Описание алгоритма

1. Общая структура

1. **Размерность:**

Игра ведётся на доске размером 8×8 (константа BOARD_SIZE = 8).

Каждая клетка может содержать:

- '.' — пустое поле,
- 'w'/'b' — простая белая/чёрная шашка,
- 'W'/'B' — дамка белая/чёрная.

2. **Структуры данных:**

MoveStep: описывает один «микро-ход» (начало/конец, например startRow, startCol → endRow, endCol).

MoveSequence: полный ход (возможно, многократный бой), содержит:

- Вектор шагов (steps),
- Поле capturesCount (сколько фигур побито).

3. **Инициализация доски:**

Функция initBoard заполняет верхние 3 строки чёрными шашками (на тёмных клетках) и нижние 3 строки белыми шашками, создавая классическую начальную позицию русских шашек.

4. **Вывод:**

Функция printBoard показывает всю доску в консоли, обозначая столбцы буквами A..H и строки 1..8.

2. Основные функции для ходов

1. **Обычные ходы (без боя):**

Простая шашка (белая) идёт «вверх» на 1 клетку по диагонали, (чёрная) — вниз (то есть на +1 по индексу строки).

Дамка ('W' или 'B') может ходить на любое количество свободных клеток по диагонали.

2. **Бой (взятие):**

Простой шашке нужно перепрыгнуть через вражескую (если она рядом по диагонали), приземлиться на свободную клетку. Это может происходить и вперёд, и назад.

Дамка может дальним боем перескочить через вражескую на любое количество клеток (при условии, что промежуточные клетки пусты).

Возможен многократный бой: после первого взятия проверяем, можно ли продолжать.

Обязательность взятия: если есть хоть одно взятие, нужно выполнять именно его.

3. **Превращение в дамку:**

Если белая шашка дошла до 0-й строки, становится 'W'. Если чёрная достигла 7-й строки, становится 'B'.

4. **Рекурсивный поиск боёв:**

Функция (searchCaptures) ищет все цепочки (многократные прыжки), учитывая правило, что нельзя бить одну и ту же фигуру более одного раза за ход.

3. Параллельный поиск ходов (через «чанки»)

1. **Идея разбиения (chunking):**

Мы делим строки доски 0..7 на несколько кусков, размер которых вычисляется из std::thread::hardware_concurrency().

Например, если есть 4 аппаратных потока, будут чанки по 2 строки (или схожим образом).

2. **Асинхронные задачи:**

Для каждого чанка запускается std::async (с флагом std::launch::async), который перебирает фигуры нужного цвета на своих строках, генерирует ходы (либо боевые, либо обычные) и складывает в локальный список.

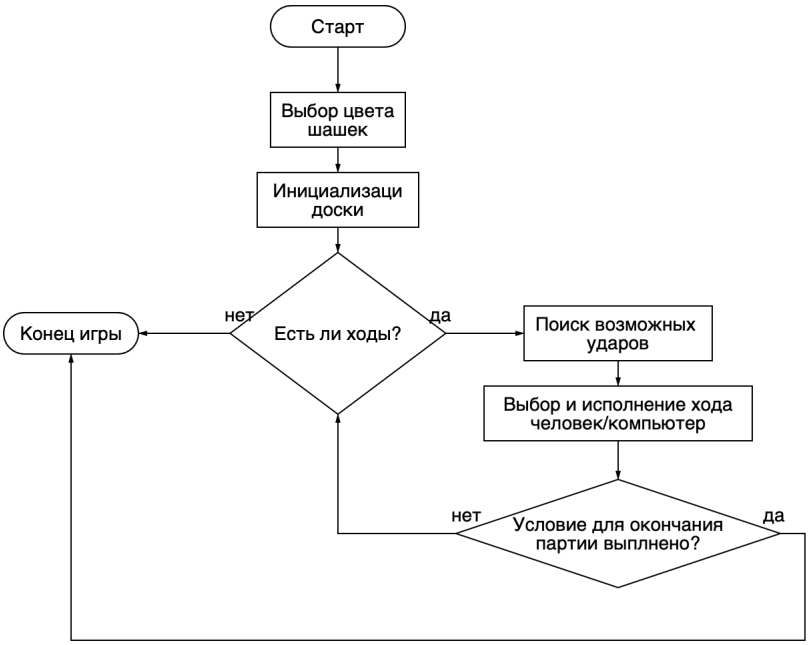
По завершении мы собираем результаты из std::future, объединяем их в общий вектор.

3. **Основные функции:**

findAllCaptures (для боёв),

findAllNormalMoves (для обычных ходов).

4. Основной игровой цикл



1. **Выбор стороны:**

В начале main спрашиваем, кем хочет играть пользователь: белыми (ходят первыми) или чёрными (ходят вторыми).

2. **Инициализация:**

initBoard заполняет массив фигур.

2. **Цикл:**
    a. Печать доски (printBoard).
    b. Проверка ходов (hasAnyMove) — если нет, текущий игрок проигрывает.
    c. Поиск боёв (findAllCaptures). Если результат не пуст — надо бить. Иначе findAllNormalMoves.
    d. Выбор хода:
        i. Если человек (человек ходит своими шашками/дамками), вводит две координаты, например «A3 B4». Программа ищет совпадающий ход.
        ii. Если компьютер, берётся случайный вариант (chooseComputerMove).
    e. Выполняем (makeMoveSequence).
    f. Проверяем, не выбиты ли все шашки у соперника (или у него нет ходов). Если да — игра завершается.
    g. Снимаем время на ход, выводим.
    h. Переключаем ход (белые/чёрные).

4. **Завершение:**

Когда у одной стороны нет ходов (или фигур), игра завершается, выводим результат.

Анализ вариантов решения

1. **Без параллелизма (последовательный перебор)**

Самый простой вариант: один цикл по всем клеткам (r=0..7, c=0..7).

Меньше кода, проще отладка, но может быть медленнее, если мы масштабируем логику (например, расширим доску или будем исследовать глубокие варианты ходов).

2. **Отдельная задача для каждой клетки**

Можно теоретически запустить std::async для каждой из 64 клеток. Но для доски 8×8 это, скорее всего, слишком много (64 потока или 64 задач).

Нагрузки на планировщик и система управления потоками могут ухудшить производительность.

3. **Разбиение на «чанки»**

Текущий код — создаём число задач, примерно равное числу доступных аппаратных потоков. Каждая задача обрабатывает свои полосы строк.

Это даёт параллелизм без перегрузки из-за слишком большого числа потоков.

4. **Использование OpenMP /** <execution>

Могли бы применить #pragma omp parallel for, но требуется поддержка компилятора/библиотеки OpenMP.

5. **Расширение алгоритма (ИИ)**

В коде компьютер делает ход случайно; можно было бы реализовать **Minimax**/альфа-бету и распараллелить **поиск в глубину**.

Это позволило бы компьютеру играть сильнее, а выгода от параллельности выросла бы (больше работы на вычисление).

Источники

Википедия https://ru.wikipedia.org/wiki/Русские_шашки

ФШР https://shashki.ru/variations/draughts64/

Введение в многопоточность: C++ https://rekovalev.site/multithreading-3-cpp/