Source Code

Shubh

199301305

ChessMain.java:

*// package mini\_project ;*

*import* java.util.\*;

public class ChessMain {

*// main function for testing*

    public static void main(String[] args) {

        System.out.println();

        System.out.println(" Hello! And welcome to Chess!"); *// these two lines are always shown when game starts*

        boolean mainMenu *=* true; *// true --> game is still playing*

*while* (mainMenu) {

*try* {

                System.out.println();

                System.out.println(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

                System.out.println("                 MAIN MENU             ");

                System.out.println(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

                System.out.println("            1. Play a new game.         ");

                System.out.println(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

                System.out.println("         2. Display the help menu.      ");

                System.out.println(" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

                Scanner userInput *=* *new* Scanner(System.in);

                String inputString;

                System.out.print(">>Please type in option number: ");

                inputString *=* userInput.nextLine();

*if* (inputString.charAt(0) *==* '1') { *// start a new game!*

                    System.out.println();

                    ChessBoard chessBoard *=* *new* ChessBoard();

                    Move mover *=* *new* Move(chessBoard);

                    HelpMenu help *=* *new* HelpMenu();

                    Player white *=* *new* Player(0);

                    System.out.println("Hello Player 1. Please input your desired user name: ");

                    white.setName(userInput.nextLine());

                    System.out.println();

                    Player black *=* *new* Player(1);

                    System.out.println("And hello Player 2. Please input your desired user name: ");

                    black.setName(userInput.nextLine());

                    System.out.println();

                    System.out.println("Thank you both very much.");

                    chessBoard.initBoard(); *// initiate the board, start game*

                    boolean turn *=* true; *// white starts the game*

                    String source, destin;

                    play*:* *while* (true) {

*if* (turn) { *//Whites Turn*

                            chessBoard.setTurn(0);

                            System.out.println("Type 'H' for help and to access specific commands.");

                            System.out.println();

                            System.out.println("Input current coordinates of the piece that you want to move.");

*if* (mover.checkCheck(white.getColor())) {

                                System.out

                                        .println(white.getName() *+* "(White), you are in check. Proceed with caution.");

                            } *else* {

                                System.out.println(white.getName() *+* "(White) it is your turn. Choose wisely.");

                            }

                            source *=* userInput.nextLine();

*if* (source.charAt(0) *==* 'H') { *// help menu!*

                                help.display();

*continue* play; *// if help just exited normally, restart player's turn*

                            }

                            System.out.println("Input coordinates of the destination space.");

                            destin *=* userInput.nextLine();

*if* (destin.charAt(0) *==* 'H') { *// help menu!*

                                help.display();

*continue* play; *// if help just exited normally, restart player's turn*

                            }

*try* {

*if*(*!*mover.move(source, destin, white)) *// check if move is valid*

                                    System.out.println("OOPS Illegal Move!!!! Try Again");

*else*

                                    turn *=* false;

                            } *catch* (NullPointerException e) {

                                System.out.println("\nOOPS Illegal Move!!!! Try Again");

                            }

                        }

*else* { *// black player's turn!*

                            chessBoard.setTurn(1);

                            System.out.println("Type 'H' for help and to access specific commands.");

                            System.out.println();

                            System.out.println("Input current coordinates of the piece that you want to move.");

*if* (mover.checkCheck(black.getColor())) {

                                System.out

                                        .println(black.getName() *+* "(Black), you are in check. Proceed with caution.");

                            } *else* {

                                System.out.println(black.getName() *+* "(Black) it is your turn. Choose wisely.");

                            }

                            source *=* userInput.nextLine();

*if* (source.charAt(0) *==* 'H') { *// help menu!*

                                help.display();

*continue* play; *// if help just exited normally, restart player's turn*

                            }

                            System.out.println("Input coordinates of the destination space.");

                            destin *=* userInput.nextLine();

*if* (destin.charAt(0) *==* 'H') { *// help menu!*

                                help.display();

*continue* play; *// if help just exited normally, restart player's turn*

                            }

*try* {

*if* (*!*mover.move(source, destin, black))

                                    System.out.println("OOPS Illegal Move!!!! Try Again");

*else*

                                    turn *=* true;

                            } *catch* (NullPointerException e) {

                                System.out.println("\nOOPS Illegal Move!!!! Try Again");

                            }

                        }

                    }

                } *else* *if* (inputString.charAt(0) *==* '2') { *// display help!*

                    HelpMenu help *=* *new* HelpMenu();

                    help.display();

                } *else* {

                    System.out.println("I did not recognize that command. Please try again.");

                }

            } *catch* (Exception e) {

                e.printStackTrace();

                System.out.println("OOPS Try again");

            }

        }

    }

}

ChessBoard.java:

*// package mini\_project ;*

*import* java.util.\*;

public class ChessBoard {

    private Piece[][] board *=* *new* Piece[8][8]; *// dynamic game board array*

    private ArrayList<Piece> pieces *=* *new* ArrayList<Piece>(32); *// dynamic ArrayList of pieces*

    private int turn; *// white = 0, black = 1*

*// show current game state, for viewing game in terminal*

    public void currentGameState() {

        System.out.println();

        System.out.println(" ------------------------------------");

*for* (int i *=* 0; i *<* 8; i*++*) {

            System.out.print(" |");

            System.out.print(8 *-* i *+* " |");

*for* (int j *=* 0; j *<* 8; j*++*) {

*if* (board[i][j] *==* null) { *// empty space*

                    System.out.print("   ");

                    System.out.print("|");

                } *else* { *// piece*

                    System.out.print(board[i][j].getPieceName());

                    System.out.print("|");

                }

            }

*if* (i *<* 8) { *// border between rows*

                System.out.println();

                System.out.println(" |--|-------------------------------|");

            }

        }

*// print a - h for completeness of board notation*

        System.out.println(" |  | a | b | c | d | e | f | g | h |");

        System.out.println(" ------------------------------------");

    }

*// remove a piece from the board, r = row, c = col*

    public void removePiece(int r, int c) {

        ArrayList<Piece> pieces *=* getPieces(); *// current pieces*

*for* (Piece p *:* pieces) {

*if* (p.getRow() *==* r *&&* p.getCol() *==* c) { *// find desired piece, remove, update game status*

                pieces.remove(p);

                updatePieces(pieces);

                updateGameBoard();

*break*;

            }

        }

    }

*// add a piece to the board*

    public void addPiece(Piece p, int r, int c) {

        ArrayList<Piece> pieces *=* getPieces(); *// current pieces*

*if* (*!*pieceOnSpace(r, c)) { *// if space is clear, add piece*

            pieces.add(p);

            updatePieces(pieces); *// update game status*

            updateGameBoard();

        } *else* {

            System.out.println("There is already a piece in this space! You cannot add a piece here.");

        }

    }

*// clear the board if necessary*

    public void clearBoard() {

*for* (int i *=* 0; i *<* 8; i*++*) {

*for* (int j *=* 0; j *<* 8; j*++*) {

                board[i][j] *=* null;

            }

        }

    }

*// check input location for a piece*

    public boolean pieceOnSpace(int r, int c) {

*for* (Piece p *:* getPieces()) {

*if* (p.getRow() *==* r *&&* p.getCol() *==* c) {

*return* true; *// piece found*

            }

        }

*return* false; *// no piece found*

    }

*// create pieces -- Note on color value: white = 0, black = 1*

    public void createPieces() {

        Scanner scan *=* *new* Scanner(System.in); *// scanner for user input*

        System.out.print("\nCreating Board and filling pieces "); *// ask user for game choice --> CHANGE THIS NAME!!*

        ArrayList<Piece> pieces *=* getPieces(); *// ArrayList of all 32 pieces in game*

*for* (int i *=* 0; i *<* 8; i*++*) { *// 8 pawns/player*

            pieces.add(*new* Piece(1, Piece.PAWN, 1, i)); *// Piece(color, type, row, col)*

            pieces.add(*new* Piece(0, Piece.PAWN, 6, i));

        }

*// 2 rooks/player*

        pieces.add(*new* Piece(1, Piece.ROOK, 0, 0));

        pieces.add(*new* Piece(1, Piece.ROOK, 0, 7));

        pieces.add(*new* Piece(0, Piece.ROOK, 7, 0));

        pieces.add(*new* Piece(0, Piece.ROOK, 7, 7));

*// 2 bishops/player*

        pieces.add(*new* Piece(1, Piece.BISHOP, 0, 2));

        pieces.add(*new* Piece(1, Piece.BISHOP, 0, 5));

        pieces.add(*new* Piece(0, Piece.BISHOP, 7, 2));

        pieces.add(*new* Piece(0, Piece.BISHOP, 7, 5));

*// 2 knights/player*

        pieces.add(*new* Piece(1, Piece.KNIGHT, 0, 1));

        pieces.add(*new* Piece(1, Piece.KNIGHT, 0, 6));

        pieces.add(*new* Piece(0, Piece.KNIGHT, 7, 1));

        pieces.add(*new* Piece(0, Piece.KNIGHT, 7, 6));

*// 1 queen/player*

        pieces.add(*new* Piece(1, Piece.QUEEN, 0, 3));

        pieces.add(*new* Piece(0, Piece.QUEEN, 7, 3));

*// 1 king/player*

        pieces.add(*new* Piece(1, Piece.KING, 0, 4));

        pieces.add(*new* Piece(0, Piece.KING, 7, 4));

    }

*// populate board with pieces*

    private void populateBoard() {

        ArrayList<Piece> pieces *=* getPieces(); *// current pieces*

*for* (Piece p *:* pieces) {

            board[p.getRow()][p.getCol()] *=* p; *// place pieces*

        }

    }

*// bool to set type of game*

    public boolean chooseGameType(int choice) {

*if* (choice *==* 0) { *// normal game*

*return* false;

        } *else* { *// scramble*

*return* true;

        }

    }

*// initiate the game*

    public void initBoard() {

        clearBoard(); *// start with clear board*

        createPieces(); *// create pieces, update game state*

        populateBoard();

        currentGameState();

    }

*// update the pieces*

    public void updatePieces(ArrayList<Piece> pieces) {

        this.pieces *=* pieces;

    }

*// update board*

    public void updateGameBoard() {

        clearBoard();

        populateBoard();

    }

*// get pieces*

    public ArrayList<Piece> getPieces() {

*return* pieces;

    }

*// get turn*

    public int getTurn() {

*return* turn;

    }

*// set turn*

    public void setTurn(int turn) {

        this.turn *=* turn;

    }

*// promote piece at input row, column location*

    public void promotePiece(Piece newPiece, int r, int c) {

        ArrayList<Piece> pieces *=* getPieces();

*for* (Piece p *:* pieces) {

*// find piece*

*if* (p.getRow() *==* r *&&* p.getCol() *==* c) {

                removePiece(r, c); *// remove piece currently there*

                addPiece(newPiece, r, c); *// add new piece*

                updatePieces(pieces); *// update game status*

                updateGameBoard();

*break*;

            }

        }

    }

*// check at end of each turn for pawns that reached the enemy's end line -->*

*// promote to queen*

    public void pawnPromotion(int color) {

        int endRow;

*if* (color *==* 0) {

            endRow *=* 0;

        } *else* {

            endRow *=* 7;

        }

        ArrayList<Piece> pieces *=* getPieces(); *// current pieces*

        int count *=* 0; *// iteration counter*

*for* (Piece p *:* pieces) { *// iterate through pieces*

*if* (p.getRow() *==* endRow *&&* p.getType() *==* 6 *&&* p.getColor() *==* color) { *// pawn on endrow found*

                p.setType(2); *// change to queen*

                pieces.set(count, p); *// put piece back in ArrayList*

                updatePieces(pieces); *// update game state*

                updateGameBoard();

*break*;

            }

            count*++*;

        }

    }

*// get piece at given location*

    public Piece getPieces(int r, int c) {

        ArrayList<Piece> pieces *=* getPieces(); *// current pieces*

*for* (Piece p *:* pieces) { *// iterate through pieces to find piece*

*if* (p.getRow() *==* r *&&* p.getCol() *==* c) {

*return* p;

            }

        }

*return* *new* Piece(); *// no piece found, blank space*

    }

*// get piece at given location with given color*

    public Piece getPieces(int r, int c, int color) {

        ArrayList<Piece> pieces *=* getPieces(); *// current pieces*

*for* (Piece p *:* pieces) { *// iterate through pieces to find piece*

*if* ((p.getRow() *==* r *&&* p.getCol() *==* c) *&&* p.getColor() *==* color) { *// if same location and color, return*

*// piece*

*return* p;

            }

        }

*return* *new* Piece(); *// no piece found, blank space*

    }

*// swap two input pieces*

    public void swapPieces(Piece one, Piece two) {

        ArrayList<Piece> pieces *=* getPieces(); *// current pieces*

        Piece hold *=* one; *// set piece one equal to a holding piece*

        int count *=* 0;

*for* (Piece p *:* pieces) { *// iterate through pieces to find piece*

*if* (p *==* one) { *// find Piece one*

                p.setType(two.getType()); *// set properties of piece one to piece two*

                p.setRow(two.getRow());

                p.setCol(two.getCol());

                p.setColor(two.getColor());

                pieces.set(count, p); *// put piece back in ArrayList*

                updatePieces(pieces); *// update game state*

                updateGameBoard();

*break*;

            }

            count*++*;

        }

        count *=* 0; *// reset count*

*for* (Piece p *:* pieces) {

*if* (p *==* two) {

                p.setType(hold.getType()); *// set properties of piece one to piece two*

                p.setRow(hold.getRow());

                p.setCol(hold.getCol());

                p.setColor(hold.getColor());

                pieces.set(count, p); *// put piece back in ArrayList*

                updatePieces(pieces); *// update game state*

                updateGameBoard();

*break*;

            }

            count*++*;

        }

    }

}

HelpMenu.java :

*// package mini\_project ;*

*import* java.util.\*;

public class HelpMenu {

    private ChessBoard board;

*// get board method*

    public ChessBoard getBoard() {

*return* board;

    }

*// app info*

    public void appInfo() {

*for* (int i *=* 0; i *<* 5; i*++*) {

            System.out.println();

        }

        System.out.println("Basic Chess App");

        System.out.println("Developed and built using Java");

    }

*// Basic Chess Rules*

    public void basicChessInfo() {

*for* (int i *=* 0; i *<* 5; i*++*) {

            System.out.println();

        }

        System.out.println("To learn about the fantastic game of chess..");

        System.out.println("Watch any Chess video on Youtube, as this place is just not enough to teach you CHESS");

    }

*// quit game*

    public void quitGame() {

*for* (int i *=* 0; i *<* 5; i*++*) {

            System.out.println();

        }

        System.out.println("\nGAME OVER!!!!!! \nExiting........");

        System.exit(1);

    }

*// display help menu, should allow user to navigate through help*

    public void display() {

*for* (int i *=* 0; i *<* 5; i*++*) {

            System.out.println();

        }

        Scanner scn *=* *new* Scanner(System.in);

        System.out.println("Help is here to HELP YOU!!");

*while* (true) {

            System.out.println("The following are your options:");

            System.out.println("1. App Info");

            System.out.println("2. Basic Chess Rules");

            System.out.println("3. Return to Game");

            System.out.println("3. Quit Game");

            System.out.println();

            System.out.println("Please enter the number of your choice.");

            String choice *=* scn.nextLine();

*if* (choice.charAt(0) *==* '1') {

                appInfo();

*break*;

            } *else* *if* (choice.charAt(0) *==* '2') {

                basicChessInfo();

*break*;

            } *else* *if* (choice.charAt(0) *==* '3') {

*break*;

            } *else* *if* (choice.charAt(0) *==* '4') {

                quitGame();

*break*;

            } *else* {

                System.out.println("That is not an option, try again.");

            }

        }

    }

}

Move.java :

*// package mini\_project;*

*import* java.util.\*;

public class Move{

    private ChessBoard board; *// game board*

*// constructor*

    public Move(ChessBoard board){

        this.board *=* board;

    }

*// get board*

    public ChessBoard getBoard(){

*return* board;

    }

*// find piece with given row, col*

    public Piece findPiece(int row, int col){

        boolean flag *=* false; *// piece found = true*

        ArrayList<Piece> pieces *=* getBoard().getPieces(); *// current pieces*

        Piece foundPiece *=* null;

        findLoop*:* *for*(Piece p*:* pieces){ *// iterate to find piece*

            foundPiece *=* p;

*if*(foundPiece.getRow() *==* row *&&* foundPiece.getCol() *==* col){

                flag *=* true; *// piece found, break loop*

*break* findLoop;

            }

        }

*// if the flag is true then return the found piece, otherwise null*

*if*(flag){

*return* foundPiece;

        } *else* {

*return* null;

        }

    }

*// convert String input into row integer*

    private int inputToRow(String input){

        char r *=* input.charAt(1);

        int row;

*switch*(r){ *// '0' value corresponds to '8', etc*

*case* '8'*:*

            row *=* 0;

*break*;

*case* '7'*:*

            row *=* 1;

*break*;

*case* '6'*:*

            row *=* 2;

*break*;

*case* '5'*:*

            row *=* 3;

*break*;

*case* '4'*:*

            row *=* 4;

*break*;

*case* '3'*:*

            row *=* 5;

*break*;

*case* '2'*:*

            row *=* 6;

*break*;

*case* '1'*:*

            row *=* 7;

*break*;

*default:*

            row *=* *-*1;

*break*;

        }

*return* row;

    }

*// convert String input into column integer*

    private int inputToCol(String input){

        char c *=* input.charAt(0);

        int col;

*switch*(c){ *// '0' value corresponds to 'a', etc*

*case* 'a'*:*

            col *=* 0;

*break*;

*case* 'b'*:*

            col *=* 1;

*break*;

*case* 'c'*:*

            col *=* 2;

*break*;

*case* 'd'*:*

            col *=* 3;

*break*;

*case* 'e'*:*

            col *=* 4;

*break*;

*case* 'f'*:*

            col *=* 5;

*break*;

*case* 'g'*:*

            col *=* 6;

*break*;

*case* 'h'*:*

            col *=* 7;

*break*;

*default:*

            col *=* *-*1;

*break*;

        }

*return* col;

    }

*// move piece in selected space to destination returns true if the move was*

*// successful*

     boolean move(String curSpace, String destSpace, Player player){

*// convert input Strings into row, col integers*

        int curRow, curCol, destRow, destCol;

        curRow *=* inputToRow(curSpace);

        curCol *=* inputToCol(curSpace);

        destRow *=* inputToRow(destSpace);

        destCol *=* inputToCol(destSpace);

*if*((curRow *>=* 0 *&&* curRow *<* 8) *&&* (curCol *>=* 0 *&&* curCol *<* 8)){ *// check source space*

*if*((destRow *>=* 0 *&&* destRow *<* 8) *&&* (destCol *>=* 0 *&&* destCol *<* 8)){ *// check destination space*

                Piece piece *=* findPiece(curRow, curCol);

*if*(piece.getColor() *==* player.getColor()){ *// color match check*

                    ArrayList<ArrayList<Integer>> allowedMoves *=* legalPieceMoves(piece, false); *// get legal moves*

                    ArrayList<Integer> r *=* allowedMoves.get(0); *// row*

                    ArrayList<Integer> c *=* allowedMoves.get(1); *// column*

                    ListIterator<Integer> rowIter *=* r.listIterator(); *// iterate through row, column*

                    ListIterator<Integer> colIter *=* c.listIterator();

                    int rNext, cNext;

*while*(rowIter.hasNext() *&&* colIter.hasNext()){ *// while the iterators still have values*

                        rNext *=* rowIter.next();

                        cNext *=* colIter.next();

*if*(destRow *==* rNext *&&* destCol *==* cNext){ *// if it is a valid move...*

                            getBoard().removePiece(destRow, destCol); *// remove target piece (if enemy on space)*

                            piece.setRow(destRow); *// set row, column to new space*

                            piece.setCol(destCol);

                            getBoard().updateGameBoard(); *// update game status*

                            getBoard().currentGameState();

                            getBoard().pawnPromotion(player.getColor()); *// promote pawns if necessary*

*// moveValid = true; // successful move!*

*return* true;

                        }

                    }

                }

            } *else* {

                System.out.println("Not a valid destination space.  again!");

*// moveValid = false; // DO NOT THINK I ACTAULLY NEED THIS ONE!! --> TAKE OUT IF NOT*

            }

        } *else* {

            System.out.println("Not a valid source space. Try again!");

*// moveValid = false;*

        }

*return* false; *// move failed*

    }

*// check if king piece is on input square*

    public boolean kingOnSpace(int r, int c, int color){

*for*(Piece p *:* getBoard().getPieces()){ *// iterate through pieces*

*if*(p.getColor() *==* color){ *// color check*

*if*(p.getRow() *==* r *&&* p.getCol() *==* c){ *// space check*

*if*(p.getType() *==* 1){ *// check that type = king*

*return* true;

                    }

                }

            }

        }

*return* false; *// no king on space*

    }

*// legal move check for input piece, returns list of allowed moves*

    public ArrayList<ArrayList<Integer>> legalPieceMoves(Piece piece, boolean checkMate){

        ArrayList<Integer> row, col; *// list of coordinates*

        row *=* *new* ArrayList<Integer>();

        col *=* *new* ArrayList<Integer>();

        ArrayList<ArrayList<Integer>> king, queen, rook, knight, bishop, pawn; *// list for each different type of piece*

        ArrayList<ArrayList<Integer>> legalMoves *=* *new* ArrayList<ArrayList<Integer>>(); *// list to be returned with legal move set*

*switch*(piece.getType()){ *// switch between different types*

*// king*

*case* 1*:*

                king *=* possKingMoves(piece, checkMate); *// fill king with all possible King moves*

                row *=* king.get(0); *// row values are stored in first ArrayList*

                col *=* king.get(1); *// column values are stored in second ArrayList*

*break*;

*// queen*

*case* 2*:*

                queen *=* possQueenMoves(piece, checkMate);

                row *=* queen.get(0);

                col *=* queen.get(1);

*break*;

*// rook*

*case* 3*:*

                rook *=* possRookMoves(piece, checkMate);

                row *=* rook.get(0);

                col *=* rook.get(1);

*break*;

*// knight*

*case* 4*:*

                knight *=* possKnightMoves(piece, checkMate);

                row *=* knight.get(0);

                col *=* knight.get(1);

*break*;

*// bishop*

*case* 5*:*

                bishop *=* possBishopMoves(piece, checkMate);

                row *=* bishop.get(0);

                col *=* bishop.get(1);

*break*;

*// pawn*

*case* 6*:*

                pawn *=* possPawnMoves(piece, checkMate);

                row *=* pawn.get(0);

                col *=* pawn.get(1);

*break*;

*default:*

*break*;

        }

        legalMoves.add(row); *// add row, col to legalMoves*

        legalMoves.add(col);

*return* legalMoves; *// return list of legal moves*

    }

*// check if input row, col coordinates are on the board*

    private boolean onBoardCheck(int r, int c){

*if*((r *>=* 0 *&&* r *<=* 7) *&&* (c *>=* 0 *&&* c *<=* 7)){

*return* true;

        } *else* {

*return* false;

        }

    }

*// check for piece on input space, input: row, column*

    public boolean pieceOnSpace(int r, int c){

*for*(Piece p *:* getBoard().getPieces()){ *// iterate through pieces on board*

*if*(p.getRow() *==* r *&&* p.getCol() *==* c){ *// find piece*

*return* true;

            }

        }

*return* false; *// no piece found*

    }

*// check for piece on input space, input: row, column, color*

    public boolean pieceOnSpace(int r, int c, int color){

*for*(Piece p *:* getBoard().getPieces()){ *// iterate through pieces*

*if*(p.getColor () *==* color){ *// color check*

*if*(p.getRow() *==* r *&&* p.getCol() *==* c){ *// find piece*

*return* true;

                }

            }

        }

*return* false; *// no piece found*

    }

*// possible pawn movements: returns ArrayList<ArrayList<Integer>>*

    private ArrayList<ArrayList<Integer>> possPawnMoves(Piece piece, boolean checkMate){

        ArrayList<Integer> row, col; *// row, column lists*

        row *=* *new* ArrayList<Integer>();

        col *=* *new* ArrayList<Integer>();

        ArrayList<ArrayList<Integer>> rowAndCol *=* *new* ArrayList<ArrayList<Integer>>(); *// combined list to return*

        int r *=* piece.getRow(); *// row*

        int c *=* piece.getCol(); *// column*

        int oppColor *=* piece.getEnemyColor(); *// enemy color*

*// regular forward movement*

*if*(piece.getType() *==* 6 *&&* piece.getColor() *==* 0){ *// check if pawn / right color*

*if*(r *==* 6) { *// if pawn has not moved, can move one or two spaces*

*if*(*!*pieceOnSpace(r*-*1, c)){ *// make sure space is clear*

                    row.add(r*-*1); *// add row, column to possible moves*

                    col.add(c);

*if*(*!*pieceOnSpace(r*-*2, c)){

                        row.add(r*-*2);

                        col.add(c);

                    }

                }

            } *else* { *// not on home row, can only move one space forward*

*if*((r*-*1)*>=*0 *&&* *!*pieceOnSpace(r*-*1, c)){ *// make sure space is clear / pawn will not go off board*

                    row.add(r*-*1);

                    col.add(c);

                }

            }

*// diagonal above: right/left is for capturing opponents*

*if*(pieceOnSpace(r*-*1, c*+*1, oppColor)){ *// check for opponent in space (above right)*

*if*(checkMate){ *// if checkMate = true, game is over and pawn can take king*

                    row.add(r*-*1);

                    col.add(c*+*1);

                } *else* {  *// game is not over yet*

*if*(*!*kingOnSpace(r*-*1, c*+*1, oppColor)){ *// make sure king is not on space*

                        row.add(r*-*1);

                        col.add(c*+*1);

                    }

                }

            }

*if*(pieceOnSpace(r*-*1, c*-*1, oppColor)){ *// above left*

*if*(checkMate){

                    row.add(r*-*1);

                    col.add(c*-*1);

                } *else* {

*if*(*!*kingOnSpace(r*-*1, c*-*1, oppColor)){

                        row.add(r*-*1);

                        col.add(c*-*1);

                    }

                }

            }

        } *else* *if*(piece.getType() *==* 6 *&&* piece.getColor() *==* 1){ *// black pawn*

*// regular forward movement*

*if*(r *==* 1) {

*if*(*!*pieceOnSpace(r*+*1, c)){

                    row.add(r*+*1);

                    col.add(c);

*if*(*!*pieceOnSpace(r*+*2, c)){

                        row.add(r*+*2);

                        col.add(c);

                    }

                }

            } *else* {

*if*((r*+*1)*>=*0 *&&* *!*pieceOnSpace(r*+*1, c)){

                    row.add(r*+*1);

                    col.add(c);

                }

            }

*// diagonal below: right/left is for capturing opponents*

*if*(pieceOnSpace(r*+*1, c*+*1, oppColor)){ *// below right*

*if*(checkMate){

                    row.add(r*+*1);

                    col.add(c*+*1);

                } *else* {

*if*(*!*kingOnSpace(r*+*1, c*+*1, oppColor)){

                        row.add(r*+*1);

                        col.add(c*+*1);

                    }

                }

            }

*if*(pieceOnSpace(r*+*1, c*-*1, oppColor)){ *// below left*

*if*(checkMate){

                    row.add(r*+*1);

                    col.add(c*-*1);

                } *else* {

*if*(*!*kingOnSpace(r*+*1, c*-*1, oppColor)){

                        row.add(r*+*1);

                        col.add(c*-*1);

                    }

                }

            }

        }

        rowAndCol.add(row); *// add row, column to combined list*

        rowAndCol.add(col);

*return* rowAndCol; *// return pawn moves*

    }

*// possible bishop movements: returns ArrayList<ArrayList<Integer>>*

    private ArrayList<ArrayList<Integer>> possBishopMoves(Piece piece, boolean checkMate){

        ArrayList<Integer> row, col; *// lists for row, col*

        row *=* *new* ArrayList<Integer>();

        col *=* *new* ArrayList<Integer>();

        ArrayList<ArrayList<Integer>> rowAndCol *=* *new* ArrayList<ArrayList<Integer>>(); *// combined list to return*

        int r *=* piece.getRow(); *// row*

        int c *=* piece.getCol(); *// column*

        int color *=* piece.getColor(); *// color*

        int oppColor *=* piece.getEnemyColor(); *// enemy color*

*// check all diagonal above, right*

        int j *=* c*+*1; *// column count*

        int i *=* r*-*1; *// row count*

*while*(i *>=* 0 *&&* j *<=* 7){ *// while on the board*

*if*(pieceOnSpace(i, j, color)){ *// if same color piece is on space, no add, break*

*break*;

            } *else* *if*(pieceOnSpace(i, j, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// check for checkmate, end game if true*

                    row.add(i);

                    col.add(j);

                } *else* { *// not checkmate*

*if*(*!*kingOnSpace(i, j, oppColor)){ *// if the piece is not the king, can take the piece*

                        row.add(i);

                        col.add(j);

                    }

                }

*break*;

            } *else* { *// empty space so add*

                row.add(i);

                col.add(j);

            }

            i*--*; *// iterate through counters*

            j*++*;

        }

*// check all diagonal above, left*

        j *=* c*-*1;

        i *=* r*-*1;

*while*(i *>=* 0 *&&* j *>=* 0){ */// while on board*

*if*(pieceOnSpace(i, j, color)){ *// if same color piece, no add, break*

*break*;

            } *else* *if*(pieceOnSpace(i, j, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                    row.add(i);

                    col.add(j);

                } *else* { *// not checkmate, make sure piece is not king*

*if*(*!*kingOnSpace(i, j, oppColor)){

                        row.add(i);

                        col.add(j);

                    }

                }

*break*;

            } *else* { *// empty space so add*

                row.add(i);

                col.add(j);

            }

            i*--*; *// iterate counters*

            j*--*;

        }

*// check all diagonal below, right*

        j *=* c*+*1;

        i *=* r*+*1;

*while*(i *<=* 7 *&&* j *<=* 7){ *// while on board*

*if*(pieceOnSpace(i, j, color)){ *// if same color piece, no add, break*

*break*;

            } *else* *if*(pieceOnSpace(i, j, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                    row.add(i);

                    col.add(j);

                } *else* { *// not checkmate, make sure piece is not king*

*if*(*!*kingOnSpace(i, j, oppColor)){

                        row.add(i);

                        col.add(j);

                    }

                }

*break*;

            } *else* { *// empty space so add*

                row.add(i);

                col.add(j);

            }

            i*++*; *// iterate counters*

            j*++*;

        }

*// check all diagonal, left*

        j *=* c*-*1;

        i *=* r*+*1;

*while*(i *<=* 7 *&&* j *>=* 0){ *// while on board*

*if*(pieceOnSpace(i, j, color)){ *// if same color piece, no add, break*

*break*;

            } *else* *if*(pieceOnSpace(i, j, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                    row.add(i);

                    col.add(j);

                } *else* { *// not checkmate, make sure piece is not king*

*if*(*!*kingOnSpace(i, j, oppColor)){

                        row.add(i);

                        col.add(j);

                    }

                }

*break*;

            } *else* { *// empty space so add*

                row.add(i);

                col.add(j);

            }

            i*++*; *// iterate counters*

            j*--*;

        }

        rowAndCol.add(row); *// add row, column to combination ArrayList*

        rowAndCol.add(col);

*return* rowAndCol; *// return combination*

    }

*// possible knight movements: returns ArrayList<ArrayList<Integer>>*

    private ArrayList<ArrayList<Integer>> possKnightMoves(Piece piece, boolean checkMate){

        ArrayList<Integer> row, col; *// row, column lists*

        row *=* *new* ArrayList<Integer>();

        col *=* *new* ArrayList<Integer>();

        ArrayList<ArrayList<Integer>> rowAndCol *=* *new* ArrayList<ArrayList<Integer>>(); *// combined list to return*

        int r *=* piece.getRow(); *// row*

        int c *=* piece.getCol(); *// column*

        int color *=* piece.getColor(); *// color*

        int oppColor *=* piece.getEnemyColor(); *// enemy color*

*// check above, right spaces (2 options)*

*// option 1: up two, right one*

*if*(pieceOnSpace(r*-*2, c*+*1, color)){

*// if same color piece on space, do not add*

        } *else* *if*(pieceOnSpace(r*-*2, c*+*1, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                row.add(r*-*2);

                col.add(c*+*1);

            } *else* { *// not checkmate, make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*-*2, c*+*1, oppColor)){

                    row.add(r*-*2);

                    col.add(c*+*1);

                }

            }

        } *else* { *// empty space so add*

            row.add(r*-*2);

            col.add(c*+*1);

        }

*// option 2: up one, right two*

*if*(pieceOnSpace(r*-*1, c*+*2, color)){

*// if same color piece on space, do not add*

        } *else* *if*(pieceOnSpace(r*-*1, c*+*2, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                row.add(r*-*1);

                col.add(c*+*2);

            } *else* { *// not checkmate, make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*-*1, c*+*2, oppColor)){

                    row.add(r*-*1);

                    col.add(c*+*2);

                }

            }

        } *else* { *// empty space so add*

            row.add(r*-*1);

            col.add(c*+*2);

        }

*// check above, left spaces (2 options)*

*//option 1: up two, left one*

*if*(pieceOnSpace(r*-*2, c*-*1, color)){

*// if same color piece on space, do not add*

        } *else* *if*(pieceOnSpace(r*-*2, c*-*1, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                row.add(r*-*2);

                col.add(c*-*1);

            } *else* { *// not checkmate, make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*-*2, c*-*1, oppColor)){

                    row.add(r*-*2);

                    col.add(c*-*1);

                }

            }

        } *else* { *// empty space so add*

            row.add(r*-*2);

            col.add(c*-*1);

        }

*// option 2: up one, left two*

*if*(pieceOnSpace(r*-*1, c*-*2, color)){

*// if same color piece on space, do not add*

        } *else* *if*(pieceOnSpace(r*-*1, c*-*2, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                row.add(r*-*1);

                col.add(c*-*2);

            } *else* { *// not checkmate, make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*-*1, c*-*2, oppColor)){

                    row.add(r*-*1);

                    col.add(c*-*2);

                }

            }

        } *else* { *// empty space so add*

            row.add(r*-*1);

            col.add(c*-*2);

        }

*// check below, right spaces (2 options)*

*//option 1: down two right one*

*if*(pieceOnSpace(r*+*2, c*+*1, color)){

*// if same color piece on space, do not add*

        } *else* *if*(pieceOnSpace(r*+*2, c*+*1, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                row.add(r*+*2);

                col.add(c*+*1);

            } *else* { *// not checkmate, make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*+*2, c*+*1, oppColor)){

                    row.add(r*+*2);

                    col.add(c*+*1);

                }

            }

        } *else* { *// empty space so add*

            row.add(r*+*2);

            col.add(c*+*1);

        }

*// option 2: down one, right two*

*if*(pieceOnSpace(r*+*1, c*+*2, color)){

*// if same color piece on space, do not add*

        } *else* *if*(pieceOnSpace(r*+*1, c*+*2, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                row.add(r*+*1);

                col.add(c*+*2);

            } *else* { *// not checkmate, make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*+*1, c*+*2, oppColor)){

                    row.add(r*+*1);

                    col.add(c*+*2);

                }

            }

        } *else* { *// empty space so add*

            row.add(r*+*1);

            col.add(c*+*2);

        }

*// check below, left spaces (2 options)*

*//option 1: down two, left one*

*if*(pieceOnSpace(r*+*2, c*-*1, color)){

*// if same color piece on space, do not add*

        } *else* *if*(pieceOnSpace(r*+*2, c*-*1, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                row.add(r*+*2);

                col.add(c*-*1);

            } *else* { *// not checkmate, make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*+*2, c*-*1, oppColor)){

                    row.add(r*+*2);

                    col.add(c*-*1);

                }

            }

        } *else* { *// empty space so add*

            row.add(r*+*2);

            col.add(c*-*1);

        }

*// option 2: down one, left two*

*if*(pieceOnSpace(r*+*1, c*-*2, color)){

*// if same color piece on space, do not add*

        } *else* *if*(pieceOnSpace(r*+*1, c*-*2, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                row.add(r*+*1);

                col.add(c*-*2);

            } *else* { *// not checkmate, make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*+*1, c*-*2, oppColor)){

                    row.add(r*+*1);

                    col.add(c*-*2);

                }

            }

        } *else* { *// empty space so add*

            row.add(r*+*1);

            col.add(c*-*2);

        }

        rowAndCol.add(row); *// add row, column to combined list*

        rowAndCol.add(col);

*return* rowAndCol; *// return combined list*

    }

*// possible rook movements: returns ArrayList<ArrayList<Integer>>*

    private ArrayList<ArrayList<Integer>> possRookMoves(Piece piece, boolean checkMate){

        ArrayList<Integer> row, col; *// list for row, col*

        row *=* *new* ArrayList<Integer>();

        col *=* *new* ArrayList<Integer>();

        ArrayList<ArrayList<Integer>> rowAndCol *=* *new* ArrayList<ArrayList<Integer>>(); *// combined list to return*

        int r *=* piece.getRow(); *// row*

        int c *=* piece.getCol(); *// column*

        int color *=* piece.getColor(); *// color*

        int oppColor *=* piece.getEnemyColor(); *// enemy color*

*// check all spaces above piece*

*for*(int i *=* r*-*1; i *>=* 0; i*--*){

*if*(pieceOnSpace(i, c, color)){ *// if same color piece, no add, break*

*break*;

            } *else* *if*(pieceOnSpace(i, c, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                    row.add(i);

                    col.add(c);

                } *else* { *// not checkmate so check if piece is enemy king*

*if*(*!*kingOnSpace(i, c, oppColor)){

                        row.add(i);

                        col.add(c);

                    }

                }

*break*;

            } *else* { *// empty space so add*

                row.add(i);

                col.add(c);

            }

        }

*// check all spaces below piece*

*for*(int i *=* r*+*1; i *<* 8; i*++*){

*if*(pieceOnSpace(i, c, color)){ *// if same color piece, no add, break*

*break*;

            } *else* *if*(pieceOnSpace(i, c, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                    row.add(i);

                    col.add(c);

                } *else* { *// not checkmate so check if piece is enemy king*

*if*(*!*kingOnSpace(i, c, oppColor)){

                        row.add(i);

                        col.add(c);

                    }

                }

*break*;

            } *else* { *// empty space so add*

                row.add(i);

                col.add(c);

            }

        }

*// check all spaces to the right of piece*

*for*(int i *=* c*+*1; i *<* 8; i*++*){

*if*(pieceOnSpace(i, c, color)){ *// if same color piece, no add, break*

*break*;

            } *else* *if*(pieceOnSpace(i, c, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                    row.add(i);

                    col.add(c);

                } *else* { *// not checkmate so check if piece is enemy king*

*if*(*!*kingOnSpace(i, c, oppColor)){

                        row.add(i);

                        col.add(c);

                    }

                }

*break*;

            } *else* { *// empty space so add*

                row.add(i);

                col.add(c);

            }

        }

*// check all spaces to the left of piece*

*for*(int i *=* c*-*1; i *>=* 0; i*--*){

*if*(pieceOnSpace(i, c, color)){ *// if same color piece, no add, break*

*break*;

            } *else* *if*(pieceOnSpace(i, c, oppColor)){ *// check for opponent on space, add then break*

*if*(checkMate){ *// capture king*

                    row.add(i);

                    col.add(c);

                } *else* { *// not checkmate so check if piece is enemy king*

*if*(*!*kingOnSpace(i, c, oppColor)){

                        row.add(i);

                        col.add(c);

                    }

                }

*break*;

            } *else* { *// empty space so add*

                row.add(i);

                col.add(c);

            }

        }

        rowAndCol.add(row); *// add row, column to combined list*

        rowAndCol.add(col);

*return* rowAndCol; *// return combined list*

    }

*// possible king movements: returns ArrayList<ArrayList<Integer>>*

    private ArrayList<ArrayList<Integer>> possKingMoves(Piece piece, boolean checkMate){

        ArrayList<Integer> row, col; *// row, column list*

        row *=* *new* ArrayList<Integer>();

        col *=* *new* ArrayList<Integer>();

        ArrayList<ArrayList<Integer>> rowAndCol *=* *new* ArrayList<ArrayList<Integer>>(); *// combined list to be returned*

        int r *=* piece.getRow(); *// row*

        int c *=* piece.getCol(); *// column*

        int color *=* piece.getColor(); *// color*

        int oppColor *=* piece.getEnemyColor(); *// enemy color*

*// NOTE: DOES THERE EVEN HAVE TO BE AN ONBOARD CHECK?? --> WOULD NOT BE TOO HARD TO TAKE AWAY, BUT IS NOT TOO CRAZY*

*// possible horizontal / vertical / diagonal movements*

*// space to the right*

*if*(onBoardCheck(r, c*+*1) *&&* *!*pieceOnSpace(r, c*+*1, color)){ *// on board and not a piece that is same color in space, add space*

*if*(checkMate){ *// capture king*

                row.add(r);

                col.add(c*+*1);

            } *else* { *// not checkmate so make sure piece is not enemy king*

*if*(*!*kingOnSpace(r, c*+*1, oppColor)){

                    row.add(r);

                    col.add(c*+*1);

                }

            }

        }

*// space to the left*

*if*(onBoardCheck(r, c*-*1) *&&* *!*pieceOnSpace(r, c*-*1, color)){ *// on board and not a piece that is same color in space, add space*

*if*(checkMate){ *// capture king*

                row.add(r);

                col.add(c*-*1);

            } *else* { *// not checkmate so make sure piece is not enemy king*

*if*(*!*kingOnSpace(r, c*-*1, oppColor)){

                    row.add(r);

                    col.add(c*-*1);

                }

            }

        }

*// space above*

*if*(onBoardCheck(r*-*1, c) *&&* *!*pieceOnSpace(r*-*1, c, color)){ *// on board and not a piece that is same color in space, add space*

*if*(checkMate){ *// capture king*

                row.add(r*-*1);

                col.add(c);

            } *else* { *// not checkmate so make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*-*1, c, oppColor)){

                    row.add(r*-*1);

                    col.add(c);

                }

            }

        }

*// space below*

*if*(onBoardCheck(r*+*1, c) *&&* *!*pieceOnSpace(r*+*1, c, color)){ *// on board and not a piece that is same color in space, add space*

*if*(checkMate){ *// capture king*

                row.add(r*+*1);

                col.add(c);

            } *else* { *// not checkmate so make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*+*1, c, oppColor)){

                    row.add(r*+*1);

                    col.add(c);

                }

            }

        }

*// space diagonal up, right*

*if*(onBoardCheck(r*-*1, c*+*1) *&&* *!*pieceOnSpace(r*-*1, c*+*1, color)){ *// on board and not a piece that is same color in space, add space*

*if*(checkMate){ *// capture king*

                row.add(r*-*1);

                col.add(c*+*1);

            } *else* { *// not checkmate so make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*-*1, c*+*1, oppColor)){

                    row.add(r*-*1);

                    col.add(c*+*1);

                }

            }

        }

*// space diagonal up, left*

*if*(onBoardCheck(r*-*1, c*-*1) *&&* *!*pieceOnSpace(r*-*1, c*-*1, color)){ *// on board and not a piece that is same color in space, add space*

*if*(checkMate){ *// capture king*

                row.add(r*-*1);

                col.add(c*-*1);

            } *else* { *// not checkmate so make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*-*1, c*-*1, oppColor)){

                    row.add(r*-*1);

                    col.add(c*-*1);

                }

            }

        }

*// space diagonal down, right*

*if*(onBoardCheck(r*+*1, c*+*1) *&&* *!*pieceOnSpace(r*+*1, c*+*1, color)){ *// on board and not a piece that is same color in space, add space*

*if*(checkMate){ *// capture king*

                row.add(r*+*1);

                col.add(c*+*1);

            } *else* { *// not checkmate so make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*+*1, c*+*1, oppColor)){

                    row.add(r*+*1);

                    col.add(c*+*1);

                }

            }

        }

*// space diagonal down, left*

*if*(onBoardCheck(r*+*1, c*-*1) *&&* *!*pieceOnSpace(r*+*1, c*-*1, color)){ *// on board and not a piece that is same color in space, add space*

*if*(checkMate){ *// capture king*

                row.add(r*+*1);

                col.add(c*-*1);

            } *else* { *// not checkmate so make sure piece is not enemy king*

*if*(*!*kingOnSpace(r*+*1, c*-*1, oppColor)){

                    row.add(r*+*1);

                    col.add(c*-*1);

                }

            }

        }

        rowAndCol.add(row); *// add row, column to combined list*

        rowAndCol.add(col);

*return* rowAndCol; *// return combined list*

    }

*// possible queen movements: returns ArrayList<ArrayList<Integer>>*

    private ArrayList<ArrayList<Integer>> possQueenMoves(Piece piece, boolean checkMate){

        ArrayList<Integer> row, col; *// row, col list*

        row *=* *new* ArrayList<Integer>();

        col *=* *new* ArrayList<Integer>();

        ArrayList<ArrayList<Integer>> rowAndCol *=* *new* ArrayList<ArrayList<Integer>>(); *// combination array to be returned*

*// queen is a combination of bishop and rook movements. So: add both to row and col*

        ArrayList<ArrayList<Integer>> bishop, rook;

        bishop *=* possBishopMoves(piece, checkMate);

        rook *=* possRookMoves(piece, checkMate);

        row.addAll(bishop.get(0));

        col.addAll(bishop.get(1));

        row.addAll(rook.get(0));

        col.addAll(rook.get(1));

        rowAndCol.add(row); *// add row, column lists to combined list*

        rowAndCol.add(col);

*return* rowAndCol; *// return combined list*

    }

*// possible enemy targets for input piece to capture*

    public ArrayList<Piece> possEnemyTargets(Piece piece){

        ArrayList<Piece> enemyTargets *=* *new* ArrayList<Piece>(); *// list of targets to be returned*

        ArrayList<ArrayList<Integer>> allowedMoves *=* legalPieceMoves(piece, false); *// legal moves for piece*

        ArrayList<Integer> r *=* allowedMoves.get(0); *// row list*

        ArrayList<Integer> c *=* allowedMoves.get(1); *// column list*

        ListIterator<Integer> rowIter *=* r.listIterator(); *// iterate through both row, column lists*

        ListIterator<Integer> colIter *=* c.listIterator();

        int rNext, cNext;

*while*(rowIter.hasNext() *&&* colIter.hasNext()){ *// while the iterators still have values*

            rNext *=* rowIter.next();

            cNext *=* colIter.next();

*if*(pieceOnSpace(rNext, cNext, piece.getEnemyColor())){ *// check if enemy is on space, if so add it to list*

                enemyTargets.add(board.getPieces(rNext, cNext, piece.getEnemyColor()));

            }

        }

*return* enemyTargets; *// return list full of targets*

    }

*// check for check! --> so if the king of the current player (current color) is in the list for any of the opponent's possEnemyTargets then they are in check and the flag should be thrown to show so*

    public boolean checkCheck(int color){

*// ArrayList with the current pieces*

        ArrayList<Piece> pieces *=* getBoard().getPieces();

*for*(Piece p *:* pieces){ *// iterate over all pieces*

            ArrayList<Piece> enemyTargets *=* possEnemyTargets(p); *// fill enemyTargets with Piece p's targets*

            ListIterator<Piece> enemyIter *=* enemyTargets.listIterator();

            Piece nextEnemy *=* *new* Piece();

*while*(enemyIter.hasNext()){ *// while iterator has more values*

                nextEnemy *=* enemyIter.next();

*if*(nextEnemy.getType() *==* 1 *&&* (p.getColor() *!=* color)){ *// if the next available target for that piece is a king and the opposite color*

*// CHANGE THIS TO BE MORE FLUID!*

*//System.out.println("You are in check!");*

*return* true;

                }

            }

        }

*return* false; *// no pieces are putting king in check*

    }

}

Piece.java:

*// package mini\_project;*

public class Piece {

*// constant integer piece type*

    public static final int KING *=* 1;

    public static final int QUEEN *=* 2;

    public static final int ROOK *=* 3;

    public static final int KNIGHT *=* 4;

    public static final int BISHOP *=* 5;

    public static final int PAWN *=* 6;

    private int color; *// white = 0, black = 1*

    private int type; *// type of piece*

    private int row, col; *// position on board*

*// default constructor, used for blank space*

    public Piece() {

    }

*// constructor to create pieces*

    public Piece(int color, int type, int row, int col) {

        this.color *=* color;

        this.type *=* type;

        this.row *=* row;

        this.col *=* col;

    }

*// get piece name for board*

    public String getPieceName() {

        String name *=* "";

*if* (getColor() *==* 0 *&&* getType() *!=* 0) {

            name *=* "w";

        } *else* {

            name *=* "b";

        }

*switch* (getType()) {

*case* 1*:*

                name *+=* "KI";

*break*;

*case* 2*:*

                name *+=* "QU";

*break*;

*case* 3*:*

                name *+=* "RK";

*break*;

*case* 4*:*

                name *+=* "KN";

*break*;

*case* 5*:*

                name *+=* "BI";

*break*;

*case* 6*:*

                name *+=* "PN";

*break*;

*default:* *// show error to terminal, break*

                System.out.println("The given piece type is not in the valid range.");

*break*;

        }

*return* name;

    }

*// get type*

    public int getType() {

*return* type;

    }

*// set type*

    public void setType(int type) {

        this.type *=* type;

    }

*// get color*

    public int getColor() {

*return* color;

    }

*// get enemy color*

    public int getEnemyColor() {

*if* (color *==* 0) {

*return* 1;

        } *else* {

*return* 0;

        }

    }

*// set color*

    public void setColor(int color) {

        this.color *=* color;

    }

*// get row*

    public int getRow() {

*return* row;

    }

*// set row*

    public void setRow(int row) {

        this.row *=* row;

    }

*// get column*

    public int getCol() {

*return* col;

    }

*// set column*

    public void setCol(int col) {

        this.col *=* col;

    }

}

Player.java :

*// package mini\_project;*

*import* java.util.\*;

public class Player {

    private ChessBoard board; *// current board*

    private int color; *// white = 0, black = 1*

    private String name; *// player name*

*// constructor to choose color*

    public Player(int color) {

        this.color *=* color;

    }

*// constructor to input name as well as color*

    public Player(int color, String name) {

        this.color *=* color;

        this.name *=* name;

    }

*// get board*

    private ChessBoard getBoard() {

*return* board;

    }

*// get color*

    public int getColor() {

*return* color;

    }

*// set color*

    public void setColor(int color) {

        this.color *=* color;

    }

*// return enemy color*

    public int getEnemyColor() {

*if* (getColor() *==* 0) {

*return* 1;

        } *else* {

*return* 0;

        }

    }

*// get name*

    public String getName() {

*return* name;

    }

*// set name*

    public void setName(String name) {

        this.name *=* name;

    }

*// get number of pieces remaining for player*

    public int getNumPiecesLeft() {

        ArrayList<Piece> pieces *=* getBoard().getPieces(); *// get current pieces*

        int numLeft *=* 0; *// assume no pieces*

*for* (Piece p *:* pieces) {

*if* (p.getColor() *==* color) { *// if same color, increment numLeft*

                numLeft*++*;

            }

        }

*return* numLeft;

    }

}