

**АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ  
ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ  
МОСКОВСКИЙ МЕЖДУНАРОДНЫЙ КОЛЛЕДЖ ЦИФРОВЫХ  
ТЕХНОЛОГИЙ «АКАДЕМИЯ ТОП»**

**УЧЕБНЫЙ ПРОЕКТ**

Уровень профессионального образования:

Среднее профессиональное образование

Программа подготовки специалистов среднего звена

по специальности:

Разработчик Программного Обеспечения

Учебный предмет: Основы алгоритмизации и программирование

Тема: **Шахматный движок для ИИ-соперника**

Преподаватель:

Я. А. Рыхичин

\_\_\_\_\_

подпись

\_\_\_\_\_

инициалы фамилия

Обучающийся:

И.И. Еремеев

\_\_\_\_\_

подпись дата

\_\_\_\_\_

инициалы фамилии

**Санкт-Петербург, 2025**

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
ЦЕЛИ СОЗДАНИЕ ПРИЛОЖЕНИЯ.....	4
ТЕХНОЛОГИИ И ИНСТРУМЕНТЫ РАЗРАБОТКИ.....	6
ОСНОВНЫЕ МОДУЛИ.....	7
1. Представление доски и Генерация ходов (movegen.rs).....	7
2. Оценка позиции (evaluation.rs).....	7
3. Поиск лучшего хода (search.rs).....	8
4. Упорядочивание ходов (move_ordering.rs).....	9
5. Таблица транспозиции (transposition_table.rs).....	10
6. Коммуникация и Протоколы (main.rs / UCI & XBoard).....	10
ТЕСТИРОВАНИЕ.....	12
ЗАКЛЮЧЕНИЕ.....	13
СПИСОК ЛИТЕРАТУРЫ.....	14

## **ВВЕДЕНИЕ**

На сегодняшний день интеллектуальные игры и приложения с элементами искусственного интеллекта занимают одну из ведущих позиций в быстро развивающейся сфере программного обеспечения. С ростом интереса к геймификации обучения, доступностью современных фреймворков разработки и увеличением числа пользователей, желающих развивать стратегическое мышление, такие приложения становятся всё более привлекательными для разработчиков. Шахматы с ИИ-соперником позволяют пользователям погружаться в классическую игру, совершенствовать тактические навыки и прокачивать мастерство против умного алгоритма — от развлекательных партий до серьёзного тренировочного инструмента. В связи с этим создание шахматного приложения с интеграцией ИИ не только способствует развитию навыков программирования и проектирования, но и помогает разработчикам освоить современные технологии в алгоритмах теории игр, битборд-представлении и межъязыковой интеграции.

## ЦЕЛИ СОЗДАНИЕ ПРИЛОЖЕНИЯ

Основной целью данной курсовой работы является изучение современных технологий разработки шахматных движков и применение полученных знаний для создания полноценного шахматного движка с ИИ-соперником. В процессе работы над проектом будут реализованы следующие цели:

1. Изучение технологий разработки шахматных движков: Освоение bitboard-представления доски, Zobrist-хеширования и алгоритмов теории игр (минимакс) для создания высокопроизводительного ИИ на Rust.

2. Интерактивность и шахматная логика: Реализация полной валидации ходов фигур (включая рокировку, взятие на проходе, шах/мат), генерации легальных ходов и проверки специальных условий.

3. ИИ и оценка позиций: Создание интеллектуального соперника с использованием piece-square таблиц (PST), материальной оценки и алгоритмов выбора оптимального хода.

Функции, которые могут быть реализованы с помощью приложения

Разработанное шахматное приложение будет обладать следующими функциями:

1. Генерация ходов: Полная генерация легальных ходов для всех фигур с учетом блокировок, атак и специальных правил (рокировка, en passant).

2. ИИ-соперник: Автоматический выбор лучшего хода через минимакс-алгоритм с оценкой позиций

3. Эвалюация доски: Анализ материального баланса и позиционных преимуществ с использованием PST для всех типов фигур.

Таким образом, данное приложение не только станет практическим примером применения знаний по алгоритмам ИИ и системному программированию, но и обеспечит пользователям увлекательный опыт стратегических партий против умного соперника.

## ТЕХНОЛОГИИ И ИНСТРУМЕНТЫ РАЗРАБОТКИ

1. Rust — мощный системный язык программирования, используемый для реализации шахматного движка и основной логики приложения. Rust обеспечивает:

- Безопасность памяти без использования сборщика мусора
- Высокую производительность, критичную для вычисления ходов в реальном времени
- Эффективную работу с побитовыми операциями (bitboards)
- Типизацию, которая помогает предотвратить многие ошибки на этапе компиляции

2. Visual Studio Code — кроссплатформенная интегрированная среда разработки, используемая для написания, отладки и тестирования кода движка. VS Code обеспечивает:

- Поддержку языка Rust через расширение rust-analyzer: интеллектуальное завершение кода, переход к определению, подсказки по типам, диагностика ошибок в реальном времени и рефакторинг, что значительно ускоряет разработку на Rust.
- Встроенный терминал и поддержку инструментов Cargo: позволяет запускать сборку (cargo build), тесты (cargo test) и другие инструменты
- Мощный встроенный отладчик (с настройкой для Rust): пошаговое выполнение кода, инспекция переменных и состояния bitboard
- Интеграцию с системами контроля версий (Git): удобное управление репозиторием, просмотр изменений и истории прямо в интерфейсе.

# ОСНОВНЫЕ МОДУЛИ

## 1. Представление доски и Генерация ходов (movegen.rs)

Для чего используется:

Этот модуль отвечает за понимание того, где находятся фигуры, и расчет всех возможных легальных ходов для текущей позиции (тихие ходы, взятия, рокировки, превращения пешек).

Как работает:

- Битборды (Bitboards): Движок использует 64-битные целые числа (u64), где каждый бит соответствует клетке шахматной доски. Это позволяет хранить положение всех белых пешек или всех черных коней в одной переменной.
- Побитовые операции: Расчет ходов происходит с помощью быстрых логических операций (AND, OR, XOR, сдвиги). Например, чтобы найти ходы всех пешек вперед, достаточно сдвинуть битборд пешек на 8 бит.
- Precomputed Tables: Для дальнобойных фигур (слоны, ладьи) и прыгающих фигур (кони, короли) используются заранее вычисленные таблицы атак, что исключает сложные вычисления в реальном времени.

## 2. Оценка позиции (evaluation.rs)

Для чего используется:

Статическая функция оценки (SFE), которая определяет, насколько выгодна текущая позиция для стороны, делающей ход, если бы игра закончилась прямо сейчас.

Как работает:

- Материал: Базовый подсчет стоимости фигур (пешка = 100, конь = 320 и т.д.).
- Таблицы позиционных весов (PST): Используются таблицы Piece-Square Tables, которые дают бонусы за хорошее расположение фигур (например, конь в центре доски получает бонус, король в центре в дебюте — штраф).
- Функция возвращает число (в сантипешках), где положительное значение означает преимущество белых, а отрицательное — черных (относительно стороны хода).

### 3. Поиск лучшего хода (search.rs)

Для чего используется:

Это "мозг" движка. Модуль анализирует дерево возможных вариантов на несколько ходов вперед, чтобы выбрать оптимальное решение.

Как работает:

- Минимакс (Minimax): Алгоритм, который предполагает, что соперник всегда сделает лучший ответный ход.
- Альфа-бета отсечение (Alpha-Beta Pruning): Оптимизация минимакса. Если движок находит ход, который гарантированно хуже

уже найденного альтернативного варианта, он прекращает расчет этой ветки ("отсекает" её), экономя ресурсы.

- Итеративное углубление (Iterative Deepening): Поиск начинается с глубины 1, затем 2, 3 и т.д., пока не истечет время. Это позволяет всегда иметь готовый ответ, если время внезапно закончится.
- Quiescence Search (Поиск спокойствия): После завершения основного поиска движок продолжает просчитывать только взятия фигур, чтобы избежать "эффекта горизонта" (ситуации, когда движок оценивает позицию как хорошую, не видя, что следующим ходом у него съедят ферзя).

#### 4. Упорядочивание ходов (move\_ordering.rs)

Для чего используется:

Сортирует список возможных ходов перед тем, как отдать их алгоритму поиска. Чем раньше будет проверен "лучший" ход, тем эффективнее сработает альфа-бета отсечение.

- Как работает: MVV-LVA: (Most Valuable Victim - Least Valuable Attacker) — приоритет отдается взятию ценной фигуры наименее ценной (например, пешка бьет ферзя).
- Killer Moves: Ходы, которые вызвали отсечение (оказались очень сильными) на той же глубине в соседних ветках поиска.
- History  
Heuristic: Статистика ходов, которые часто оказывались успешными в прошлом, даже если они не являются взятиями.

## 5. Таблица транспозиции (transposition\_table.rs)

Для чего используется: Кэш памяти для хранения уже просчитанных позиций. В шахматах часто можно прийти к одной и той же позиции разными путями (перестановка ходов).

Как работает:

- Zobrist Hashing: Каждой позиции присваивается уникальный 64-битный хэш-ключ.
- При анализе новой позиции движок сначала проверяет таблицу. Если позиция уже встречалась и была просчитана на достаточную глубину, движок берет готовую оценку из памяти, не запуская тяжелый поиск заново.

## 6. Коммуникация и Протоколы (main.rs / UCI & XBoard)

Для чего используется:

Обеспечивает связь "немого" алгоритма с графическим интерфейсом (GUI), таким как Arena или WinBoard.

Как работает:

- Слушает стандартный ввод (stdin) в бесконечном цикле.
- Распознает команды протоколов (например, position startpos moves e2e4, go wtime 300000).
- Управляет временем (Time Management): рассчитывает, сколько времени можно потратить на ход, исходя из остатка времени на часах.

- Отправляет результаты (лучший ход, текущую глубину, оценку) в стандартный вывод (stdout).

## ТЕСТИРОВАНИЕ

Для тестирования шахматного движка требуется использовать UCI-совместимый GUI, например Arena Chess GUI. Он запускает движок как дочерний процесс и отправляет/получает данные через буфер. В Arena Chess GUI вся логика действий пользователя продумана, нужно просто подключить пользователя или другой движок, также можно сделать так чтобы два движка играли против друг друга



Рисунок 1. Игра против движка в Arena Chess GUI

## **ЗАКЛЮЧЕНИЕ**

В результате проделанной работы был создан производительный шахматный движок на языке Rust, реализующий основные принципы разработки интеллектуальных игровых систем. Движок успешно применяет эффективные алгоритмы поиска (минимакс с альфа-бета отсечением, итеративное углубление) и оптимизированные структуры данных (битборды, таблицы транспозиции) для быстрого и качественного анализа позиций. Реализация поддержки протоколов UCI и XBoard обеспечивает полную совместимость с популярными графическими интерфейсами, позволяя использовать программу как полноценного шахматного партнера или инструмент для анализа партий. Разработанная модульная архитектура проекта закладывает надежный фундамент для дальнейшего совершенствования оценочных функций и внедрения более сложных эвристик.

## СПИСОК ЛИТЕРАТУРЫ

1. GeeksforGeeks. Алгоритм минимакса в искусственном интеллекте [Электронный ресурс]. URL: <https://www.geeksforgeeks.org/artificial-intelligence/mini-max-algorithm-in-artificial-intelligence/> (дата обращения: 11.12.2025). (Описание фундаментального алгоритма поиска ходов, включая принцип работы, псевдокод и оптимизацию альфа-бета отсечением).

2. Chessprogramming wiki. UCI (Universal Chess Interface) [Электронный ресурс]. URL: <https://www.chessprogramming.org/UCI> (дата обращения: 11.12.2025). (Спецификация универсального протокола взаимодействия шахматного движка с графическим интерфейсом, рассматриваются принципы дизайна, преимущества и критика).

3. Chessprogramming wiki. Bitboards [Электронный ресурс]. URL: <https://www.chessprogramming.org/Bitboards> (дата обращения: 11.12.2025). (Подробное описание структуры данных на основе битовых полей для представления шахматной позиции, включая историю, основные техники и анализ эффективности).

4. Chessprogramming wiki. Transposition Table [Электронный ресурс]. URL: [https://www.chessprogramming.org/Transposition\\_Table](https://www.chessprogramming.org/Transposition_Table) (дата обращения: 11.12.2025). (Описание хеш-таблиц для сохранения результатов поиска, принципов работы, стратегий разрешения коллизий и замены записей).

5. Chessprogramming wiki. Zobrist Hashing [Электронный ресурс]. URL: [https://www.chessprogramming.org/Zobrist\\_Hashing](https://www.chessprogramming.org/Zobrist_Hashing) (дата обращения: 11.12.2025).