**CHESSBOARD GAME**



A project report submitted to

Rajiv Gandhi Proudyogiki Vishwavidhyalaya, Bhopal

towards partial fulfillment of

the degree of

**MASTER OF COMPUTER APPLICATION**

**{year 2020-21}**

**Submitted by:**

Moonis Ali

0801CA191017

**Guided By:**

Mr Upendra Singh

*Department of Information*

*Technology*

Department of Information Technology

**SHRI G.S. INSTITUTE OF TECHNOLOGY AND SCIENCE**

**INDORE (M.P.)**



**Recommendation**

The project report entitled **“*ChessBoard Game*”** submitted by **Moonis Ali** student of MCA Second year in the session 2020-21, towards partial fulfillment of the degree of **Master of Computer Applications** of Rajiv Gandhi Proudyogiki Vishwavidhyalaya, Bhopal, is a satisfactory account of his work and is recommended for the award of degree**.**

**Upendra Singh**

Project Guide

Department of IT

**Head**

Department of IT

**Dean(Academics)**

**S.G.S.I.T.S.,Indore**

**SHRI G.S. INSTITUTE OF TECHNOLOGY AND SCIENCE**

**INDORE (M.P.)**



**Certificate**

The project report entitled **“*ChessBoard Game*”** submitted by **Moonis Ali** student of MCA second year in the session 2020-21, towards partial fulfillment of the degree of **Master of Computer Applications** of Rajiv Gandhi Proudyogiki Vishwavidhyalaya, Bhopal, is a satisfactory account of their work and is approved for award of the degree.

**Internal Examiner External Examiner**

**Date**

**Acknowledgement**

I am heartily pleased to acknowledge all those people who have helped me in the successful completion of this project. With great pleasure i express my heartfelt gratitude to our esteemed guide, **Mr. Upendra Singh** Lecturer Department of information Technology, S.G.S.I.T.S. Indore. His persistent encouragement, perpetual motivation, everlasting patience and valuable technical inputs in discussions have enabled the successful completion of this project. His invaluable help, advice and constant encouragement helped us a lot and provide impetus to the progress of the project. We extend our profound indebtedness to the Head of the department **Ms. Sunita Varma,** the word loose their worth for her invaluable guidance, continuous encouragement and cooperation in every respect.

I sincerely wish to express our gratitude to all the members of staff of M.C.A. who have extended their cooperation at all times and have contributed in their own way in developing the project. Successful completion of a project is not an individual effort. It is an outcome of the cumulative effort of a number of persons, each having his own importance to the objective. We are thankful to our parents for being a constant source of encouragement in all our endeavors. Indeed it is their support that helps us through the ups and downs of life. The support and suggestion of our friends are worth appreciation and thankfulness. *A blend of gratitude, pleasure, great satisfaction and indebtedness is what, we feel to convey to all those who have directly or indirectly contributed to the successful completion of our project work.*

**Moonis Ali**

**Abstract**

In this project, an attempt is made to make a chessboard game for the players who want to compete other players to make sure that there game should reach to a height where the players can even compete in national or international championship. It is design specifically for the ones who wants to improve their game since now a days many people are busy because of their day-to-day chores and doesn’t use a board to play the chess as well as today’s generation has become a tech nerd, so this project can help them to reduce time normally used by a board game and increase efficiency in a player game where player can play multiple games one after other.

The main aim of this project is to computerize the classic chessboard game with same qualities of a board game.

**Introduction**

* 1. **Objective**

The aim of this project is to implement a program which can help a player to heighten his skills and save time and unnecessary accessories which is normally used by classic board game.

* 1. **Scope**

The software can be used to perform all activities which are used in classic chessboard game with same aspects of board game.

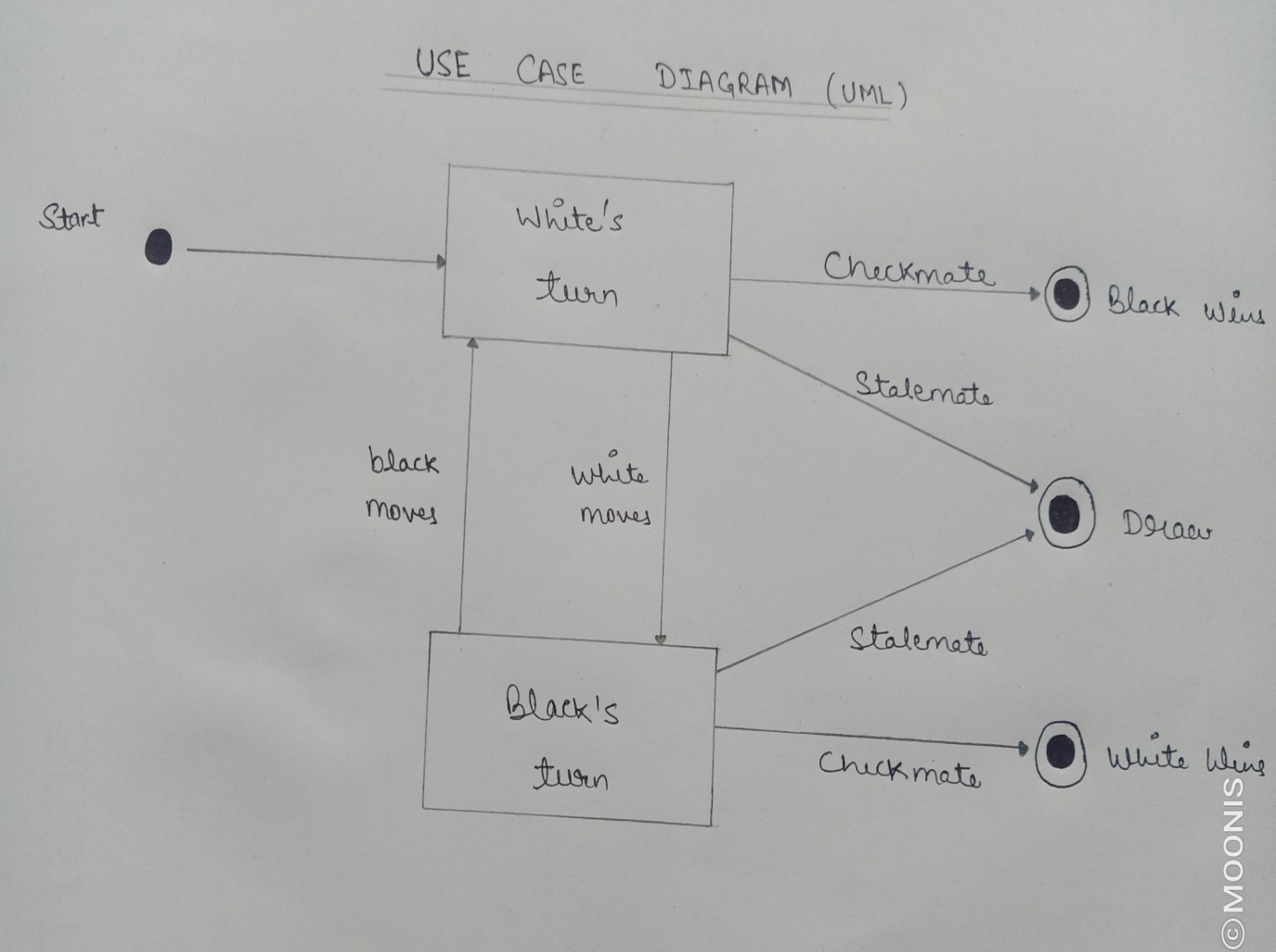
The benefits of this software are it is easy to use and less time and money consuming.

* 1. **Technologies Used**

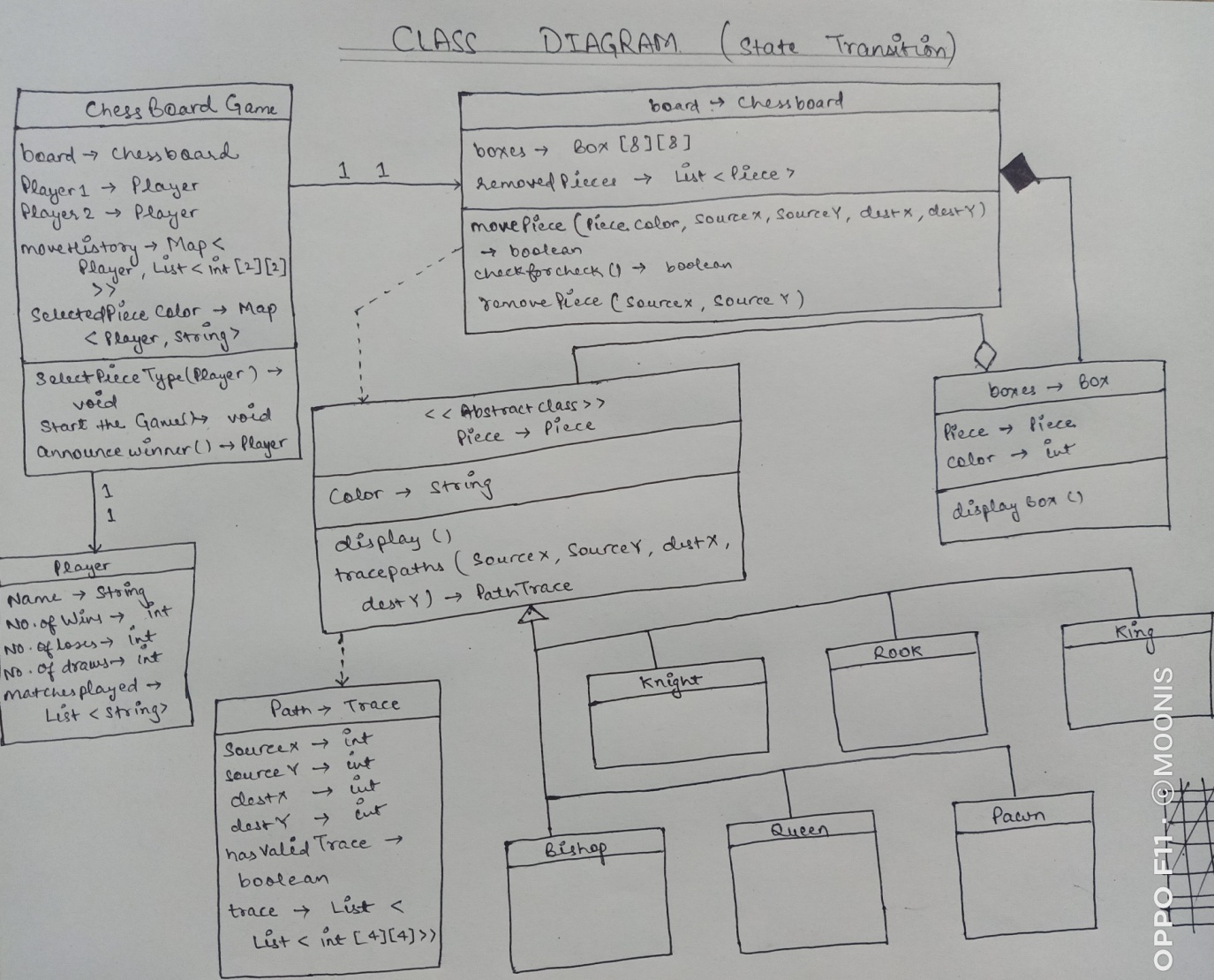
1. Java

**Analysis Diagram**

**Use Case Diagram**

****

**Class Diagram**

****

**Coding**

CHESSBOARD.JAVA -

import java.util.\*;  
  
public class ChessBoard {  
 public byte[][] board = new byte[8][8];  
 public int previousDoublePush;  
 public int turn;  
 public boolean[] kingside = new boolean[2], queenside = new boolean[2];  
 public boolean[] hascastled = new boolean[2];  
  
 public static final int *KNIGHT* = 0;  
 public static final int *BISHOP* = 1;  
 public static final int *ROOK* = 2;  
 public static final int *QUEEN* = 3;  
 public static final int *KING* = 4;  
 public static final int *PAWN* = 5;  
 public static final int *EMPTY* = 6;  
 public static final int *CBIT* = 8;  
  
 public static final int *BLACK* = 0;  
 public static final int *WHITE* = 1;  
  
 public static int colorof(byte x) {  
 return (x & *CBIT*)>>3;  
 }  
  
 public static int pieceof(byte x) {  
 return x & (~*CBIT*);  
 }  
   
 public static boolean isempty(byte x) {  
 return (x & (~*CBIT*)) == *EMPTY*;  
 }  
  
 public static byte makesquare(int p, int c) {  
 return (byte) (p | (c<<3));  
 }  
  
  
 private static class Pair {  
 int x, y;  
 Pair(int xx, int yy) {  
 x = xx;  
 y = yy;  
 }  
 }  
  
 private static class PieceDesc {  
 boolean iterate;  
 Pair[] delta;  
 PieceDesc(boolean iter, int[] deltas) {  
 iterate = iter;  
 delta = new Pair[deltas.length/2];  
 for (int i=0; i<deltas.length; i+=2) {  
 delta[i/2] = new Pair(deltas[i], deltas[i+1]);  
 }  
 }  
 }  
  
 public class SquareDesc {  
 public SquareDesc() { }  
 public SquareDesc(int t, int c, int ix, int iy) { type=t; color=c; x=ix; y=iy; }  
  
 public int type,color,x,y;  
 }  
  
 private static PieceDesc[] *pdesc* = new PieceDesc[5];  
 static {  
 *pdesc*[*KNIGHT*] = new PieceDesc(false, new int[]  
 { 2, 1 , 1, 2 , 2, -1 , 1, -2 , -2, 1 , -1,2 , -2,-1 , -1,-2 });  
   
 *pdesc*[*BISHOP*] = new PieceDesc(true, new int[]  
 { 1, 1 , 1,-1 , -1, -1 , -1,1 });  
   
 *pdesc*[*ROOK*] = new PieceDesc(true, new int[]  
 { 1, 0 , 0, 1 , -1,0 , 0,-1 });  
   
 *pdesc*[*QUEEN*] = new PieceDesc(true, new int[]  
 { 1, 0 , 0, 1 , -1,0 , 0,-1 , 1, 1 , 1,-1 , -1, -1 , -1,1 });  
   
 *pdesc*[*KING*] = new PieceDesc(false, new int[]  
 { 1, 0 , 0, 1 , -1,0 , 0,-1 , 1, 1 , 1,-1 , -1, -1 , -1,1 });  
 }  
  
 private static String *piecestr* = "nbrqkp-XNBRQKPX";  
  
 */\*\* Prints an ascii picture of a chess board. p=pawn, n=knight  
 b=bishop, r=rook, q=queen, k=king. Lower case letters are the  
 black pieces, upper case letters are the white. \*/* public String toString() {  
 String s = "";  
 s += " a b c d e f g h\n";  
 s += " +---------------+\n";  
 for (int y=7; y>=0; y--) {  
 s += (y+1) + " |";  
 for (int x=0; x<8; x++) {  
 s += *piecestr*.charAt(board[x][y]);  
 if (x<7) s += " ";  
 }  
 s += "| " + (y+1);  
 s += "\n";  
 }  
 s += " +---------------+\n";  
 s += " a b c d e f g h\n";  
 return s;  
 }  
  
 ChessBoard() {  
 turn = *WHITE*;  
 for (int x=0; x<8; x++) {  
 for (int y=0; y<8; y++) {  
 board[x][y] = *EMPTY*;  
 }  
 }  
 previousDoublePush = -1;  
 for (int color = 0; color < 2; color++) {  
 kingside[color] = queenside[color] = true;  
 hascastled[color] = false;  
 }  
 board[0][0] = *makesquare*(*ROOK*, *WHITE*);  
 board[1][0] = *makesquare*(*KNIGHT*, *WHITE*);  
 board[2][0] = *makesquare*(*BISHOP*, *WHITE*);  
 board[3][0] = *makesquare*(*QUEEN*, *WHITE*);  
 board[4][0] = *makesquare*(*KING*, *WHITE*);  
 board[5][0] = *makesquare*(*BISHOP*, *WHITE*);  
 board[6][0] = *makesquare*(*KNIGHT*, *WHITE*);  
 board[7][0] = *makesquare*(*ROOK*, *WHITE*);  
 for (int x=0; x<8; x++) {  
 board[x][7] = *makesquare*(*pieceof*(board[x][0]), *BLACK*);  
 board[x][1] = *makesquare*(*PAWN*, *WHITE*);  
 board[x][6] = *makesquare*(*PAWN*, *BLACK*);  
 }  
 }  
  
 ChessBoard(ChessBoard b) {  
 for (int x=0; x<8; x++) {  
 for (int y=0; y<8; y++) {  
 board[x][y] = b.board[x][y];  
 }  
 }  
  
 previousDoublePush = b.previousDoublePush;  
 turn = b.turn;  
 for (int i=0; i<2; i++) {  
 kingside[i] = b.kingside[i];  
 queenside[i] = b.queenside[i];  
 hascastled[i] = b.hascastled[i];  
 }  
 }  
  
 private void makeCastlingMove(Move m) {  
 board[m.destx][m.desty] = board[m.srcx][m.srcy];  
 board[m.srcx][m.srcy] = *EMPTY*;  
 if (m.destx == 6) { *// Kingside* board[5][m.desty] = board[7][m.srcy];  
 board[7][m.srcy] = *EMPTY*;   
 }  
  
 if (m.destx == 2) { *// Queenside* board[3][m.desty] = board[0][m.srcy];  
 board[0][m.srcy] = *EMPTY*;   
 }  
 kingside[turn] = queenside[turn] = false;  
 hascastled[turn] = true;  
  
 turn = 1-turn;  
 }  
  
 public SquareDesc[] getPieces(int pturn) {  
  
 LinkedList pieces = new LinkedList();  
  
 for (int x=0; x<8; x++) {  
 for (int y=0; y<8; y++) {  
 if (*isempty*(board[x][y])) continue;  
 if (*colorof*(board[x][y]) != pturn) continue;  
 pieces.addFirst(new SquareDesc(*pieceof*(board[x][y]), *colorof*(board[x][y]),x,y));  
 }  
 }  
  
 return (SquareDesc[]) pieces.toArray(new SquareDesc[0]);  
 }  
  
 public void makeMove(Move m) {  
  
 if ((*pieceof*(board[m.srcx][m.srcy]) == *KING*) &&   
 ((m.srcx-m.destx >= 2) || (m.srcx-m.destx <= -2))) {  
 makeCastlingMove(m);  
 return;  
 }  
   
 if ((*pieceof*(board[m.srcx][m.srcy]) == *PAWN*) &&   
 (*isempty*(board[m.destx][m.desty])) &&  
 (m.srcx != m.destx)) {  
 board[m.destx][m.srcy] = *EMPTY*;  
 }  
  
 if ((*pieceof*(board[m.srcx][m.srcy]) == *PAWN*) &&   
 (m.desty != m.srcy +1) && (m.desty != m.srcy -1)) {  
 previousDoublePush = m.srcx;  
 } else {  
 previousDoublePush = -1;  
 }  
  
 if (*pieceof*(board[m.srcx][m.srcy]) == *KING*) {  
 kingside[turn] = queenside[turn] = false;  
 }  
  
 int castleRank = (turn == *WHITE*)? 0: 7;  
 if ((m.destx == 0) && (m.desty == castleRank)) queenside[turn] = false;  
 if ((m.destx == 7) && (m.desty == castleRank)) kingside[turn] = false;  
   
 board[m.destx][m.desty] = board[m.srcx][m.srcy];  
 board[m.srcx][m.srcy] = *EMPTY*;  
   
 if ((*pieceof*(board[m.destx][m.desty]) == *PAWN*) &&   
 (m.desty == 7 || m.desty == 0)) {  
 */\* promote to a queen \*/* board[m.destx][m.desty] = *makesquare*(*QUEEN*, turn);  
 }  
 turn = 1-turn;  
 }  
  
 public int getTurn() {  
 return turn;  
 }  
  
 public SquareDesc getSquare(int x, int y) {  
 if (*isempty*(board[x][y]))  
 return new SquareDesc(*EMPTY*, *EMPTY*, x, y);  
 else  
 return new SquareDesc(*pieceof*(board[x][y]), *colorof*(board[x][y]), x , y);  
 }  
  
 private boolean pawnCheck(int x, int y) {  
 int dy, dx;  
 byte enemyking = *makesquare*(*KING*, turn);  
 if (1-turn == *WHITE*) dy = 1; else dy = -1;  
 if (y+dy >= 8 || y+dy < 0) return false;  
 for (dx = -1; dx <= 1; dx += 2) {  
 if (x+dx < 0 || x+dx >= 8) continue;  
 if (board[x+dx][y+dy] == enemyking) return true;  
 }  
 return false;  
 }  
  
 private boolean pieceCheck(int x, int y) {  
 int cx, cy;  
 boolean stopnow;  
 byte enemyking = *makesquare*(*KING*, turn);  
 int piece = *pieceof*(board[x][y]);  
 for (int dir = 0; dir < *pdesc*[piece].delta.length; dir++) {  
 cx = x;  
 cy = y;  
 stopnow = !*pdesc*[piece].iterate;  
 while (true) {  
 cx += *pdesc*[piece].delta[dir].x;  
 cy += *pdesc*[piece].delta[dir].y;  
 if (cx < 0 || cy < 0 || cx >=8 || cy >= 8) break;  
 if (!*isempty*(board[cx][cy])) {  
 if (*colorof*(board[cx][cy]) == 1-turn) break;  
 stopnow = true;  
 }  
 if (board[cx][cy] == enemyking) return true;  
 if (stopnow) break;  
 }  
 }  
 return false;  
 }  
  
 public boolean inCheck() {  
 for (int x=0; x<8; x++) {  
 for (int y=0; y<8; y++) {  
 if (*isempty*(board[x][y])) continue;  
 if (*colorof*(board[x][y]) != 1-turn) continue;  
 if (*pieceof*(board[x][y]) == *PAWN*) {  
 if (pawnCheck(x, y)) return true;  
 } else {  
 if (pieceCheck(x, y)) return true;  
 }  
 }  
 }  
 return false;  
 }  
  
 private void generatePawnMoves(LinkedList moveList, int x, int y) {  
 int dy, dx, ex, ey;  
 ChessBoard newp;  
   
 if (turn == *WHITE*) dy = 1; else dy = -1;  
 if (y+dy >= 8 || y+dy < 0) return;  
 */\* push 1 \*/* if (*isempty*(board[x][y+dy])) {  
 newp = new ChessBoard(this);  
 newp.board[x][y+dy] = newp.board[x][y];  
 newp.board[x][y] = *EMPTY*;  
 if (!newp.inCheck()) moveList.addFirst(new Move(x, y, x, y+dy, false));  
 */\* push 2 \*/* if (((turn == *WHITE* && y == 1) || (turn == *BLACK* && y == 6)) && *isempty*(board[x][y+2\*dy])) {  
 newp.board[x][y+2\*dy] = newp.board[x][y+dy];  
 newp.board[x][y+dy] = *EMPTY*;   
 if (!newp.inCheck()) moveList.addFirst(new Move(x, y, x, y+2\*dy, false));  
 }  
 }  
  
 for (dx = -1; dx <= 1; dx += 2) {  
 if (x+dx < 0 || x+dx >= 8) continue;  
 if (!*isempty*(board[x+dx][y+dy])) {  
 if (*colorof*(board[x+dx][y+dy]) == turn) continue;  
  
 ex = x;  
 ey = y;  
 } else {  
 if (x+dx != previousDoublePush) continue;  
 if (!((turn == *WHITE* && y == 4) || (turn == *BLACK* && y == 3))) continue;  
 ex = x+dx;  
 ey = y;  
 }  
 newp = new ChessBoard(this);  
 newp.board[x+dx][y+dy] = newp.board[x][y];  
 newp.board[x][y] = *EMPTY*;  
 newp.board[ex][ey] = *EMPTY*;  
 if (!newp.inCheck()) moveList.addFirst(new Move(x, y, x+dx, y+dy, true));  
 }  
 }  
  
 private void generatePieceMoves(LinkedList moveList, int x, int y) {  
 int cx, cy, dir;  
 boolean stopnow, capture;  
 ChessBoard newp;  
 int piece = *pieceof*(board[x][y]);  
 stopnow = !*pdesc*[piece].iterate;  
 for (dir = 0; dir < *pdesc*[piece].delta.length; dir++) {  
 cx = x;  
 cy = y;  
 capture = false;  
 while (true) {  
 cx += *pdesc*[piece].delta[dir].x;  
 cy += *pdesc*[piece].delta[dir].y;  
 */\* (cx, cy) is the current position \*/* if (cx < 0 || cy < 0 || cx >=8 || cy >= 8) break;  
 if (!*isempty*(board[cx][cy])) {  
 if (*colorof*(board[cx][cy]) == turn) break;  
 */\* hit an enemy piece \*/* capture = true;  
 }  
  
 newp = new ChessBoard(this);  
 newp.board[cx][cy] = newp.board[x][y];  
 newp.board[x][y] = *EMPTY*;  
 if (!newp.inCheck()) moveList.addFirst(new Move(x, y, cx, cy, capture));  
 if (stopnow || capture) break;  
 }  
 }  
 }  
  
 private void generateQueenCastling(int castleRank, LinkedList moveList) {  
 if (board[0][castleRank] != *makesquare*(*ROOK*, turn)) return;  
 if (!*isempty*(board[3][castleRank])) return;  
 if (!*isempty*(board[2][castleRank])) return;  
 if (!*isempty*(board[1][castleRank])) return;  
  
 ChessBoard newp = new ChessBoard(this);  
 if (newp.inCheck()) return;  
 newp.board[3][castleRank] = newp.board[4][castleRank];  
 newp.board[4][castleRank] = *EMPTY*;  
 if (newp.inCheck()) return;  
 newp.board[2][castleRank] = newp.board[3][castleRank];  
 newp.board[3][castleRank] = *EMPTY*;  
 if (newp.inCheck()) return;  
 moveList.addFirst(new Move(4, castleRank, 2, castleRank, false));  
 }  
  
 private void generateKingCastling(int castleRank, LinkedList moveList) {  
 if (board[7][castleRank] != *makesquare*(*ROOK*, turn)) return;  
 if (!*isempty*(board[5][castleRank])) return;  
 if (!*isempty*(board[6][castleRank])) return;  
  
 ChessBoard newp = new ChessBoard(this);  
 if (newp.inCheck()) return;  
 newp.board[5][castleRank] = newp.board[4][castleRank];  
 newp.board[4][castleRank] = *EMPTY*;  
 if (newp.inCheck()) return;  
 newp.board[6][castleRank] = newp.board[5][castleRank];  
 newp.board[5][castleRank] = *EMPTY*;  
 if (newp.inCheck()) return;  
 moveList.addFirst(new Move(4, castleRank, 6, castleRank, false));  
 }  
  
 private void generateCastlingMoves(LinkedList moveList) {  
 int castleRank = (turn == *WHITE*)? 0: 7;  
 if (board[4][castleRank] != *makesquare*(*KING*, turn)) return;  
 if (kingside[turn]) generateKingCastling(castleRank, moveList);  
 if (queenside[turn]) generateQueenCastling(castleRank, moveList);  
 }  
  
 public List generateMoves() {  
 LinkedList moveList = new LinkedList();  
  
 for (int x=0; x<8; x++) {  
 for (int y=0; y<8; y++) {  
 if (*isempty*(board[x][y])) continue;  
 if (*colorof*(board[x][y]) != turn) continue;  
 if (*pieceof*(board[x][y]) == *PAWN*) {  
 generatePawnMoves(moveList, x, y);  
 } else {  
 generatePieceMoves(moveList, x, y);  
 }  
 }  
 }  
 generateCastlingMoves(moveList);  
 return moveList;  
 }  
}

ENGINE.JAVA –

import java.util.\*;  
  
public class Engine {  
  
 private ChessBoard board;  
 private long startTime;  
 private long allocated;  
 private int initialTime, increment;  
  
 private static final int *WHITE* = ChessBoard.*WHITE*;  
 private static final int *BLACK* = ChessBoard.*BLACK*;  
 private static final int *INFINITY*=1000000, *MATE* = 300000;  
  
 private static final String *programName* = "Put Your Name Here";  
  
 public String getName() {  
 return *programName*;  
 }  
  
 public void newGame(int time, int inc) {  
 initialTime = time;  
 increment = inc;  
 board = new ChessBoard();  
 }  
  
 public void applyMove(Move m) {  
 board.makeMove(m);  
 }  
  
 public ChessBoard getBoard() {  
 return board;  
 }  
  
 public Move computeMove(int timeleft, int optime) {  
  
 Move move;  
 startTime = System.*currentTimeMillis*();  
 allocated = allocateTime(timeleft, optime);  
 System.*out*.println("ALLOCATED: " + allocated + "ms");  
 return search();  
 }  
  
 private int allocateTime(int timeleft, int optime) {  
 double t = increment + timeleft/30.0;  
 if (t > timeleft) t = .9\*timeleft;  
 return (int) t;  
 }  
  
 private boolean timeup() {  
  
 if ((System.*currentTimeMillis*()-startTime) > allocated) {  
 return true;  
 }  
 return false;  
 }  
  
 private Move search() {  
 return randomMove();  
 }  
  
 private Move randomMove() {  
 Random rand = new Random();  
 List moveList = board.generateMoves();  
 if (moveList.size() == 0) return null;  
  
 Move [] moveArray = (Move[]) moveList.toArray(new Move[0]);  
 Move [] captureArray = new Move[moveArray.length];  
 int capCount = 0;  
 for (int i=0; i<moveArray.length; i++) if (moveArray[i].capture) captureArray[capCount++] = moveArray[i];  
 if (capCount > 0) return captureArray[rand.nextInt(capCount)];  
 return moveArray[rand.nextInt(moveArray.length)];  
 }  
  
 private int alpha\_beta(ChessBoard board, int depth, int alpha, int beta) {  
 return 0;  
 }  
}

EVALUATOR.JAVA –

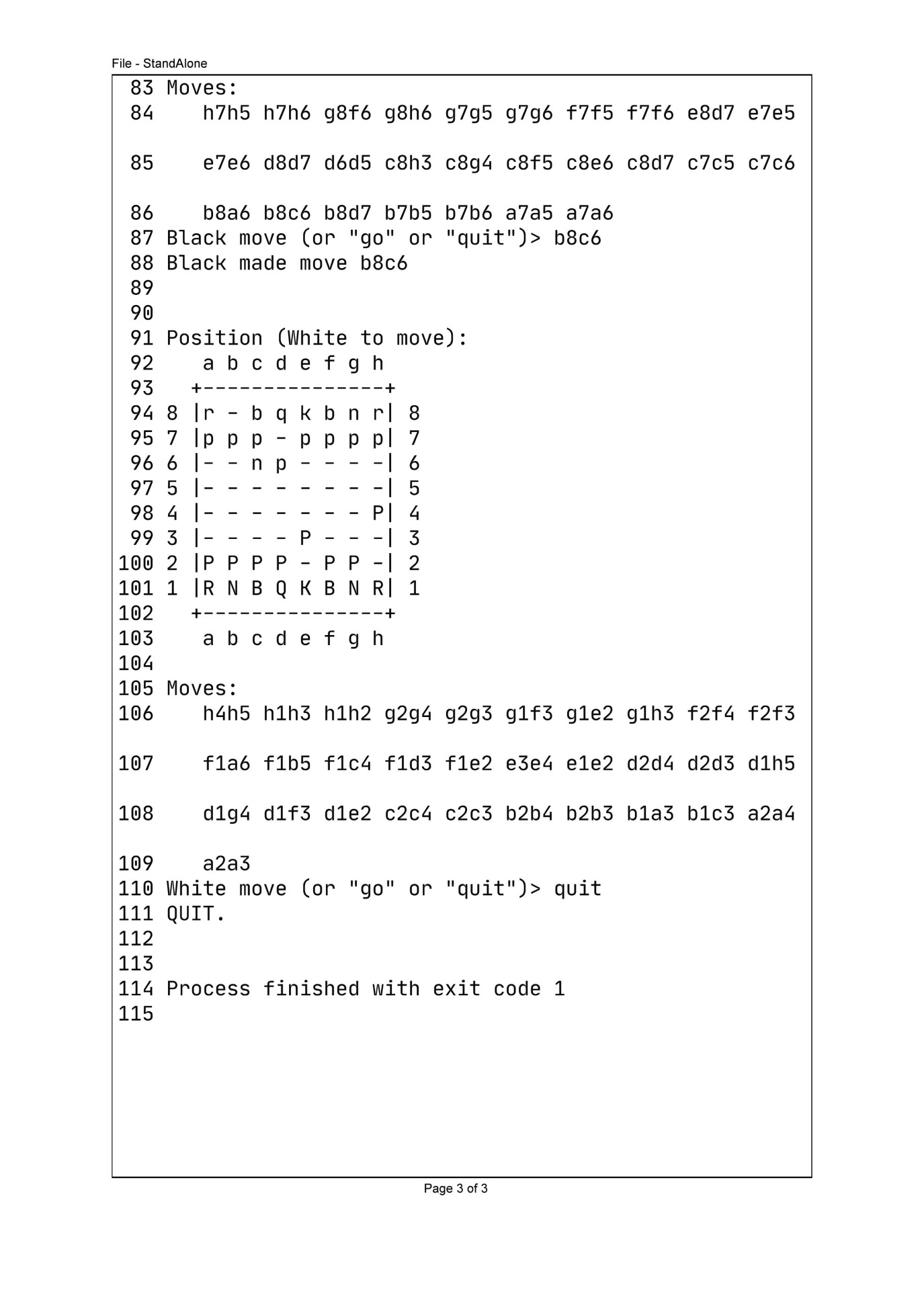
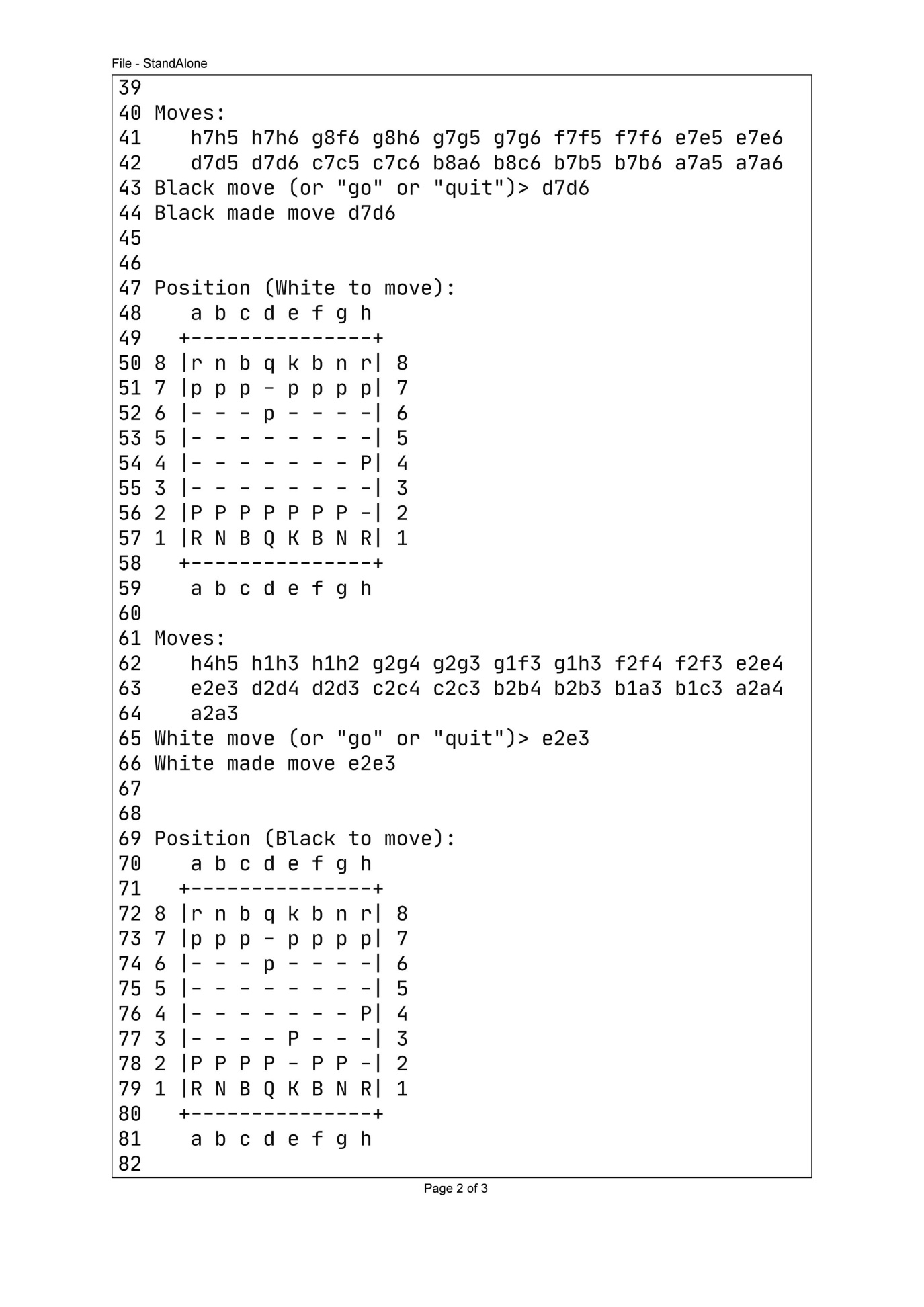
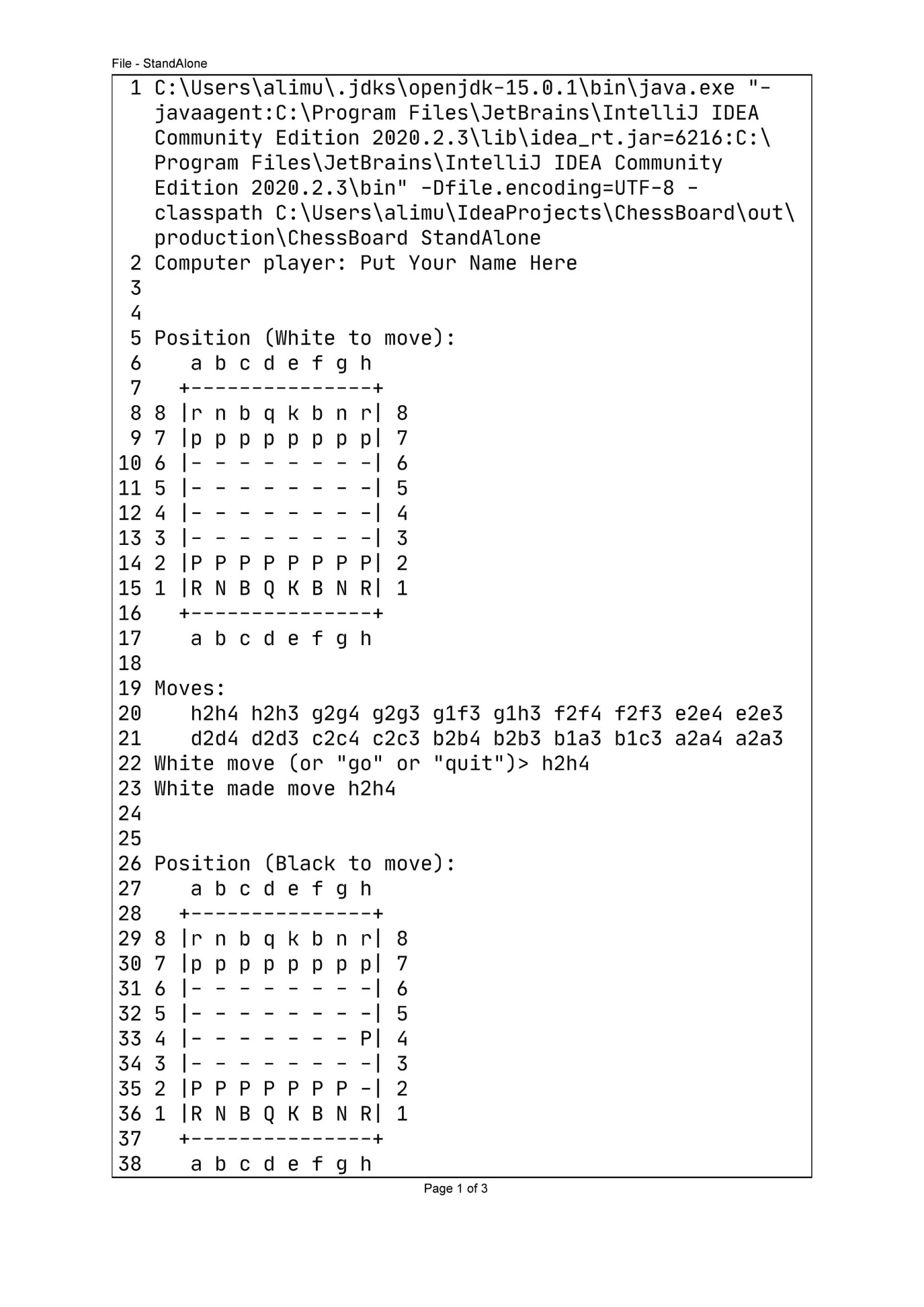
public class Evaluator {  
  
 public static int eval(ChessBoard board) {  
 return 0;  
 }  
  
 private static int[] *pieceValue* = new int[7];  
 static {  
 *pieceValue*[ChessBoard.*PAWN*] = 100;  
 *pieceValue*[ChessBoard.*KNIGHT*] = 300;  
 *pieceValue*[ChessBoard.*BISHOP*] = 300;  
 *pieceValue*[ChessBoard.*ROOK*] = 500;  
 *pieceValue*[ChessBoard.*QUEEN*] = 900;  
 *pieceValue*[ChessBoard.*KING*] = 0;  
 *pieceValue*[ChessBoard.*EMPTY*] = 0;  
 }  
   
 private static int *bishoppos*[] = {-5,-5,-5,-5,-5,-5,-5,-5,  
 -5,10,5,8,8,5,10,-5,  
 -5,5,3,8,8,3,5,-5,  
 -5,3,10,3,3,10,3,-5,  
 -5,3,10,3,3,10,3,-5,  
 -5,5,3,8,8,3,5,-5,  
 -5,10,5,8,8,5,10,-5,  
 -5,-5,-5,-5,-5,-5,-5,-5};  
   
 private static int *knightpos*[] = {-10,-5,-5,-5,-5,-5,-5,-10,  
 -8,0,0,3,3,0,0,-8,  
 -8,0,10,8,8,10,0,-8,  
 -8,0,8,10,10,8,0,-8,  
 -8,0,8,10,10,8,0,-8,  
 -8,0,10,8,8,10,0,-8,  
 -8,0,0,3,3,0,0,-8,  
 -10,-5,-5,-5,-5,-5,-5,-10};  
   
 private static int *pawnpos*[] = {0,0,0,0,0,0,0,0,  
 0,0,0,-5,-5,0,0,0,   
 0,2,3,4,4,3,2,0,  
 0,4,6,10,10,6,4,0,  
 0,6,9,10,10,9,6,0,  
 4,8,12,16,16,12,8,4,  
 5,10,15,20,20,15,10,5,  
 0,0,0,0,0,0,0,0};  
}

MOVE.JAVA –

public class Move {  
 int srcx, srcy, destx, desty;  
 boolean capture;  
  
 Move() {}  
 Move(int x0, int y0, int x1, int y1, boolean c) {  
 srcx = x0;  
 srcy = y0;  
 destx = x1;  
 desty = y1;  
 capture = c;  
 }  
   
 Move(String s) {  
 srcx = s.charAt(0) - 'a';  
 srcy = s.charAt(1) - '1';  
 destx = s.charAt(2) - 'a';  
 desty = s.charAt(3) - '1';  
 }  
  
 public void copyMove(Move m) {  
 srcx = m.srcx;  
 srcy = m.srcy;  
 destx = m.destx;  
 desty = m.desty;  
 capture = m.capture;   
 }  
  
 public boolean equals(Move m) {  
 return (m.srcx == srcx && m.srcy == srcy && m.destx == destx && m.desty == desty);  
 }  
   
 public String toString() {  
 return new String (new byte[] {  
 (byte) ('a'+srcx), (byte)('1'+srcy), (byte)('a'+destx), (byte)('1'+desty)});  
 }  
}

STANDALONE.JAVA –

import java.util.\*;  
import java.io.\*;  
  
public class StandAlone {  
   
 private static boolean *logfile* = false;  
   
public static void main(String[] args) throws IOException {  
 InputStreamReader stdin;   
 if( args.length >= 2 && args[0].equals("-l") ){  
 stdin = new FileReader(args[1]);  
 *logfile* = true;  
 System.*out*.println(" Reading command script from file '"+args[1]+"'...");  
 }  
 else{  
 stdin = new InputStreamReader(System.*in*);   
 }  
   
 boolean printMoves = true;  
 if( args.length >= 1 && args[0].equals("-m") ){  
 printMoves = false;  
 }  
 else if( args.length >= 3 && args[2].equals("-m")){  
 printMoves = false;  
 }  
  
 Move[] moveArray;  
 ChessBoard b;  
 String command, prompt;  
 Move m;  
 Engine player = new Engine();  
 System.*out*.println("Computer player: "+player.getName());  
   
 while(true) {  
 player.newGame(5\*60\*1000, 0);  
 while (true) {  
 b = player.getBoard();  
 if (b.getTurn() == ChessBoard.*WHITE*) prompt = "White"; else prompt = "Black";  
 System.*out*.println("\n\nPosition ("+prompt+" to move):\n"+b);  
 moveArray = (Move[]) b.generateMoves().toArray(new Move[0]);  
 if (moveArray.length == 0) {  
 if (b.inCheck()) System.*out*.println("Checkmate");  
 else System.*out*.println("Stalemate");  
 break;  
 }  
   
 if(printMoves){  
 System.*out*.println("Moves:");  
 System.*out*.print(" ");  
 for (int i=0; i<moveArray.length; i++) {  
 if ((i % 10) == 0 && i>0) System.*out*.print("\n ");  
 System.*out*.print(moveArray[i]+" ");  
 }  
 }  
 System.*out*.println();  
 while(true) {  
 System.*out*.print(prompt + " move (or \"go\" or \"quit\")> ");  
 command = *readCommand*(stdin);  
 if (command.equals("go")) {  
 m = player.computeMove(1\*60\*1000, 0);  
 System.*out*.println("Computer Moves: " + m);  
 break;  
 } else if (command.equals("quit")) {  
 System.*out*.println("QUIT.\n");  
 System.*exit*(1);  
 } else {  
 m = null;  
 for (int i=0; i<moveArray.length; i++) {  
 if (command.equals(moveArray[i].toString())) {  
 m = moveArray[i];  
 break;  
 }  
 }  
 if (m != null) break;  
 System.*out*.println("\""+command+"\" is not a legal move");  
 }  
 }  
 player.applyMove(m);  
 System.*out*.println(prompt + " made move "+m);  
 }  
  
 while(true) {  
 System.*out*.print("Play again? (y/n):");  
 command = *readCommand*(stdin);  
 if (command.equals("n")) System.*exit*(1);  
 if (command.equals("y")) break;  
 }  
 }  
 }  
 static String readCommand(InputStreamReader stdin) throws IOException {  
 final int MAX = 100;  
 int len = 0;  
 char[] cbuf = new char[MAX];  
 for(int i=0; i<cbuf.length; i++){  
 if(*logfile* && !stdin.ready()) return "quit";  
   
 cbuf[i] = (char)stdin.read();  
 len++;  
 if(cbuf[i] == '\n')  
 break;  
 if(cbuf[i] == -1){  
 System.*out*.println("An error occurred reading input");  
 System.*exit*(1);  
 }  
 }  
 return new String(cbuf, 0, len).trim();  
 }  
}

**Output**

**References**

* [www.google.com](http://www.google.com)
* [www.geeksforgeeks.com](http://www.geeksforgeeks.com)
* [www.stackoverflow.com](http://www.stackoverflow.com)