Оглавление

[Введение 2](#_Toc121913599)

[1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству 3](#_Toc121913600)

[1. 1. Анализ существующих прототипов 3](#_Toc121913601)

[1. 2. Постановка задачи 6](#_Toc121913602)

[2. Моделирование предметной области 7](#_Toc121913603)

[2. 1. Правила игры «Шахматы» 7](#_Toc121913604)

[2. 2. Описание протокола UCI 8](#_Toc121913605)

[2. 3. Битборды 9](#_Toc121913606)

[3. Проектирование программного средства 10](#_Toc121913607)

[3. 1. Проектирование структуры генератора ходов 10](#_Toc121913608)

[3. 2. Проектирование алгоритма вычисления хэша в позиции 10](#_Toc121913609)

[3. 3. Проектирование алгоритма получения всех ходов в позиции 11](#_Toc121913610)

[4. Конструирование программного средства 13](#_Toc121913611)

[4. 1. Структура модулей программы 13](#_Toc121913612)

[5. Тестирование, проверка работоспособности и анализ полученных результатов 16](#_Toc121913613)

[6. Руководство по установке и пользованию: 18](#_Toc121913614)

[Заключение 18](#_Toc121913615)

[7. Список использованных источников 19](#_Toc121913616)

[Приложение A (обязательное) Блок-схема алгоритма 20](#_Toc121913617)

[20](#_Toc121913618)

[Приложение Б (обязательное) Исходный код программы 21](#_Toc121913619)

# Введение

Шахматы настольная логическая игра с шахматными фигурами на 64-клеточной доске.

В шахматы обычно играют два игрока (именуемые шахматистами) друг против друга. Также возможна игра одной группы шахматистов против другой или против одного игрока, такие партии зачастую именуются консультационными. Кроме того, существует практика сеансов одновременной игры, когда против одного сильного игрока играет несколько противников, каждый на отдельной доске.

Игра происходит на доске, поделённой на равные квадратные клетки, или поля. Размер доски — 8×8 клеток. Вертикальные ряды полей (вертикали) обозначаются латинскими буквами от *а* до *h* слева направо, горизонтальные ряды (горизонтали) — цифрами от 1 до 8 снизу вверх; каждое поле обозначается сочетанием соответствующих буквы и цифры. Поля раскрашены в тёмный и светлый цвета (и называются, соответственно, чёрными и белыми) так, что соседние по вертикали и горизонтали поля раскрашены в разные цвета. Доска располагается так, чтобы ближнее угловое поле справа от игрока было белым (для белых это поле h1, для чёрных — поле а8).

Выигрыш фиксируется в следующих случаях:

Мат. Игрок, поставивший мат, выигрывает.

Один из игроков сдаётся. Игрок, решивший, что дальнейшее сопротивление бессмысленно, может сдаться в любой момент, для этого ему достаточно остановить часы. Его противник объявляется победителем.

Один из игроков просрочил время. В этом случае победа за некоторыми исключениями, описанными в разделе «Контроль времени», присуждается другой стороне.

Данная пояснительная записка содержит следующие основные разделы:

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
2. Моделирование предметной области;
3. Проектирование программного средства;
4. Создание (конструирование) программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию.

# Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству

## Анализ существующих прототипов

Для успешного проектирования программы нужно провести анализ аналогов и прототипов, чтобы выяснить какие наработки уже существуют и как они используются на практике.

**«Arena»** — это полноценный графический интерфейс для шахматных движков. Программа уже поставляется с движком, который позволяет вам играть против компьютера, но также поддерживает почти все популярные движки. Программа легко настраивается и проста в использовании, поэтому вы можете создать именно ту игру, которую хотите. Благодаря регулируемой силе вы можете выбрать уровень сложности и повышать его по мере того, как вы лучше знакомитесь с игрой и более умелыми.

Интерфейс программы приведен на рисунке 1.1.

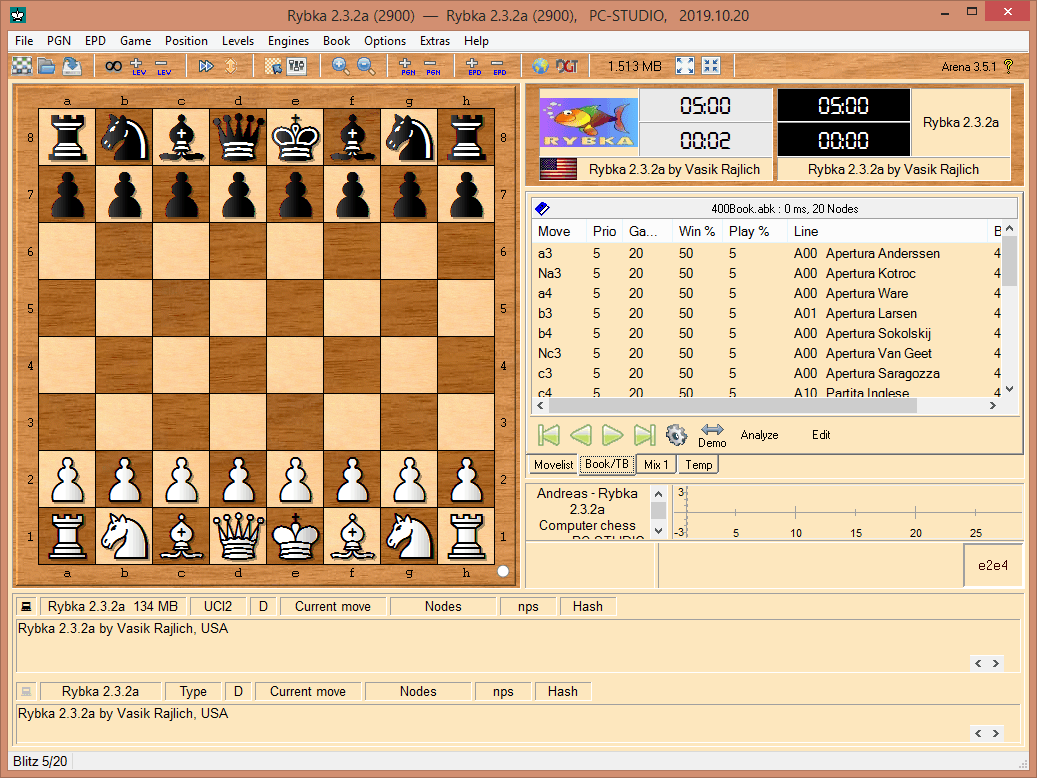


Рисунок 1.1 – Интерфейс «Papergames.io»

Достоинства:

* Возможность создать профиль и соревноваться с другими игроками;
* Возможность самостоятельно создать турнир;
* Возможность выбора условий игры: максимальное количество камней в ряд, время на ход;
* Возможность игры против робота;
* Возможность игры с друзьями;
* Необязательная регистрация.

Недостатки:

* Отсутствие возможности выбора сложности робота (робот всегда играет сильно);
* При игре по ссылке с друзьями сервис не всегда работает корректно.

**«Lichess»** — бесплатный сервис для игры в шахматы.

Интерфейс программы приведен на рисунке 1.2.



Рисунок 1.2 – Интерфейс «Liches»

Достоинства:

* Возможность сохранить и загрузить игру;
* Возможность выбора из двадцати уровней сложности при игре с роботом;

При использовании приложения мной не были найдены недостатки.

**«Chess.com»** — бесплатная мобильная онлайн игра - шахматы. 

Рисунок 1.3 – Интерфейс «Сhess.com»

Достоинства:

* Большое количество сильных игроков онлайн;

Недостатки:

* Реклама появляется после каждой игры;
* Неудобный интерфейс;
* Нет возможности играть с компьютером;
* Правила игры не полностью соответствуют оригинальным правилам;

## Постановка задачи

Для максимально удобного взаимодействия пользователя с проектируемым программным средством должен быть создан информативный и интуитивно понятный пользовательский интерфейс. Также программное средство должно иметь достаточный функционал для обеспечения удобной работы.

В проектируемом программном средстве должны быть реализованы следующие функции:

* Наличие режима игры с другом;
* Наличие режима игры с компьютером;
* Подсветка следующего хода.

Функция игры с компьютером будет реализована при помощи использования стороннего программного обеспечения (UCI compatible chess engine)

Пользовательский интерфейс для программного средства будет спроектирован при помощи Windows API, используя язык C++.

В качестве языка программирования выбран язык С++ и среда разработки Visual Studio 2022.

# Моделирование предметной области

## Правила игры «Шахматы»

Игра происходит на доске, поделённой на равные квадратные клетки, или поля. Размер доски — 8×8 клеток. Вертикальные ряды полей (вертикали) обозначаются латинскими буквами от а до h слева направо, горизонтальные ряды (горизонтали) — цифрами от 1 до 8 снизу вверх; каждое поле обозначается сочетанием соответствующих буквы и цифры. Поля раскрашены в тёмный и светлый цвета (и называются, соответственно, чёрными и белыми) так, что соседние по вертикали и горизонтали поля раскрашены в разные цвета. Доска располагается так, чтобы ближнее угловое поле справа от игрока было белым (для белых это поле h1, для чёрных — поле а8).

Шахматные фигуры, слева направо: король, ладья, ферзь, пешка, конь, слон

У игроков в начале игры имеется по одинаковому набору фигур. Фигуры одного из игроков условно называются «белыми», другого — «чёрными». Белые фигуры окрашены в светлый цвет, чёрные — в тёмный. Сами игроки называются «белые» и «чёрные» по цвету своих фигур.

В каждый комплект фигур входят: король (♔, ♚), ферзь (♕, ♛), две ладьи (♖, ♜), два слона (♗, ♝), два коня (♘, ♞) и восемь пешек (♙, ♟). В начальной позиции фигуры обеих сторон размещаются так, как показано на диаграмме. Белые занимают первую и вторую горизонтали, чёрные — седьмую и восьмую. Пешки расположены на второй и седьмой горизонталях соответственно.

## Описание протокола UCI

Формат UCI (Universal Chess Interface) используется для взаимодействия шахматных движков с их графическими оболочками. Шахматный движок представляет собой обычное приложение (чаще всего консольное). Оно получает различные команды по стандартному устройству ввода (stdin), как-то их обрабатывает и результат подаёт на стандартное устройство вывода (stdout). Таким образом, теоретически движок можно использовать и вручную: вводить команды на клавиатуре и смотреть, какие результаты будут выводиться. Однако это достаточно громоздко, да и требует хорошего знания UCI. Чтобы всё эту работу упростить, используются графические оболочки (GUI), которые общение с движком берут на себя и представляют все необходимые данные в удобном виде. Формат UCI описывает, какие команды движок может получить на вход, и как он должен на них реагировать.

ОБЩИЕ ПОЛОЖЕНИЯ.

1. После запуска движок должен войти в состояние ожидания команд. Он не должен начинать обдумывание без соответствующей команды.

2. Чтобы запуск происходил побыстрее, движок должен дождаться команд "isready" или "setoption" перед тем, как начать установку своих внутренних параметров. Далеко не все движки придерживаются этого правила. Например Rybka 3 при запуске производит инициализацию каких-то своих параметров, в течение которой ввод команд оказывается невозможным.

3. Движок должен быть готовым принимать команды на обработку в любое время, даже во время обдумывания.

4. Все передаваемые движку команды должны заканчиваться переводом строки ("\n"). Символом "\n" должна оканчиваться и каждая строка на выходе движка. В разных системах "\n" может кодироваться разными символами (0x0A, 0x0A0D и т.п.). Было бы неплохо, если бы движки могли воспринимать разные варианты перевода строки, однако, к сожалению, мало кто это умеет.

5. Между отдельными словами в командах можно вставлять любое число пробелов и символов табуляции.

6. Если на входе появляется команда, неизвестная движку, то он должен просто проигнорировать её и пытаться обработать строку так, будто команда эта отсутствует. То же самое касается и известных команд, появившихся в неположенном месте.

## Битборды

  Битборды — представление позиции в виде нескольких беззнаковых целых чисел, каждый бит которого отвечает за состояние некоторого элемента игры, например клетки. Обычно использование битбордов даёт выигрыш по производительности и по объёму используемой памяти, но связано с более изощрённым программированием. При этом часто возникает задача получения значения определённых бит в битборде, например, для последующего обращения к таблице. Есть два основных подхода к решению этой задачи. Первый — использование и поддержка избыточного представления в виде дополнительных битбордов с перенумерацией битов. Такие битборды асто называют вращаемые. Второй способ — умножение на магическую константу, сдвиг и обращение к таблице

Одни из самых быстрых алгоритмов получения списка всех ходов используют битборды. В случае шахмат это представление позиции в виде 64-х битных чисел, где каждый бит отвечает за одну клетку доски. Всего используется несколько битбордов, один для хранения расположения белых пешек, другой для хранения расположения черных коней и т. д. Для генерации всех ходов в шахматах используются два основных метода: «вращаемых битбордов» и «магических битбордов»

.

# Проектирование программного средства

## Проектирование структуры генератора ходов

Основные свойства генератора ходов: Вычисление ходов во всех возможных позициях со всеми возможными наборами блокеров (остальных фигур) производится перед началос рассчета вариантов(инициализация), для представления возможных ходов используюся магические битборды.

Генератор ходов состоит из двух компонент: массивов масок и массивов кэшей фигур, маски это массивы из 64 элементов которые показывают сколько полей может бить фигура заданного типа на данной клетке. Массивы кэшей имеют измерения клетки и набора блокеров, то есть при работе программы для генерации ходов идет простое обращение к памяти которое возвращает битборд всех возможных ходов фигуры из заданной клетки для заданного набора блокеров. Генерация всех ходов осуществляется проходом по всем битбордам позиции для активного в позиции цвета, выделении всех фигур и формирование массива ходов который содержит поля откуда и куда пошла фигура, тип фигуры, тип хода, тип взятой фигуры если это взятие или тип фигуры в который мы превратимся если тип хода превращение (union)

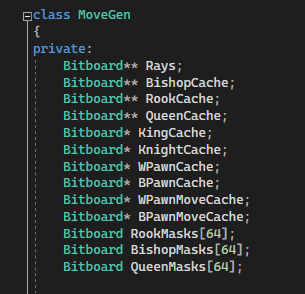


Рисунок 2.1 – Структура генератора ходов

## Проектирование алгоритма вычисления хэша в позиции

хеширование Zobrist (также называемое ключи Zobrist или подписи Zobrist ) - это конструкция хеш-функции , используемая в компьютерные программы , которые играют в абстрактные настольные игры , такие как шахматы и Go , для реализации таблиц транспонирования , особого вида хеш-таблица , которая индексируется позицией на доске и используется, чтобы не анализировать одну и ту же позицию более одного раза. Зобристское хеширование названо в честь его изобретателя Альберта Линдси Зобриста.

Хеширование Zobrist начинается с случайного генерирования битовых строк для каждый возможный элемент настольной игры, то есть для каждой комбинации фигуры и позиции (в шахматной игре это 12 фигур × 64 позиции на доске или 16 x 64, если король все еще может рокироваться, и пешка, которая может брать en passant обрабатываются отдельно для обоих цветов). Теперь любая конфигурация платы может быть разбита на независимые части / компоненты позиции, которые отображаются на случайные последовательности битов, сгенерированные ранее. Окончательный хэш Zobrist вычисляется путем объединения этих битовых строк с помощью побитового XOR .

На рисунке 3.2 представлена блок-схема данного алгоритма.

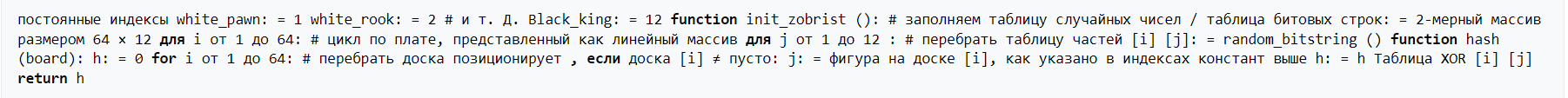


Рисунок 2.2 – Псевдокод Zobrist-hash

## Проектирование алгоритма получения всех ходов в позиции

Для того чтобы получить все ходы в заданной позиции нужно просто пройтись по битбордам заданного цвета и по всем типам фигур, изъять из битбордов численные позиции клетки на которой стоит фигура (нумерация 0-63 с верхнего левого угла) и получить битборды всех возможных ходов фигуры из клетки и набора блокеров (все белые фигуры + все черные фигуры), далее извлекая из них индексы клеток заполнять массив возможных ходов который имеет структуру: поле откуда походила фигура, поле куда походила фигура, тип хода, тип сьеденной фигуры если тип хода взятие и тип превращенной фигуры если тип хода превращение.



# Конструирование программного средства

## Структура модулей программы

Программа будет состоять из нескольких классов, основными из которых будут являться: класс представления позиции и класс генератора ходов.

Класс доски будет представлен полями и методами, необходимыми для верификации ходов и получения состояния доски для отрисовки. Подробное описание методов класса представлено в таблице 4.1.

|  |  |  |
| --- | --- | --- |
| Сигнатура метода | Назначение метода | Входные параметры |
| Position(); | Инициализация полей объекта. |  |
| void MakeMove(Move move) | Сделать ход | move – ход который нужно сделать. |
| void InitPosition(string FEN); | Задать позицию по строке FEN | FEN – строка в нотации FEN для задания позиции |
| string GetPositionFEN(); | Получить представление позиции FEN | - |
| void UnMakeMove(Move move); | Отменить ход | Move – ход который нужно отменить |
| Position(const Position& other); | Конструктор копирования для создания копий обьектов | Other – обьект по которому нужно сделать клона |
| ~Position(); | Деструктор для освобождения ресурсов | - |

Подробное описание полей класса представлено в таблице 4.2.

|  |  |
| --- | --- |
| Поле класса | Назначение |
| Bitboard Pieces[2][6] | Битборды для фигур |
| int ActiveColor; | Цвет текущего игрока. |
| std::vector<Move> CurrMoves; | Массив возможных ходов в позиции |
| Unsignedint MoveCounter; | Количество ходов в партии |

Класс генератора ходов – используется для заполнения массива кэшей фигур, для быстрого получения битбордов атак в runtime. Подробное описание методов класса представлено в таблице 4.3.

|  |  |  |
| --- | --- | --- |
| Сигнатура метода | Назначение метода | Входные параметры |
| MoveGen(); | Инициализация полей класса. | - |
| void InitRays(); | Заполнение массива лучей | - |
| Bitboard get\_king\_attack(int sq); | Получение ходов короля | Sq – индекс клетки |
| Bitboard get\_pawn\_attack(int sq, PieceColors color); | Получение атак пешек | Sq – индекс клетки, color –цвет пешки |
| Bitboard get\_queen\_attack(int sq, Bitboard blockers); | Получение атак ферзя | Sq – индекс клетки, blockers – набор блокеров |
| Bitboard get\_bishop\_attack(int sq, Bitboard blockers);); | Получение атак слона | Sq – индекс клетки, blockers – набор блокеров |
| Bitboard get\_rook\_attack(int sq, Bitboard blockers); | Получение атак ладьи | Sq – индекс клетки, blockers – набор блокеров |
| Bitboard get\_knight\_attack(int sq); | Получение атак коня. | Sq – индекс клетки |
| void InitMasks(); | Инициализация масок фигур | - |
| bool is\_under\_attack(int sq, const Position& position, int color); | Проверка находится ли поле под атакой | Sq – клетка, position – позиция где производится анализ, color – цвет фигуры |
| bool IsKingUnderAttack(const Position& position, int color); | Проверка находится ли король под атакой | Position – позиция,где производистя анализ, color – цвет короля |
| bool IsCastlePossible(const Position& position, int color, int CastleType); | Проверка на возможность рокировки | Position – позиция,где производистя анализ, color – активный цвет, CastleType – тип рокирови |
| inline Bitboard east\_n(Bitboard board, int n); | Вспомогательная функция для инициализации лучей | Board – битборд, n - клетка |
| inline Bitboard west\_n(Bitboard board, int n); | Вспомогательная функция для инициализации лучей | Board – битборд, n - клетка |
| inline int row(int sq); | Вспомогательная функция для инициализации масок | Sq – индекс клетки |
| inline int col(int sq); | Вспомогательная функция для инициализации масок | Sq – индекс клетки |

Подробное описание полей класса представлено в таблице 4.3.

|  |  |
| --- | --- |
| Поле класса | Назначение |
| Bitboard Masks[6][64]; | Для получения масок ходов всех фигур в любоей клетке |
| Bitboard Rays[8][64]; | Для получения лучей во всех направлениях из любой клетки |
| Bitboard[2][6][] Cache; | Кэш ходов всех фигур |
| int state; | Переменная состояния |

# Тестирование, проверка работоспособности и анализ полученных результатов

|  |  |  |
| --- | --- | --- |
| Смысл теста | Expected | Equal |
| Тестирование функции MakeMove(Получение при опредленном ходе верной строки fen)  Исходный fen:  rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1  Исходный ход e2e4 | rnbqkbnr/pppppppp  /8/8/4P3/8  /PPPP1PPP  /RNBQKBNR w KQkq - 0 1 | rnbqkbnr/pppppppp/  8/8/4P3/8/  PPPP1PPP/R  NBQKBNR w KQkq - 0 1 |
| Тестирование функции GetKnightAttack(35)  Должен вывестись битборд с ходами коня в 35 клетке | 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,1,0,1,0,0,0,  0,1,0,0,0,1,0,0,  0,0,0,2,0,0,0,0,  0,1,0,0,0,1,0,0,  0,0,1,0,1,0,0,0,  0,0,0,0,0,0,0,0, | 0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,1,0,1,0,0,0,  0,1,0,0,0,1,0,0,  0,0,0,2,0,0,0,0,  0,1,0,0,0,1,0,0,  0,0,1,0,1,0,0,0,  0,0,0,0,0,0,0,0, |
| Тестирование функции GetQueenAttack(35,0)  Должен вывестись битборд с ходами ферзя в 35 клетке на пустой доске | 0,0,0,1,0,0,0,1,  1,0,0,1,0,0,1,0,  0,1,0,1,0,1,0,0,  0,0,1,1,1,0,0,0,  1,1,1,2,1,1,1,1,  0,0,1,1,1,0,0,0,  0,1,0,1,0,1,0,0,  1,0,0,1,0,0,1,0, | 0,0,0,1,0,0,0,1,  1,0,0,1,0,0,1,0,  0,1,0,1,0,1,0,0,  0,0,1,1,1,0,0,0,  1,1,1,2,1,1,1,1,  0,0,1,1,1,0,0,0,  0,1,0,1,0,1,0,0,  1,0,0,1,0,0,1,0, |
| Тестирование функции GetQueenAttack(35,325235736432)  Должен вывестись битборд с ходами ферзя в 35 клетке на пустой доске с заданным набором блокеров | 0 1 0 0 1 0 0 1  0 0 1 0 1 0 1 0  0 0 0 1 1 1 0 0  0 1 1 1 0 1 1 0  0 0 0 1 1 1 0 0  0 0 0 0 0 0 1 0  0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 | 0 1 0 0 1 0 0 1  0 0 1 0 1 0 1 0  0 0 0 1 1 1 0 0  0 1 1 1 0 1 1 0  0 0 0 1 1 1 0 0  0 0 0 0 0 0 1 0  0 0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 |
| Проверка функции GetPositionFen(Position)  Задаётся начальная позиция с ходами е2е4 с7с5 | rnbqkbnr/pp1ppppp  /8/2p5/4P3/8  /PPPP1PPP/  RNBQKBNR w KQkq - 0 3 | rnbqkbnr/pp1ppppp  /8/2p5/4P3/8  /PPPP1PPP/  RNBQKBNR w KQkq - 0 3 |
| Тестирование функции InitPosition(fen)  Fen = r1bqk2r/1p2bppp  /p1n1pn2/2pp4/4P3  /1BN2N2/PPPP1PPP  /R1BQ1RK1 w kq - 3 15 |  |  |
| Тестирование функции IsKingUnderAttack()  Исходная позиция: | true | true |
| Тестирование функции IsCastlePossible(black)  Исходная позиция: | false | false |

# Руководство по установке и пользованию:

Для запуска данной программы потребуется склонировать репозиторий GitHub и запустить .exe file UCI\_GUI.exe

Для гибкого использования был предусмотрен пакет движков которые содержатся в папке Engines исходного проекта, достаточно поменять путь к исполняемому файлу движка в файле IPC.h, пересобрать решение и запустить .exe файл. Это предусмотрено для того чтобы оценивать ходы разных движков и сопутствующую информацию в одинаковых позициях.

Также в программе предусмотрена возможность менять стили фигур и доски и общего оформления, для этого достаточно поменять константу Style в модуле Resources.h, пересобрать решение и запустить файл.

Также в программе существует возможность выбирать какой тип игры будет использоваться. Всего существует 3 варианта игры: человек с человеком, компьютер с человеком, также можно понаблюдать за шахматным совершенством, включив режим игры движок против движка, для этого нужно поменять константу gameStyle в файле Game.h

Заключение

В процессе выполнения курсового проекта была проанализирована предметная область, рассмотрены существующие аналоги, выявлены их преимущества и недостатки.

На этапе проектирования были изучены основные принципы работы с динамическими структурами данных, принципы работы с графикой на языках программирования C++, разработаны блок-схемы алгоритмов и графический интерфейс пользователя.

В соответствии с поставленной целью было разработано приложение, которое представляет собой игру «Гомоку».

Проведено тестирование работоспособности разработанной программной части. Поставленная цель была выполнена в полном объеме, работоспособность подтверждена тестированием программного средства. Разработано руководство пользователя.

# Список использованных источников

[1] Корнилов, Е. Н. Программирование шахмат и других логических игр. – СПб.: БХВ-Петербург, 2005. – 272 с.

[2] От крестиков-ноликов до шахмат [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/383907/>. – Дата доступа: 10.10.2022.

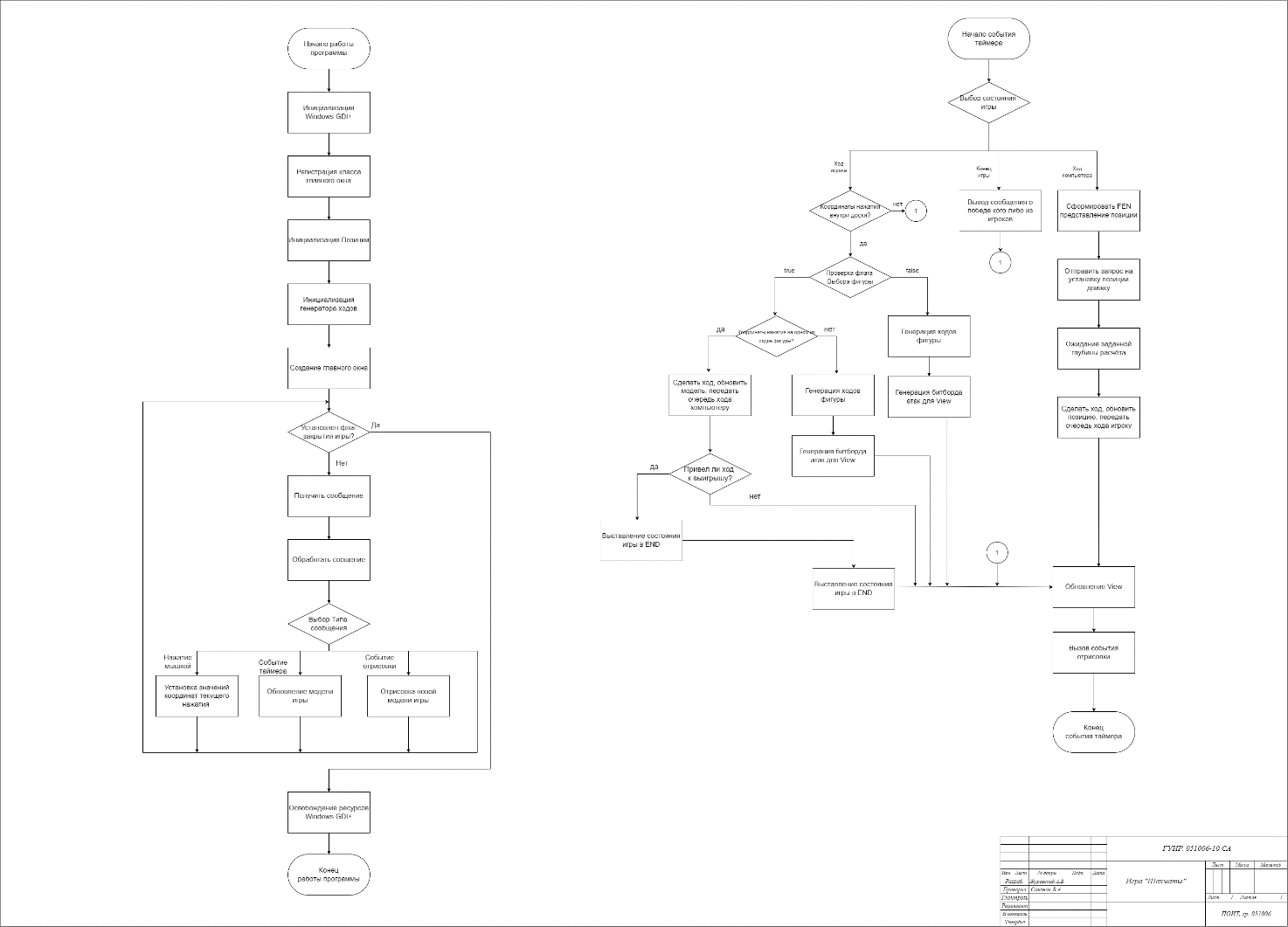
[3] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/. – Дата доступа: 20.10.2022.

[4] How to make multiple windows using Win32 API [Электронный ресурс]. – Режим доступа: https://stackoverflow.com/questions/2886609/how-to-make-multiple-windows-using-win32-api. – Дата доступа: 01.11.2022.

[5] Поиск решений в логических играх на примере шахмат [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/202264/. – Дата доступа: 20.10.2022.

[6] Минимаксный алгоритм [Электронный ресурс]. – Режим доступа: https://ru.frwiki.wiki/wiki/Algorithme\_minimax/. – Дата доступа: 20.10.2022.

Приложение A  
(обязательное)  
**Блок-схема алгоритма**



Приложение Б  
(обязательное)  
Исходный код программы

#include "Board.h"

Board::Board()

{

this->board = new Cell[size];

this->Attacks = Bitboard(0);

}

void Board::InitBoard(Position& pos)

{

for (int color = PieceColors::WHITE; color <= PieceColors::BLACK; color++)

{

for (int piece = PieceTypes::KING; piece <= PieceTypes::PAWN; piece++)

{

Bitboard temp = pos.Pieces[color][piece];

while (temp)

{

DWORD\* index = new DWORD;

BitScanForward64(index, temp);

board[63 - \*index].SetCell(piece, color);

temp &= (temp - 1);

}

}

}

}

void Board::UpdateBoard(Position& pos)

{

for (int i = 0; i < 64; i++)

board[i].PieceColor = board[i].PieceType = nullptr;

for (int color = PieceColors::WHITE; color <= PieceColors::BLACK; color++)

{

for (int piece = PieceTypes::KING; piece <= PieceTypes::PAWN; piece++)

{

Bitboard temp = pos.Pieces[color][piece];

while (temp)

{

DWORD\* index = new DWORD;

BitScanForward64(index, temp);

board[63 - \*index].SetCell(piece, color);

temp &= (temp - 1);

}

}

}

}

Cell\* Board::GetCell(int index)

{

return &board[index];

}

Cell\* Board::GetBoard()

{

return this->board;

}

string Board::GetPositionFEN()

{

std::string dictionary = "KQRBNPkqrbnp";

string FEN;

for (int i = 0; i < 8; i++)

{

int counter = 0;

for (int j = 0; j < 8; j++)

{

if (board[8 \* i + j].PieceColor != nullptr)

{

if (counter > 0)

{

FEN += char(counter) + '0';

counter = 0;

}

int index = (\*board[8 \* i + j].PieceColor) \* 6 + \*board[8 \* i + j].PieceType;

FEN += dictionary[index];

}

else ++counter;

if (counter == 8) FEN += '8';

else if (j == 7 && counter != 0) FEN += char(counter) + '0';

}

FEN += '/';

}

FEN.erase(FEN.end() - 1);

return FEN;

}

void Board::SetAttacks(Bitboard attacks)

{

this->Attacks = attacks;

}

Bitboard Board::GetAttacks()

{

return this->Attacks;

}

Board::~Board()

{

delete[] this->board;

}

#include "Cell.h"

Cell::Cell()

{

PieceColor = nullptr;

PieceType = nullptr;

}

void Cell::SetCell(int pieceType, int pieceColor)

{

this->PieceType = new int;

\*PieceType = pieceType;

this->PieceColor = new int;

\*PieceColor = pieceColor;

}

Cell::~Cell()

{

if (PieceColor) delete PieceColor;

if (PieceType) delete PieceType;

}

#include "framework.h"

int MyBitScanReverse(Bitboard bb)

{

if(bb == 0) return -1;

const int index64[64] = {

0, 47, 1, 56, 48, 27, 2, 60,

57, 49, 41, 37, 28, 16, 3, 61,

54, 58, 35, 52, 50, 42, 21, 44,

38, 32, 29, 23, 17, 11, 4, 62,

46, 55, 26, 59, 40, 36, 15, 53,

34, 51, 20, 43, 31, 22, 10, 45,

25, 39, 14, 33, 19, 30, 9, 24,

13, 18, 8, 12, 7, 6, 5, 63

};

const Bitboard mask = 0x03f79d71b4cb0a89;

return index64[((bb ^ (bb - 1)) \* mask) >> 58];

}

int MyBitScanForward(Bitboard bb)

{

if (bb == 0) return -1;

const int index64[64] = {

0, 47, 1, 56, 48, 27, 2, 60,

57, 49, 41, 37, 28, 16, 3, 61,

54, 58, 35, 52, 50, 42, 21, 44,

38, 32, 29, 23, 17, 11, 4, 62,

46, 55, 26, 59, 40, 36, 15, 53,

34, 51, 20, 43, 31, 22, 10, 45,

25, 39, 14, 33, 19, 30, 9, 24,

13, 18, 8, 12, 7, 6, 5, 63

};

const Bitboard mask = 0x03f79d71b4cb0a89;

bb |= bb >> 1;

bb |= bb >> 2;

bb |= bb >> 4;

bb |= bb >> 8;

bb |= bb >> 16;

bb |= bb >> 32;

return index64[(bb \* mask) >> 58];

}

#include "Game.h"

Game::Game(HWND window)

{

this->window = window;

res = new Resources();

pos = new Position();

board = new Board();

moveGen = new MoveGen();

ipc = new IPC();

kal.open("C:\\Users\\user\\Desktop\\debug.txt");

kal << "NEW\_GAME\n";

kal.close();

}

int Game::GetCellIndex(int mouse\_x, int mouse\_y)

{

int offtop = 36, x, y;

if (mouse\_x >= offtop && mouse\_x <= res->BoardSize - offtop &&

mouse\_y >= offtop && mouse\_y <= res->BoardSize - offtop)

{

y = int((double(mouse\_y) - offtop) / (res->BoardSize - 2 \* offtop) \* 8);

x = int((double(mouse\_x) - offtop) / (res->BoardSize - 2 \* offtop) \* 8);

return y \* 8 + x;

}

else return -1;

}

void Game::Init()

{

pos->InitPosition("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1");

board->InitBoard(\*pos);

string response = ipc->ReceiveResponse();

state = GameStates::COMPUTER\_MOVE;

isFigureChoosen = false;

}

void Game::Draw(Gdiplus::Graphics& gr)

{

gr.DrawImage(res->BoardImage,0,0,res->BoardSize,res->BoardSize);

Cell\* temp = this->board->GetBoard();

Bitboard attacks = board->GetAttacks();

Bitboard mask = Bitboard(1)<<63;

int x = 35; int y = 35;

for (int i = 0; i < 64; i++)

{

if (temp[i].PieceColor)

{

gr.DrawImage(res->PiecesImages[\*temp[i].PieceColor][\*temp[i].PieceType], x, y, res->PieceSize, res->PieceSize);

}

if (attacks & mask)

{

gr.DrawImage(res->MoveImage, x, y, res->PieceSize, res->PieceSize);

}

x += 86;

if (x == 35+8\*86)

{

x = 35;

y += 85;

}

mask >>= 1;

}

}

string Game::GetPositionFen()

{

return board->GetPositionFEN() + pos->GetPositionFEN();

}

void Game::Update(int mouse\_x, int mouse\_y)

{

int index = GetCellIndex(mouse\_x, mouse\_y);

switch (state)

{

case GameStates::PLAYER\_MOVE:

{

// Если никуда не нажали выходим из update

if (index == -1) return;

Cell\* currCell = board->GetCell(index);

// Если фигура уже выбрана

if (isFigureChoosen)

{

// Если после выбора фигуры нажали на один из ее ходов

if ((Bitboard(1) << (63 - index)) & board->GetAttacks())

{

for (auto& move : pos->CurrMoves)

{

if (move.To == 63-index)

{

pos->MakeMove(move);

board->UpdateBoard(\*pos);

board->SetAttacks(Bitboard(0));

pos->CurrMoves.clear();

isFigureChoosen = false;

if (isWin())

{

state = GameStates::END;

return;

}

else

{

state = GameStates::COMPUTER\_MOVE;

return;

}

return;

}

}

}

// Если выбрали другую фигуру

else

{

if (currCell->PieceColor != nullptr && \*currCell->PieceColor == pos->ActiveColor)

{

isFigureChoosen = true;

std::vector<Move> AllMoves = moveGen->GetAllMoves(\*pos, pos->ActiveColor);

std::vector<Move> CurrMoves;

Bitboard bbCurrMoves = 0;

for (auto& move : AllMoves)

{

if (move.From == 63 - index)

{

CurrMoves.push\_back(move);

bbCurrMoves |= Bitboard(1) << move.To;

}

}

board->SetAttacks(bbCurrMoves);

pos->CurrMoves = CurrMoves;

}

else return;

}

}

else

{

// Если фигурка есть на клетке и цвета игрока то выбираем ее и генерирум ее ходы

if (currCell->PieceColor != nullptr && \*currCell->PieceColor == pos->ActiveColor)

{

isFigureChoosen = true;

std::vector<Move> AllMoves = moveGen->GetAllMoves(\*pos, pos->ActiveColor);

std::vector<Move> CurrMoves;

Bitboard bbCurrMoves = 0;

for (auto& move : AllMoves)

{

if (move.From == 63 - index)

{

CurrMoves.push\_back(move);

bbCurrMoves |= Bitboard(1) << move.To;

}

}

board->SetAttacks(bbCurrMoves);

pos->CurrMoves = CurrMoves;

}

}

break;

}

case GameStates::COMPUTER\_MOVE:

{

string FEN = GetPositionFen();

string temp = FEN + '\n';

kal.open("C:\\Users\\user\\Desktop\\debug.txt",std::ios::app);

kal << temp;

kal.close();

ipc->SendRequest("position fen " + FEN + '\n');

Sleep(100);

ipc->SendRequest("go\n");

Sleep(500);

string searchinfo = ipc->ReceiveResponse();

Sleep(100);

ipc->SendRequest("stop\n");

Sleep(100);

string bestmove = ipc->ReceiveResponse();

int index = bestmove.find("bestmove");

bestmove = bestmove.substr(index + 9, 4);

std::vector<Move> moves = moveGen->GetAllMoves(\*pos, pos->ActiveColor);

std::vector<string> MoveStrings;

for (int i = 0; i < moves.size(); i++)

MoveStrings.push\_back(moves[i].ToString());

for (auto& move : moves)

{

if (move.ToString().\_Equal(bestmove))

{

pos->MakeMove(move);

board->UpdateBoard(\*pos);

board->SetAttacks(Bitboard(0));

pos->CurrMoves.clear();

if (isWin())

{

state = GameStates::END;

return;

}

else

{

//state = GameStates::PLAYER\_MOVE;

return;

}

return;

}

}

break;

}

case GameStates::END:

{

if (pos->ActiveColor == WHITE)

MessageBox(window, L"Black won", L"Game over", MB\_OK);

else

MessageBox(window, L"White won ", L"Game over", MB\_OK);

exit(0);

break;

}

}

}

bool Game::isWin()

{

if (moveGen->GetAllMoves(\*pos, pos->ActiveColor).size() == 0

&& moveGen->IsKingUnderAttack(\*pos, pos->ActiveColor))

return true;

return false;

}

Game::~Game()

{

delete board;

delete ipc;

delete moveGen;

delete pos;

}

#include "IPC.h"

IPC::IPC()

{

SECURITY\_ATTRIBUTES sa;//атрибуты защиты канала

sa.lpSecurityDescriptor = NULL; //защита по умолчанию

sa.nLength = sizeof(SECURITY\_ATTRIBUTES);

sa.bInheritHandle = true;//разрешаем наследование дескрипторов

//создаем анонимные каналы

CreatePipe(&parent\_out, &child\_in, &sa, 0); //канал для stdin дочернего процесса

CreatePipe(&child\_out, &parent\_in, &sa, 0); //канал для stdout дочернего процесса

STARTUPINFO si;

ZeroMemory(&si, sizeof(STARTUPINFO));

si.dwFlags = STARTF\_USESTDHANDLES | STARTF\_USESHOWWINDOW;

si.wShowWindow = SW\_HIDE;//скрываем окно дочернего процесса

//подменяем дескрипторы

si.hStdOutput = parent\_in;

si.hStdError = parent\_in;

si.hStdInput = parent\_out;

//Создаём дочерний процесс

WCHAR ModuleName[] = L"C:\\Users\\user\\desktop\\Курсач\\stockfish\\stockfish.exe";

BOOL bsucces = CreateProcess(ModuleName, NULL, NULL, NULL, TRUE, ABOVE\_NORMAL\_PRIORITY\_CLASS, NULL, NULL, &si, &pi);

}

void IPC::SendRequest(string message)

{

// количество записанных байт

DWORD num;

// записываем в буфер сообщение

wsprintfA(send, message.c\_str());

// пишем в пайп

WriteFile(child\_in, send, lstrlenA(send), &num, NULL);

}

string IPC::ReceiveResponse()

{

// Строка ответа

string response;

// количество прочитанных байт

DWORD num;

BOOL bsuccses;

do {

bsuccses = ReadFile(child\_out, read, 1023, &num, NULL); //читаем ответ в буфер //и взводим событие готовности

for (int i = 0; i < 1023; i++)

response.push\_back(read[i]);

ZeroMemory(read, 1024);

if (num < 1023) break;

} while (num > 0);

return response;

}

IPC::~IPC()

{

CloseHandle(child\_in);//закрываем канал stdin дочернего процесса

CloseHandle(parent\_out);//закрываем родительский принимающий пайп

CloseHandle(parent\_in);

CloseHandle(child\_out);

CloseHandle(hePipeReply);

CloseHandle(hePipeStart);

CloseHandle(pi.hThread);

CloseHandle(pi.hProcess);

}

#include "Move.h"

Move::Move()

{

}

Move::Move(int from, int to, int piecetype, int movetype, int opPieceType)

{

From = from;

To = to;

PieceType = piecetype;

MoveType = movetype;

OpPieceType = opPieceType;

}

string Move::ToString()

{

string Files = "abcdefgh";

string move = "";

move += Files[7 - (From % 8)];

move += '1' + (char)(From / 8);

move += Files[7 - (To % 8)];

move += '1' + (char)(To / 8);

return move;

}

#include "Position.h"

Position::Position()

{

Pieces = new Bitboard \* [2]{0};

for (int i = 0; i < 2; i++)

Pieces[i] = new Bitboard[6]{0};

}

Position::Position(const Position& other)

{

ActiveColor = other.ActiveColor;

AllBlackPeaces = other.AllBlackPeaces;

AllWhitePeaces = other.AllWhitePeaces;

BlackLongCastle = other.BlackLongCastle;

BlackShortCastle = other.BlackShortCastle;

FiftyMovesRuleCounter = other.FiftyMovesRuleCounter;

MagicCaptureField = other.MagicCaptureField;

MoveCounter = other.MoveCounter;

Pieces = new Bitboard \* [2];

for (int i = 0; i < 2; i++)

{

Pieces[i] = new Bitboard[6];

for (int j = 0; j < 6; j++)

Pieces[i][j] = other.Pieces[i][j];

}

WhiteLongCastle = other.WhiteLongCastle;

WhiteShortCastle = other.WhiteShortCastle;

CurrMoves = other.CurrMoves;

}

void Position::MakeMove(Move move)

{

++MoveCounter; // счетчик ходов увелчичваетсся

// проверка на правило 50 ходов

if ((move.PieceType) != PAWN || move.MoveType != CAPTURE

|| move.MoveType != CAPTURE\_TRANSFORM || move.MoveType != MAGIC\_CAPTURE) ++FiftyMovesRuleCounter;

else FiftyMovesRuleCounter = 0;

this->MagicCaptureField = Bitboard(0); // убираем поле взятия на проходе

switch (move.MoveType)

{

case NORMAL\_MOVE:

Pieces[ActiveColor][move.PieceType] &= ~(Bitboard(1) << move.From); // убираем фигуру с места где она стояла

Pieces[ActiveColor][move.PieceType] |= (Bitboard(1) << move.To); // ставим фигуру туда куда она пошла

if (ActiveColor == WHITE)

{

AllWhitePeaces &= ~(Bitboard(1) << move.From);

AllWhitePeaces |= (Bitboard(1) << move.To);

if (move.PieceType == KING)

WhiteLongCastle = WhiteShortCastle = false;

}

else

{

AllBlackPeaces &= ~(Bitboard(1) << move.From);

AllBlackPeaces |= (Bitboard(1) << move.To);

if (move.PieceType == KING)

BlackLongCastle = BlackShortCastle = false;

}

break;

case CAPTURE:

Pieces[ActiveColor][move.PieceType] &= ~(Bitboard(1) << move.From); // убираем фигуру с места где она стояла

Pieces[ActiveColor][move.PieceType] |= (Bitboard(1) << move.To); // ставим фигуру туда куда она пошла

if (ActiveColor == WHITE)

{

AllWhitePeaces &= ~(Bitboard(1) << move.From);

AllWhitePeaces |= (Bitboard(1) << move.To);

AllBlackPeaces &= ~(Bitboard(1) << move.To);

Pieces[BLACK][move.OpPieceType] &= ~(Bitboard(1) << move.To);

if (move.PieceType == KING)

WhiteLongCastle = WhiteShortCastle = false;

}

else

{

AllBlackPeaces &= ~(Bitboard(1) << move.From);

AllBlackPeaces |= (Bitboard(1) << move.To);

AllWhitePeaces &= ~(Bitboard(1) << move.To);

Pieces[WHITE][move.OpPieceType] &= ~(Bitboard(1) << move.To);

if (move.PieceType == KING)

BlackLongCastle = BlackShortCastle = false;

}

break;

case FIRST\_PAWN\_MOVE:

Pieces[ActiveColor][PAWN] &= ~(Bitboard(1) << move.From);

Pieces[ActiveColor][PAWN] |= (Bitboard(1) << move.To);

if (ActiveColor == WHITE)

{

MagicCaptureField = (Bitboard(1) << (move.To - 8));

AllWhitePeaces &= ~(Bitboard(1) << move.From);

AllWhitePeaces |= (Bitboard(1) << move.To);

}

else

{

MagicCaptureField = (Bitboard(1) << (move.To + 8));

AllBlackPeaces &= ~(Bitboard(1) << move.From);

AllBlackPeaces |= (Bitboard(1) << move.To);

}

break;

case MAGIC\_CAPTURE:

Pieces[ActiveColor][PAWN] &= ~(Bitboard(1) << move.From);

Pieces[ActiveColor][PAWN] |= (Bitboard(1) << move.To);

if (ActiveColor == WHITE)

{

Pieces[BLACK][PAWN] &= ~(Bitboard(1) << (move.To - 8));

AllBlackPeaces &= ~(Bitboard(1) << (move.To - 8));

AllWhitePeaces &= ~(Bitboard(1) << move.From);

AllWhitePeaces |= (Bitboard(1) << move.To);

}

else

{

Pieces[WHITE][PAWN] &= ~(Bitboard(1) << (move.To + 8));

AllWhitePeaces &= ~(Bitboard(1) << (move.To + 8));

AllBlackPeaces &= ~(Bitboard(1) << move.From);

AllBlackPeaces |= (Bitboard(1) << move.To);

}

break;

case LONG\_CASTLE:

if (ActiveColor == WHITE)

{

WhiteLongCastle = false;

WhiteShortCastle = false;

Pieces[WHITE][KING] = 0X20;

Pieces[WHITE][ROOK] &= ~(0X80);

Pieces[WHITE][ROOK] |= 0X10;

AllWhitePeaces &= ~(0X88);

AllWhitePeaces |= 0X30;

}

else

{

BlackLongCastle = false;

BlackShortCastle = false;

Pieces[BLACK][KING] = Bitboard(1) << 61;

Pieces[BLACK][ROOK] &= ~(Bitboard(1) << 63);

Pieces[BLACK][ROOK] |= (Bitboard(1) << 60);

AllBlackPeaces &= ~0X8800000000000000;

AllBlackPeaces |= 0X3000000000000000;

}

break;

case SHORT\_CASTLE:

if (ActiveColor == WHITE)

{

WhiteLongCastle = false;

WhiteShortCastle = false;

Pieces[WHITE][KING] = 0X2;

Pieces[WHITE][ROOK] &= ~0X1;

Pieces[WHITE][ROOK] |= 0X4;

AllWhitePeaces &= ~(0X9);

AllWhitePeaces |= 0X6;

}

else

{

BlackLongCastle = false;

BlackShortCastle = false;

Pieces[BLACK][KING] = 0X0200000000000000;

Pieces[BLACK][ROOK] &= ~(0X0100000000000000);

Pieces[BLACK][ROOK] |= 0X0400000000000000;

AllBlackPeaces &= ~0X0900000000000000;

AllBlackPeaces |= 0X0600000000000000;

}

break;

case TRANSFORM:

if (ActiveColor == WHITE)

{

Pieces[WHITE][PAWN] &= ~(Bitboard(1) << move.From);

Pieces[WHITE][move.OpPieceType] |= (Bitboard(1) << move.To);

AllWhitePeaces &= ~(Bitboard(1) << move.From);

AllWhitePeaces |= (Bitboard(1) << move.To);

}

else

{

Pieces[BLACK][PAWN] &= ~(Bitboard(1) << move.From);

Pieces[BLACK][move.OpPieceType] |= (Bitboard(1) << move.To);

AllBlackPeaces &= ~(Bitboard(1) << move.From);

AllBlackPeaces |= (Bitboard(1) << move.To);

}

break;

case CAPTURE\_TRANSFORM:

if (ActiveColor == WHITE)

{

Pieces[BLACK][move.OpPieceType] &= ~(Bitboard(1) << move.To);

AllBlackPeaces &= ~(Bitboard(1) << move.To);

Pieces[WHITE][PAWN] &= ~(Bitboard(1) << move.From);

Pieces[WHITE][QUEEN] |= (Bitboard(1) << move.To);

AllWhitePeaces &= ~(Bitboard(1) << move.From);

AllWhitePeaces |= (Bitboard(1) << move.To);

}

else

{

Pieces[WHITE][move.OpPieceType] &= ~(Bitboard(1) << move.To);

AllWhitePeaces &= ~(Bitboard(1) << move.To);

Pieces[BLACK][PAWN] &= ~(Bitboard(1) << move.From);

Pieces[BLACK][QUEEN] |= (Bitboard(1) << move.To);

AllBlackPeaces &= ~(Bitboard(1) << move.From);

AllBlackPeaces |= (Bitboard(1) << move.To);

}

break;

}

// смена активного цвета

if (ActiveColor == WHITE) ActiveColor = BLACK;

else ActiveColor = WHITE;

}

void Position::UnMakeMove(Move move)

{

}

void Position::InitPosition()

{

InitPosition("rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1");

}

// rnbqkbnr/pppp1ppp/8/4p3/4P3/8/PPPP1PPP/RNBQKBNR w KQkq e6 0 2

void Position::InitPosition(string FEN)

{

std::string dictionary = "KQRBNPkqrbnp";

char arr[64];

int i = 0, index = 0;

while (FEN[i] != ' ')

{

if (FEN[i] == '/') i++;

else

{

if (((FEN[i] - '0') <= 9) && ((FEN[i] - '0') >= 0))

{

for (int j = 0; j < FEN[i] - '0'; j++) arr[index++] = '0';

}

else

{

arr[index++] = FEN[i];

}

i++;

}

}

for (int j = 0; j < dictionary.length(); j++)

{

Bitboard temp = 0;

for (int k = 0; k < 64; k++)

{

if (arr[k] == dictionary[j])

{

temp |= (Bitboard(1) << (63 - k));

}

}

Pieces[j / 6][j % 6] = temp;

}

ActiveColor = (FEN[++i] == 'w') ? PieceColors::WHITE : PieceColors::BLACK;

i += 2;

BlackShortCastle = BlackLongCastle = WhiteLongCastle = WhiteShortCastle = false;

while (FEN[i] != ' ')

{

if (FEN[i] == 'K') WhiteShortCastle = true;

else if (FEN[i] == 'k') BlackShortCastle = true;

else if (FEN[i] == 'q') BlackLongCastle = true;

else if (FEN[i] == 'Q') WhiteLongCastle = true;

i++;

}

i++;

int sq = 0;

if (FEN[i] != '-')

{

sq = 'h' - FEN[i] + ((FEN[i + 1] - '0' - 1) << 3);

MagicCaptureField = Bitboard(1) << sq;

i++;

}

else MagicCaptureField = Bitboard(0);

i += 2;

this->FiftyMovesRuleCounter= FEN[i] - '0';

i += 2;

MoveCounter = FEN[i] - '0';

this->AllBlackPeaces= AllWhitePeaces = Bitboard(0);

for (int i = 0; i < 6; i++)

{

AllWhitePeaces |= Pieces[WHITE][i];

AllBlackPeaces |= Pieces[BLACK][i];

}

}

string Position::GetPositionFEN()

{

// w KQkq - 0 1

string FEN = " ";

FEN += (ActiveColor == PieceColors::WHITE) ? "w " : "b ";

if (WhiteShortCastle) FEN += 'K';

if (WhiteLongCastle) FEN += 'Q';

if (BlackShortCastle) FEN += 'k';

if (BlackLongCastle) FEN += 'q';

if (!WhiteShortCastle && !WhiteLongCastle && !BlackShortCastle && !BlackLongCastle) FEN += '-';

FEN += ' ';

if (MagicCaptureField)

{

DWORD index;

BitScanReverse64(&index, MagicCaptureField);

int rank = index / 8;

int file = index % 8;

FEN += 'h' - char(file);

FEN += char(rank) + '1';

FEN += ' ';

}

else FEN += "- ";

std::stringstream a;

a << FiftyMovesRuleCounter;

string k; a >> k;

FEN += '0' + ' ';

std::stringstream b;

b << MoveCounter;

string l; b >> l;

FEN += l;

return FEN;

}

Position::~Position()

{

for (int i = 0; i < 2; i++)

delete[] Pieces[i];

delete[] Pieces;

}

string Position::ViewBitboard(Bitboard a)

{

string res;

for (int i = 0; i < 64; i++)

{

if (a & 0x8000000000000000) res += "1 ";

else res += "0 ";

a <<= 1;

if (((i + 1) & 7) == 0) res += '\n';

}

return res;

}

// UCI\_GUI.cpp : Определяет точку входа для приложения.

//

#include "framework.h"

#include "UCI\_GUI.h"

#include <fcntl.h>

#include <io.h>

#include "Game.h"

// TODO:

// 1) Валидность генерируемых ходов (Для этого написать MakeMove and UnMakeMove)

// 2) Конец игры

// 3) IPC

#define MAX\_LOADSTRING 100

// Глобальные переменные:

HINSTANCE hInst; // текущий экземпляр

WCHAR szTitle[MAX\_LOADSTRING]; // Текст строки заголовка

WCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окна

Game\* game = nullptr;

int mouse\_x = 5, mouse\_y = 5;

int k = 0;

// Отправить объявления функций, включенных в этот модуль кода:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

Gdiplus::GdiplusStartupInput gdiplusStartupInput;

ULONG\_PTR gdiplusToken;

// Initialize GDI+.

GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

// Инициализация глобальных строк

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_UCIGUI, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_UCIGUI));

MSG msg;

// Цикл основного сообщения:

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

Gdiplus::GdiplusShutdown(gdiplusToken);

return (int) msg.wParam;

}

//

// ФУНКЦИЯ: MyRegisterClass()

//

// ЦЕЛЬ: Регистрирует класс окна.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_UCIGUI));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_UCIGUI);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

//

// ФУНКЦИЯ: InitInstance(HINSTANCE, int)

//

// ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное окно

//

// КОММЕНТАРИИ:

//

// В этой функции маркер экземпляра сохраняется в глобальной переменной, а также

// создается и выводится главное окно программы.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

350, 0, 800, 800, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// ЦЕЛЬ: Обрабатывает сообщения в главном окне.

//

// WM\_COMMAND - обработать меню приложения

// WM\_PAINT - Отрисовка главного окна

// WM\_DESTROY - отправить сообщение о выходе и вернуться

//

//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_CREATE:

{

// Инициализируем игру и выставляем начальную позицию

game = new Game(hWnd);

game->Init();

string fen = game->GetPositionFen();

SetTimer(hWnd, 1, 25, NULL);

break;

}

case WM\_TIMER:

{

game->Update(mouse\_x, mouse\_y);

mouse\_x = mouse\_y = 0;

PAINTSTRUCT ps;

HDC hdcWin = GetDC(hWnd), hdc;

HBITMAP hBitmap;

hdc = CreateCompatibleDC(hdcWin);

hBitmap = CreateCompatibleBitmap(hdcWin, 800, 800);

SelectObject(hdc, hBitmap);

Rectangle(hdc, -1, -1, 801, 801); // очистка

Gdiplus::Graphics gr(hdc);

game->Draw(gr);

BitBlt(hdcWin, 0, 0, 800, 800, hdc, 0, 0, SRCCOPY);

DeleteDC(hdc);

DeleteObject(hBitmap);

ReleaseDC(hWnd, hdcWin);

break;

}

case WM\_LBUTTONDOWN:

{

// Получаем координаты нажатия

mouse\_x = LOWORD(lParam);

mouse\_y = HIWORD(lParam);

break;

}

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

break;

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdcWin = BeginPaint(hWnd, &ps),hdc;

EndPaint(hWnd, &ps);

break;

}

case WM\_DESTROY:

{

game->~Game();

PostQuitMessage(0);

break;

}

default:

{

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

return 0;

}

// Обработчик сообщений для окна "О программе".

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}