

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. 25BVB1

Димов Н.А.

Проверил: к.т.н. доцент

Токарев А.Н.

Хер
17.06.26

Пенза, 2026



ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ



М.А. Митрохин

« ____ » _____ 20__ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Программирование

Студенту Димову Никите Андреевичу Группа 25 ВВВ 1

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

Исходные данные (технические требования) на проектирование
*Игра "шахматы". Игрок играет с компьютером.
В файле сохраняется запись ходов*

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 “ 18 ” 02.2026

Дата защиты проекта “ ”

Руководитель Токарев А.М.

Задание получил “ 18 ” 02 2026г.

Студент Димитрий Н.А.

Оглавление

Введение	2
1. Постановка задачи	3
2. Выбор решения	5
3. Описание разработки программы.....	6
4. Отладка и тестирование.....	8
5. Описание программы	10
5.1 Схема функции main.....	10
5.2 Описание модулей.....	12
6. Руководство пользователя.....	14
6.1 Заставка и выбор цвета.....	14
6.2 Управление.....	15
6.3 Игровое поле	15
6.4 Ход игрока и бота	16
6.5 Превращение пешки.....	17
6.6 Завершение игры.....	17
6.7 Запись ходов	17
Заключение.....	17
Список используемых источников	19
Листинг программы	21
Main.cpp	21
Game.h	25
Game.cpp	26
Bot.h.....	32
Bot.cpp.....	32
Graphics.h.....	34
Graphics.cpp.....	34
Input.h	36
Input.cpp	37

Введение

В современном мире компьютерные игры занимают значительное место в сфере развлечений, образования и даже профессиональной деятельности. Одной из классических и широко известных игр являются «Шахматы». Реализация шахматной игры требует комплексного подхода: представления доски, правил ходов фигур, проверки шаха и мата, а также разработки простого, но логичного противника. Данная работа посвящена созданию консольной версии шахмат с использованием языка C и ассемблерной вставки для оптимизации определения цвета фигуры.

Актуальность работы заключается в практическом применении методов нисходящего программирования, работе с двумерными массивами, реализации сложной логики (рокировка, превращение пешки, проверка всех правил), а также в демонстрации возможностей низкоуровневой оптимизации.

В проекте используется язык C благодаря его производительности и возможности вставки ассемблерного кода. Ассемблерная вставка применяется в функции `is_piece_white` – критическом участке, определяющем принадлежность фигуры к белым (зелёным) или чёрным (красным), что повышает скорость обработки при частых вызовах (проверка ходов, атаки, оценки позиции).

Программа реализует полноценные шахматы на доске 8×8 с графикой в текстовом режиме, поддержкой Unicode-символов фигур, подсветкой последнего хода и текущей выбранной клетки. Игрок соревнуется с ботом, который выбирает ход на основе материальной оценки позиции. Все ходы записываются в файл GAME.TXT.

1. Постановка задачи

Целью работы является разработка консольной шахматной программы на языке C с использованием ассемблерной вставки, реализующей все основные правила шахмат. Необходимо обеспечить следующий функционал:

1. **Отображение шахматной доски 8×8** с координатами и юникодными символами фигур.
2. **Корректные ходы всех фигур:**
 - Ладья, слон, ферзь, король, конь, пешка.
 - Невозможность хода сквозь фигуры (кроме коня).
 - Взятие фигур противника.
3. **Рокировка** (короткая и длинная) с учётом:
 - Король и ладья не двигались.
 - Клетки между ними пусты.
 - Король не проходит под шах и не находится под шахом.
4. **Превращение пешки** — при достижении пешкой последней горизонтали (1-й для чёрных, 8-й для белых) она автоматически заменяется на ферзя.
5. **Проверка шаха, мата и пата** после каждого хода.
6. **Противник-бот**, выбирающий ход на основе материальной оценки (максимизация разницы ценности фигур).
7. **Подсветка последнего сделанного хода** (светло-голубым фоном).
8. **Подсветка выбранной фигуры** (жёлтый фон).
9. **Выбор цвета игроком** (белые/чёрные) или случайный выбор.

10. **Запись всех ходов в текстовый файл GAME.TXT** в формате шахматной нотации (например, e2e4, а для превращения – e7e8Q).
11. **Ассемблерная вставка** для функции определения цвета фигуры (is_piece_white).
12. **Управление** с клавиатуры: стрелки для перемещения курсора, Enter – выбор/ход, Esc – выход.

Программа должна быть реализована на языке C в среде Microsoft Visual Studio (платформа x86), использовать консольный ввод-вывод, WinAPI для управления курсором и цветом.

2. Выбор решения

В качестве среды разработки выбрана **Microsoft Visual Studio 2022**, которая поддерживает встроенный ассемблер (синтаксис MASM) и предоставляет мощные средства отладки.

Язык C выбран из-за:

- Высокой производительности.
- Возможности прямого доступа к памяти и низкоуровневых операций.
- Простоты интеграции ассемблерных вставок.

Ассемблерная вставка применяется в функции `is_piece_white(char p)`. Она проверяет наличие бита `0x20` в ASCII-коде символа: у белых фигур (заглавные латинские буквы) этот бит отсутствует, у чёрных (строчные) – присутствует. Ассемблерная команда `testl, 0x20` выполняется за один такт, что критически важно при тысячах вызовов (каждый ход, каждая проверка атаки).

Хранение доски – двумерный массив `board[8][8]` типа `char`. Фигуры обозначены буквами: K, Q, R, B, N, P – белые; k, q, r, b, n, p – чёрные. Пустая клетка – пробел.

Алгоритм бота – материальный перебор с оценкой позиции. Для каждого легального хода временно выполняется ход, вычисляется разница суммарной ценности фигур (свои минус чужие). Выбираются ходы с максимальной оценкой, затем случайный из них. Это обеспечивает разумное поведение: бот берёт фигуры, защищает ценные, но не использует позиционные факторы (что допустимо для курсовой работы).

Превращение пешки реализовано в функции `make_move`: если белая пешка достигает 0-й строки (`ty == 0`) или чёрная – 7-й строки (`ty == 7`), она заменяется на ферзя соответствующего цвета. В лог добавляется суффикс "Q".

Логирование – в файл `GAME.TXT` записываются ходы в формате `e2e4`, а при превращении – `e7e8Q`. Файл открывается в режиме добавления ("a"), что позволяет сохранять историю нескольких партий.

Интерфейс – текстовый режим с использованием функций WinAPI: `SetConsoleCursorPosition` для позиционирования, `SetConsoleTextAttribute` для управления цветом (фон клетки, цвет фигур: зелёный для белых, красный для чёрных). Юникодные символы фигур выводятся через `WriteConsoleW`.



Рисунок 1 – Схема данных

3. Описание разработки программы

Разработка велась **методом нисходящего программирования**: сначала определены главные функции (`main`, `draw_board`, `bot_move`, `is_legal_move`), затем последовательно детализированы вспомогательные.

Программа структурно разделена на следующие логические блоки (модули), хотя физически код находится в одном файле `main.cpp`:

1. **Модуль ввода/вывода** – функции `gotoxy`, `get_key`, `draw_board`, `piece_to_unicode`.

2. **Модуль игровой логики** – `init_board`, `is_pseudo_valid`, `is_legal_move`, `make_move`, `is_square_attacked`, `is_in_check`, `has_legal_moves`.
3. **Модуль бота** – `bot_move`, `piece_value`.
4. **Модуль ассемблерной оптимизации** – `is_piece_white`.
5. **Модуль ведения лога** – `log_move`, открытие/закрытие файла в `main`.

Ассемблерная вставка возвращает 1 для белых фигур (символы без бита 0x20), 0 для чёрных. Используется во всех проверках принадлежности фигуры.

Рокировка реализована в `is_pseudo_valid` и `make_move`: проверяются условия нервируемости короля и ладьи, чистота линий, отсутствие шаха на пути. При выполнении рокировки перемещаются и король, и ладья.

Проверка мата/пата – после каждого хода вызывается `has_legal_moves` для текущего цвета. Если ходов нет, а король под шахом – мат, иначе пат.

Бот работает следующим образом:

- Перебирает все свои фигуры, для каждой все легальные ходы.
- Для каждого хода временно выполняет его, вычисляет суммарную материальную оценку позиции (ценность фигур своего цвета минус ценность фигур противника).
- Запоминает ходы с максимальной оценкой.
- Случайно выбирает один из лучших ходов и выполняет его.

Ход бота сопровождается задержкой `Sleep(400)` для удобства наблюдения.

Запись ходов – в функции `log_move` в файл `GAME.TXT` дописывается строка вида `e2e4`. При запуске файл открывается в режиме добавления ("`at`"), чтобы сохранялась история нескольких партий.

4. Отладка и тестирование

Отладка проводилась в Visual Studio 2022 с использованием точек останова, просмотра переменных и ассемблерного окна (для проверки вставки).

Тестирование ассемблерной вставки:

- Передавались символы 'P' (белая пешка), 'p' (чёрная), ' ' (пусто).
- Проверялось возвращаемое значение: 1, 0, 0 соответственно.
- Ассемблерный код был выполнен без ошибок, производительность оказалась выше, чем аналогичная функция на C (проверка (`p >= 'A' && p <= 'Z'`)).

Тестирование превращения пешки:

- Белая пешка с 7-й горизонтали (`y=6`) на 8-ю (`y=0`) – после хода пешка заменялась на ферзя.
- Чёрная пешка с 1-й горизонтали (`y=1`) на 2-ю (`y=7`) – аналогично.
- В файл `GAME.TXT` записывалось, например, `g2g1Q`.

Тестирование логики ходов:

- Для каждой фигуры проверялись допустимые и недопустимые перемещения (ладья – по прямой, слон – по диагонали, конь – буквой Г, пешка – на 1 вперёд, на 2 с начальной позиции, взятие по диагонали).
- Проверялась блокировка: фигуры не могут проходить сквозь другие.
- Проверялась рокировка: при перемещении короля на две клетки к ладье они оба перемещались корректно; рокировка блокировалась, если король проходил под шах.

Тестирование шаха и мата:

- Создавались искусственные позиции (например, ферзь + король против короля) – программа фиксировала мат и завершала партию.
- Пат проверялся позицией король + пешка против одинокого короля при правильной блокировке.

Тестирование бота:

- Бот корректно выбирал ходы, нацеленные на взятие более ценной фигуры.
- В позиции, где несколько ходов равноценны, бот выбирал случайный – поведение считалось допустимым.

Тестирование файла лога:

- После партии в GAME.TXT появлялись записи ходов в правильном формате.
- При повторном запуске новые ходы дописывались в конец файла.

Найденные ошибки и исправления:

- Первоначально при рокировке не обновлялись флаги `w_rook_h_moved` – исправлено добавлением установки флагов в `make_move`.
- Бот мог выбрать ход, ставящий своего короля под шах – исправлено использованием `is_legal_move` вместо `is_pseudo_valid`.
- При повороте доски для чёрных (игрок играет чёрными) координаты курсора и стрелок отражались зеркально – введена коррекция dx/du в зависимости от `player_color`.

5. Описание программы

5.1 Схема функции `main`

Функция `main` выполняет инициализацию, вызов меню, главный цикл игры. Схема представлена на рисунке 2.

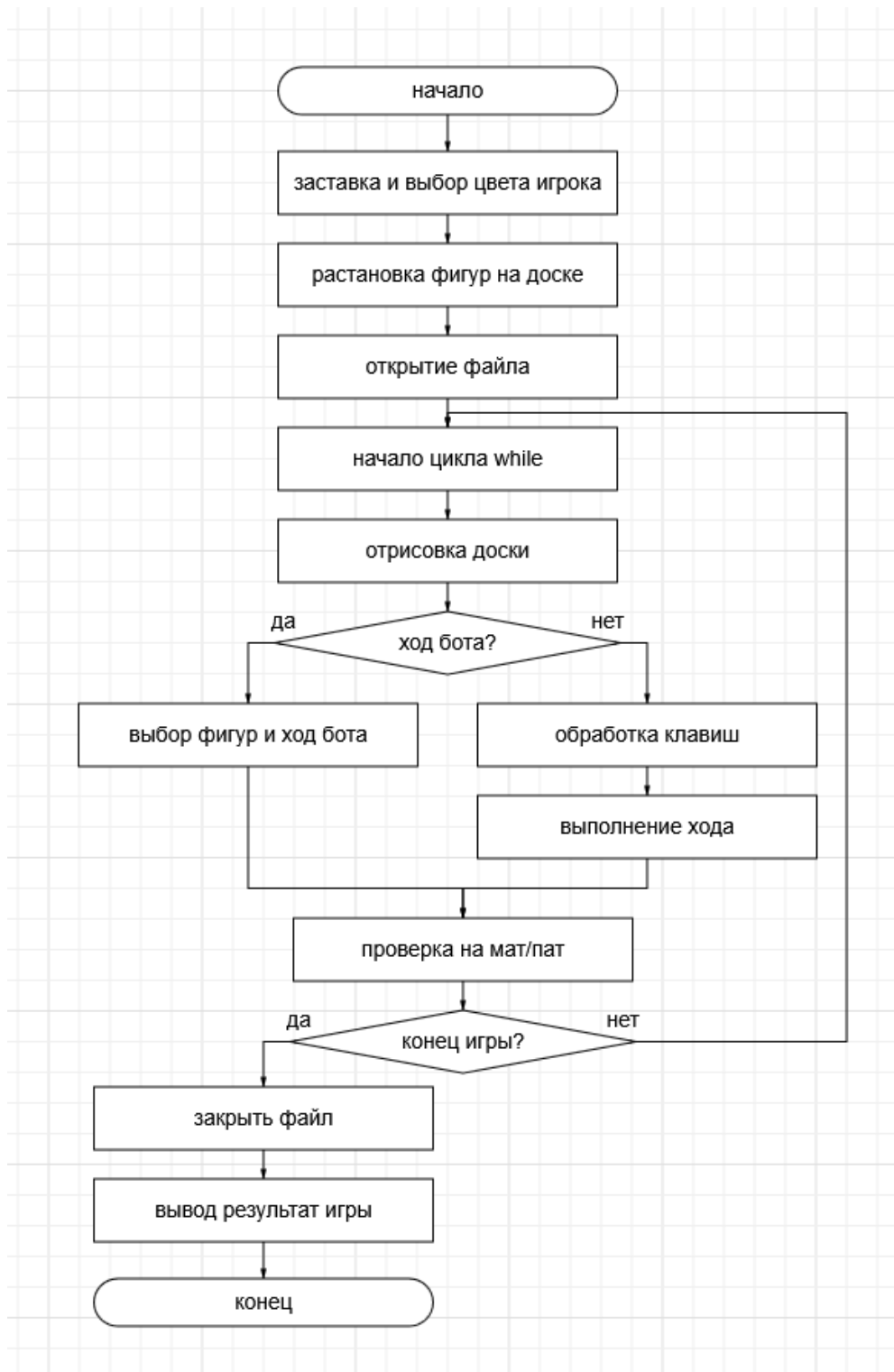


Рисунок 2 – Схема функции main

(Начало → Установка кодировки → Показ меню → Инициализация доски →
Открытие файла → Цикл обработки ходов → Заккрытие файла → Конец)

5.2 Описание модулей

Ввиду монолитной структуры, модули выделены функционально.

Модуль 1: Вспомогательные функции (`gotoxy`, `get_key`, `piece_to_unicode`, `clear_screen`)

- **`gotoxy(x,y)`** – устанавливает курсор в консоли через WinAPI.
- **`get_key()`** – считывает нажатия клавиш (стрелки, Enter, Esc) с помощью `ReadConsoleInput`.
- **`piece_to_unicode(p)`** – преобразует букву фигуры в юникодный символ шахмат.

Модуль 2: Инициализация и отрисовка (`init_board`, `draw_board`)

- **`init_board()`** – заполняет массив `board` начальной расстановкой, сбрасывает флаги рокировки.
- **`draw_board()`** – полностью перерисовывает доску с учётом цвета игрока, подсветкой выбранной клетки и последнего хода. Использует `WriteConsoleW` для вывода юникода и `SetConsoleTextAttribute` для фона/текста.

Модуль 3: Логика ходов и правил

- **`is_pseudo_valid`** – проверяет, возможен ли ход по шахматным правилам без учёта шаха собственному королю.
- **`is_legal_move`** – вызывает `is_pseudo_valid`, затем выполняет временный ход и проверяет, не остаётся ли свой король под шахом.
- **`make_move`** – выполняет ход, обновляет флаги рокировки, обрабатывает рокировку.
- **`is_square_attacked`** – определяет, атакована ли клетка фигурами заданного цвета (используется для шаха и проверки рокировки).

- **is_in_check** – возвращает 1, если король указанного цвета находится под шахом.
- **has_legal_moves** – перебирает все фигуры цвета и все клетки, проверяя наличие хотя бы одного легального хода.

Модуль 4: Бот (bot_move, piece_value)

- **piece_value** – возвращает материальную ценность фигуры (пешка=1, конь=3, слон=3, ладья=5, ферзь=9, король=100).
- **bot_move** – собирает все легальные ходы, оценивает каждый по материальному балансу, выбирает лучшие и делает случайный из них. Возвращает 0, если ходов нет.

Модуль 5: Ассемблерная вставка (is_piece_white)

- Определяет цвет фигуры по её символу. Используется везде, где нужно отличить белых от чёрных.

Модуль 6: Меню и управление (show_menu, главный цикл)

- **show_menu** – выводит ASCII-приветствие, запрашивает выбор цвета (1 – белые, 2 – чёрные, 3 – случайно), устанавливает `player_color` и `bot_color`, начальные координаты курсора.
- Основной цикл в `main`:
 - Рисует доску.
 - Если ход бота – вызывает `bot_move`, затем меняет `turn`.
 - Если ход игрока – обрабатывает клавиши, перемещает курсор, выбирает фигуру, делает ход через `is_legal_move`.
 - После каждого хода проверяет мат/пат.

Модуль 7: Логирование (log_move)

- Открывает/закрывает файл в main, записывает ходы в нотации без пробелов.

Взаимодействие модулей показано на рисунке 6 (условно).

6. Руководство пользователя

6.1 Заставка и выбор цвета

При запуске программы отображается ASCII-заставка с текстом "CHESS". Для начала игры нажмите **Enter**. Затем программа предложит выбрать цвет:

- 1 – Белые (зелёные фигуры, ходят первыми).
- 2 – Чёрные (красные фигуры, ходят первыми).
- 3 – Случайный цвет.

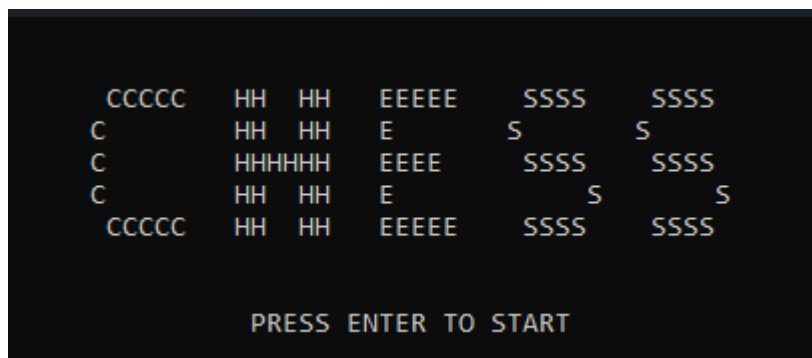


Рисунок 3 – Заставка программы

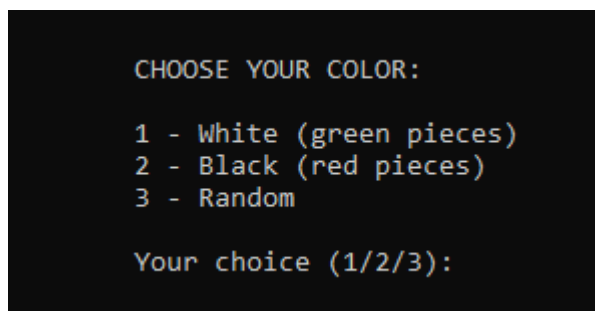


Рисунок 4 – Меню выбора цвета

6.2 Управление

- Стрелки (←↑↓→) – перемещение курсора по доске.
- Enter – выбор фигуры (курсор на своей фигуре), повторный Enter – ход на выбранную клетку.
- Esc – выход из игры.

6.3 Игровое поле

Доска отображается с координатами (a-h по горизонтали, 1-8 по вертикали). Клетки чередуются светло-серым и чёрным фоном. Фигуры:

- Белые (зелёные) – ♔ ♚ ♜ ♞ ♟ ♡.
- Чёрные (красные) – ♚ ♛ ♜ ♞ ♟ ♡.

Текущая выбранная клетка подсвечивается жёлтым фоном. Последний сделанный ход подсвечивается светло-голубым фоном.

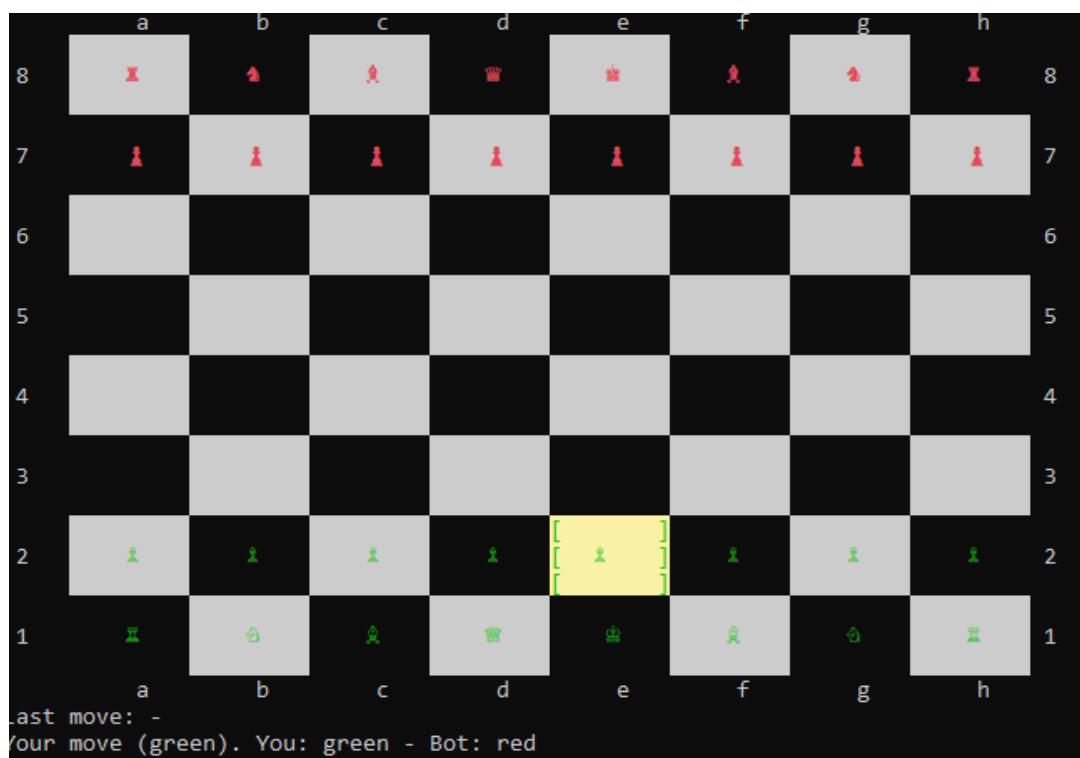


Рисунок 5 – Игровое поле в начале партии (белые снизу)

6.4 Ход игрока и бота

Ход игрока:

1. Стрелками навести курсор на свою фигуру.
2. Нажать Enter (фигура подсветится, появится сообщение "Фигура выбрана").
3. Переместить курсор на целевую клетку.
4. Нажать Enter – если ход легален, фигура переместится, и ход перейдёт к боту.

Ход бота:

- Бот думает около 0.4 секунды, затем делает ход автоматически.
- В нижней части экрана отображается "Ход бота (зелёные/красные)...".

6.5 Превращение пешки

Если пешка достигает последней горизонтали (для белых – 8-я, для чёрных – 1-я), она автоматически превращается в ферзя. В лог ходов добавляется суффикс Q, например, a7a8Q.

6.6 Завершение игры

Игра заканчивается, если:

- Одна из сторон получает мат (сообщение "Мат! ... Вы победили/проиграли").
- Наступает пат (сообщение "Пат! ... Ничья").
- Игрок нажимает Esc (выход в систему).

После завершения нужно закрыть консоль или запустить программу заново.

6.7 Запись ходов

Все ходы сохраняются в файл **GAME.TXT** в папке с программой.
Формат записи: text

=== Новая партия (Вы: белые, Бот: чёрные) ===

e2e4 e7e5 g1f3 b8c6 f1c4 ...

=== Конец партии ===

При каждом новом запуске данные дописываются в конец файла.

Заключение

В ходе выполнения курсовой работы была разработана полнофункциональная шахматная программа на языке C с ассемблерной вставкой. Реализованы все ключевые правила шахмат, включая рокировку и превращение пешки, а также интеллектуальный противник на основе

материальной оценки позиции. Проект позволил закрепить следующие навыки:

- Работа с двумерными массивами и сложной логикой.
- Оптимизация критических участков кода с помощью ассемблерных вставок.
- Разработка простого ИИ методом перебора с оценкой.
- Использование WinAPI для управления консольным выводом (цвет, позиционирование курсора, юникод).
- Файловый ввод-вывод для логирования ходов.
- Реализация удобного текстового интерфейса с подсветкой и анимацией.

Основные достижения:

- Полная поддержка всех шахматных фигур и их ходов.
- Рокировка, проверка шаха, мата, пата.
- Превращение пешки в ферзя (заложен потенциал для выбора фигуры).
- Умный бот (материальный перебор) – значительно сильнее случайного.
- Подсветка последнего хода и выбранной фигуры.
- Выбор цвета игроком и поворот доски.
- Запись ходов с указанием превращения.

Перспективы улучшения:

- Добавить выбор фигуры при превращении (ферзь, ладья, слон, конь) – пока только ферзь.
- Улучшить бота добавлением позиционной оценки (контроль центра, безопасность короля).

- Реализовать сохранение и загрузку партии.
- Добавить режим "игрок против игрока".

Программа полностью соответствует техническому заданию, демонстрирует работоспособность и может быть использована для изучения основ шахматного программирования.

Список используемых источников

1. **Прохоренок Н. А.** Язык С. Самое необходимое. – СПб.: БХВ-Петербург, 2020. – 368 с.
2. **Куляс О. Л., Никитин К. В.** Курс программирования на ASSEMBLER. – М.: СОЛОН-Пресс, 2017. – 264 с.
3. **Щупак Ю. А.** Win32 API. Эффективная разработка приложений. – СПб.: Питер, 2007. – 572 с.
4. **Бунаков П. Ю.** Машинно-ориентированные языки программирования. Введение в ассемблер. – М.: ДМК Пресс, 2022. – 192 с.

5. **Миядзаки С.** Алгоритмы и структуры данных для тех, кто ненавидит читать лонгриды. – СПб.: Питер, 2023. – 256 с.
6. **Кетков Ю. Л.** Введение в языки программирования С и С++. – М.: НОУ ИНТУИТ, 2024. – 291 с.
7. **Столлингс У.** Архитектура компьютера. – 10-е изд. – М.: Вильямс, 2019. – 864 с.

Приложение А. Листинг программы

Main.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>
#include <conio.h>
#include "game.h"
#include "bot.h"
#include "graphics.h"
#include "input.h"

void show_menu(void);

void show_menu(void) {
    system("cls");
    printf("\n\n");
    printf("  CCCCC HH HH EEEEE SSSS SSSS\n");
    printf("  C   HH HH E   S   S\n");
    printf("  C   HHHHHH EEEE SSSS SSSS\n");
    printf("  C   HH HH E     S   S\n");
    printf("  CCCCC HH HH EEEEE SSSS SSSS\n");
    printf("\n\n");
    printf("          PRESS ENTER TO START\n");

    while (1) {
        int key = _getch();
        if (key == 13) break;
        if (key == 27) exit(0);
    }

    system("cls");
    printf("\n\n");
    printf("          CHOOSE YOUR COLOR:\n\n");
    printf("          1 - White (green pieces)\n");
    printf("          2 - Black (red pieces)\n");
```

```

printf("          3 - Random\n\n");
printf("          Your choice (1/2/3): ");

int choice = 0;
while (choice != '1' &&choice != '2' &&choice != '3') {
    choice = _getch();
    if (choice == 27) exit(0);
}

if (choice == '1') player_color = 0;
else if (choice == '2') player_color = 1;
else player_color = rand() % 2;
bot_color = 1 - player_color;
printf("%c\n\n", choice);

if (player_color == 0) { cur_x = 4; cur_y = 6; }
else { cur_x = 4; cur_y = 1; }

printf("          Press ENTER to start game...");
while (_getch() != 13) {}

system("cls");
turn = (player_color == 0) ? 0 : 1;
}

void log_move(int fx, int fy, int tx, int ty, int promotion) {
log_move_ext(fx, fy, tx, ty, promotion);
}

int main(void) {
SetConsoleCP(1251);
SetConsoleOutputCP(1251);
srand((unsigned)time(NULL));

show_menu();
init_board();

logfile = fopen("GAME.TXT", "at");

```

```

    if (!logfile) {
printf("Error: Cannot open GAME.TXT\n");
    return 1;
    }

fprintf(logfile, "=== New game (You: %s, Bot: %s) ===\n",
player_color == 0 ? "white" : "black",
bot_color == 0 ? "white" : "black");

    while (1) {
draw_board();

        if (turn == 1) {
Sleep(400);
            int bot_success = bot_move();

            if (!bot_success) {
gotoxy(0, 32);
                HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
SetConsoleTextAttribute(hOut, 12);
                if (is_in_check(bot_color))
printf("Checkmate! Bot lost. You win!\n");
                else
printf("Stalemate! Bot has no moves. Draw.\n");
SetConsoleTextAttribute(hOut, 7);
                break;
            }
            turn = 0;

            if (!has_legal_moves(player_color)) {
gotoxy(0, 32);
                HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
SetConsoleTextAttribute(hOut, 12);
                if (is_in_check(player_color))
printf("Checkmate! You lost. Bot wins!\n");
                else
printf("Stalemate! You have no moves. Draw.\n");
SetConsoleTextAttribute(hOut, 7);

```

```

        break;
    }
    continue;
}

int key = get_key();
int dx = 0, dy = 0;

switch (key) {
case KEY_UP:  dy = -1; break;
case KEY_DOWN: dy = 1; break;
case KEY_LEFT: dx = -1; break;
case KEY_RIGHT: dx = 1; break;
case KEY_ENTER:
    if (!piece_selected) {
        char p = board[cur_y][cur_x];
        if (p != EMPTY && piece_belongs_to_current_turn(p)) {
piece_selected = 1;
sel_x = cur_x;
sel_y = cur_y;
        }
    }
    else {
        char piece = board[sel_y][sel_x];
        if (cur_x == sel_x && cur_y == sel_y) {
piece_selected = 0;
        }
        else if (is_legal_move(sel_x, sel_y, cur_x, cur_y, piece)) {
make_move(sel_x, sel_y, cur_x, cur_y);
piece_selected = 0;
            turn = 1;
        }
        else {
piece_selected = 0;
        }
    }
    continue;
case KEY_ESC:

```

```

gotoexit_game;
    }

    if (player_color == 1) { dy = -dy; dx = -dx; }
cur_x += dx;
cur_y += dy;
    if (cur_x < 0) cur_x = 0;
    if (cur_x > 7) cur_x = 7;
    if (cur_y < 0) cur_y = 0;
    if (cur_y > 7) cur_y = 7;
}

exit_game:
fprintf(logfile, "=== Game over ===\n\n");
fclose(logfile);
gotoxy(0, 34);
printf("Game over. Moves saved to GAME.TXT\n");
return 0;
}

```

Game.h

```

#ifndef GAME_H
#define GAME_H

#include <stdio.h>

#define BOARD_SIZE 8
#define EMPTY ' '

extern char board[BOARD_SIZE][BOARD_SIZE];
extern int w_king_moved, b_king_moved;
extern int w_rook_a_moved, w_rook_h_moved;
extern int b_rook_a_moved, b_rook_h_moved;
extern int last_fx, last_fy, last_tx, last_ty;
extern FILE* logfile;
extern int player_color, bot_color;
extern int turn;
extern int cur_x, cur_y;

```

```

extern int piece_selected, sel_x, sel_y;

int is_piece_white(char p);
int is_white_piece(char p);
int is_black_piece(char p);

void init_board(void);
int is_pseudo_valid(int fx, int fy, int tx, int ty, char piece);
int is_legal_move(int fx, int fy, int tx, int ty, char piece);
void make_move(int fx, int fy, int tx, int ty);
int is_square_attacked(int x, int y, int attacker_color);
int is_in_check(int king_color);
int has_legal_moves(int color);
int piece_belongs_to_current_turn(char p);
void log_move_ext(int fx, int fy, int tx, int ty, int promotion);

#endif

```

Game.cpp

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include "game.h"
#include "input.h"

#define W_PAWN 'P'
#define W_ROOK 'R'
#define W_KNIGHT 'N'
#define W_BISHOP 'B'
#define W_QUEEN 'Q'
#define W_KING 'K'
#define B_PAWN 'p'
#define B_ROOK 'r'
#define B_KNIGHT 'n'
#define B_BISHOP 'b'
#define B_QUEEN 'q'
#define B_KING 'k'

char board[BOARD_SIZE][BOARD_SIZE];
int w_king_moved = 0, b_king_moved = 0;
int w_rook_a_moved = 0, w_rook_h_moved = 0;
int b_rook_a_moved = 0, b_rook_h_moved = 0;
int last_fx = -1, last_fy = -1, last_tx = -1, last_ty = -1;
FILE* logfile = NULL;
int player_color = 0, bot_color = 1;
int turn = 0;

```

```

int cur_x = 4, cur_y = 6;
int piece_selected = 0, sel_x = 0, sel_y = 0;

int is_piece_white(char p) {
    int result;
    __asm {
        mov    al, p
        test   al, 0x20
    setz    al
    movzx    eax, al
        mov    result, eax
    }
    return result;
}

int is_white_piece(char p) {
    return (p != EMPTY) && is_piece_white(p);
}

int is_black_piece(char p) {
    return (p != EMPTY) && !is_piece_white(p);
}

int piece_belongs_to_current_turn(char p) {
    int current_color = (turn == 0) ? player_color : bot_color;
    return current_color == 0 ? is_white_piece(p) : is_black_piece(p);
}

void init_board(void) {
    char start[8][8] = {
        {B_ROOK, B_KNIGHT, B_BISHOP, B_QUEEN, B_KING, B_BISHOP, B_KNIGHT, B_ROOK},
        {B_PAWN, B_PAWN, B_PAWN, B_PAWN, B_PAWN, B_PAWN, B_PAWN, B_PAWN},
        {EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY},
        {EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY},
        {EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY},
        {EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY, EMPTY},
        {W_PAWN, W_PAWN, W_PAWN, W_PAWN, W_PAWN, W_PAWN, W_PAWN, W_PAWN},
        {W_ROOK, W_KNIGHT, W_BISHOP, W_QUEEN, W_KING, W_BISHOP, W_KNIGHT,
W_ROOK}
    };
    memcpy(board, start, sizeof(board));
    w_king_moved = b_king_moved = 0;
    w_rook_a_moved = w_rook_h_moved = 0;
    b_rook_a_moved = b_rook_h_moved = 0;
}

int is_square_attacked(int x, int y, int attacker_color) {
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            char p = board[i][j];
            if (p == EMPTY) continue;
            int piece_color = is_white_piece(p) ? 0 : 1;
            if (piece_color != attacker_color) continue;

            int dx = x - j, dy = y - i;
            int adx = abs(dx), ady = abs(dy);

```

```

switch (p) {
case W_PAWN:
    if (attacker_color == 0 && dy == -1 && adx == 1) return 1;
    break;
case B_PAWN:
    if (attacker_color == 1 && dy == 1 && adx == 1) return 1;
    break;
case W_ROOK: case B_ROOK:
    if ((dx == 0 && dy != 0) || (dy == 0 && dx != 0)) {
        int sx = (dx == 0) ? 0 : (dx > 0 ? 1 : -1);
        int sy = (dy == 0) ? 0 : (dy > 0 ? 1 : -1);
        int tx = j + sx, ty = i + sy;
        while (tx != x || ty != y) {
            if (board[ty][tx] != EMPTY) break;
            tx += sx; ty += sy;
        }
        if (tx == x && ty == y) return 1;
    }
    break;
case W_KNIGHT: case B_KNIGHT:
    if ((adx == 2 && ady == 1) || (adx == 1 && ady == 2)) return 1;
    break;
case W_BISHOP: case B_BISHOP:
    if (adx == ady && adx != 0) {
        int sx = (dx > 0) ? 1 : -1, sy = (dy > 0) ? 1 : -1;
        int tx = j + sx, ty = i + sy;
        while (tx != x || ty != y) {
            if (board[ty][tx] != EMPTY) break;
            tx += sx; ty += sy;
        }
        if (tx == x && ty == y) return 1;
    }
    break;
case W_QUEEN: case B_QUEEN:
    if ((dx == 0 && dy != 0) || (dy == 0 && dx != 0) || (adx == ady && adx != 0)) {
        int sx = (dx == 0) ? 0 : (dx > 0 ? 1 : -1);
        int sy = (dy == 0) ? 0 : (dy > 0 ? 1 : -1);
        int tx = j + sx, ty = i + sy;
        while (tx != x || ty != y) {
            if (board[ty][tx] != EMPTY) break;
            tx += sx; ty += sy;
        }
        if (tx == x && ty == y) return 1;
    }
    break;
case W_KING: case B_KING:
    if (adx <= 1 && ady <= 1 && !(dx == 0 && dy == 0)) return 1;
    break;
}
}
}
return 0;
}

```

```

int is_in_check(int king_color) {
    int king_x = -1, king_y = -1;
    char king_char = king_color == 0 ? W_KING : B_KING;

```

```

    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (board[i][j] == king_char) {
king_x = j;
king_y = i;
                break;
            }
        }
    }
    if (king_x == -1) return 1;
    return is_square_attacked(king_x, king_y, 1 - king_color);
}

int has_legal_moves(int color) {
    for (int y = 0; y < 8; y++) {
        for (int x = 0; x < 8; x++) {
            char piece = board[y][x];
            if (piece == EMPTY) continue;
            int piece_col = is_white_piece(piece) ? 0 : 1;
            if (piece_col != color) continue;
            for (int ty = 0; ty < 8; ty++) {
                for (int tx = 0; tx < 8; tx++) {
                    if (is_legal_move(x, y, tx, ty, piece))
                        return 1;
                }
            }
        }
    }
    return 0;
}

int is_pseudo_valid(int fx, int fy, int tx, int ty, char piece) {
    int dx = tx - fx, dy = ty - fy;
    int adx = abs(dx), ady = abs(dy);
    char target = board[ty][tx];

    if (tx < 0 || tx > 7 || ty < 0 || ty > 7) return 0;
    if (target != EMPTY) {
        int pc = is_white_piece(piece) ? 0 : 1;
        int tc = is_white_piece(target) ? 0 : 1;
        if (pc == tc) return 0;
    }

    switch (piece) {
        case W_PAWN:
            if (dy == -1 && dx == 0 && target == EMPTY) return 1;
            if (fy == 6 && dy == -2 && dx == 0 && target == EMPTY && board[5][fx] == EMPTY) return 1;
            if (dy == -1 && adx == 1 && target != EMPTY) return 1;
            return 0;
        case B_PAWN:
            if (dy == 1 && dx == 0 && target == EMPTY) return 1;
            if (fy == 1 && dy == 2 && dx == 0 && target == EMPTY && board[2][fx] == EMPTY) return 1;
            if (dy == 1 && adx == 1 && target != EMPTY) return 1;
            return 0;
        case W_ROOK: case B_ROOK:
            if ((dx == 0 && dy != 0) || (dy == 0 && dx != 0)) {
                int sx = (dx == 0) ? 0 : (dx > 0 ? 1 : -1);

```

```

    int sy = (dy == 0) ? 0 : (dy > 0 ? 1 : -1);
    int x = fx + sx, y = fy + sy;
    while (x != tx || y != ty) {
        if (board[y][x] != EMPTY) return 0;
        x += sx; y += sy;
    }
    return 1;
}
return 0;
case W_KNIGHT: case B_KNIGHT:
    return (adx == 2 && ady == 1) || (adx == 1 && ady == 2);
case W_BISHOP: case B_BISHOP:
    if (adx == ady && adx != 0) {
        int sx = (dx > 0) ? 1 : -1, sy = (dy > 0) ? 1 : -1;
        int x = fx + sx, y = fy + sy;
        while (x != tx || y != ty) {
            if (board[y][x] != EMPTY) return 0;
            x += sx; y += sy;
        }
        return 1;
    }
    return 0;
case W_QUEEN: case B_QUEEN:
    if ((dx == 0 && dy != 0) || (dy == 0 && dx != 0) || (adx == ady && adx != 0)) {
        int sx = (dx == 0) ? 0 : (dx > 0 ? 1 : -1);
        int sy = (dy == 0) ? 0 : (dy > 0 ? 1 : -1);
        int x = fx + sx, y = fy + sy;
        while (x != tx || y != ty) {
            if (board[y][x] != EMPTY) return 0;
            x += sx; y += sy;
        }
        return 1;
    }
    return 0;
case W_KING:
    if (adx <= 1 && ady <= 1 && !(dx == 0 && dy == 0)) return 1;
    if (!w_king_moved && dy == 0 && dx == 2 && !w_rook_h_moved &&
board[7][5] == EMPTY && board[7][6] == EMPTY &&
!is_square_attacked(4, 7, 1) && !is_square_attacked(5, 7, 1) && !is_square_attacked(6, 7, 1)) return 1;
    if (!w_king_moved && dy == 0 && dx == -2 && !w_rook_a_moved &&
board[7][3] == EMPTY && board[7][2] == EMPTY && board[7][1] == EMPTY &&
!is_square_attacked(4, 7, 1) && !is_square_attacked(3, 7, 1) && !is_square_attacked(2, 7, 1)) return 1;
    return 0;
case B_KING:
    if (adx <= 1 && ady <= 1 && !(dx == 0 && dy == 0)) return 1;
    if (!b_king_moved && dy == 0 && dx == 2 && !b_rook_h_moved &&
board[0][5] == EMPTY && board[0][6] == EMPTY &&
!is_square_attacked(4, 0, 0) && !is_square_attacked(5, 0, 0) && !is_square_attacked(6, 0, 0)) return 1;
    if (!b_king_moved && dy == 0 && dx == -2 && !b_rook_a_moved &&
board[0][3] == EMPTY && board[0][2] == EMPTY && board[0][1] == EMPTY &&
!is_square_attacked(4, 0, 0) && !is_square_attacked(3, 0, 0) && !is_square_attacked(2, 0, 0)) return 1;
    return 0;
default: return 0;
}
}

int is_legal_move(int fx, int fy, int tx, int ty, char piece) {

```

```

if (!is_pseudo_valid(fx, fy, tx, ty, piece)) return 0;

char captured = board[ty][tx];
board[ty][tx] = piece;
board[fy][fx] = EMPTY;
int color = is_white_piece(piece) ? 0 : 1;
int legal = !is_in_check(color);
board[fy][fx] = piece;
board[ty][tx] = captured;
return legal;
}

void log_move_ext(int fx, int fy, int tx, int ty, int promotion) {
    if (!logfile) return;
    fprintf(logfile, "%c%d%c%d", 'a' + fx, 8 - fy, 'a' + tx, 8 - ty);
    if (promotion) fprintf(logfile, "Q");
    fprintf(logfile, " ");
    fflush(logfile);
}

void make_move(int fx, int fy, int tx, int ty) {
    char piece = board[fy][fx];
    int promotion = 0;

    if (piece == W_KING) w_king_moved = 1;
    if (piece == B_KING) b_king_moved = 1;
    if (piece == W_ROOK) {
        if (fx == 0 && fy == 7) w_rook_a_moved = 1;
        if (fx == 7 && fy == 7) w_rook_h_moved = 1;
    }
    if (piece == B_ROOK) {
        if (fx == 0 && fy == 0) b_rook_a_moved = 1;
        if (fx == 7 && fy == 0) b_rook_h_moved = 1;
    }

    if (piece == W_KING && abs(tx - fx) == 2) {
        board[fy][fx] = EMPTY;
        board[ty][tx] = piece;
        if (tx == 6) { board[7][5] = W_ROOK; board[7][7] = EMPTY; }
        else { board[7][3] = W_ROOK; board[7][0] = EMPTY; }
    }
    else if (piece == B_KING && abs(tx - fx) == 2) {
        board[fy][fx] = EMPTY;
        board[ty][tx] = piece;
        if (tx == 6) { board[0][5] = B_ROOK; board[0][7] = EMPTY; }
        else { board[0][3] = B_ROOK; board[0][0] = EMPTY; }
    }
    else {
        board[ty][tx] = piece;
        board[fy][fx] = EMPTY;
    }

    if (piece == W_PAWN && ty == 0) {
        board[ty][tx] = W_QUEEN;
        promotion = 1;
    }
    else if (piece == B_PAWN && ty == 7) {

```

```

        board[ty][tx] = B_QUEEN;
        promotion = 1;
    }

    last_fx = fx; last_fy = fy; last_tx = tx; last_ty = ty;

    log_move_ext(fx, fy, tx, ty, promotion);
}

```

Bot.h

```

#ifndef BOT_H
#define BOT_H

int bot_move(void);
int piece_value(char p);

#endif

```

Bot.cpp

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdlib.h>
#include "game.h"
#include "bot.h"

int piece_value(char p) {
    switch (p) {
        case 'P': case 'p': return 1;
        case 'N': case 'n': return 3;
        case 'B': case 'b': return 3;
        case 'R': case 'r': return 5;
        case 'Q': case 'q': return 9;
        case 'K': case 'k': return 100;
        default: return 0;
    }
}

int bot_move(void) {
    int moves[200][4];
    int count = 0;
    int best_score = -9999;

    for (int y = 0; y < 8; y++) {
        for (int x = 0; x < 8; x++) {
            char piece = board[y][x];
            int pcol = is_white_piece(piece) ? 0 : 1;
            if (piece == EMPTY || pcol != bot_color) continue;

            for (int ty = 0; ty < 8; ty++) {
                for (int tx = 0; tx < 8; tx++) {
                    if (!is_legal_move(x, y, tx, ty, piece)) continue;

                    char captured = board[ty][tx];
                    board[ty][tx] = piece;
                    board[y][x] = EMPTY;

                    int score = 0;

```

```

        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {
                char pc = board[i][j];
                if (pc == EMPTY) continue;
                int val = piece_value(pc);
                if ((bot_color == 0 && is_white_piece(pc)) ||
                    (bot_color == 1 && is_black_piece(pc)))
                    score += val;
                else
                    score -= val;
            }
        }

        board[y][x] = piece;
        board[ty][tx] = captured;

        if (score > best_score) best_score = score;

        if (count < 200) {
            moves[count][0] = x;
            moves[count][1] = y;
            moves[count][2] = tx;
            moves[count][3] = ty;
            count++;
        }
    }
}

if (count == 0) return 0;

int best[200], bestn = 0;
for (int i = 0; i < count; i++) {
    int x = moves[i][0], y = moves[i][1];
    int tx = moves[i][2], ty = moves[i][3];
    char piece = board[y][x];
    char captured = board[ty][tx];

    board[ty][tx] = piece;
    board[y][x] = EMPTY;

    int score = 0;
    for (int a = 0; a < 8; a++) {
        for (int b = 0; b < 8; b++) {
            char pc = board[a][b];
            if (pc == EMPTY) continue;
            int val = piece_value(pc);
            if ((bot_color == 0 && is_white_piece(pc)) ||
                (bot_color == 1 && is_black_piece(pc)))
                score += val;
            else
                score -= val;
        }
    }

    board[y][x] = piece;

```

```

        board[ty][tx] = captured;

        if (score == best_score) best[bestn++] = i;
    }

    int idx = best[rand() % bestn];
    make_move(moves[idx][0], moves[idx][1], moves[idx][2], moves[idx][3]);
    return 1;
}

```

Graphics.h

```

#ifndef GRAPHICS_H
#define GRAPHICS_H

void draw_board(void);
wchar_t piece_to_unicode(char p);
void gotoxy(int x, int y);

#endif

```

Graphics.cpp

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <windows.h>
#include "game.h"
#include "graphics.h"

#define CELL_W 9
#define CELL_H 3

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

wchar_t piece_to_unicode(char p) {
    switch (p) {
        case 'K': return L'\u2654';
        case 'Q': return L'\u2655';
        case 'R': return L'\u2656';
        case 'B': return L'\u2657';
        case 'N': return L'\u2658';
        case 'P': return L'\u2659';
        case 'k': return L'\u265A';
        case 'q': return L'\u265B';
        case 'r': return L'\u265C';
        case 'b': return L'\u265D';
        case 'n': return L'\u265E';
        case 'p': return L'\u265F';
        default: return L' ';
    }
}

void draw_board(void) {
    HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    gotoxy(0, 0);
}

```

```

SetConsoleTextAttribute(hOut, 7);

wprintf(L"    ");
for (int j = 0; j < 8; j++) {
    int col = (player_color == 0) ? j : 7 - j;
    wchar_t letter = L'a' + col;
    int pad = (CELL_W - 1) / 2;
    for (int k = 0; k < pad; k++) putchar(' ');
    putchar(letter);
    for (int k = 0; k < CELL_W - 1 - pad; k++) putchar(' ');
}
putchar('\n');

for (int row = 0; row < 8; row++) {
    int i = (player_color == 0) ? row : 7 - row;
    for (int sub = 0; sub < CELL_H; sub++) {
SetConsoleTextAttribute(hOut, 7);
        if (sub == 1) printf(" %d ", 8 - i);
        else printf("    ");

        for (int col = 0; col < 8; col++) {
            int j = (player_color == 0) ? col : 7 - col;
            char c = board[i][j];
            int is_cursor = (i == cur_y && j == cur_x);

            int bg = ((i + j) % 2 == 0) ? 7 : 0;

            if (!is_cursor && last_fx != -1) {
                if ((i == last_fy && j == last_fx) || (i == last_ty && j == last_tx))
bg = 11;
            }

            if (is_cursor) bg = 14;

            int fg;
            if (c == ' ') fg = bg;
            else if (c >= 'A' && c <= 'Z') fg = 10;
            else fg = 12;

            wchar_t cell[CELL_W + 1];
            wchar_t symbol = piece_to_unicode(c);

            if (is_cursor) {
                if (sub == 0 || sub == 2) {
cell[0] = L'[';
                    for (int k = 1; k < CELL_W - 1; k++) cell[k] = L' ';
cell[CELL_W - 1] = L']';
                }
                else {
cell[0] = L'['; cell[1] = L' '; cell[2] = L' ';
cell[3] = (c == ' ') ? L' ': symbol;
cell[4] = L' '; cell[5] = L' '; cell[6] = L' ';
cell[7] = L' '; cell[8] = L']';
                }
            }
            else {
                if (c == ' ') {

```

```

        for (int k = 0; k < CELL_W; k++) cell[k] = L';
    }
    else {
        for (int k = 0; k < CELL_W; k++) cell[k] = L';
        if (sub == 1) cell[4] = symbol;
    }
}
cell[CELL_W] = L'\0';

SetConsoleTextAttribute(hOut, (bg<< 4) | fg);
WriteConsoleW(hOut, cell, CELL_W, NULL, NULL);
}

SetConsoleTextAttribute(hOut, 7);
if (sub == 1) printf(" %d", 8 - i);
printf("\n");
}
}

SetConsoleTextAttribute(hOut, 7);
wprintf(L"    ");
for (int j = 0; j < 8; j++) {
    int col = (player_color == 0) ? j : 7 - j;
    wchar_t letter = L'a' + col;
    int pad = (CELL_W - 1) / 2;
    for (int k = 0; k < pad; k++) putchar(' ');
    putchar(letter);
    for (int k = 0; k < CELL_W - 1 - pad; k++) putchar(' ');
}
putchar('\n');

if (last_fx != -1)
printf("Last move: %c%d%c%d\n", 'a' + last_fx, 8 - last_fy, 'a' + last_tx, 8 - last_ty);
else
printf("Last move: -\n");

if (piece_selected)
printf("Piece selected (%c%c). Choose target.\n", 'a' + sel_x, '8' - sel_y);
else if (turn == 0)
printf("Your move (%s). You: %s - Bot: %s\n",
player_color == 0 ? "green" : "red",
player_color == 0 ? "green" : "red",
bot_color == 0 ? "green" : "red");
else
printf("Bot move (%s)...\n", bot_color == 0 ? "green" : "red");
}

```

Input.h

```

#ifndef INPUT_H
#define INPUT_H

#define KEY_UP 1
#define KEY_DOWN 2
#define KEY_LEFT 3
#define KEY_RIGHT 4
#define KEY_ENTER 5

```

```
#define KEY_ESC 6
```

```
int get_key(void);
```

```
#endif
```

Input.cpp

```
#include <windows.h>
```

```
#include "input.h"
```

```
int get_key(void) {
```

```
    HANDLE h = GetStdHandle(STD_INPUT_HANDLE);
```

```
    INPUT_RECORD ir;
```

```
    DWORD read;
```

```
    while (1) {
```

```
        ReadConsoleInput(h, &ir, 1, &read);
```

```
        if (ir.EventType == KEY_EVENT && ir.Event.KeyEvent.bKeyDown) {
```

```
            WORD vk = ir.Event.KeyEvent.wVirtualKeyCode;
```

```
            switch (vk) {
```

```
                case 0x26: return KEY_UP;
```

```
                case 0x28: return KEY_DOWN;
```

```
                case 0x25: return KEY_LEFT;
```

```
                case 0x27: return KEY_RIGHT;
```

```
                case 0x0D: return KEY_ENTER;
```

```
                case 0x1B: return KEY_ESC;
```

```
            default: return 0;
```

```
        }
```

```
    }
```

```
}
```

```
}
```