МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ижевский государственный технический университет имени М. Т. Калашникова»

Институт «Информатика и вычислительная техника»

Кафедра «Программное обеспечение»

Работа защищена с оценкой

«\_\_\_\_\_\_\_\_\_\_\_\_»

Дата\_\_\_\_\_\_\_\_\_\_

Подпись\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Математические основы искусственного интеллекта»

на тему:«Программирование игры с использованием ИИ»

Выполнил:

Студент группы Б18-191-2 Гумметов Р. А.

Семенов Я. П.

Руководитель:

к. т. н., доцент Коробейников А. В.

Рецензия:

степень достижения поставленной цели работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

полтона разработки темы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

уровень самостоятельности работы обучающегося\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

недостатки работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

СОДЕРЖАНИЕ

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc73530870)

[КРАТКОЕ ОПИСАНИЕ ИГРЫ 4](#_Toc73530871)

[ФОРМАЛИЗАЦИЯ ДЕРЕВА СОСТОЯНИЙ 7](#_Toc73530872)

[КРАТКОЕ ОПИСАНИЕ ИСПОЛЬЗОВАННЫХ АЛГОРИТМОВ 8](#_Toc73530873)

[МИНИМАКСНЫЙ АЛГОРИТМ НА ОГРАНИЧЕННУЮ ГЛУБИНУ 8](#_Toc73530874)

[ОПИСАНИЕ АДАПТАЦИИ АЛГОРИТМА MINIMAX ДЛЯ ШАШЕК 9](#_Toc73530875)

[СТРУКТУРА АЛГОРИТМА 11](#_Toc73530876)

[ПРИМЕР РАБОТЫ ПРОГРАММЫ 13](#_Toc73530877)

[ЗАКЛЮЧЕНИЕ 16](#_Toc73530878)

[СПИСОК ЛИТЕРАТУРЫ 17](#_Toc73530879)

[ПРИЛОЖЕНИЕ 1 18](#_Toc73530880)

[ТЕКСТ ПРОГРАММЫ 18](#_Toc73530881)

# ПОСТАНОВКА ЗАДАЧИ

Решить одну из игровых задач или головоломку методом поиска по дереву игровых состояний. Разработать алгоритм решения и реализовать его в виде программы на языке высокого уровня.

При просмотре дерева состояний использовать один из методов:

* просмотр дерева состояний на полную глубину;
* просмотр дерева состояний на ограниченную глубину.

При просмотре дерева состояний использовать один из алгоритмов:

* «жадный» алгоритм;
* минимаксный алгоритм;
* минимаксный алгоритм с альфа-бета-отсечением.

При просмотре дерева состояний на ограниченную глубину предложить ряд эвристических параметров для оценки состояний. Для каждой эвристики ввести свой весовой коэффициент.

# КРАТКОЕ ОПИСАНИЕ ИГРЫ

Шашки – это настольная игра для двух участников, в которую играют на специально расчерченном поле (доске) с помощью разноцветных фишек (шашек). Игра в шашки была любимым увлечением у вельмож и фараонов Древнего Египта, о чем свидетельствуют многочисленные находки в их гробницах. В том числе из гробницы Тутанхамона была извлечена доска для игры в шашки, состоявшая из тридцати клеток.

* Игра предназначена для двух игроков.
* У каждого игрока есть по 12 шашек одного цвета, выстроенных на доске по определенной схеме.
* Игра проходит на специальной шашечной доске (8 х 8), причем шашки могут ходить только по черным клеткам в количестве тридцати двух штук.
* Первым ходит игрок, владеющий белыми шашками, далее в порядке очереди.
* Шашки бывают двух видов: обычные и дамки. Дамки обозначаются перевернутыми вверх шашками. Дамкой шашка становится, дойдя до последнего поля соперника. Дамки могут двигаться по диагонали на любое количество полей.
* В шашках возможна ничья.

Целью игры является "побить" или "запереть" все шашки своего соперника. Запертые шашки не являются побитыми, однако сами ходить они не могут. Выбор цвета игроками определяется жребием или по договоренности. Шашки расставляются на трех, ближних к игроку, рядах на темных клетках. Право первого хода обычно принадлежит игроку, который играет белым (светлым) цветом. Ходы осуществляются соперниками поочередно. Ход считается сделанным, если участник игры после перемещения шашки отпустил руку. Если игрок дотронулся до шашки, он обязан ей сделать ход. Если, кто-либо из соперников хочет поправить шашки, обязан предупредить заранее.

Математическое решение

Разработаем математическую модель для игры в шашки, будем использовать для этого игровое поле 4\*4. Обозначим наши переменные следующим образом:

Пусть - переменная для белых шашек, где x = 1,2,3,4,5,6,7,8.

Тогда - переменная для черных шашек, где у = 1,2,3,4,5,6,7,8.

Из этого следует что:

= 0 - если клетка игрового поля пуста,

= 1- если на клетке стоит белая шашка,

= 0 - если клетка игрового поля пуста,

= 1- если на клетке стоит черная шашка.

ОПИСАНИЕ ПРЕДЛОЖЕННЫХ ЭВРИСТИК

Эвристика - это простые стратегии принятия решений, используемые для быстрого и эффективного достижения. Это способ оценки выгоды от того или иного хода. В самом простейшем случае, можно выделить некоторые состояния игрового поля (ничья, победа и поражение) и присвоить каждому из них некоторый числовой коэффициент(оценку).Такая оценка строится не для каждого узла дерева игры, а только для его листьев. Оценка же каждого узла получается из следующих соображений: на каждом этапе раскрытия дерева решений, начиная с последнего (на котором получаются листы с оценками), выбирается ход, наиболее предпочтительный для игрока, чей ход приводит к порождению узлов. При этом, для игрока, за которого ведется игра, оценка должна стремиться к максимуму, а для противника к минимуму.

Так, играя за «белых», каждый нечетный ход представляет их интересы и выбирается ход с наибольшей оценкой (игрок MAX), а каждый четный - интересы «черных», выбирается ход с наименьшей оценкой (игрок MIN). Такой подход называется минимаксной процедурой.

# ФОРМАЛИЗАЦИЯ ДЕРЕВА СОСТОЯНИЙ

Для шашек, размер пространства состояний составляет 3^32 = 1,8530202 × 10^15. (Есть три состояния для каждой ячейки и 32 ячейки). Чтобы связать дерево игры, есть 7 возможных начальных ходов, 7 возможных ответов.

Пример дерева состояний представлен на рис. 1.

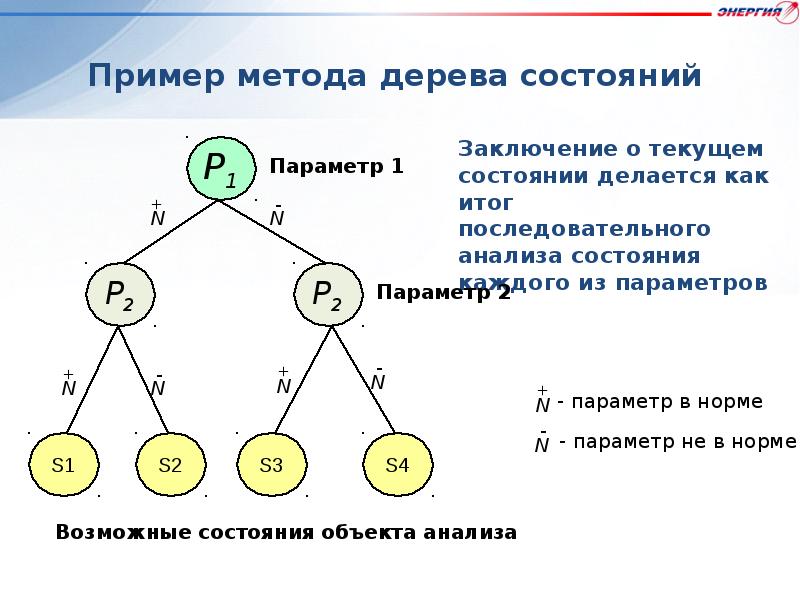


Рис.1 – Дерево состояний

# КРАТКОЕ ОПИСАНИЕ ИСПОЛЬЗОВАННЫХ АЛГОРИТМОВ

## МИНИМАКСНЫЙ АЛГОРИТМ НА ОГРАНИЧЕННУЮ ГЛУБИНУ

В некоторых играх, процесс игры между двумя игроками (которые делают ход по очереди) можно представить в виде так называемого дерева вариантов. В каждой конкретной позиции каждый игрок, обычно, имеет выбор между разными вариантами своего хода. И в ответ на каждый из этих вариантов соперник может также походить разными способами.  
 Поскольку в любой момент игры о состоянии игрового поля есть полная информация, текущее состояние позиции всегда можно точно оценить. Такая функция носит название статическая функция оценки или сокращённо СФО. При этом, чем большее значение принимает эта функция при оценке конкретной позиции, тем более выгодна позиция для одного игрока (назовём его — максимизирующий игрок). Чем меньше при оценке позиции числовое значение этой функции, тем более выгодна позиция для второго игрока (назовём его — минимизирующий игрок).  
 После каждого хода позиция меняется, а значит и меняется её оценка. При рассмотрении дерева вариантов, каждому игроку необходимо не просто предпочесть те ветки, в которых оценка наиболее благоприятна для него. Также следует избегать тех веток, в которых оценка позиции благоприятна для соперника.  
 Предполагается, что соперник также руководствуется рационализмом и также избегает вариантов, которые могут привести его к проигрышу. То есть каждый игрок при выборе варианта исходит из максимизации своей выгоды и одновременно минимизации выгоды противника.

# ОПИСАНИЕ АДАПТАЦИИ АЛГОРИТМА MINIMAX ДЛЯ ШАШЕК

Минимакс - это искусственный интеллект, применяемый в играх с двумя игроками, таких как крестики-нолики, шашки, шахматы и го. Эти игры известны как игры с нулевой суммой, потому что в математическом представлении: один игрок выигрывает (+1), а другой игрок проигрывает (-1) или оба игрока не выигрывают (0).

Алгоритм рекурсивно ищет лучший ход, который приведет максимального игрока к победе или не проиграет (ничья). Он учитывает текущее состояние игры и доступные ходы в этом состоянии, затем для каждого допустимого хода он играет (чередуя min и max), пока не найдет конечное состояние (выигрыш, ничья или проигрыш).

Функция минимакс в программе(рис.2):



Рис.2 – функцияминимакс

* bf: состояние доски
* depth: индекс узла в игровом дереве
* alpha: отсечение альфа параметров
* beta: отсечение бета параметров
* turn:чья очередь

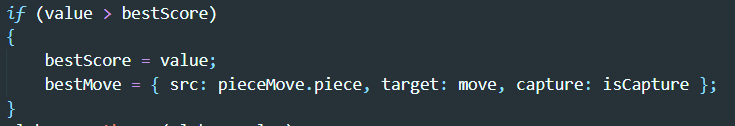


Рис.3. – Проверка лучшего хода



Рис.3. – Оценка состояния

Если глубина равна нулю, то на доске нет новых пустых ячеек для игры. Или, если игрок выигрывает, то игра заканчивается на МАКСИМУМ или МИНИМУМ. Таким образом, оценка для этого состояния будет возвращена.

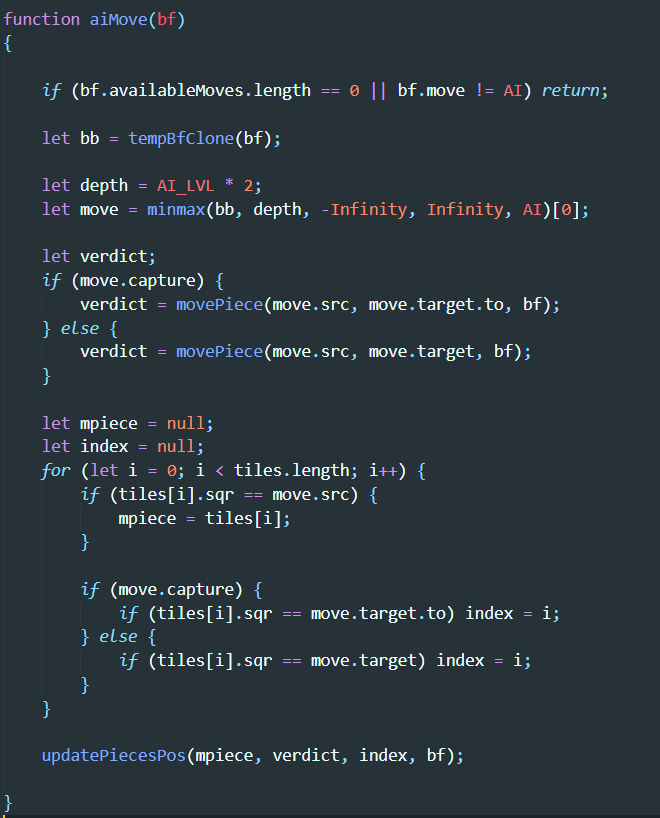


Рис.4. – Основная часть алгоритма

bb– состояние поля

depth-глубина поиска

move- наилучший вариант перемещения

verdict- состояние всей доски после хода

mpiece – клетка куда сходили

updatePiecePos- функция обновляющая позиции

tiles[]-массив клеток

# СТРУКТУРА АЛГОРИТМА

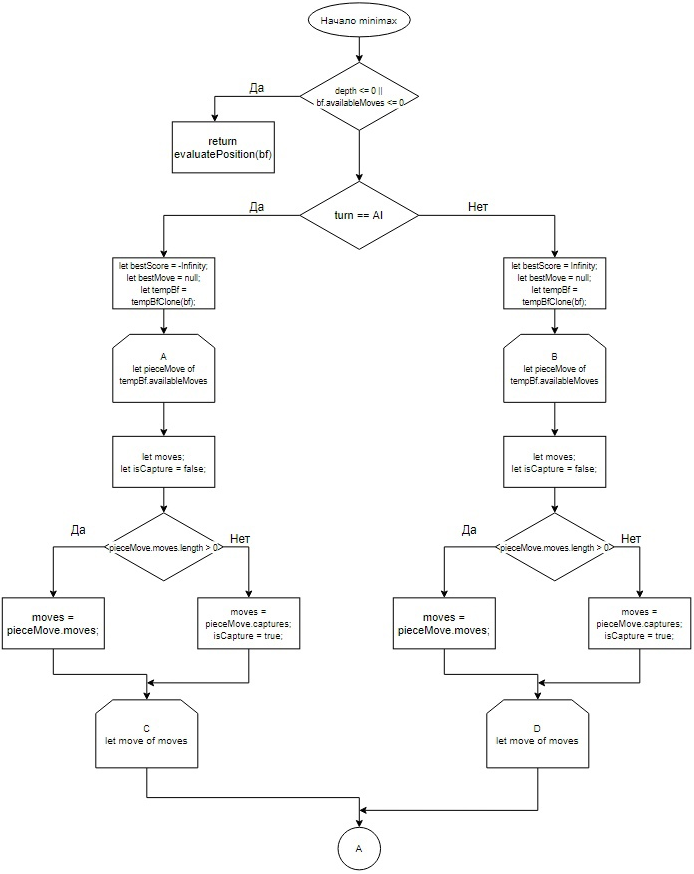


Рис. 5

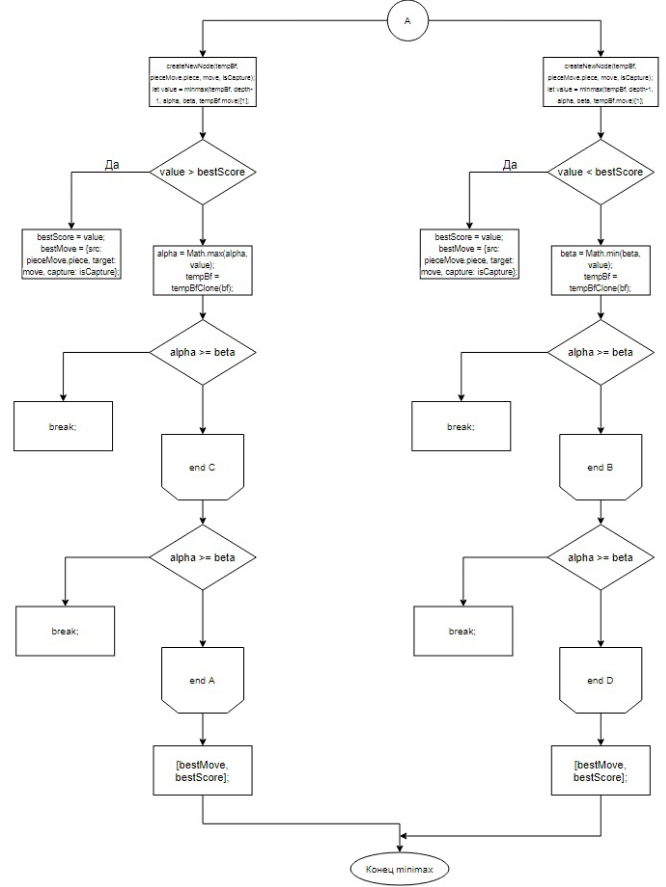


Рис. 6

# ПРИМЕР РАБОТЫ ПРОГРАММЫ

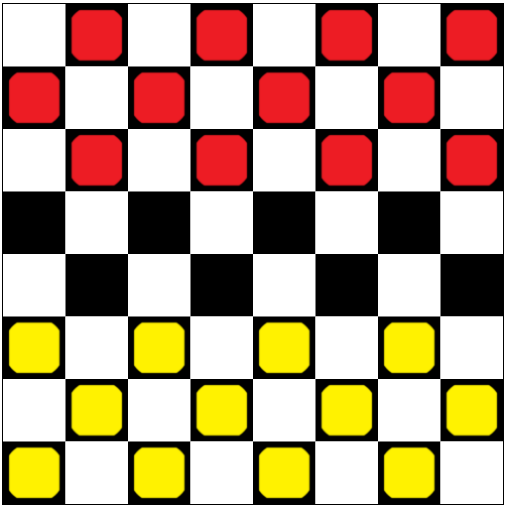


Рис. 5 - Начальное поле

Пользователь играет желтыми шашками, ИИ - красными. Сделаем первый ход:

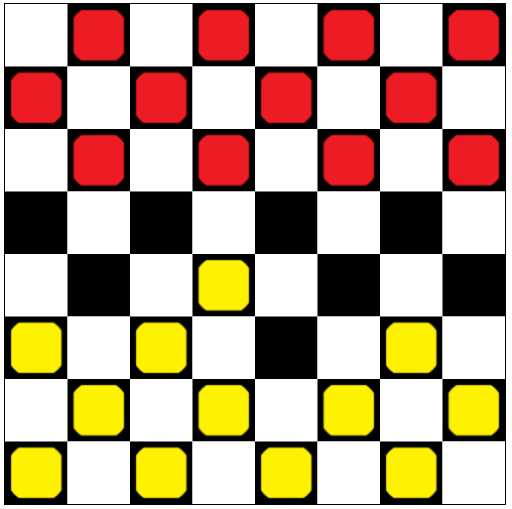


Рис. 6

ИИ сделал первый ход:

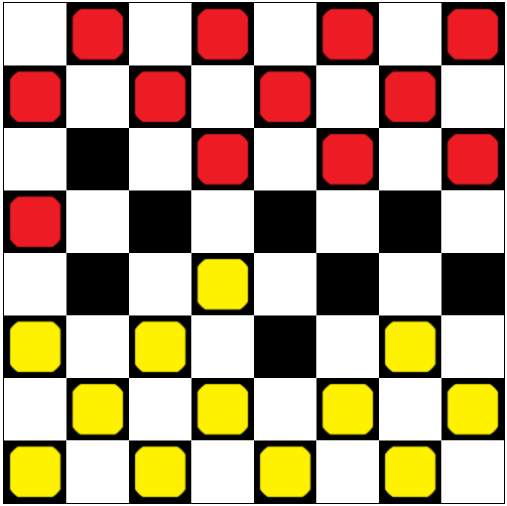


Рис. 7

Процесс игры:

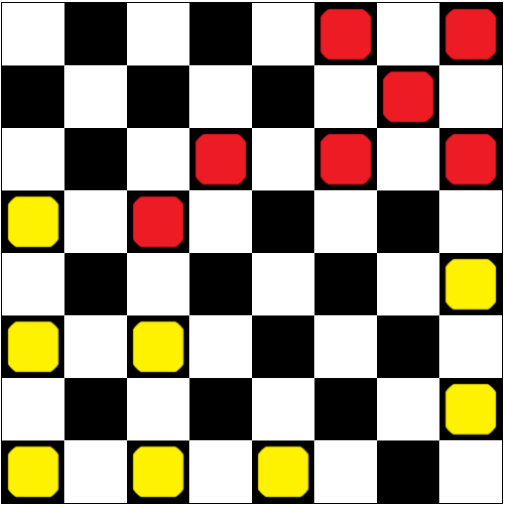


Рис. 8

ИИ выиграл:

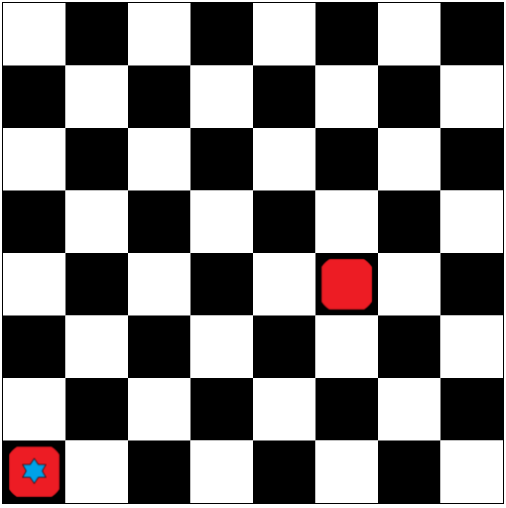


Рис. 9

Сложность игры против ИИ зависит от глубины поиска. Если глубина поиска небольшая, то иногда удается обыграть ИИ, если глубину поиска установить больше, то выиграть становится почти невозможно.

# ЗАКЛЮЧЕНИЕ

В ходе данной курсовой работы мы решили игровую задачу «шашки» методом поиска по дереву игровых состояний. Разработали алгоритм решения и реализовали его в виде программы на языке JavaScript. При просмотре дерева состояний использовали метод просмотра на ограниченную глубину и минимаксный алгоритм с альфа-бета-отсечением.

# СПИСОК ЛИТЕРАТУРЫ

1. Люггер, Д. Ф. Искусственный интеллект. Стратегии и методы решения сложных проблем. – М. : Вильямс, 2003. – 864 с.
2. Корнилов, Е. Программирование шахмат и других логических игр. – СПб. : БХВ-Петербург, 2005. – 272 с.
3. Ясницкий, Л. Н. Введение в искусственный интеллект. – М. : ACADEMIA, 2005. – 176 с
4. Book: George T. Heineman; Gary Pollice; Stanley Selkow. Algorithms in a nutshell. O'Reilly, 2009.
5. Wikipédia: https://en.wikipedia.org/wiki/Minimax

# ПРИЛОЖЕНИЕ 1

# ТЕКСТ ПРОГРАММЫ

Файл index.html:

<!DOCTYPE *html*>

<html>

<head>

    <title>Dama</title>

</head>

<body>

    <script *type*="text/javascript" *src*="board.js"></script>

    <script *type*="text/javascript" *src*="piece.js"></script>

    <script *type*="text/javascript" *src*="tile.js"></script>

    <script *type*="text/javascript" *src*="ai.js"></script>

    <script *type*="text/javascript" *src*="main.js"></script>

</body>

</html>

Файлmain.js:

const canvas = document.createElement('canvas');

const context = canvas.getContext('2d');

canvas.width = 500;

canvas.height = 500;

canvas.style.border = '1px solid black';

document.body.append(canvas);

const tilesize = canvas.width / 8;

const tiles = [];

let moving = false;

let mousepos = 0;

let piecemoving = null;

function init(){

    let colorblack = false;

    let indexOfSqr = 0;

*for*(let r=0; r<8; r++){

*for*(let c=0; c<8; c++){

            let p = null;

*if*(colorblack){

*if*(BOARD\_DEF.board[sqr48[indexOfSqr]] != PIECE\_TYPE.NO\_PIECE)

                {

                    p = new Piece(c\*tilesize, r\*tilesize, tilesize, BOARD\_DEF.board[sqr48[indexOfSqr]]);

                }

                tiles.push(new Tile(c\*tilesize, r\*tilesize, tilesize, colorblack, sqr48[indexOfSqr], p));

                indexOfSqr++;

            }*else*{

                tiles.push(new Tile(c\*tilesize, r\*tilesize, tilesize, colorblack, SQR.NONE, p));

            }

            colorblack = !colorblack;

        }

        colorblack = !colorblack;

    }

}

function draw(){

    context.clearRect(0, 0, canvas.width, canvas.height);

*for*(let i=0; i<tiles.length; i++){

        tiles[i].draw(context);

    }

*for*(let i=0; i<tiles.length; i++){

        tiles[i].drawPiece(context);

    }

}

function collide( source, target ) {

*return* !(

        ( ( source.y + source.h ) < ( target.y ) ) ||

        ( source.y > ( target.y + target.h ) ) ||

        ( ( source.x + source.w ) < target.x ) ||

        ( source.x > ( target.x + target.w ) )

    );

}

canvas.onmouseup = function(e){

    moving = false;

    let mp = getPos(canvas, e);

*if*(BOARD\_DEF.move == AI && withAI){

*return*;

    }

    let donePlayerMove = false;

*for*(let i=0; i<tiles.length; i++){

        let t = {x: tiles[i].x, y: tiles[i].y, w: tiles[i].size, h: tiles[i].size};

*if*(collide(mp, t) && piecemoving != null && tiles[i] != piecemoving && tiles[i].sqr != SQR.NONE){

*if*(piecemoving.piece == null) *return*;

            let move = piecemoving.piece.move(piecemoving.sqr, tiles[i].sqr);

*if*(move == null) *return*;

*if*(!updatePiecesPos(piecemoving, move, i, BOARD\_DEF)) *break*;

            donePlayerMove = true;

*break*;

        }

    }

*if*(piecemoving != null && piecemoving.piece != null && !donePlayerMove){

        piecemoving.piece.x = piecemoving.x;

        piecemoving.piece.y = piecemoving.y;

        piecemoving.piece.size = piecemoving.size;

        piecemoving = null;

    }

}

function updatePiecesPos(piecemoving, move, i, bf){

*if*(!move[0]) *return* false;

*switch*(move[1])

    {

*case* MOVE\_TYPE.MOVE\_NORMAL:

        {

            tiles[i].piece = piecemoving.piece;

            tiles[i].piece.x = tiles[i].x;

            tiles[i].piece.y = tiles[i].y;

            tiles[i].piece.size = tiles[i].size;

            piecemoving.piece = null;

            piecemoving = null;

            updateSuperPiece(bf.rPieces, PIECE\_TYPE.SUPER\_RED);

            updateSuperPiece(bf.yPieces, PIECE\_TYPE.SUPER\_YELLOW);

*break*;

        }

*case* MOVE\_TYPE.MOVE\_CAPTURE:

        {

            tiles[i].piece = piecemoving.piece;

            tiles[i].piece.x = tiles[i].x;

            tiles[i].piece.y = tiles[i].y;

            tiles[i].piece.size = tiles[i].size;

*for*(let toRemove of tiles){

*if* (toRemove.sqr == move[2]){

                    toRemove.piece = null;

*break*;

                }

            }

            piecemoving.piece = null;

            piecemoving = null;

            updateSuperPiece(bf.rPieces, PIECE\_TYPE.SUPER\_RED);

            updateSuperPiece(bf.yPieces, PIECE\_TYPE.SUPER\_YELLOW);

        }

    }

*return* true;

}

function updateSuperPiece(piecePos, pieceType){

*for*(let tile of tiles){

*for*(let rP of piecePos){

*if*(BOARD\_DEF.board[rP] == pieceType && tile.sqr == rP){

                tile.piece.pieceType = pieceType;

*break*;

            }

        }

    }

}

canvas.onmousedown =  function(e){

*if*(BOARD\_DEF.move == AI && withAI){

*return*;

    }

    let mp = getPos(canvas, e);

*for*(let i=0; i<tiles.length; i++){

        let t = {x: tiles[i].x, y: tiles[i].y, w: tiles[i].size, h: tiles[i].size};

*if*(collide(mp, t) && tiles[i].piece != null && tiles[i].sqr != SQR.NONE){

            let pTypeSuper = tiles[i].piece.pieceType == PIECE\_TYPE.SUPER\_RED ? PLAYER.P1 : PLAYER.P2;

*if*(BOARD\_DEF.move != tiles[i].piece.pieceType){

*if*(pTypeSuper != BOARD\_DEF.move)

*break*;

            }

*for*(let moves of BOARD\_DEF.availableMoves){

*if*(tiles[i].sqr == moves.piece){

                    piecemoving = tiles[i];

                    mp.size = tilesize;

                    piecemoving.piece.mouseMove(mp);

                    moving = true;

*break*;

                }

            }

*break*;

        }

    }

}

canvas.onmousemove = function(e){

*if*(moving && piecemoving != null){

        let pos = getPos(canvas, e);

        pos.size = tilesize;

        piecemoving.piece.mouseMove(pos);

    }

}

function getPos(c, e){

    var rect = c.getBoundingClientRect();

*return* {x:e.clientX - rect.left, y: e.clientY - rect.top, h: 1, w:1};

}

initBoard(BOARD\_DEF);

init();

*if*(DEBUG){

    var zz = document.createElement('div');

    document.body.append(zz);

}

let counterAlert = 0;

let aiMoveCounter = 0;

let stopLoop = false;

function loop() {

*if*(DEBUG){

        zz.innerHTML = "";

        zz.innerHTML = printBoard();

    }

*if*(withAI && BOARD\_DEF.move == AI){

        aiMoveCounter++;

*if*(aiMoveCounter > 20){

*if*(BOARD\_DEF.availableMoves.length > 0) {

                aiMove(BOARD\_DEF);

                aiMoveCounter = 0;

            }

        }

    }

    draw();

*if*(BOARD\_DEF.availableMoves.length <= 0){

        counterAlert++;

    }

*if*(counterAlert > 30){

        let p = BOARD\_DEF.move == PLAYER.P1 ? "Yellow" : "Red";

        stopLoop = true;

        alert(p +" player wins!");

    }

*if*(!stopLoop)

        requestAnimationFrame(loop);

}

requestAnimationFrame(loop);

Файлboard.js

var DEBUG = false;

var withAI = true;

const imgs = [null, new Image(), new Image(), new Image(), new Image()];

imgs[1].src = 'r.png';

imgs[2].src = 'y.png';

imgs[3].src = 'r\_super.png';

imgs[4].src = 'y\_super.png';

const PLAYER = {P1: 1, P2: 2};

const PIECE\_TYPE = {

    NO\_PIECE: 0,

    RED\_PIECE: 1,

    YELLOW\_PIECE: 2,

    SUPER\_RED: 3,

    SUPER\_YELLOW: 4

};

const SQR = {

    A1:7, A2:13, A3:19, A4:25, A5:31, A6:37, A7:43, A8:49,

    B1:8, B2:14, B3:20, B4:26, B5:32, B6:38, B7:44, B8:50,

    C1:9, C2:15, C3:21, C4:27, C5:33, C6:39, C7:45, C8:51,

    D1:10, D2:16, D3:22, D4:28, D5:34, D6:40, D7:46, D8:52,

    NONE: 0

};

const FILE = {

    FILE\_A:0, FILE\_B:1, FILE\_C:2, FILE\_D:3, FILE\_NONE: 4

};

const RANK = {

    RANK\_1:0, RANK\_2:1, RANK\_3:2, RANK\_4:3,

    RANK\_5:4, RANK\_6:5, RANK\_7:6, RANK\_8:7,

    RANK\_NONE: 9

};

const OFF\_BOARD = [

    0,1,2,3,4,5,

    6,12,18,24,30,36,42,48,54,

    11,17,23,29,35,41,47,53,59,

    55,56,57,58

];

const EVEN\_RANK\_SQR = [

    SQR.A2, SQR.B2, SQR.C2, SQR.D2, SQR.A4, SQR.B4, SQR.C4, SQR.D4,

    SQR.A6, SQR.B6, SQR.C6, SQR.D6, SQR.A8, SQR.B8, SQR.C8, SQR.D8

];

const ODD\_RANK\_SQR = [

    SQR.A1, SQR.B1, SQR.C1, SQR.D1, SQR.A3, SQR.B3, SQR.C3, SQR.D3,

    SQR.A5, SQR.B5, SQR.C5, SQR.D5, SQR.A7, SQR.B7, SQR.C7, SQR.D7

];

const SIDE\_SQR = [SQR.A2, SQR.A4, SQR.A6, SQR.A8, SQR.D1, SQR.D3, SQR.D5, SQR.D7];

const BOARD\_SIZE = 60;

const BOARD\_DEF = {};

const sqr48 = [];

BOARD\_DEF.board = new Array(BOARD\_SIZE);

BOARD\_DEF.move = PLAYER.P2;

BOARD\_DEF.availableMoves = null;

BOARD\_DEF.rPieces = [

    SQR.A1, SQR.B1, SQR.C1, SQR.D1,

    SQR.A2, SQR.B2, SQR.C2, SQR.D2,

    SQR.A3, SQR.B3, SQR.C3, SQR.D3

];

BOARD\_DEF.yPieces = [

    SQR.A8, SQR.B8, SQR.C8, SQR.D8,

    SQR.A7, SQR.B7, SQR.C7, SQR.D7,

    SQR.A6, SQR.B6, SQR.C6, SQR.D6

];

const MOVE\_TYPE = {

    MOVE\_NORMAL : 0,

    MOVE\_CAPTURE : 1

};

const SUPER\_SQR = [[SQR.A8, SQR.B8, SQR.C8, SQR.D8], [SQR.A1, SQR.B1, SQR.C1, SQR.D1]];

function frToSqr(f, r){

*return* (7 + f) + (r \* 6);

}

function initBoard(bf){

*for*(let i=0; i<BOARD\_SIZE; i++){

        bf.board[i] = PIECE\_TYPE.NO\_PIECE;

    }

*for*(let r=RANK.RANK\_1; r<Object.keys(RANK).length-1; r++){

*for*(let f=FILE.FILE\_A; f<Object.keys(FILE).length-1; f++){

            sqr48.push(frToSqr(f, r));

        }

    }

    Object.keys(SQR).forEach(function(key) {

*if*(hasPieceAt(bf.rPieces, SQR[key])){

            bf.board[SQR[key]] = PIECE\_TYPE.RED\_PIECE;

        }

*if*(hasPieceAt(bf.yPieces, SQR[key])){

            bf.board[SQR[key]] = PIECE\_TYPE.YELLOW\_PIECE;

        }

    });

    bf.availableMoves = generateMove(bf.move, bf);

}

function movePiece(src, target, bf){

*if*(bf.board[src] != PIECE\_TYPE.NO\_PIECE && bf.board[target] == PIECE\_TYPE.NO\_PIECE){

*for*(let m of bf.availableMoves){

*if*(m.piece == src && m.moves.includes(target)){

                insertPiece(src, target, bf);

                switchPlayer(bf);

*return* [true, MOVE\_TYPE.MOVE\_NORMAL];

            }*else* *if*(m.piece == src && m.captures.length > 0){

*for*(let capture of m.captures){

*if*(target == capture.to){

                        removePiece(capture.remove, bf);

                        insertPiece(src, target, bf);

*if*(!hasMultipleCapture(target, bf)){

                            switchPlayer(bf);

                        }

*return* [true, MOVE\_TYPE.MOVE\_CAPTURE, capture.remove];

                    }

                }

            }

        }

    }

*return* [false];

}

function removePiece(sqr, bf){

*if*(bf.board[sqr] == PLAYER.P1){

        bf.rPieces.splice(bf.rPieces.indexOf(sqr), 1);

    }*else* *if*(bf.board[sqr] == PLAYER.P2){

        bf.yPieces.splice(bf.yPieces.indexOf(sqr), 1);

    }*else* *if*(PIECE\_TYPE.SUPER\_RED == bf.board[sqr]){

        bf.rPieces.splice(bf.rPieces.indexOf(sqr), 1);

    }*else* *if*(PIECE\_TYPE.SUPER\_YELLOW == bf.board[sqr]){

        bf.yPieces.splice(bf.yPieces.indexOf(sqr), 1);

    }

    bf.board[sqr] = PIECE\_TYPE.NO\_PIECE;

}

function insertPiece(src, target, bf){

    bf.board[target] = bf.board[src];

    bf.board[src] = PIECE\_TYPE.NO\_PIECE;

*if*(PIECE\_TYPE.RED\_PIECE == bf.board[target]){

        bf.rPieces[bf.rPieces.indexOf(src)] = target;

    }*else* *if*(PIECE\_TYPE.YELLOW\_PIECE == bf.board[target]){

        bf.yPieces[bf.yPieces.indexOf(src)] = target;

    }*else* *if*(PIECE\_TYPE.SUPER\_RED == bf.board[target]){

        bf.rPieces[bf.rPieces.indexOf(src)] = target;

    }*else* *if*(PIECE\_TYPE.SUPER\_YELLOW == bf.board[target]){

        bf.yPieces[bf.yPieces.indexOf(src)] = target;

    }

}

function upgradePiece(bf){

*for*(let i=0; i<bf.rPieces.length; i++){

*if*(SUPER\_SQR[0].includes(bf.rPieces[i])){

            bf.board[bf.rPieces[i]] = PIECE\_TYPE.SUPER\_RED;

        }

    }

*for*(let i=0; i<bf.yPieces.length; i++){

*if*(SUPER\_SQR[1].includes(bf.yPieces[i])){

            bf.board[bf.yPieces[i]] = PIECE\_TYPE.SUPER\_YELLOW;

        }

    }

}

function switchPlayer(bf){

    upgradePiece(bf);

    bf.move = bf.move == PLAYER.P1 ? PLAYER.P2 : PLAYER.P1;

    bf.availableMoves = generateMove(bf.move, bf);

}

function hasMultipleCapture(target, bf){

    let moves = generateMove(bf.move, bf);

*for*(let move of moves){

*if*(move.captures.length > 0 && move.piece == target){

            bf.availableMoves = moves.filter(pp => pp.piece == target);

*return* true;

        }

    }

*return* false;

}

function generateMove(player, bf){

    let possibleMoves = [];

    let pieceCount = 0;

    let pieces = [];

*if*(player == PLAYER.P1){

        pieceCount = bf.rPieces.length;

        pieces = bf.rPieces;

    }*else* *if*(player == PLAYER.P2){

        pieceCount = bf.yPieces.length;

        pieces = bf.yPieces;

    }

    let hasCaptureMove = false;

    let superCap = false;

*for*(let i=0; i<pieceCount; i++){

        let moves = moveCount(player, pieces[i]);

        let m = {piece: pieces[i], moves: [], captures: []};

        let move1 = pieces[i] + moves[0];

        let move2 = pieces[i] + moves[1];

        let move3 = pieces[i] + moves[2];

        let move4 = pieces[i] + moves[3];

*if* (bf.board[pieces[i]] == PIECE\_TYPE.SUPER\_YELLOW || bf.board[pieces[i]] == PIECE\_TYPE.SUPER\_RED)

        {

            superCap = superPieceCapture(m, pieces[i], bf) ? true : superCap;

            hasCaptureMove = hasCaptureMove ? true : superCap;

        }

        hasCaptureMove = checkCaptureMoves(m, move1, player, 0, bf) ? true : hasCaptureMove;

        hasCaptureMove = checkCaptureMoves(m, move2, player, 1, bf) ? true : hasCaptureMove;

        hasCaptureMove = checkCaptureMoves(m, move3, player, 2, bf) ? true : hasCaptureMove;

        hasCaptureMove = checkCaptureMoves(m, move4, player, 3, bf) ? true : hasCaptureMove;

*if*(m.captures.length > 0)

            possibleMoves.push(m)

    }

*if*(!hasCaptureMove){

*for*(let i=0; i<pieceCount; i++){

            let moves = moveCount(player, pieces[i]);

            let m = {piece: pieces[i], moves: [], captures: []};

            let move1 = pieces[i] + moves[0];

            let move2 = pieces[i] + moves[1];

*if*(bf.board[pieces[i]] == PIECE\_TYPE.SUPER\_RED || bf.board[pieces[i]] == PIECE\_TYPE.SUPER\_YELLOW){

                superPieceMove(m, pieces[i], bf);

*if*(m.moves.length > 0) possibleMoves.push(m);

*continue*;

            }

*if*(bf.board[move1] == PIECE\_TYPE.NO\_PIECE && !inOffset(move1)){

                m.moves.push(pieces[i] + moves[0]);

            }

*if*(bf.board[move2] == PIECE\_TYPE.NO\_PIECE && !inOffset(move2)){

                m.moves.push(pieces[i] + moves[1]);

            }

*if*(m.moves.length > 0) possibleMoves.push(m);

        }

    }

*return* possibleMoves;

}

function superPieceCapture(piece, sqr, bf){

    let mUpRight = sqr + sqrPieceEvenOrOdd(sqr)[0];

    let mUpLeft = sqr + sqrPieceEvenOrOdd(sqr)[1];

    let mDownLeft= sqr +  sqrPieceEvenOrOdd(sqr)[2];

    let mDownRight = sqr +  sqrPieceEvenOrOdd(sqr)[3];

    let hasCap = false;

    hasCap = sPieceCap(mUpRight, 0, piece, bf) ? true : hasCap;

    hasCap = sPieceCap(mUpLeft, 1, piece, bf) ? true : hasCap;

    hasCap = sPieceCap(mDownLeft, 2, piece, bf) ? true : hasCap;

    hasCap = sPieceCap(mDownRight, 3, piece, bf) ? true : hasCap;

*return* hasCap;

}

function sPieceCap(direct, i, piece, bf){

    let hasCap = false;

*while*(!inOffset(direct)){

*if*(bf.board[direct] != PIECE\_TYPE.NO\_PIECE){

            let sPieceType = bf.move == PLAYER.P1 ? PIECE\_TYPE.SUPER\_RED : PIECE\_TYPE.SUPER\_YELLOW;

*if*(bf.board[direct] == bf.move || bf.board[direct] == sPieceType) *break*;

            let toRemove = direct;

            direct += sqrPieceEvenOrOdd(direct)[i];

*if*(bf.board[direct] == PIECE\_TYPE.NO\_PIECE && !inOffset(direct)){

                piece.captures.push({to: direct, remove: toRemove});

                hasCap = true;

            }

*break*;

        }

        direct += sqrPieceEvenOrOdd(direct)[i];

    }

*return* hasCap;

}

function superPieceMove(piece, sqr, bf){

    let mUpRight = sqr + sqrPieceEvenOrOdd(sqr)[0];

    let mUpLeft = sqr + sqrPieceEvenOrOdd(sqr)[1];

    let mDownLeft= sqr +  sqrPieceEvenOrOdd(sqr)[2];

    let mDownRight = sqr +  sqrPieceEvenOrOdd(sqr)[3];

*while*(bf.board[mUpRight] == PIECE\_TYPE.NO\_PIECE && !inOffset(mUpRight)){

        piece.moves.push(mUpRight);

        mUpRight += sqrPieceEvenOrOdd(mUpRight)[0];

    }

*while*(bf.board[mUpLeft] == PIECE\_TYPE.NO\_PIECE && !inOffset(mUpLeft)){

        piece.moves.push(mUpLeft);

        mUpLeft += sqrPieceEvenOrOdd(mUpLeft)[1];

    }

*while*(bf.board[mDownLeft] == PIECE\_TYPE.NO\_PIECE && !inOffset(mDownLeft)){

        piece.moves.push(mDownLeft);

        mDownLeft += sqrPieceEvenOrOdd(mDownLeft)[2];

    }

*while*(bf.board[mDownRight] == PIECE\_TYPE.NO\_PIECE && !inOffset(mDownRight)){

        piece.moves.push(mDownRight);

        mDownRight += sqrPieceEvenOrOdd(mDownRight)[3];

    }

}

function sqrPieceEvenOrOdd(sqr){

*if*(EVEN\_RANK\_SQR.includes(sqr))

*return* [-6, -7, 5, 6];

*return*[-5, -6, 6, 7];

}

function checkCaptureMoves(m, move, player, tmove, bf){

*if*(bf.board[move] != PIECE\_TYPE.NO\_PIECE && player != bf.board[move] && !inOffset(move)){

        let moveTemp = moveCount(player, move);

        let sPieceType = player == PLAYER.P1 ? PIECE\_TYPE.SUPER\_RED : PIECE\_TYPE.SUPER\_YELLOW;

*if*(bf.board[move] == sPieceType)

*return* false;

*if*(bf.board[move + moveTemp[tmove]] == PIECE\_TYPE.NO\_PIECE  && !inOffset(move + moveTemp[tmove])){

            m.captures.push({to: move + moveTemp[tmove], remove: move});

*return* true;

        }

    }

*return* false;

}

function moveCount(player, sqr){

*if*(player == PLAYER.P1){

*if*(EVEN\_RANK\_SQR.includes(sqr)){

*return* [5, 6, -6, -7];

        }*else* *if*(ODD\_RANK\_SQR.includes(sqr)){

*return* [6, 7, -5, -6];

        }

    }

*else* *if*(player == PLAYER.P2)

    {

*if*(EVEN\_RANK\_SQR.includes(sqr)){

*return* [-6, -7, 5, 6];

        }*else* *if*(ODD\_RANK\_SQR.includes(sqr)){

*return* [-5, -6, 6, 7];

        }

    }

*return* [0,0];

}

function inOffset(pos){

*return* OFF\_BOARD.includes(pos);

}

function hasPieceAt(pos, index){

*return* pos.includes(index);

}

function printBoard(){

    let s = "";

    s += "<center>";

    let x = 0;

    let y = false;

*for*(let i=0; i<sqr48.length\*2; i++){

*if*(i % 8 == 0)

        {

            s += "<br>";

            y = !y;

        }

*if*(i % 2 == 0){

*if*(y) s += "--";

            s += ""+BOARD\_DEF.board[sqr48[x]];

*if*(!y) s += "--";

            x++;

        }

    }

   s += "</center>";

*return* s;

}

ФайлPiece.js:

class Piece{

    constructor(x, y, size, pType){

*this*.x = x;

*this*.y = y;

*this*.size = size;

*this*.pieceType = pType;

    }

    draw(context){

        context.drawImage(imgs[*this*.pieceType], *this*.x, *this*.y, *this*.size, *this*.size);

    }

    mouseMove(mp){

*this*.size = mp.size \* 1.5;

*this*.x = mp.x - *this*.size /2;

*this*.y = mp.y - *this*.size /2;

    }

    move(src, target, tiles){

*return* movePiece(src, target, BOARD\_DEF);

    }

}

ФайлTile.js:

class Tile{

    constructor(x, y, size, c, sqr, p=null){

*this*.x = x;

*this*.y = y;

*this*.size = size;

*this*.sqr = sqr;

*this*.piece = p;

*if*(c)

*this*.color = 'black';

*else*

*this*.color = 'white';

    }

    draw(context){

        context.fillStyle = *this*.color;

        context.fillRect(*this*.x, *this*.y, *this*.size, *this*.size);

*if*(DEBUG){

*if*(*this*.color == 'black'){

            context.fillStyle = "blue";

            context.font = "30px Arial";

            context.fillText(*this*.sqr, *this*.x, *this*.y);

            }

        }

    }

    drawPiece(context){

*if*(*this*.piece != null){

*this*.piece.draw(context);

        }

    }

}

Файлai.js:

var AI = PLAYER.P1;

let AI\_LVL = 7;

function aiMove(bf)

{

*if* (bf.availableMoves.length == 0 || bf.move != AI) *return*;

    let bb = tempBfClone(bf);

    let depth = AI\_LVL \* 2;

    let move = minmax(bb, depth, -Infinity, Infinity, AI)[0];

    let verdict;

*if* (move.capture) {

        verdict = movePiece(move.src, move.target.to, bf);

    } *else* {

        verdict = movePiece(move.src, move.target, bf);

    }

    let mpiece = null;

    let index = null;

*for* (let i = 0; i < tiles.length; i++) {

*if* (tiles[i].sqr == move.src) {

            mpiece = tiles[i];

        }

*if* (move.capture) {

*if* (tiles[i].sqr == move.target.to) index = i;

        } *else* {

*if* (tiles[i].sqr == move.target) index = i;

        }

    }

    updatePiecesPos(mpiece, verdict, index, bf);

}

function minmax(bf, depth, alpha, beta, turn) {

*if* (depth <= 0 || bf.availableMoves <= 0) *return* evaluatePosition(bf);

*if* (turn == AI) {

        let bestScore = -Infinity;

        let bestMove = null;

        let tempBf = tempBfClone(bf);

*for* (let pieceMove of tempBf.availableMoves) {

            let moves;

            let isCapture = false;

*if* (pieceMove.moves.length > 0) {

                moves = pieceMove.moves;

            } *else* {

                moves = pieceMove.captures;

                isCapture = true;

            }

*for* (let move of moves) {

                createNewNode(tempBf, pieceMove.piece, move, isCapture);

                let value = minmax(tempBf, depth - 1, alpha, beta, tempBf.move)[1];

*if* (value > bestScore)

                {

                    bestScore = value;

                    bestMove = { src: pieceMove.piece, target: move, capture: isCapture };

                }

                alpha = Math.max(alpha, value);

                tempBf = tempBfClone(bf);

*if* (alpha >= beta) *break*;

            }

*if* (alpha >= beta) *break*;

        }

*return* [bestMove, bestScore];

    } *else* {

        let bestScore = Infinity;

        let bestMove = null;

        let tempBf = tempBfClone(bf);

*for* (let pieceMove of tempBf.availableMoves) {

            let moves;

            let isCapture = false;

*if* (pieceMove.moves.length > 0) {

                moves = pieceMove.moves;

            } *else* {

                moves = pieceMove.captures;

                isCapture = true;

            }

*for* (let move of moves) {

                createNewNode(tempBf, pieceMove.piece, move, isCapture);

                let value = minmax(tempBf, depth - 1, alpha, beta, tempBf.move)[1];

*if* (value < bestScore) {

                    bestScore = value;

                    bestMove = { src: pieceMove.piece, target: move, capture: isCapture };

                }

                beta = Math.min(beta, value);

                tempBf = tempBfClone(bf);

*if* (alpha >= beta) *break*;

            }

*if* (alpha >= beta) *break*;

        }

*return* [bestMove, bestScore];

    }

}

function createNewNode(bf, src, target, isCapture) {

*if* (isCapture) {

        movePiece(src, target.to, bf);

    } *else* {

        movePiece(src, target, bf);

    }

}

function evaluatePosition(bf) {

    let rPieceSum = 0;

    let yPieceSum = 0;

*for* (let p of bf.rPieces) {

*if* (bf.board[p] == PIECE\_TYPE.SUPER\_RED) {

            rPieceSum += 3;

        } *else* *if* (bf.board[p] == PLAYER.P1) {

            rPieceSum++;

        }

    }

*for* (let p of bf.yPieces) {

*if* (bf.board[p] == PIECE\_TYPE.SUPER\_YELLOW) {

            yPieceSum += 3;

        } *else* *if* (bf.board[p] == PLAYER.P2) {

            yPieceSum++;

        }

    }

    let eval = rPieceSum - yPieceSum;

*return* [null, eval];

}

function tempBfClone(bf) {

    let retBoard = {};

    retBoard.board = new Array(BOARD\_SIZE);

    retBoard.yPieces = [];

    retBoard.rPieces = [];

    retBoard.availableMoves = [];

    retBoard.move = bf.move;

*for* (let i = 0; i < BOARD\_SIZE; i++) {

        retBoard.board[i] = bf.board[i];

    }

*for* (let i = 0; i < bf.rPieces.length; i++) {

        retBoard.rPieces[i] = bf.rPieces[i];

    }

*for* (let i = 0; i < bf.yPieces.length; i++) {

        retBoard.yPieces[i] = bf.yPieces[i];

    }

*for* (let i = 0; i < bf.availableMoves.length; i++) {

        retBoard.availableMoves[i] = bf.availableMoves[i];

    }

*return* retBoard;

}