МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Программа, играющая в Английские шашки 12x12

Отчет по курсовой работе

Этап №2

Вариант № 17

по дисциплине

Алгоритмы и структуры данных

РУКОВОДИТЕЛЬ:

_____ Капранов С.Н.

СТУДЕНТ:

_____ Сапожников В.О.

19-ИВТ-3

Работа защищена «___» _____

С оценкой _____

Нижний Новгород 2020

Содержание

1. Текст задачи

Написать программу, играющую в Английские шашки на поле 12x12.

Замечание: стандартные Английские шашки играются на поле 8x8, данная модификация играется на поле 12x12 (Размер поля и расстановка шашек аналогичная Канадским шашкам).

Правила игры:

Цель игры — уничтожить все шашки противника или лишить их возможности хода («запереть»).

Игра проводится на доске **12×12** клеток. В начальной позиции у каждого игрока по **30** шашек, расположенных на первых пяти рядах на черных клетках.

Первый ход делают белые шашки (Игрок - человек). Простая шашка ходит по диагонали вперёд на одну клетку.

При достижении любого поля последней горизонтали, простая шашка превращается в дамку. Дамка может ходить на одно поле по диагонали как вперёд так и назад.

Взятие обязательно, если оно возможно. Шашки снимаются с доски лишь после того, как берущая шашка остановилась. При нескольких вариантах взятия игрок выбирает вариант взятия по своему усмотрению, и в выбранном варианте необходимо бить все доступные для взятия шашки. При взятии дамка побьет только через одно поле в любую сторону, а не на любое поле диагонали, как в русских или международных шашках.

2. Алгоритм Минимакса

2.1. Описание вычисления

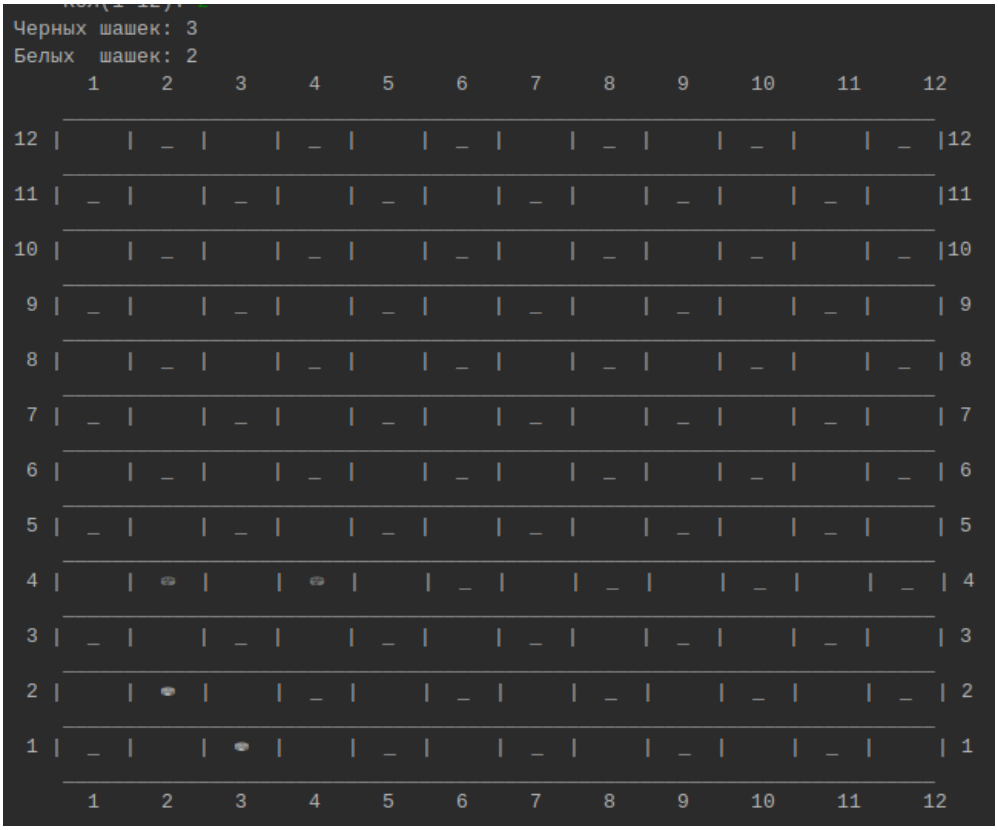
Алгоритм Минимакса — правило принятия решений, используемое в теории игр, теории принятия решений, исследовании операций, статистике и философии для минимизации возможных потерь их тех, которые лицу, принимающему решение, нельзя предотвратить при развитии событий по наихудшему сценарию.

Строится дерево ходов (решений) заданной глубины. После постройки дерева получаем результаты оценочной функции для каждого листа (состояния поля) и начинаем двигаться вверх, к корню дерева. На уровне **MIN** выбирается минимальное значение дочерних узлов и транслируется на данный уровень, а на уровне **MAX** максимальное значение дочерних узлов. Последними операциями выбора и трансляции получаем значение для корня дерева. Как только получено значение корня дерева делаем ход согласно той ветви, которая привела к данному результату. Если к одинаковому значению корня приводят несколько ветвей дерева, то направления движения выбирается произвольно.

2.1. Пример расчета

Рассмотрим состояние поля, представленное на скриншоте 1. Ход делает компьютер (**MAX** – черные шашки).

Замечание: для рассмотрения алгоритма специально было выбрано такое состояние поля, при котором дерево минимакса имеет малое кол-во ветвей, что является более приемлемым для визуального восприятия.



Скриншот 1

Для данного состояния возможно построение следующего дерева минимакса (рис.1)

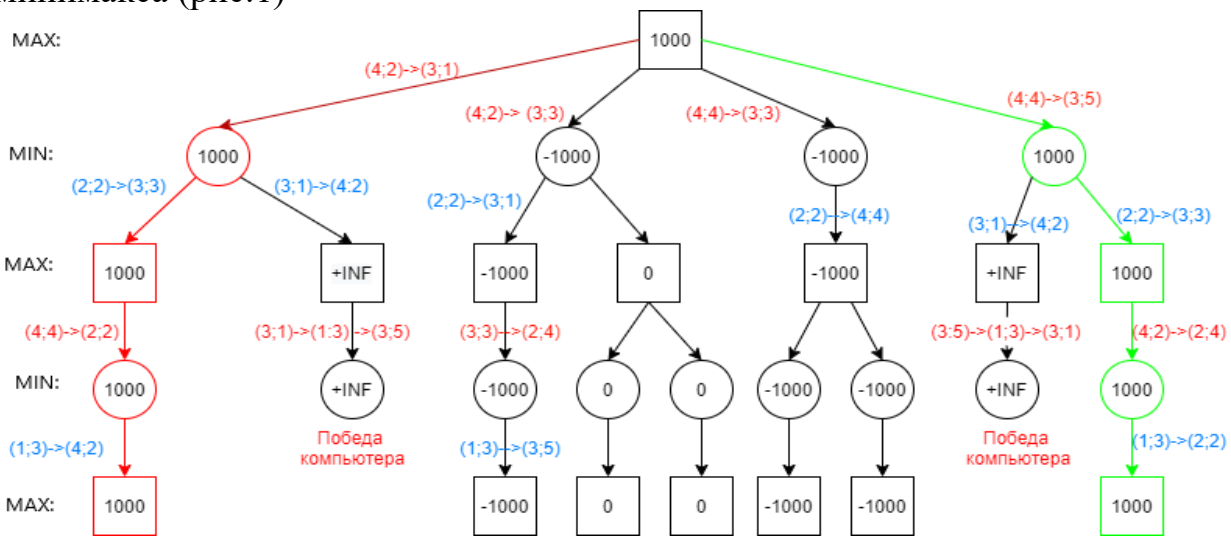
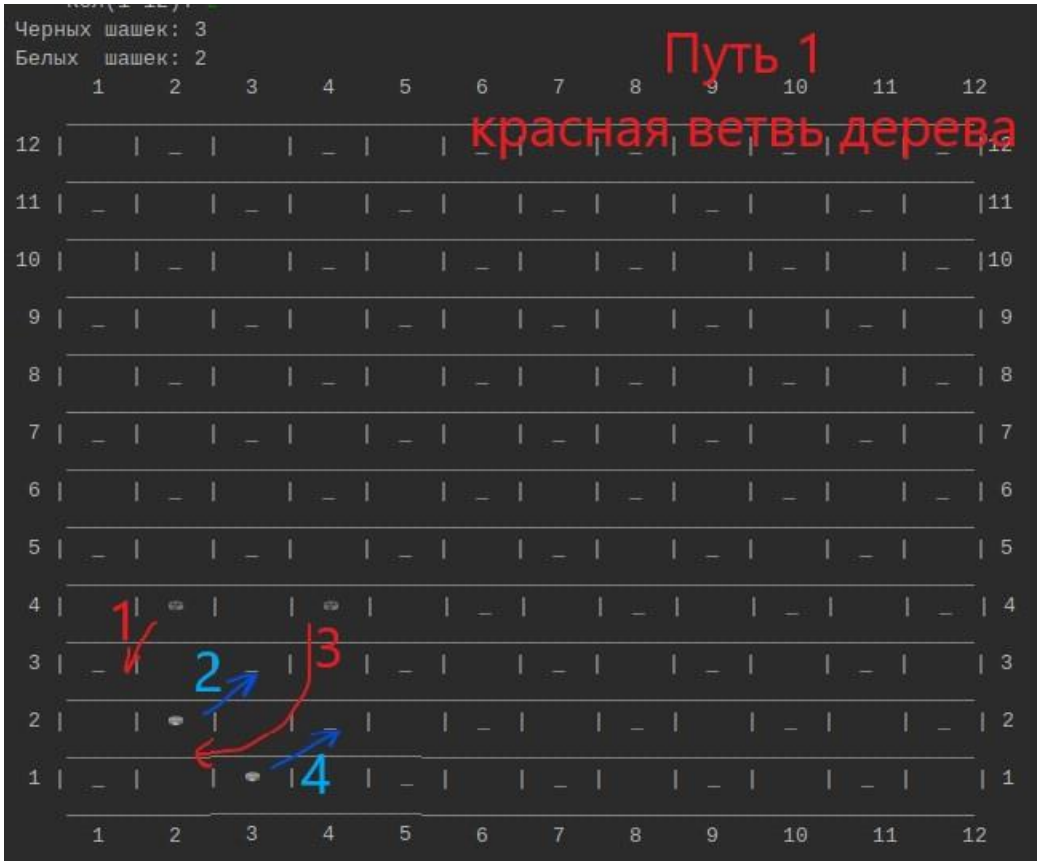


рис.1

где красные координаты – ходы компьютера, синие – ходы человека (координаты указаны не везде, дабы не загромождать рисунок), красная и зеленая ветви - те возможные последовательности ходов, которые привели к данному значению корня дерева.

Рассмотрим последовательность ходов красной ветви (скриншот 2):



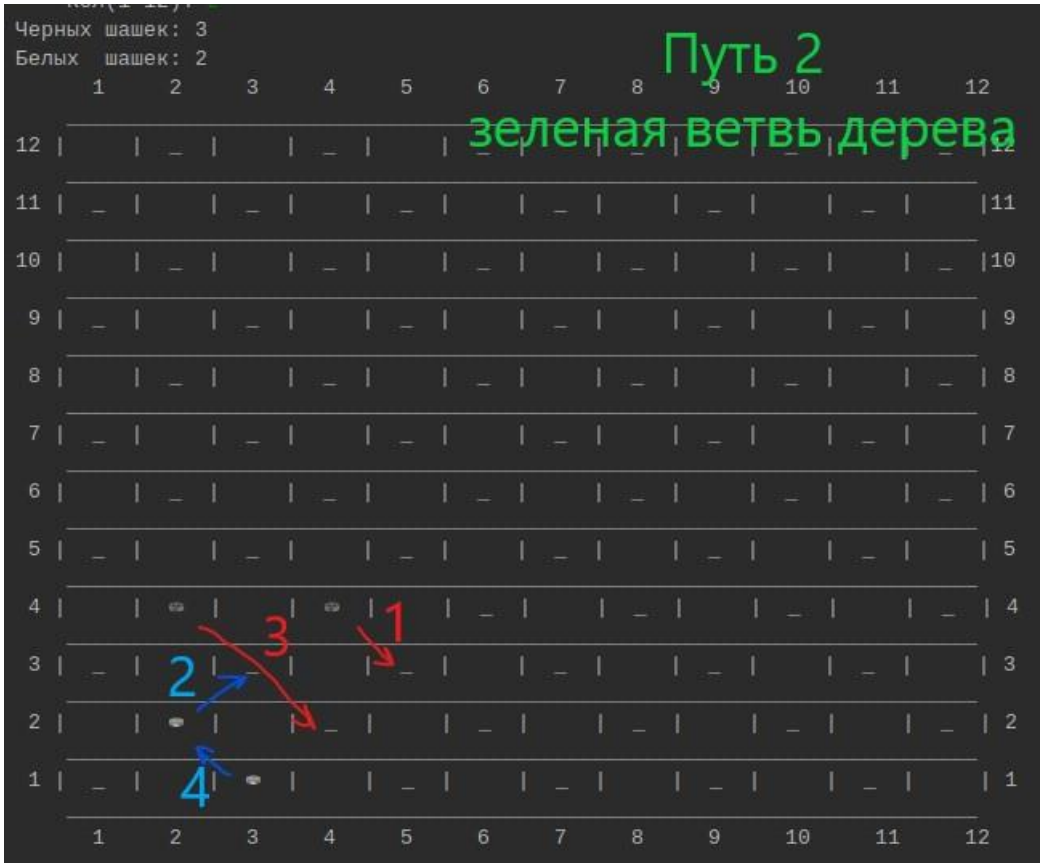
Ходы:

1. (4;2)→(3;1)
2. (2;2)→(3;3)
3. (4;4)→(2;2)
4. (1;3)→(4;2)

Скриншот 2

В итоге данной последовательности игроку **MAX**(компьютер – черные) удалось съесть одну шашку противника и получить значение ОФ = 1000.

Рассмотрим последовательность ходов зеленой ветви (скриншот 3):



Ходы:

1. (4;4)→(3;5)
2. (2;2)→(3;3)
3. (4;2)→(2;4)
4. (1;3)→(4;2)

Скриншот 3

В итоге данной последовательности игроку **MAX**(компьютер – черные) удалось съесть одну шашку противника и получить значение ОФ = 1000.

Обе рассмотренные ситуации устраивают компьютер, значит предпочтительным является ход (4;2)→(3;1) или (4;4)→(3;5).

2.3. Программный код

Замечание: реализации классов Board и Move не являются итоговыми.

```
/**
 * Алгоритм Minimax'a
 *
 * @param board   - текущее состояние доски
 * @param command - 'команда' для которой обрабатывается ход
 * @param depth   - текущий уровень глубины
 * */
private static int miniMax(Board board, Command command,
                           int depth)
{
    //Если мы НЕ можем дальше строить дерево, то
    //оцениваем текущее положение
    if(!canExploreFurther(board, command, depth))
    {
        return fieldAssessment(board);
    }

    //создаем список возможных ходов и заполняем его
    Vector<Vector<Move>> possibleMoveSeq =
                                expandMoves(board, command);

    //создаем список возможных состояний доски и заполняем
    //его
    Vector<Board> possibleBoardConf =
      getPossibleBoardConf(board, possibleMoveSeq, command);

    int value;

    //если ход принадлежит черным(компьютер, MAX)
    if(command == Command.black)
    {
        value = Integer.MIN_VALUE;

        //перебор по возможным состояниям доски
        for (int i = 0; i < possibleBoardConf.size(); i++)
        {
            Board cur_boards     = possibleBoardConf.get(i);
            Vector<Move> moveSeq = possibleMoveSeq.get(i);

            value = miniMax(cur_boards, Command.white,
                            depth + 1);
        }
    }

    //иначе ход принадлежит белым(человек, MIN)
    else
    {
        value = Integer.MAX_VALUE;

        //перебо по возможным состояниям доски
```

```
            for (int i = 0; i < possibleBoardConf.size(); i++)
            {
                Board cur_boards     = possibleBoardConf.get(i);
                Vector<Move> moveSeq = possibleMoveSeq.get(i);

                value = miniMax(cur_boards, Command.white,
                                depth + 1);
            }
        }

        return value;
    }

    /**
     * Вспомогательная функция проверки возможности
     * дальнейшего построения дерева MiniMax'a.
     *
     * @param board   - текущее состояние доски
     * @param command - 'команда' для которой обрабывается ход
     * @param depth   - текущий уровень глубины
     * */
    private static boolean canExploreFurther(Board board,
                                     Command command, int depth)
    {
        //Если игра кончилась или нет невозможный ходов
        //возвращаем false - дальнейшее построение
        //дерева минимакса невозможно
        if((board.CheckGameComplete()||
            board.CheckGameDraw(command)))
        {
            return false;
        }

        //если текущая глубина не равна максимальной,
        //то возвращает true и строим дерево минимакса дальше
        return depth != MAX_DEPTH;
    }
```

Board.java

```
/**
 * Класс отвечающий за игровое поле
 * */
public class Board
{
    private final MoveEvaluator oracle = new MoveEvaluator();
    int blackCheckers;               //кол-во черных
    int whiteCheckers;               //кол-во белых

    static final int rows = 12;      //всего строк доски
    static final int cols = 12;      //всего колон доски
    CellContents[][]    cell;        //двумерный массив клеток
```

```
    /**
     * Конструктор по умолчанию.
     * Доска в начальном положении
     * */
    Board()
    {
        this.blackCheckers = 30;    //начальное кол-во черных
        this.whiteCheckers = 30;    //начальное кол-во белых

        //Инициализация клеток поля
        //CellContents.white   - в клетке белая шашка
        //CellContents.inaccessible - неиспользуемая клетка (в
        //нее невозможно сделать ход)
        //CellContents.empty   - пустая клетка (в нее возможно
        //сделать ход)
        //CellContents.black,  - в клетке черная шашка
        this.cell = new CellContents[][]
        {
          //описание массива 12x12 скоращено.
        };
    }
```

Move.java

```
/**
 * Класс 'ходов'.
 * */
public class Move
{
    int initialRow;    //начальная строка хода
    int initialCol;    //начальная колона хода
    int finalRow;      //конечная  строка хода
    int finalCol;      //конечная  колона хода

    /**
     * Конструктор с параметрами.
     * Создает объект - ход с переданными параметрами.
     *
     * @param r1, c1 - координаты начала хода
     *
     * @param r2, c2 - координаты конца  хода
     * */
    Move(int r1, int c1, int r2, int c2)
    {
        this.initialRow = r1;
        this.initialCol = c1;
        this.finalRow = r2;
        this.finalCol = c2;
    }

    /**
     * Проверка на соответствие данного хода переданному ходу.
     *
     * @param Move – переданный ход для проверки.
```

```
     * */
    public boolean conformity(Move move)
    {
        return this.initialRow == move.initialRow
                && this.initialCol == move.initialCol
                && this.finalRow == move.finalRow
                && this.finalCol == move.finalCol;
    }

    /**
     * Проверка: есть ли данный ход в
     * списке возможных ходов.
     *
     * @param moves - список возможных ходов.
     * */
    public boolean existsInVector(Vector<Move> moves)
    {
        for (int i = 0; i<moves.size(); i++)
        {
            if (this.conformity(moves.elementAt(i)))
            {
                return true;
            }
        }
        return false;
    }

    /**
     * Вывод информации о ходе на экран.
     * */
    public void display(){
        System.out.print("(" + (this.initialRow + 1) + "," +
                           (this.initialCol + 1) + ") -->" + "
                           ("+ (this.finalRow + 1) + ", " +
                           (this.finalCol + 1) + ")");
    }
}
```