool has_moves = has_legal_moves(g, active_player_white);

    if (!has_moves) {
        if (in_chk) {
            // Мат! Тот, у кого нет ходов и кто под шахом, проиграл
            return active_player_white ? GAME_BLACK_WIN :
GAME_WHITE_WIN;
        } else {
            // Пат! Ничья
            return GAME_DRAW;
        }
    }

    // Резервная проверка на существование королей
    bool white_king = false;
    bool black_king = false;
    for (int r = 0; r < BOARD_SIZE; ++r) {
        for (int c = 0; c < BOARD_SIZE; ++c) {
            if (g->board[r][c] == 'K') white_king = true;
            if (g->board[r][c] == 'k') black_king = true;
        }
    }
    if (!white_king) return GAME_BLACK_WIN;
    if (!black_king) return GAME_WHITE_WIN;
```

```
    return GAME_ONGOING;
}

void save_game_history(const Game *g, GameResult result) {
    FILE *file = fopen(HISTORY_FILE, "a");
    if (!file) {
        return;
    }

    time_t now = time(NULL);
    struct tm *tm_info = localtime(&now);
    char buf[64];
    strftime(buf, sizeof(buf), "%Y-%m-%d %H:%M:%S", tm_info);

    fprintf(file, "Дата: %s\n", buf);
    fprintf(file, "Результат: ");
    switch (result) {
        case GAME_WHITE_WIN: fprintf(file, "%s\n", g->player_is_white ?
"Победа игрока" : "Победа компьютера"); break;
        case GAME_BLACK_WIN: fprintf(file, "%s\n", g->player_is_white ?
"Победа компьютера" : "Победа игрока"); break;
        case GAME_DRAW: fprintf(file, "Ничья\n"); break;
        default: fprintf(file, "Неопределено\n"); break;
    }
    fprintf(file, "Ходы:\n");
    for (int i = 0; i < g->move_count; ++i) {
        fprintf(file, "%s ", g->moves[i]);
        if ((i + 1) % 8 == 0) {
            fprintf(file, "\n");
        }
    }
    fprintf(file, "\n---\n");
    fclose(file);
}

void show_game_history(void) {
    FILE *file = fopen(HISTORY_FILE, "r");
    if (!file) {
        printf("\nФайл истории игр не найден.\n");
        return;
    }

    printf("\n=== История игр ===\n");
    char line[256];
    while (fgets(line, sizeof(line), file)) {
```

```
        printf("%s", line);
    }
    fclose(file);
}

#ifndef WIN32_GUI
static void run_game(void) {
    Game g;
    game_init(&g);
    GameResult result = GAME_ONGOING;

    while (result == GAME_ONGOING) {
        print_board(&g);
        if (is_bot_turn(&g)) {
            printf("\nХод компьютера...\n");
            if (bot_make_move(&g) == 0) {
                printf("Компьютер не нашёл ход.\n");
                result = GAME_DRAW;
                break;
            }
        } else {
            char move_text[32];
            printf("\nВаш ход (например, e2e4): ");
            if (!fgets(move_text, sizeof(move_text), stdin)) {
                continue;
            }
            move_text[strcspn(move_text, "\r\n")] = '\0';
            if (strcmp(move_text, "exit") == 0) {
                printf("Выход из партии.\n");
                return;
            }
            if (!make_player_move(&g, move_text)) {
                printf("Неверный ход. Попробуйте снова.\n");
                continue;
            }
        }

        result = check_game_result(&g);
    }

    print_board(&g);
    printf("\nРезультат партии: ");
    if (result == GAME_WHITE_WIN) {
        printf("Победа игрока\n");
    } else if (result == GAME_BLACK_WIN) {
```

```
        printf("Победа компьютера\n");
    } else {
        printf("Ничья\n");
    }
    save_game_history(&g, result);
}
#endif

#ifdef WIN32_GUI
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, int nCmdShow) {
    (void)hInstance;
    (void)hPrevInstance;
    (void)lpCmdLine;
    (void)nCmdShow;
    srand((unsigned int)time(NULL));
    run_gui();
    return 0;
}
#else
int main(void) {
    SetConsoleOutputCP(CP_UTF8);  // Включаем UTF-8
    SetConsoleCP(CP_UTF8);
    srand((unsigned int)time(NULL));
    while (1) {
        print_menu();
        int choice = 0;
        if (scanf("%d", &choice) != 1) {
            scanf("%*s");
            continue;
        }
        int ch = getchar();
        (void)ch;

        switch (choice) {
            case 1:
                run_game();
                break;
            case 2:
                show_game_history();
                break;
            case 3:
                printf("Выход из программы.\n");
                return 0;
            default:
```

```
            printf("Неверный выбор. Попробуйте снова.\n");
            break;
        }
    }
}
#endif
```

asm_util.c:

bot.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "../include/chess.h"

bool is_bot_turn(const Game *g) {
    return g && (g->white_turn != g->player_is_white);
}

static int get_piece_value(char piece) {
    char p = (piece >= 'a' && piece <= 'z') ? (piece - 'a' + 'A') : piece;
    switch (p) {
        case 'P': return 10;
        case 'N': return 30;
        case 'B': return 30;
        case 'R': return 50;
        case 'Q': return 90;
        case 'K': return 900;
        default: return 0;
    }
}

static int move_score(const Game *g, int tr, int tc) {
    char target = g->board[tr][tc];
    if (target != '.') {
        // Взятие фигуры противника оценивается по её ценности
        return 10 + get_piece_value(target);
    }
    return 1;
}

int bot_make_move(Game *g) {
```

```
if (!g) {
    return 0;
}

int best_score = -1;
int best_fr = -1, best_fc = -1, best_tr = -1, best_tc = -1;

bool bot_is_white = !g->player_is_white;
// Бот перебирает все свои фигуры
for (int fr = 0; fr < BOARD_SIZE; ++fr) {
    for (int fc = 0; fc < BOARD_SIZE; ++fc) {
        char piece = g->board[fr][fc];
        if (piece == '.' || piece == 0) {
            continue;
        }
        bool is_piece_white = (piece >= 'A' && piece <= 'Z');
        if (is_piece_white != bot_is_white) {
            continue;
        }
        for (int tr = 0; tr < BOARD_SIZE; ++tr) {
            for (int tc = 0; tc < BOARD_SIZE; ++tc) {
                if (!is_legal_move(g, fr, fc, tr, tc)) {
                    continue;
                }
                int score = move_score(g, tr, tc);
                // Добавим немного случайности при равных ходах
                if (score > best_score || (score == best_score && (rand() % 5 == 0))) {
                    best_score = score;
                    best_fr = fr;
                    best_fc = fc;
                    best_tr = tr;
                    best_tc = tc;
                }
            }
        }
    }
}

if (best_score < 0) {
    return 0; // Бот не нашел допустимых ходов
}

char piece = g->board[best_fr][best_fc];
char move_text[MAX_MOVE_TEXT];
```

```
    move_text[0] = 'a' + best_fc;
    move_text[1] = '8' - best_fr;
    move_text[2] = 'a' + best_tc;
    move_text[3] = '8' - best_tr;
    move_text[4] = '\0';

    // Совершаем ход
    g->board[best_tr][best_tc] = piece;
    g->board[best_fr][best_fc] = '.';

    // Авто-продвижение пешки бота на последней горизонтали
    if (piece == 'P' && best_tr == 0) {
        g->board[best_tr][best_tc] = 'Q';
    } else if (piece == 'p' && best_tr == 7) {
        g->board[best_tr][best_tc] = 'q';
    }

    if (g->move_count < MAX_MOVES) {
        snprintf(g->moves[g->move_count], MAX_MOVE_TEXT, "%s",
move_text);
    }
    g->move_count++;
    g->white_turn = !g->white_turn; // Смена очереди хода
    return 1;
}
```

gui.c:

```
#ifndef UNICODE
#define UNICODE
#endif
#ifndef _UNICODE
#define _UNICODE
#endif

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include "../include/chess.h"

// Константы размеров
#define BOARD_OFFSET_X 40
```

```
#define BOARD_OFFSET_Y 40
#define CELL_SIZE 50

// Идентификаторы кнопок
#define ID_BTN_NEW 101
#define ID_BTN_HISTORY 102
#define ID_BTN_EXIT 103
#define ID_BTN_PLAY 104

// Глобальные переменные для GUI
static Game g_game;
static int selected_row = -1;
static int selected_col = -1;
static HFONT hFontUI = NULL;
static HFONT hFontPieces = NULL;
static HFONT hFontLabels = NULL;
static bool game_over_notified = false;
static bool g_in_splash_screen = true;

// Типы для GDI+ Flat API
typedef void* GpGraphics;
typedef void* GpImage;
typedef int GpStatus;

typedef GpStatus (WINAPI *GdiplusStartup_fn)(ULONG_PTR*, const void*,
void*);
typedef void (WINAPI *GdiplusShutdown_fn)(ULONG_PTR);
typedef GpStatus (WINAPI *GdipCreateFromHDC_fn)(HDC, GpGraphics**);
typedef GpStatus (WINAPI *GdipDeleteGraphics_fn)(GpGraphics*);
typedef GpStatus (WINAPI *GdipLoadImageFromFile_fn)(const WCHAR*,
GpImage*);
typedef GpStatus (WINAPI *GdipDisposeImage_fn)(GpImage*);
typedef GpStatus (WINAPI *GdipDrawImageRectI_fn)(GpGraphics*, GpImage*,
INT, INT, INT, INT);

static HMODULE hGdiplus = NULL;
static ULONG_PTR gdiplusToken = 0;
static GdiplusStartup_fn pGdiplusStartup = NULL;
static GdiplusShutdown_fn pGdiplusShutdown = NULL;
static GdipCreateFromHDC_fn pGdipCreateFromHDC = NULL;
static GdipDeleteGraphics_fn pGdipDeleteGraphics = NULL;
static GdipLoadImageFromFile_fn pGdipLoadImageFromFile = NULL;
static GdipDisposeImage_fn pGdipDisposeImage = NULL;
static GdipDrawImageRectI_fn pGdipDrawImageRectI = NULL;
```

```
static GpImage img_white_king = NULL;
static GpImage img_black_king = NULL;

// Загрузка GDI+
static void init_gdiplus(void) {
    hGdiplus = LoadLibraryW(L"gdiplus.dll");
    if (!hGdiplus) return;

    pGdiplusStartup = (GdiplusStartup_fn)GetProcAddress(hGdiplus,
"GdiplusStartup");
    pGdiplusShutdown = (GdiplusShutdown_fn)GetProcAddress(hGdiplus,
"GdiplusShutdown");
    pGdipCreateFromHDC = (GdipCreateFromHDC_fn)GetProcAddress(hGdiplus,
"GdipCreateFromHDC");
    pGdipDeleteGraphics = (GdipDeleteGraphics_fn)GetProcAddress(hGdiplus,
"GdipDeleteGraphics");
    pGdipLoadImageFromFile =
(GdipLoadImageFromFile_fn)GetProcAddress(hGdiplus,
"GdipLoadImageFromFile");
    pGdipDisposeImage = (GdipDisposeImage_fn)GetProcAddress(hGdiplus,
"GdipDisposeImage");
    pGdipDrawImageRectI = (GdipDrawImageRectI_fn)GetProcAddress(hGdiplus,
"GdipDrawImageRectI");

    if (pGdiplusStartup && pGdiplusShutdown && pGdipCreateFromHDC &&
pGdipDeleteGraphics &&
        pGdipLoadImageFromFile && pGdipDisposeImage &&
pGdipDrawImageRectI) {

        struct {
            UINT32 GdiplusVersion;
            void* DebugEventCallback;
            BOOL SuppressBackgroundThread;
            BOOL SuppressExternalCodecs;
        } startupInput = {1, NULL, FALSE, FALSE};

        ULONG_PTR token = 0;
        if (pGdiplusStartup(&token, &startupInput, NULL) == 0) {
            gdiplusToken = token;
            // Пытаемся загрузить картинки королей
            pGdipLoadImageFromFile(L"white_king.png", &img_white_king);
            pGdipLoadImageFromFile(L"black_king.png", &img_black_king);
        }
    } else {
        FreeLibrary(hGdiplus);
```

```
        hGdiplus = NULL;
    }
}

static void shutdown_gdiplus(void) {
    if (hGdiplus) {
        if (img_white_king) pGdipDisposeImage(img_white_king);
        if (img_black_king) pGdipDisposeImage(img_black_king);
        if (pGdiplusShutdown && gdiplusToken) pGdiplusShutdown(gdiplusToken);
        FreeLibrary(hGdiplus);
        hGdiplus = NULL;
    }
}

// Отрисовка фигуры
static void DrawPiece(HDC hdc, char piece, int x, int y) {
    bool is_white = (piece >= 'A' && piece <= 'Z');
    wchar_t wstr[2] = {0};

    // Если это король и загружена картинка
    if (hGdiplus) {
        GpGraphics* graphics = NULL;
        if (pGdipCreateFromHDC(hdc, &graphics) == 0) {
            if (piece == 'K' && img_white_king) {
                pGdipDrawImageRectI(graphics, img_white_king, x, y, CELL_SIZE,
CELL_SIZE);
                pGdipDeleteGraphics(graphics);
                return;
            } else if (piece == 'k' && img_black_king) {
                pGdipDrawImageRectI(graphics, img_black_king, x, y, CELL_SIZE,
CELL_SIZE);
                pGdipDeleteGraphics(graphics);
                return;
            }
            pGdipDeleteGraphics(graphics);
        }
    }

    // Встроенный Unicode-символ фигуры в качестве основного метода или
резервного варианта
    switch (piece) {
        // Белые и черные фигуры маппятся на одинаковые залитые Unicode-
глифы.
        // Их цвет определяется цветом кисти hBrush в зависимости от is_white.
        case 'P': case 'p': wstr[0] = 0x265F; break; // Пешка ♟
```

```
        case 'R': case 'r': wstr[0] = 0x265C; break; // Ладья ♜
        case 'N': case 'n': wstr[0] = 0x265E; break; // Конь ♞
        case 'B': case 'b': wstr[0] = 0x265D; break; // Слон ♝
        case 'Q': case 'q': wstr[0] = 0x265B; break; // Ферзь ♛
        case 'K': case 'k': wstr[0] = 0x265A; break; // Король ♚
        default: return;
    }

    HFONT hOldFont = (HFONT)SelectObject(hdc, hFontPieces);
    SetBkMode(hdc, TRANSPARENT);

    // Координаты для рисования по центру клетки
    int text_x = x + 8;
    int text_y = y + 2;

    // Рисуем черный контур смещением в 8 направлениях
    SetTextColor(hdc, RGB(0, 0, 0));
    TextOutW(hdc, text_x - 1, text_y,     wstr, 1);
    TextOutW(hdc, text_x + 1, text_y,     wstr, 1);
    TextOutW(hdc, text_x,     text_y - 1, wstr, 1);
    TextOutW(hdc, text_x,     text_y + 1, wstr, 1);
    TextOutW(hdc, text_x - 1, text_y - 1, wstr, 1);
    TextOutW(hdc, text_x + 1, text_y - 1, wstr, 1);
    TextOutW(hdc, text_x - 1, text_y + 1, wstr, 1);
    TextOutW(hdc, text_x + 1, text_y + 1, wstr, 1);

    // Рисуем тело фигуры цветом стороны
    COLORREF fillColor = is_white ? RGB(255, 255, 255) : RGB(45, 45, 45);
    SetTextColor(hdc, fillColor);
    TextOutW(hdc, text_x, text_y, wstr, 1);

    SelectObject(hdc, hOldFont);
}

// Отображение истории игр
static void show_game_history_gui(HWND hwndOwner) {
    FILE *file = fopen(HISTORY_FILE, "r");
    if (!file) {
        MessageBoxW(hwndOwner, L"Файл истории игр не найден.", L"История
игр", MB_OK | MB_ICONINFORMATION);
        return;
    }

    char buffer[16384] = {0};
    size_t bytesRead = fread(buffer, 1, sizeof(buffer) - 1, file);
    fclose(file);
```

```
    if (bytesRead == 0) {
        MessageBoxW(hwndOwner, L"История игр пуста.", L"История игр",
MB_OK | MB_ICONINFORMATION);
        return;
    }

    int wlen = MultiByteToWideChar(CP_UTF8, 0, buffer, -1, NULL, 0);
    if (wlen <= 0) {
        // Попробуем дефолтную кодировку
        wlen = MultiByteToWideChar(CP_ACP, 0, buffer, -1, NULL, 0);
    }

    wchar_t *wbuf = (wchar_t*)malloc(wlen * sizeof(wchar_t));
    if (wbuf) {
        MultiByteToWideChar(CP_UTF8, 0, buffer, -1, wbuf, wlen);
        MessageBoxW(hwndOwner, wbuf, L"История игр", MB_OK |
MB_ICONINFORMATION);
        free(wbuf);
    }
}

// Проверка окончания игры
static void check_and_handle_game_over(HWND hwnd) {
    if (game_over_notified) return;

    GameResult result = check_game_result(&g_game);
    if (result != GAME_ONGOING) {
        game_over_notified = true;
        const wchar_t *msg = L"";
        if (result == GAME_WHITE_WIN) {
            if (g_game.player_is_white) {
                msg = L"Игра окончена!\nМат! Победа игрока (Белые)!";
            } else {
                msg = L"Игра окончена!\nМат! Победа компьютера (Белые)!";
            }
        } else if (result == GAME_BLACK_WIN) {
            if (g_game.player_is_white) {
                msg = L"Игра окончена!\nМат! Победа компьютера (Черные)!";
            } else {
                msg = L"Игра окончена!\nМат! Победа игрока (Черные)!";
            }
        } else {
            msg = L"Игра окончена!\nПат! Ничья!";
        }
```

```
        save_game_history(&g_game, result);
        MessageBoxW(hwnd, msg, L"Результат матча", MB_OK |
MB_ICONINFORMATION);
    }
}

// Оконная процедура
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM
wParam, LPARAM lParam) {
    switch (msg) {
        case WM_CREATE: {
            game_init(&g_game);
            init_gdiplus();

            // Создаем шрифты
            hFontUI = CreateFontW(17, 0, 0, 0, FW_MEDIUM, FALSE, FALSE,
FALSE, DEFAULT_CHARSET,
                                  OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
CLEARTYPE_QUALITY, DEFAULT_PITCH | FF_DONTCARE, L"Segoe UI");
            hFontLabels = CreateFontW(16, 0, 0, 0, FW_BOLD, FALSE, FALSE,
FALSE, DEFAULT_CHARSET,
                                      OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
CLEARTYPE_QUALITY, DEFAULT_PITCH | FF_DONTCARE, L"Segoe UI");
            hFontPieces = CreateFontW(38, 0, 0, 0, FW_NORMAL, FALSE, FALSE,
FALSE, DEFAULT_CHARSET,
                                      OUT_DEFAULT_PRECIS, CLIP_DEFAULT_PRECIS,
CLEARTYPE_QUALITY, DEFAULT_PITCH | FF_DONTCARE, L"Segoe UI
Symbol");

            // Создаем кнопки управления (изначально скрытые, кроме кнопки
"Играть")
            HWND btnNew = CreateWindowW(L"BUTTON", L"Новая игра",
WS_CHILD | BS_PUSHBUTTON,
                                        480, 80, 140, 35, hwnd, (HMENU)ID_BTN_NEW,
((LPCREATESTRUCT)lParam)->hInstance, NULL);
            HWND btnHistory = CreateWindowW(L"BUTTON", L"История игр",
WS_CHILD | BS_PUSHBUTTON,
                                            480, 130, 140, 35, hwnd,
(HMENU)ID_BTN_HISTORY, ((LPCREATESTRUCT)lParam)->hInstance,
NULL);
            HWND btnExit = CreateWindowW(L"BUTTON", L"Выход",
WS_CHILD | BS_PUSHBUTTON,
                                        480, 180, 140, 35, hwnd, (HMENU)ID_BTN_EXIT,
((LPCREATESTRUCT)lParam)->hInstance, NULL);
```

```
        HWND btnPlay = CreateWindowW(L"BUTTON", L"Играть",
WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                                     250, 220, 150, 50, hwnd, (HMENU)ID_BTN_PLAY,
((LPCREATESTRUCT)lParam)->hInstance, NULL);

        // Задаем красивый шрифт для кнопок
        SendMessageW(btnNew, WM_SETFONT, (WPARAM)hFontUI, TRUE);
        SendMessageW(btnHistory, WM_SETFONT, (WPARAM)hFontUI,
TRUE);
        SendMessageW(btnExit, WM_SETFONT, (WPARAM)hFontUI, TRUE);
        SendMessageW(btnPlay, WM_SETFONT, (WPARAM)hFontUI, TRUE);
        break;
    }

    case WM_PAINT: {
        PAINTSTRUCT ps;
        HDC hdc = BeginPaint(hwnd, &ps);

        if (g_in_splash_screen) {
            RECT rect;
            GetClientRect(hwnd, &rect);
            HDC hdcMem = CreateCompatibleDC(hdc);
            HBITMAP hbmMem = CreateCompatibleBitmap(hdc, rect.right,
rect.bottom);
            HBITMAP hOldBmp = (HBITMAP)SelectObject(hdcMem, hbmMem);

            HBRUSH hBrBg = CreateSolidBrush(RGB(240, 240, 242));
            FillRect(hdcMem, &rect, hBrBg);
            DeleteObject(hBrBg);

            HFONT hFontTitle = CreateFontW(48, 0, 0, 0, FW_BOLD, FALSE,
FALSE, FALSE, DEFAULT_CHARSET,
                                           OUT_DEFAULT_PRECIS,
CLIP_DEFAULT_PRECIS, CLEARTYPE_QUALITY, DEFAULT_PITCH |
FF_DONTCARE, L"Segoe UI");
            HFONT hOldFont = (HFONT)SelectObject(hdcMem, hFontTitle);
            SetTextColor(hdcMem, RGB(181, 136, 99));
            SetBkMode(hdcMem, TRANSPARENT);

            RECT titleRect = { 0, 80, rect.right, 160 };
            DrawTextW(hdcMem, L"ШАХМАТЫ", -1, &titleRect,
DT_SINGLELINE | DT_CENTER | DT_VCENTER);

            SelectObject(hdcMem, hFontUI);
            SetTextColor(hdcMem, RGB(100, 100, 100));
            RECT subRect = { 0, 160, rect.right, 200 };
```

```
            DrawTextW(hdcMem, L"Пешки и Короли", -1, &subRect,
DT_SINGLELINE | DT_CENTER | DT_VCENTER);

            SelectObject(hdcMem, hFontPieces);
            SetTextColor(hdcMem, RGB(181, 136, 99));
            TextOutW(hdcMem, rect.right / 2 - 180, 95, L"\x265A", 1);
            TextOutW(hdcMem, rect.right / 2 + 140, 95, L"\x265E", 1);

            BitBlt(hdc, 0, 0, rect.right, rect.bottom, hdcMem, 0, 0, SRCCOPY);

            SelectObject(hdcMem, hOldFont);
            SelectObject(hdcMem, hOldBmp);
            DeleteObject(hbmMem);
            DeleteObject(hdcMem);
            DeleteObject(hFontTitle);
            EndPaint(hwnd, &ps);
            break;
        }

        // Двойная буферизация для отсутствия мерцания при отрисовке
        RECT rect;
        GetClientRect(hwnd, &rect);
        HDC hdcMem = CreateCompatibleDC(hdc);
        HBITMAP hbmMem = CreateCompatibleBitmap(hdc, rect.right,
rect.bottom);
        HBITMAP hOldBmp = (HBITMAP)SelectObject(hdcMem, hbmMem);

        // Заливка фона (красивый светлый серый)
        HBRUSH hBrBg = CreateSolidBrush(RGB(243, 243, 245));
        FillRect(hdcMem, &rect, hBrBg);
        DeleteObject(hBrBg);

        // Отрисовка шахматной доски
        HBRUSH hBrLight = CreateSolidBrush(RGB(240, 217, 181));
        HBRUSH hBrDark = CreateSolidBrush(RGB(181, 136, 99));

        for (int r = 0; r < BOARD_SIZE; ++r) {
            for (int c = 0; c < BOARD_SIZE; ++c) {
                RECT cellRect;
                cellRect.left = BOARD_OFFSET_X + c * CELL_SIZE;
                cellRect.top = BOARD_OFFSET_Y + r * CELL_SIZE;
                cellRect.right = cellRect.left + CELL_SIZE;
                cellRect.bottom = cellRect.top + CELL_SIZE;

                // Выбор цвета клетки
                HBRUSH hBr = ((r + c) % 2 == 0) ? hBrLight : hBrDark;
```

```
            FillRect(hdcMem, &cellRect, hBr);

            // Если клетка выбрана игроком — подсвечиваем её
            if (r == selected_row && c == selected_col) {
               HPEN hPenSel = CreatePen(PS_SOLID, 3, RGB(255, 215, 0)); // Золотая рамка
               HPEN hOldPen = (HPEN)SelectObject(hdcMem, hPenSel);
               HBRUSH hOldBrush = (HBRUSH)SelectObject(hdcMem, GetStockObject(NULL_BRUSH));

               Rectangle(hdcMem, cellRect.left, cellRect.top, cellRect.right, cellRect.bottom);

               SelectObject(hdcMem, hOldBrush);
               SelectObject(hdcMem, hOldPen);
               DeleteObject(hPenSel);
            }

            // Отрисовка фигуры
            char piece = g_game.board[r][c];
            if (piece != '.' && piece != 0) {
               DrawPiece(hdcMem, piece, cellRect.left, cellRect.top);
            }
         }
      }

      DeleteObject(hBrLight);
      DeleteObject(hBrDark);

      // Отрисовка разметки доски (буквы a-h, цифры 1-8)
      HFONT hOldFont = (HFONT)SelectObject(hdcMem, hFontLabels);
      SetTextColor(hdcMem, RGB(60, 60, 60));
      SetBkMode(hdcMem, TRANSPARENT);

      for (int i = 0; i < BOARD_SIZE; ++i) {
         wchar_t colLetter[2] = { L'a' + i, 0 };
         wchar_t rowDigit[2] = { L'8' - i, 0 };

         // Буквы по горизонтали
         TextOutW(hdcMem, BOARD_OFFSET_X + i * CELL_SIZE + 20, BOARD_OFFSET_Y - 22, colLetter, 1);
         TextOutW(hdcMem, BOARD_OFFSET_X + i * CELL_SIZE + 20, BOARD_OFFSET_Y + BOARD_SIZE * CELL_SIZE + 5, colLetter, 1);

         // Цифры по вертикали
```

```
            TextOutW(hdcMem, BOARD_OFFSET_X - 22, BOARD_OFFSET_Y
+ i * CELL_SIZE + 17, rowDigit, 1);
            TextOutW(hdcMem, BOARD_OFFSET_X + BOARD_SIZE *
CELL_SIZE + 8, BOARD_OFFSET_Y + i * CELL_SIZE + 17, rowDigit, 1);
        }
        SelectObject(hdcMem, hOldFont);

        // Вывод статуса игры
        hOldFont = (HFONT)SelectObject(hdcMem, hFontUI);
        SetTextColor(hdcMem, RGB(40, 40, 40));
        RECT textRect = { 480, 40, 630, 75 };

        GameResult result = check_game_result(&g_game);
        if (result == GAME_ONGOING) {
            bool in_chk = is_in_check(&g_game, g_game.white_turn);
            bool player_turn = (g_game.white_turn == g_game.player_is_white);
            if (player_turn) {
                if (in_chk) {
                    SetTextColor(hdcMem, RGB(220, 20, 60)); // Crimson Red
                    if (g_game.player_is_white) {
                        DrawTextW(hdcMem, L"Шах! Ваш ход (Белые)", -1,
&textRect, DT_SINGLELINE | DT_LEFT | DT_VCENTER);
                    } else {
                        DrawTextW(hdcMem, L"Шах! Ваш ход (Черные)", -1,
&textRect, DT_SINGLELINE | DT_LEFT | DT_VCENTER);
                    }
                } else {
                    if (g_game.player_is_white) {
                        DrawTextW(hdcMem, L"Ваш ход (Белые)", -1, &textRect,
DT_SINGLELINE | DT_LEFT | DT_VCENTER);
                    } else {
                        DrawTextW(hdcMem, L"Ваш ход (Черные)", -1, &textRect,
DT_SINGLELINE | DT_LEFT | DT_VCENTER);
                    }
                }
            } else {
                if (in_chk) {
                    SetTextColor(hdcMem, RGB(220, 20, 60));
                    if (g_game.player_is_white) {
                        DrawTextW(hdcMem, L"Шах! Ход компьютера...", -1,
&textRect, DT_SINGLELINE | DT_LEFT | DT_VCENTER);
                    } else {
                        DrawTextW(hdcMem, L"Шах! Ход компьютера (Белые)...", -1,
&textRect, DT_SINGLELINE | DT_LEFT | DT_VCENTER);
                    }
```

```
            } else {
                if (g_game.player_is_white) {
                    DrawTextW(hdcMem, L"Ход компьютера...", -1, &textRect,
DT_SINGLELINE | DT_LEFT | DT_VCENTER);
                } else {
                    DrawTextW(hdcMem, L"Ход компьютера (Белые)...", -1,
&textRect, DT_SINGLELINE | DT_LEFT | DT_VCENTER);
                }
            }
        }
    } else {
        DrawTextW(hdcMem, L"Игра завершена", -1, &textRect,
DT_SINGLELINE | DT_LEFT | DT_VCENTER);
    }
    SelectObject(hdcMem, hOldFont);

    // Вывод на реальный HDC
    BitBlt(hdc, 0, 0, rect.right, rect.bottom, hdcMem, 0, 0, SRCCOPY);

    SelectObject(hdcMem, hOldBmp);
    DeleteObject(hbmMem);
    DeleteObject(hdcMem);

    EndPaint(hwnd, &ps);
    break;
}

case WM_LBUTTONDOWN: {
    if (g_in_splash_screen) {
        break;
    }
    int x = LOWORD(lParam);
    int y = HIWORD(lParam);

    // Проверяем клик по шахматной доске
    int c = (x - BOARD_OFFSET_X) / CELL_SIZE;
    int r = (y - BOARD_OFFSET_Y) / CELL_SIZE;

    if (r >= 0 && r < BOARD_SIZE && c >= 0 && c < BOARD_SIZE) {
        GameResult result = check_game_result(&g_game);
        if (result != GAME_ONGOING) {
            break;
        }

        // Ход разрешен только в ход игрока
        bool is_player_turn = (g_game.white_turn == g_game.player_is_white);
```

```
            if (!is_player_turn) {
                break;
            }

            char clickedPiece = g_game.board[r][c];
            bool is_player_piece = g_game.player_is_white ? (clickedPiece >= 'A'
&& clickedPiece <= 'Z') : (clickedPiece >= 'a' && clickedPiece <= 'z');

            if (selected_row == -1) {
                // Выбираем фигуру
                if (clickedPiece != '.' && is_player_piece) {
                    selected_row = r;
                    selected_col = c;
                    InvalidateRect(hwnd, NULL, TRUE);
                }
            } else {
                // Ходим выбранной фигурой
                if (r == selected_row && c == selected_col) {
                    // Кликнули по той же фигуре — сбрасываем выбор
                    selected_row = -1;
                    selected_col = -1;
                    InvalidateRect(hwnd, NULL, TRUE);
                } else if (clickedPiece != '.' && is_player_piece) {
                    // Выбираем другую фигуру игрока
                    selected_row = r;
                    selected_col = c;
                    InvalidateRect(hwnd, NULL, TRUE);
                } else {
                    // Пытаемся сделать ход
                    char move_text[32];
                    move_text[0] = 'a' + selected_col;
                    move_text[1] = '8' - selected_row;
                    move_text[2] = 'a' + c;
                    move_text[3] = '8' - r;
                    move_text[4] = '\0';

                    if (make_player_move(&g_game, move_text)) {
                        selected_row = -1;
                        selected_col = -1;
                        InvalidateRect(hwnd, NULL, TRUE);

                        // Проверяем результат нашего хода
                        check_and_handle_game_over(hwnd);

                        // Если игра продолжается, запускаем бота через таймер
```

```
                    if (check_game_result(&g_game) == GAME_ONGOING) {
                        InvalidateRect(hwnd, NULL, TRUE);
                        SetTimer(hwnd, 1, 500, NULL); // Запуск бота через 500мс
                    }
                } else {
                    // Неверный ход — сбрасываем выделение
                    selected_row = -1;
                    selected_col = -1;
                    InvalidateRect(hwnd, NULL, TRUE);
                }
            }
        }
        break;
    }

    case WM_TIMER: {
        if (wParam == 1) {
            KillTimer(hwnd, 1);
            if (is_bot_turn(&g_game)) {
                bot_make_move(&g_game);
                InvalidateRect(hwnd, NULL, TRUE);

                // Проверяем результат после хода компьютера
                check_and_handle_game_over(hwnd);
            }
        }
        break;
    }

    case WM_COMMAND: {
        switch (LOWORD(wParam)) {
            case ID_BTN_PLAY:
                g_in_splash_screen = false;
                ShowWindow(GetDlgItem(hwnd, ID_BTN_PLAY), SW_HIDE);
                ShowWindow(GetDlgItem(hwnd, ID_BTN_NEW), SW_SHOW);
                ShowWindow(GetDlgItem(hwnd, ID_BTN_HISTORY),
SW_SHOW);
                ShowWindow(GetDlgItem(hwnd, ID_BTN_EXIT), SW_SHOW);
                InvalidateRect(hwnd, NULL, TRUE);
                if (is_bot_turn(&g_game)) {
                    SetTimer(hwnd, 1, 500, NULL);
                }
                break;
```

```
            case ID_BTN_NEW:
                game_init(&g_game);
                selected_row = -1;
                selected_col = -1;
                game_over_notified = false;
                InvalidateRect(hwnd, NULL, TRUE);
                if (is_bot_turn(&g_game)) {
                    SetTimer(hwnd, 1, 500, NULL);
                }
                break;

            case ID_BTN_HISTORY:
                show_game_history_gui(hwnd);
                break;

            case ID_BTN_EXIT:
                DestroyWindow(hwnd);
                break;
        }
        break;
    }

    case WM_DESTROY:
        // Удаляем созданные шрифты
        if (hFontUI) DeleteObject(hFontUI);
        if (hFontPieces) DeleteObject(hFontPieces);
        if (hFontLabels) DeleteObject(hFontLabels);

        shutdown_gdiplus();
        PostQuitMessage(0);
        break;

    default:
        return DefWindowProc(hwnd, msg, wParam, lParam);
  }
  return 0;
}

// Запуск GUI
void run_gui(void) {
  HINSTANCE hInstance = GetModuleHandle(NULL);

  WNDCLASSW wc = {0};
  wc.lpfnWndProc = WndProc;
  wc.hInstance = hInstance;
  wc.hbrBackground = (HBRUSH)(COLOR_BTNFACE + 1);
```

```
    wc.lpszClassName = L"ChessGUIClass";
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);

    if (!RegisterClassW(&wc)) {
        return;
    }

    // Вычисляем размер окна, чтобы клиентская область была 650x500
    RECT rect = { 0, 0, 650, 500 };
    AdjustWindowRect(&rect, WS_OVERLAPPEDWINDOW &
~WS_THICKFRAME & ~WS_MAXIMIZEBOX, FALSE);

    HWND hwnd = CreateWindowW(L"ChessGUIClass", L"Шахматы (Пешки и
Короли)",
                              WS_OVERLAPPEDWINDOW & ~WS_THICKFRAME &
~WS_MAXIMIZEBOX,
                              CW_USEDEFAULT, CW_USEDEFAULT,
                              rect.right - rect.left, rect.bottom - rect.top,
                              NULL, NULL, hInstance, NULL);

    if (!hwnd) {
        return;
    }

    ShowWindow(hwnd, SW_SHOW);
    UpdateWindow(hwnd);

    MSG msg;
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}
```

chess.h:

```
#ifndef CHESS_H
#define CHESS_H

#include <stdbool.h>

#define BOARD_SIZE 8
#define MAX_MOVES 256
```

```
#define MAX_MOVE_TEXT 8
#define HISTORY_FILE "game_history.txt"

typedef enum {
    GAME_NONE = 0,
    GAME_ONGOING,
    GAME_WHITE_WIN,
    GAME_BLACK_WIN,
    GAME_DRAW
} GameResult;

typedef struct {
    char board[BOARD_SIZE][BOARD_SIZE];
    bool white_turn;
    int move_count;
    char moves[MAX_MOVES][MAX_MOVE_TEXT];
    bool player_is_white;
} Game;

void game_init(Game *g);
void print_board(const Game *g);
bool is_bot_turn(const Game *g);
bool is_valid_move(const Game *g, int fr, int fc, int tr, int tc);
bool is_in_check(const Game *g, bool white_king);
bool is_legal_move(const Game *g, int fr, int fc, int tr, int tc);
bool has_legal_moves(const Game *g, bool white_player);
bool make_player_move(Game *g, const char *move_text);
GameResult check_game_result(const Game *g);
int bot_make_move(Game *g);
void save_game_history(const Game *g, GameResult result);
void show_game_history(void);
int asm_abs(int x);
void run_gui(void);

#endif // CHESS_H
```