DEVELOPMENT OF AN ADAPTIVE CHESS PROGRAM

Final Report

by

Eda Okur

Selin Kavuzlu

Submitted to the Department of Computer Engineering

in partial fulfillment of the requirements

for the degree of

Bachelor of Science

in

Computer Engineering

Boğaziçi University

June 2011

TABLE OF CONTENTS

Page

1. Introduction

The current level of development in computer chess programming is fairly complicated, yet interesting as well. In this project, we were supposed to develop a chess-playing program. The program was supposed to play chess at a good level and have an adaptive property. The description of the project included the analysis of the evaluation of the pieces and board positions on the board. In addition, literature survey on methods, techniques, and heuristics used in chess playing and chess programming was supposed to be done in order to analyze how the pieces, board, and positions on the board are evaluated.

Upon doing this literature survey, we proposed an algorithm for the computer program that plays chess according to the results of our analysis; then developed and implemented this program. We also tried to make a contribution with some new approaches to chess-playing programming.

2. Background

For the completeness of this document, we will mention some details that we have already mentioned in the midterm progress document, regarding the background of the project. A general computer chess program involves somehow simplified approach at its base. We will explain the methods and algorithms used for a chess program by words at this section.

We start with a chessboard set up for the start of a game. Each player has 16 pieces. The white player starts the game all the time. At the beginning, white has 20 possible moves:

- The white player can move any pawn forward one or two positions.
- The white player can move either knight in two different ways.

The white player chooses one of these 20 moves and plays it. For the black player, the options are the same as 20 possible moves. So black chooses a move among those as well. Now white player can move again. This next move depends on the first move that white chose to make, but there are about 20 or so moves white can

make given the current board position, and then black has 20 or so moves it can make, and so on. Actually the number of the moves both players can make usually increases as the game develops.

This is how a computer program looks at chess. It thinks about it in a world of "all possible moves," and it makes a very large tree (our search tree) for all of those moves. It can be visualized as follows:

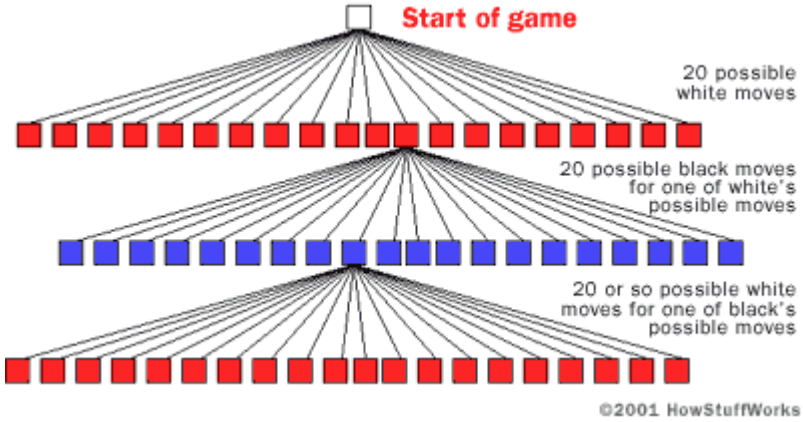


Figure 1. The Game Tree

In this tree, there are 20 possible moves for white. There are 20 * 20 = 400 possible moves for black, depending on what white does. Then there are 400 * 20 = 8,000 for white, and so on. If you were to fully develop the entire tree for all possible chess moves, this would be a very big number. Therefore we can say that chess is a pretty complicated game.

No computer is able to calculate the entire tree. What a chess computer tries to do is generate the board-position tree for nearly 5 moves into the future. Assuming that there are about 20 possible moves for any board position, a five-level tree contains 3,200,000 board positions. The depth of the tree that a computer can calculate is controlled by the speed of the computer playing the game.

Once it generates the search tree, then the computer needs to evaluate the board positions. That is, the computer has to look at the pieces on the board and decide whether that arrangement of pieces is "good" or "bad." The way it does this is by using an **evaluation function**. The simplest possible function might just count the number of pieces each side has. Obviously, for chess that is way too simple, because some pieces are more valuable than others. So our formula should apply a weight to each type of piece. It is possible to make the evaluation function more and more

complicated by adding things like board position, control of the center, vulnerability of the king to check, vulnerability of the opponent's queen, and many other parameters. No matter how complicated the function gets, however, it is condensed down to a single number that represents the "goodness" of that board position. Our proposed evaluation function will be explained in detail later.

After the evaluation function calculates a numeric value for each node of the game tree, we need to apply a certain search strategy to construct an acceptable chess program. At this search step, we are planning to apply minimax search algorithm to choose the best possible next move and alpha-beta pruning technique to increase the performance by decreasing the search space.

The basic idea underlying all two-agent search algorithms is **Minimax** method. Minimax is a recursive algorithm used for choosing the next move in a game. A tree of legal moves is built and played. Each move is evaluated using an evaluation function. The computer makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. The details of our Minimax algorithm will be explained later.

The basic idea behind the **Alpha-Beta Pruning** algorithm is that, once you have a good move, you can quickly eliminate alternatives that lead to disaster. We know that there are many of those quickly disposable moves in chess game. The alpha-beta pruning function is actually an improvement of the Minimax search method. It reduces the number of tree nodes to evaluate by eliminating a move when at least one possibility was proved worse than a previously evaluated one. The details of our alpha-beta pruning algorithm will be explained later.

We used the **evolutionary algorithms** to finely tune the evaluation function parameter weights. This adds adaptive feature to our chess program. The evolution technique is taken from Graham Kendall, Hallam Nasreddine and Hendra Suhanto Poh's paper **"**Using an Evolutionary Algorithm for the Tuning of a Chess Evaluation Function Based on a Dynamic Boundary Strategy", except that we planned to develop a different evaluation function and we did. The details will be given in the algorithm section.

With this background, we constructed our algorithms very carefully. We proposed the ideas we gathered and the algorithms that we have designed in the

midterm progress report in quite detail, but you can see the revised version in the "algorithm" section of this paper. We did the implementation using Java as the language and NetBeans IDE as the platform. Along with the implementation, some design decisions changed while some stayed the same. We will try to mention them, while we are telling about the details of our final program in the next sections.

3. Algorithms

3.1. Introduction

Before giving the implementation details, we will give the algorithms we used for the critical operations. These include the evaluation function, minimax search, alpha beta pruning and evolutionary algorithm.

3.2. Evaluation Function

The **evaluation function** should return +∞ for white's winning positions, -∞ for black's winning positions and 0, if the game is a draw. The evaluation function should take some basic factors like material, mobility, development and central position into account.

The most important feature in chess is material. Material represents all of a player's pieces and pawns on the board. The player with pieces and pawns of greater value is said to have a "material advantage". Other things being equal, the player who is ahead material will usually win the game. We know that the pieces have different values which can be measured in multiples of a pawn. Standard values are as follows: pawn = 1, knight = 3, bishop = 3, rook = 5, queen = 9, king = above all values (1000 for our project).

Mobility is the total number of moves that can be made by one's pieces in a given state. One possibility to combine material and mobility:

score = material + (0.1 * mobility)

Development measures how many pieces are moved away from their original positions. If your pieces are all "developed" (off their original squares), then they will

be ready for the ensuing battle which comes in the middle game. If you are badly behind in development, then you will be not ready for the struggle. We assume that three extra development moves are worth of approximately one pawn.

Finally central position measures the pieces in the center of the board. So if a player has more pieces in the center of the board then his/her opponent, he/she is more advantageous. For our project we exclude the king from this evaluation, because we do not want the king to be in the center of the board, since it should be secure.

When it is decided which features are included in the evaluation function, it is necessary to determine reasonable weightings for the features. Evaluation function takes form "w1 * f1 + w2 * f2 + w3 * f3 + …" where f's are various features and w's are the relative weights for these features. How to adjust weightings is a complicated issue, so we used an evolutionary algorithm with mutation to decide on better weightings.

3.3. Minimax Search

The basic idea underlying all two-agent search algorithms is **Minimax search**. Minimax search method for chess can be defined as follows:

- Assume that there is a way to evaluate a board position so that we know whether Player 1 (Max) is going to win, whether his opponent (Min) will, or whether the position will lead to a draw. This evaluation takes the form of a number: a positive number indicates that Max is leading, a negative number, that Min is ahead, and a zero, that nobody has acquired an advantage.
- Max's job is to make moves, which will increase the board's evaluation.
- Min's job is to make moves, which decrease the board's evaluation.
- Assume that both players play in a way that they never make any mistakes and always make the moves that improve their respective positions the most.

The trouble with Minimax is that there is an exponential number of possible paths which must be examined. This means that effort grows dramatically with:

- The number of possible moves by each player, called the branching factor.
- The depth of the look-ahead, and usually described as "N-ply", where N is an integer number and "ply" means one move by one player. The simplest approach we would follow is searching to a depth of 2-ply, one move per player.

Minimax Pseudocode:

```
function integer minimax(node, depth)
    if node is a terminal node or depth <= 0:
        return the heuristic value of node
    α = -∞
    for child in node:    # evaluation is identical for both players
        α = max(α, -minimax(child, depth-1))
    return α
```

3.4. Alpha-Beta Pruning Algorithm

The Alpha-Beta **Pruning** algorithm is a significant enhancement to the minimax search algorithm that eliminates the need to search large portions of the game tree. If one already has found a quite good move and search for alternatives, one refutation is enough to avoid it. No need to look for even stronger refutations. The algorithm maintains two values, alpha and beta. They represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively.

A general computer chess program consists of three main parts:
1) **Move generator**: Generates all possible moves in a given position
2) **Search function:** Looks at all possible moves and replies and try to find the best continuation.

3) **Position evaluator**. Gives a score to a position. It consists of a material, a mobility and a development score.

Here is the very general pseudocode-like order for a computer chess program:

1. Get the input as an opponent move from the user (P: player).
    a. Check whether this move is legal, prompt input again if not.
2. Generate a current board configuration based on the opponent's (P) last move.
3. Set this current state as a root node for search tree.
4. Generate all possible moves for C (C: computer) as a response to this input.
    a. Calculate the scores of each node using evaluation function.
5. For each of these moves, generate all possible user (P) responses.
    a. Calculate the scores of each node using evaluation function.
6. As we now have the search tree of depth 2, apply Minimax search algorithm with Alpha-Beta Pruning to choose the best move for computer (C).
7. Output the computer's move on the screen.
8. Check for situations like check, checkmate, draw etc. to end the game.
9. Repeat (1) to (8) until the game ends.

Alpha-Beta Pruning Pseudocode:

```
function alphabeta(node, depth, α, β, Player)
   if  depth = 0 or node is a terminal node
      return the heuristic value of node
   if  Player = MaxPlayer
      for each child of node
         α := max(α, alphabeta(child, depth-1, α, β, not(Player)))
         if β≤α
            break                    (* Beta cut-off *)
      return α
   else
      for each child of node
         β := min(β, alphabeta(child, depth-1, α, β, not(Player)))
         if β≤α
            break                    (* Alpha cut-off *)
```

```
    return β
(* Initial call *)
alphabeta(origin, depth, -infinity, +infinity, MaxPlayer)
```

3.5. Evolutionary Algorithm

The evolution algorithm with the mutation technique, works as follows: We will have "individuals" with different evaluation functions (same factors with different weights). There will be 5 individuals in a given population. They all compete with each other for survival. Two individuals (two chess programs using two different evaluation functions) play each other in a two round game where each one will take turns in starting the game first. Both the winner and its mutated version are copied into the next generation. This process continues until all individuals in the population converge to the same solution. The method of this evolutionary algorithm will be as shown below:

Member 1 vs. a random member (2 to 5) → Member 2 vs. a random member (3 to 5)
1 2 3 4 5 → **4** 2 3 **4** 5 →
Member 4 won and replaced member 1 Member 3 won and replaced member 2

Member 3 vs. a random member (4 to 5) → Member 4 vs. member 3
4 3 3 **4** 5 → **4 3 3 4** 3 →
Member 3 won and replaced member 5 Member 3 won and replaced member 4

The new population members
4 3 3 3 3

4. Development

4.1. Introduction

In this part, a more detailed explanation of the design of our project will be given. Before the implementation, with a goal of developing a complete functional chess program, we developed a model. The game is between two players, where one player is a computer (our program) playing against an opponent human player (the user).

4.2. Design

A game of chess involves chess pieces, and a chessboard. The chessboard is an 8 by 8 grid. The initial configuration of the pieces is as follows:

R	N	B	Q	K	B	N	R
P	P	P	P	P	P	P	P
P	P	P	P	P	P	P	P
R	N	B	Q	K	B	N	R

Where R=rook, N=knight, B=bishop, Q=queen, K=king, P=pawn

In our midterm progress report, we proposed a design in which each type of chess pieces is represented by a separate class. We planned to implement 6 different classes for that purpose as follows: pawn, rook, knight, bishop, queen and king. During implementation, this design choice has changed. We implemented 3 generic classes instead: Board, Move and Main. We will tell the details of these classes in the next sub section.

4.3. Classes

Here, we will summarize the general details of the classes in our object-oriented program.

4.3.1. Board.java class:

Our whole algorithm runs on a search tree as we discussed in the background section. The nodes of this tree are actually chessboards. So, to be able to construct these nodes, we created a class named Board. An object instance of this class is a board with different pieces, and with different positions.

The evaluation function, the minimax and alpha-beta pruning algorithms are all implemented in this class, because they concern board objects. For the evaluation function, many factors are taken into account, such as mobility, material, center, and development; these factors will be explained in detail later in this document. The methods that calculate the values for these factors are also all implemented in this class. The method “Play” is the most important method here, which is called from the main for the computer to make the move.

4.3.2. Move.java class:

Move is our smallest class. Every move is an object of this class. The input that is taken from the user is turned into a move object here. This move object then is given as a parameter to one of the constructors of the board class, with another parameter, the current board. This constructor creates the new board object. In other words, the move object is used to create a new board from the current board. There are also two constructors, and three other methods. One of them, “printMove” prints the move that is done either by the computer or the player.

4.3.3. Main.java class:

This is our main class. So the whole game is simulated here. The necessary inputs are taken from the user, the Boolean validity methods for both the inputs and the moves are called. Then the board constructor is called to create the new board after the user's move. The new board is printed in the format that is detailed in the "user interface" section. Then "Play" method is called, so the computer "plays". The evaluation, minimax, alpha-beta and all the methods that make the decision for the move are in the board class. This "Play" method returns the new board and then it is printed by "printBoard" method of the board class. Also if one of the players is in check, or it is a draw or a checkmate, it outputs these messages.

5. Outline of the Work Done

We developed a complete algorithm before we start the implementation. Our plan was to write methods for each piece, so the pieces can be moved according to the rules of chess. However, we changed this design along with the implementation and came up with 3 classes: "Board.java", "Move.java", "Main.java". The details of these classes will be explained in the implementation details section.

Our next step was to construct an evaluation function. As we have proposed in our midterm progress report, we constructed a new evaluation function. Obviously, we used the papers that are included in our literature survey as a reference and used the information there to construct a good function, but we did not use one that we have read in the papers or seen anywhere, instead we created a new one that may be similar to some of them. We had to write many methods before we could implement this evaluation function. Again the details of this evaluation function will be given in the implementation details section.

The next step was to develop the minimax algorithm and implement it for the program to play the game by using the evaluation function. This algorithm is known in game theory and artificial intelligence areas. Even though minimax is a very

common algorithm used in the two-player games programming, it was hard to understand it completely and apply it to our program. We developed a minimax algorithm particularly for chess and particularly with our evaluation function. We spent a lot of time here, because it was very challenging to implement the minimax with the data structures we have used.

Chess is a very complex game, and in many papers that we have read, it is mentioned that without a pruning technique the search tree for this game will be huge, so the program will either crash or run very slowly. That is because; the number of possible moves at each position is quite high. Therefore, we developed and implemented an algorithm that uses alpha-beta pruning technique for the search tree, which the minimax algorithm runs on. The minimax with alpha beta pruning is a very critical part of the project, because that is where the artificial intelligence lies beneath.

These were the main algorithms that we have developed and implemented. With these and all the other methods that we had to write for the implementation of these, we could write the code for the computer to play the game. A graphical user interface was not a requirement of our project, so we take the input from the command line as we stated in our midterm progress report. To be able to test the program, instead of just giving the computer's move to the user, we print a board composing of numbers each time a move is done, either by the computer or the user. The input and the output format will be given in the user interface section.

6. Model

As we stated above, a graphical user interface was not one of the requirements of our project. So we have a simple user interface where the player enters his/her moves in the command line and see the board after his/her move. Then the computer makes its own move and the board after its move is printed. Below the format for the input and the output can be seen.

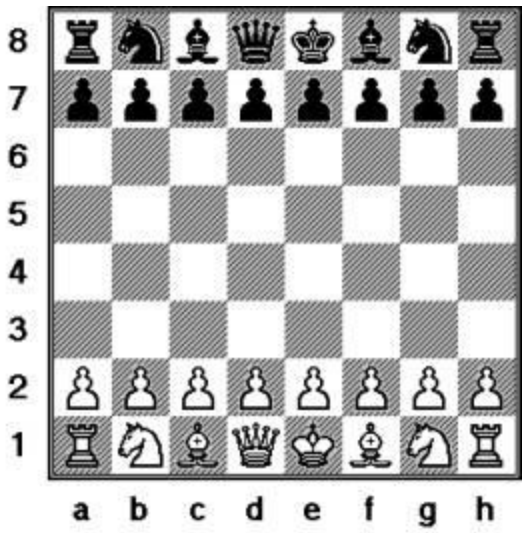
6.1. Input

First, the player chooses his/her side. The program asks the player to enter ‘w’ for white and ‘b’ for black. So the first input the user enters the program is ‘w’ or ‘b’. If the user enters w, he/she goes first to make the first move.

Since the program is a simulation of a chess playing person, the input is the opponent’s (the user in this case) move in written form. In other words, if I am playing with this program, I write my move when it is my turn and the program will continue playing accordingly. In order to test the program, we used an available chess program to be able to visualize the moves and the positions. The input is in the form:

R e4-g4

which means the rook is going to be moved from the square e4 to the square g4. The board can be seen below, to see the squares that are mentioned here:



For castling, the user enters “0-0” for castling on the king side, or “0-0-0” for castling on the queenside. The program checks for validity of all the moves, including castling. It checks the positions and if there is any piece in between. It also checks if the pieces in castling (rook and the king for the king side, rook and the queen on the queen side) have been moved before, since it would be an illegal move in that case. Therefore, checking the validity of castling is a bit more complicated than checking the validity of the other moves.

This input format made the testing process really hard and time consuming. To make it a little easier, we used a commercial chess playing game Windows-Titans and enter this commercial program's moves in the written form to our program. This made the process easier while also let us see how well our program performs, when played against another chess-playing program.

6.2. Output:

The output format we proposed in the midterm progress report was the same with the input format:

N c3-e4

However we changed it during the implementation. As we mentioned above, we print the board with numbers now. It looks like this:

```
    a  b  c  d  e  f  g  h
1: +R +N +B +Q +K +B +N +R
2: +P +P +P +P +P +P +P +P
3:  *  *  *  *  *  *  *  *
4:  *  *  *  *  *  *  *  *
5:  *  *  *  *  *  *  *  *
6:  *  *  *  *  *  *  *  *
7: -P -P -P -P -P -P -P -P
8: -R -N -B -Q -K -B -N -R
```

This is the notation for the initial board. So 1's stand for pawns, 2's stand for rooks, 3's stand for knights, 4's stand for bishops, 5's stand for queens and 6's stand for kings. The positive numbers are the pieces of the white player while the negative numbers are the pieces of the black. The board is printed this way right after each move.

If the user enters *N b1-c3*, for instance, then the program will output:

```
Current board after your move:

    a  b  c  d  e  f  g  h
1: +R  *  +B +Q +K +B +N +R
2: +P +P +P +P +P +P +P +P
3:  *  * +N  *  *  *  *  *
4:  *  *  *  *  *  *  *  *
5:  *  *  *  *  *  *  *  *
6:  *  *  *  *  *  *  *  *
7: -P -P -P -P -P -P -P -P
8: -R -N -B -Q -K -B -N -R

Computer's last move is:

Pawn(-1) from: e7(6,4) to: e5(4,4)

Current board after computer's move:

    a  b  c  d  e  f  g  h
1: +R  *  +B +Q +K +B +N +R
2: +P +P +P +P +P +P +P +P
3:  *  * +N  *  *  *  *  *
4:  *  *  *  *  *  *  *  *
5:  *  *  *  * -P  *  *  *
6:  *  *  *  *  *  *  *  *
7: -P -P -P -P  * -P -P -P
8: -R -N -B -Q -K -B -N -R

Enter your next move:
```

We can see that computer played its pawn at e7 to e5. This looks complicated at first, but it made it so much easier for us to follow the game and test it.

Then if the user enters *P e2-e4*, the output is:

```
Current board after your move:

    a  b  c  d  e  f  g  h
1: +R  *  +B +Q +K +B +N +R
2: +P +P +P +P  *  +P +P +P
3:  *  *  +N  *  *  *  *  *
4:  *  *  *  *  +P  *  *  *
5:  *  *  *  *  -P  *  *  *
6:  *  *  *  *  *  *  *  *
7: -P -P -P -P  *  -P -P -P
8: -R -N -B -Q -K -B -N -R


Current board after computer's move:

    a  b  c  d  e  f  g  h
1: +R  *  +B +Q +K +B +N +R
2: +P +P +P +P  *  +P +P +P
3:  *  *  +N  *  *  *  *  *
4:  *  *  *  *  +P  *  *  *
5:  *  *  *  *  -P  *  -Q  *
6:  *  *  *  *  *  *  *  *
7: -P -P -P -P  *  -P -P -P
8: -R -N -B  *  -K -B -N -R
```

There are specific outputs for specific points in the game too of course. When an invalid input or invalid move is entered, the program gives an error message accordingly. When the computer or the player is in check, it says “Computer is in CHECK!!!" or “Player is in CHECK!!!" When it is a checkmate, the program outputs:

"Computer is in CHECK MATE!!!"

"PLAYER WINS!!!\nGAME OVER!!!"

Finally if it is a draw it outputs:

"It is a DRAW!!!\nGAME OVER!!! "

It also prints the number of moves that have been played, which is very important for testing purposes.

6.3. Resources:

For the development of the algorithm of the implementation of the program, no external resources was used. However, for the testing part (both to test if the program works correctly and to test its success against other chess games) we used the chess game "Chess Titans" included in some versions of Microsoft Windows Vista/7. Since we did not develop a graphical user interface, it was very useful to visualize the moves better in "Chess Titans". Another resource was people when testing. We let some friends of ours play with our program, both to test its performance and to see outsiders' opinions and reactions. They did not find the interface very easy to use, but they got used to the notations after a while and played easily.

6.4. Success Criteria:

The success of the program is compared with those of other chess playing programs, by the description of the project. So the program was tested against "Chess Titans" of Microsoft Windows Vista and also real people. At first, we set our first success criteria as being able to play the game in a right manner and keep going at least for a while. This changed throughout the project. We performed good enough and we were able to implement the minimax algorithm with alpha-beta pruning and the evolutionary algorithm correctly. So now the success criteria is our program's performance against the other chess playing programs when they play two round games. In our progress report we wrote: "For now, we have to say, if our program can actually play chess and do it in an adaptive manner, it will be deemed as successful. If it wins any game towards a commercial chess program, and if it does it fast (that means our pruning technique should be working very well) then it will be regarded as over-successful." The test results will be included in the conclusion section of this document.

7. Implementation Details

7.1. Introduction

We will give the implementation details by simply explaining the methods in this section, but before that we would like to explain the development phase of the project. We will give answers to questions like where did we start from, in what order did we complete the parts, what were the most critical parts and which parts did we spend most of our time on.

As we have mentioned, the whole algorithm is based on a tree and the nodes of this tree are actually "chessboards". The root is the initial chessboard, before any move is made. For every possible move of the white player (there are 20 possible moves initially) different chessboards, which have pieces with different positions, a child of the root node is created. In other words, every child of the root stands for a possible chessboard that the white player can choose to go from the current chessboard. Then for all the children of the root, this process is repeated with the black player's possible moves. This happens in a recursive way. We will explain this process in more detail later in this document. For now, what we are trying to say is, we needed to create a "Board class" first of all, since the whole algorithm runs on the "boards." So that was what we did, we created a "board class". This is the biggest class in our program.

For every board object we needed some factors for the evaluation function. Since the program were going to play the game by evaluating the nodes (boards) in the search tree, we should have constructed an evaluation function, and to be able to do that, we needed some factors to be taken into account in this function. We take four aspects of the board into consideration while evaluating the boards:

- Material (pieces with their values)
- Mobility (number of possible moves)
- Development (number of pieces that have moved from their original positions)

- Center (number of pieces in the center of the board)

Actually, we have not read about or seen an evaluation function that includes these four factors together. Material, mobility and development are very common factors, while center is not used much. We tried to come up with a new and good evaluation function, and by trial and error, we finalized with an evaluation function that is based on these four vectors with different weights. It is calculated as shown below:

```
Val=material_w*Main.Wmat/40+  mobility_w*Main.Wmob/50
+ dev_w*Main.Wdev/16 + center_w*Main.Wcen/20;
```

Here the material_w, mobility_w, dev_w and center_w are the weights for the factors, which we determined in a mutational procedure. We scale the factors to [0,1] range so the weights are properly used. This is calculated for each player, both the white and the black, and then the final value we use is the difference of these values. For instance the value of a board for the black player is calculated as:

Val_black – Val_white

This strengthens the use of the function, because then the computer is trying to go to the board that it is advantageous and its opponent is disadvantageous.

Of course every board object has different values of these for both the white player and the black player, they are calculated for both, but in the game, only one player's values are used depending on whose turn it is. For instance, if it is the white player's turn, the evaluation is done for white's material, mobility, development and center values. Actually, it is more complex than this when it comes to the Minimax algorithm, but the details will be given later on.

The calculation methods for these four factors are included in Board class. The calculations for mobility are much more complicated than the others and it depends heavily on the pieces. So we have implemented different mobility methods for each piece (i.e. MobPawn, MobQueen etc.).

After implementing these methods, we implemented the evaluation function. These four factors have different weights in the function, which we determined by trial and error method. The board object has a field "value" that holds the numeric

value of the result of the evaluation function. These values are used in the Minimax with alpha-beta pruning procedure later on.

After the implementation of the evaluation function, we created the class "Move.java". One of the constructors of this method takes the user input as a parameter and creates a move. Then by one of the constructors of the board class, a new board is created.

The next thing we did was the implementation of Minimax and alpha-beta pruning, so the computer can give a response with a move.

The code in the main was written in parallel with all the work, so we went by testing each part of the program. Now we will explain the implementation details by the code, by taking the classes and methods one by one.

7.2. Board.java

7.2.1. Board( )

This is the default constructor.

7.2.2. Board(Board par, Move move)

This constructor is used to create a new board from the old board with the last move.

7.2.3. printBoardInfo()

Prints some valuable information about the board for our test purposes.

7.2.4. centeral(int side)

This method is used to calculate the value for the center factor of the evaluation function. The side here is the side of our computer (whether it is black or white). All of the factors are calculated with this parameter, because a value of the board for the white player and the black player is very different obviously.

7.2.5. material(int side)

This method is used to calculate the material value of a board object. It sums up the values of the pieces a player has. Again

the side is taken as the parameter to calculate the value accordingly.

7.2.6. dev(int side)

The “development” factor for the evaluation function is calculated here. It checks for each piece if it is in its original position or not for this calculation.

7.2.7. enPassant(int side)

This method implements the rule “En Passant”. The other rules were implemented in the mobility methods for each piece. However “En Passant” is a very particular situation with very different conditions, so it is implemented separately.

7.2.8. mobPawn(int side)

This method calculates pawn mobility and generates pawn moves by implementing the rules of pawn. Since pawn moves change accordingly if it is a capture or not, this is a bit more complicated than the other mobility methods.

7.2.9. mobKnight(int side)

This method calculates knight mobility and generates knight moves. Since mobility is the number of moves that the player can do at the current board, it takes side as a parameter and adds the possible moves to the moves list while also returning the number of moves the knights can do. In fact this is what is done in all the mobility methods that will be told next.

7.2.10. mobBishop(int side)

This method calculates bishop mobility and generates bishop moves.

7.2.11. mobRook(int side)

This method calculates rook mobility and generates rook moves.

7.2.12. mobQueen(int side)

This method calculates queen mobility and generates queen moves.

7.2.13. mobKing(int side)

This method calculates king mobility and generates king moves. The castling rules are also implemented here and added to the mobility value if the conditions for castling are provided.

7.2.14. printMoves()

This method is used for testing purposes. It prints all the possible moves with their values, so we could check if the computer chooses the best move.

7.2.15. set_mobility()

This is the method that calculates the final value of mobility by summing up all the values that are calculated in the previous mobility methods. So basically, the value this method returns is taken into account in the evaluation function.

7.2.16. isUnderAttack(int x, int y, int side)

This method checks if a square is under any threats. “x” and “y” are the coordinates of that square.

7.2.17. isCastling(int side, int type)

This Boolean method checks if the board is okay for castling. The “type” parameter determines if the castling we are checking for validity is on the queenside or the kingside.

7.2.18. printBoard()

This method prints the board as explained in the output part.

7.2.19. generateChildren()

Generates all possible boards from the current board both for the white player and the black player. All the possible moves with the current board is given to the method Board (Board par, Move move) to create the possible boards, which are the children of the current board in our tree.

7.2.20. moveIsValid(Move m)

This method traverses the children of the current node to check if the given move is one of them. Otherwise, the move is not valid and an error message is given to the user.

7.2.21. eval(int side)

This method is where the evaluation method is implemented with all the weights and the factors.

7.2.22. equals (Board b)

7.2.23. copy(Board b)

7.2.24. loadBoard()

After copying the new board as our current board, we need to load all the new values too. This method loads the values to the current board.

7.2.25. isInCheck(int side)

This is a Boolean method that checks if the player (white if side is 1, black if side is 0) is in check.

7.2.26. isDraw( )

Checks for the situations that the game would end as draw.

7.2.27. isCheckMate(int side)

Checks if the current board is a checkmate situation.

7.2.28. Max( )

7.2.29. Min()

7.2.30. Play()

This is the method that calls Minimax_AB and and returns the best move that is found by the minimax algorithm.

7.2.31. Minimax_AB (int side, int turn, Board current, double alpha, double beta)

This is one of the most critical methods in our program. The minimax algorithm is implemented in this method with the alpha-beta pruning technique. The algorithm is already explained in the algorithm part. It works recursively. Even though the algorithm is common and easy to find, we wrote the whole method from scratch. Our data structure is very different, especially because it is separating the operations for the white player and the black player each and every time. The algorithm is dependent on the side of the player, the board we call this method from and whose turn it is, so they are given as parameters to the method. The "alpha" and "beta" values are required for the alpha-beta pruning part. This is the method we spent most of our time on.

7.3. Move.java

7.3.1. Move ( )

Default constructor

7.3.2. Move (int x, int y, int x2, int y2, int p)

This constructor is used to create a move with the coordinates of the squares that it goes from and it goes to. The parameter "p" is for the piece that is involved in this move.

7.3.3. equals(Move m)

7.3.4. copy(Move m)

7.3.5. Move(String input)

Converts the user input into a move object with the squares and the piece.

7.3.6. printMove()

This method is used for printing the last move, in order to make it easier for the users to track it.

7.4. Main.java

7.4.1. main(String[] args)

Needless to say, this is the main method that the whole game runs in.

7.4.2. inputValid(String input)

This method is a Boolean method that checks the validity of the input. If the input is entered in an incorrect format, the program does not check if the move is valid. This is important, because the method that check the validity of the moves “moveIsValid” traverses the search tree, which would lower the performance if we call it when it is not required.

8. What Is Done?

8.1. Introduction:

As the instructor suggested, we have been following the procedure, which is explained in the Project Progress Document for this project. All required phases of the project are indicated in this document as follows:

- Literature Survey
- Proposal
- Implementation
- Presentation

Based on our plan, we have finished Literature Survey phase (4 weeks) and Proposal phase (3 weeks) before Spring Break. After Spring Break, we completed the implementation in 6 weeks and spent almost a week for the poster and the final report, which are included in the presentation phase.

In addition to the documents prepared and submitted during the literature survey and proposal phases, we also filled the Project Progress Chart showing progress in the project. This chart keeps track of dates, hours worked, progress (this week) and plan (next week); thus we were supposed to fill and revise this document

every week, and then send it to the instructor. You can see the latest version of our chart in the Appendix section.

Let us explain all the work done in the first two phases in more detail:

8.2. Phases:

1.2.1. Literature Survey Phase:

This first phase was a study of the existing work in the literature in order to familiarize with the topic. After this phase, we increased our awareness of what the topic is, what is done, and how it is done.

- We have read 9 conference and journal papers related to our project topic, which were assigned by our project advisor at the very beginning.
- Then we have also looked for additional papers, book chapters and other explanatory sources on the web to become more familiarized with our topic.
- We kept a list of the papers we have read in a document PaperList. For each paper, the following information were required: title, author(s), conference/journal/book name, volume no, pages, year, and abstract. We prepared this document and submitted to the project advisor.
- We have prepared an abstract of one page for each paper as soon as we read that paper. This was not the same thing as the abstract that appears on the paper. Instead, we needed to show how the work was done (i.e. the details of the algorithms in a pseudocode-like notation), how it was evaluated (the formulas and steps of the algorithm for evaluation), the format of the input and output, the resources used, the assumptions and restrictions for the algorithms, etc. All the abstracts were kept in a single document PaperSummary. We prepared that document too and sent it to the instructor.
- At the end of the literature survey phase, we started to prepare a literature survey report LiteratureSurvey. This report was different from the document PaperSummary. It should have combined all the information gathered.

8.2.2. Proposal Phase:

The next phase was preparing a project proposal document, of about 10 pages long, by combining the background information (literature survey) and the work that will be done in the scope of the project. The proposal shows explicitly what will be done and how it will be done. It will serve as a general plan until the end of the project and we will continue the project in concordance with this plan.

Based on all these information, we have prepared a Project Proposal Document and submitted it to the instructor. The contents of our proposal document were as follows:

- Project overview: This section explained the project as a whole. It consists of the following subsections:
  - Description
  - Input
  - Output
  - Resources
  - Process
- Project methodology: This section explained all the details of what will be done and how it will be done. It consists of the following subsections:
  - High-level description
  - Middle-level description
  - Low-level description
- Success criteria: This section explained under what conditions the project will be evaluated as successful. The success criteria should be determined according to the work in the literature. In addition, under what conditions the system will be regarded as over-successful should be mentioned.

8.2.3. Implementation Phase:

This phase includes the coding of the program along with testing. The source codes are added to this document as an appendix and the test results can be seen in the conclusion section.

8.2.4. Presentation Phase:

The preperation of the poster and the writing of this document are components of this phase. Of course the poster presentation itself, which will be done on June 10th, is also included in presentation phase.

9. Conclusion

9.1. Test Results

As we have stated in some of the above sections, we used Windows-Titans to test the program against. We tested it with against different levels of Windows-Titans and by two round games. We will give the results according to the average results:

- **1st level:** The computer wins in 21 plys. → WIN!
- **2nd level:** The computer wins in 21 plys. → WIN!
- **3rd level:** The computer wins in 27 plys. → WIN!
- **4th level:** Draw because of 50-move limit → DRAW or LOSS
  Windows-Titans wins in 39 plys.
- **5th level:** Windows-Titans wins in 20 plys. →LOSS

A ply means two moves, each one done by opponents. So when the white player plays once and the black player plays once, it is a ply.

For the testing part to be also useful, we wanted some help from our friends to play and test the game. Their feedbacks were really important to us, and we used those feedbacks to improve some points about the program. Letting some people outside of the development team play the game, let us see some details we could not see.

9.2. What did we get?

We put a lot of time and effort into this project and we learned so much from it. We learned a lot about artificial intelligence, algorithm performance and even chess. We also experienced intense pair programming and testing processes.

10. Future Work

We have done research, constructed the algorithms, implemented the project and done testing as we have stated as “future work” in our midterm progress report. So now, we have a complete project. However, of course many improvements can be done. Because we had a memory problem, we could only increase the depth as much as 3. If the computer can see further, the level of the game would go higher, but then the complexity would increase exponentially and this would lower the performance. In other words there is a trade-off between the performance of the program and the level of chess (how far the computer can see). The optimization for this can be done as an improvement. Also, the evolutionary part can be improved. Other techniques can be used for making it adaptive as well. So the future work may include the testing of different techniques for the adaptive property. This way the best evaluation function can be found for the best chess-playing program.

APPENDIX A: SOURCE CODES

Board.java:

```
package my_chess_pkg;
import java.lang.Math;
import java.util.List;
import java.util.ArrayList;


/**
 *
 * @author eda & selin
 */

public class Board {
    int[][] B;
    double value; //score of a board by evaluation function
    double value_w;
    double value_b;
    int material_w; // white(+)
    int material_b; // black(-)
    int mobility_w;
    int mobility_b;
    int dev_w; // development
    int dev_b;
    int center_w;
    int center_b;
    int[][] devArr; // development array -> 0: not moved, 1: moved
    int side; // 1: white, -1: black
    public List<Move> moves_w; //all possible moves of white
    public List<Move> moves_b;
    public List<Board> children_w; //children boards of white
    public List<Board> children_b;
    Board parent;
    double alpha;
    double beta;

    public Board(){
        B= new int[][]
            {{2,3,4,5,6,4,3,2},
             {1,1,1,1,1,1,1,1},
             {0,0,0,0,0,0,0,0},
             {0,0,0,0,0,0,0,0},
             {0,0,0,0,0,0,0,0},
             {0,0,0,0,0,0,0,0},
             {-1,-1,-1,-1,-1,-1,-1,-1},
             {-2,-3,-4,-5,-6,-4,-3,-2}};
        devArr=new int[4][8];
        for(int i=0; i<4; i++){
            for(int j=0; j<8; j++){
                devArr[i][j]=0;
            }
        }
        dev_w=dev(1);
        dev_b=dev(-1);
        material_w=material(1);
        material_b=material(-1);
        center_w=centeral(1);
        center_b=centeral(-1);

        moves_w = new ArrayList<Move>();
        moves_b = new ArrayList<Move>();
        children_w = new ArrayList<Board>();
        children_b = new ArrayList<Board>();
```

```
        parent=null;

        this.set_mobility();
        this.generateChildren();

        value_w=eval(1);
        value_b=eval(-1);
        if(Main.sideC==1) value=value_w-value_b;
        else value=value_b-value_w;
    }

    public Board(Board par, Move move){ // create a new board = old board +
last move
        this.parent = par;
        B = new int[8][8];
        devArr=new int[4][8];

        if(move.piece!=0){ //not a castling move
        for(int i=0; i<8; i++){
            for(int j=0; j<8; j++){
                if(i==move.to_x && j==move.to_y){
                    B[i][j]=move.piece; // fill in new position
                    //pawn promotion to queen
                    if(move.piece==1 && i==7) B[i][j]=5;
                    else if(move.piece==-1 && i==0) B[i][j]=-5;
                    //en passant move
                    if(move.piece=='p'){
                        if(i==5) B[i][j]=1;
                        if(i==2) B[i][j]=-1;
                    }
                }
                else if(i==move.from_x && j==move.from_y){
                    B[i][j]=0; // empty the old position
                }
                else{
                    B[i][j]=par.B[i][j];
                }
                if(move.piece=='p'){
                    B[move.from_x][move.to_y]=0;
                }
            }
        }
        //set development array
        for(int j=0; j<8; j++){
            for(int i=0; i<2; i++){ //white
                if(i==move.from_x && j==move.from_y) devArr[i][j]=1;
                else devArr[i][j]=par.devArr[i][j];
            }
            for(int i=2; i<4; i++){ //black
                if(i+4==move.from_x && j==move.from_y) devArr[i][j]=1;
                else devArr[i][j]=par.devArr[i][j];
            }
        }
        }

        else if(move.piece==0){ //castling move
            for(int i=0; i<8; i++){
                for(int j=0; j<8; j++){
                    B[i][j]=par.B[i][j];
                    if(i<4) devArr[i][j]=par.devArr[i][j];
                }
            }//white castling on queenside
            if(move.from_x==0 && move.from_y==4 && move.to_x==0 &&
move.to_y==2){
                B[0][4]=0; B[0][2]=6; B[0][0]=0; B[0][3]=2; devArr[0][0]=1;
devArr[0][4]=1;
            }//white castling on kingside
```

```
            else if(move.from_x==0 && move.from_y==4 && move.to_x==0 &&
move.to_y==6){
                B[0][4]=0; B[0][6]=6; B[0][7]=0; B[0][5]=2; devArr[0][4]=1;
devArr[0][7]=1;
            }//black castling on queenside
            else if(move.from_x==7 && move.from_y==4 && move.to_x==7 &&
move.to_y==2){
                B[7][4]=0; B[7][2]=-6; B[7][0]=0; B[7][3]=-2;
devArr[3][0]=1; devArr[3][4]=1;
            }//black castling on kingside
            else if(move.from_x==7 && move.from_y==4 && move.to_x==7 &&
move.to_y==6){
                B[7][4]=0; B[7][6]=-6; B[7][7]=0; B[7][5]=-2;
devArr[3][4]=1; devArr[3][7]=1;
            }
        }

        dev_w=dev(1);
        dev_b=dev(-1);
        material_w=material(1);
        material_b=material(-1);
        center_w=centeral(1);
        center_b=centeral(-1);

        moves_w = new ArrayList<Move>();
        moves_b = new ArrayList<Move>();
        children_w = new ArrayList<Board>();
        children_b = new ArrayList<Board>();

        this.set_mobility();
        //this.generateChildren();
        value_w=eval(1);
        value_b=eval(-1);
        if(Main.sideC==1) value=value_w-value_b;
        else value=value_b-value_w;
    }

    void printBoardInfo(){
        System.out.println("Material of white: " + material_w);
        System.out.println("Material of black: " + material_b);
        System.out.println("Mobility of white: " + mobility_w);
        System.out.println("Mobility of black: " + mobility_b);
        System.out.println("Development of white: " + dev_w);
        System.out.println("Development of black: " + dev_b);
        System.out.println("Central value of white: " + center_w);
        System.out.println("Central value of black: " + center_b);
        System.out.println("Value of white: " + value_w);
        System.out.println("Value of black: " + value_b);
        System.out.println("VALUE: " + value);
        System.out.println("Number of white children: " +
children_w.size());
        System.out.println("Number of white moves: " + moves_w.size());
        System.out.println("Number of black children: " +
children_b.size());
        System.out.println("Number of black moves: " + moves_b.size());
        if(parent==null) System.out.println("Parent is NULL");
    }


    int material(int side){ //calculate material values of board
        int m=0;
        if(side==1){ //white
            for(int i=0; i<8; i++){
                for(int j=0; j<8; j++){
                    if(B[i][j]==1){m+=1;} //pawn
                    else if(B[i][j]==2){m+=5;} //rook
                    else if(B[i][j]==3 || B[i][j]==4){m+=3;} //knight or
bishop
```

```
                    else if(B[i][j]==5){m+=9;} //queen
                    else if(B[i][j]==6){m+=1000;} //king
                }
            }
        }
        else if (side==-1){ //black
            for(int i=0; i<8; i++){
                for(int j=0; j<8; j++){
                    if(B[i][j]==-1){m+=1;} //pawn
                    else if(B[i][j]==-2){m+=5;} //rook
                    else if(B[i][j]==-3 || B[i][j]==-4){m+=3;} //knight or
bishop
                    else if(B[i][j]==-5){m+=9;} //queen
                    else if(B[i][j]==-6){m+=1000;} //king
                }
            }
        }
        return m;
    }

    int dev(int side){ //calculate development values of board
        int d=0;
        if(side==1){ //white
            for(int i=0; i<2; i++){
                for(int j=0; j<8; j++){
                    if(devArr[i][j]==1) d++;
                }
            }
        }
        else if(side==-1){ //black
            for(int i=2; i<4; i++){
                for(int j=0; j<8; j++){
                    if(devArr[i][j]==1) d++;
                }
            }
        }
        return d;
    }

    int centeral(int side){ //calculate centeral values of board
        int c=0;
        if(side==1){ //white
            for(int i=2; i<6; i++){
                for(int j=2; j<6; j++){
                    if(B[i][j]==1) c+=1;
                    else if(B[i][j]==2) c+=3;
                    else if(B[i][j]==3 || B[i][j]==4) c+=2;
                    else if(B[i][j]==5) c+=4;
                }
            }
            for(int i=3; i<5; i++){
                for(int j=3; j<5; j++){
                    if(B[i][j]==1) c+=1;
                    else if(B[i][j]==2) c+=3;
                    else if(B[i][j]==3 || B[i][j]==4) c+=2;
                    else if(B[i][j]==5) c+=4;
                }
            }
        }
        else if (side==-1){ //black
            for(int i=2; i<6; i++){
                for(int j=2; j<6; j++){
                    if(B[i][j]==-1) c+=1;
                    else if(B[i][j]==-2) c+=3;
                    else if(B[i][j]==-3 || B[i][j]==-4) c+=2;
                    else if(B[i][j]==-5) c+=4;
                }
            }
```

```
            for(int i=3; i<5; i++){
                for(int j=3; j<5; j++){
                    if(B[i][j]==-1) c+=1;
                    else if(B[i][j]==-2) c+=3;
                    else if(B[i][j]==-3 || B[i][j]==-4) c+=2;
                    else if(B[i][j]==-5) c+=4;
                }
            }
        }
        return c;
    }

    //calculate number of possible en passant moves && generate these moves
    int enPassant(int side){
        if(parent==null) return 0;
        int mp=0;
        if(side==1){ //white
            for(int j=0; j<8; j++){
                if(B[4][j]==1){
                    if(j!=0 && B[4][j-1]==-1){
                        Move m = new Move(6,j-1,4,j-1,-1);
                        for(int i=0; i<parent.moves_b.size(); i++){
                            if(parent.moves_b.get(i).equals(m)){
                                Board b = new Board
(parent,parent.moves_b.get(i));
                                //for(int k=0; k<parent.children_b.size();
k++){

//if(parent.children_b.get(k).equals(this)){
                                    if(b.equals(this)){
                                        mp++;
                                        Move m2= new Move(4,j,5,j-1,'p');
                                        moves_w.add(m2);
                                    }
                                //}
                            }
                        }
                    }
                    if(j!=7 && B[4][j+1]==-1){
                        Move m= new Move(6,j+1,4,j+1,-1);
                        for(int i=0; i<parent.moves_b.size(); i++){
                            if(parent.moves_b.get(i).equals(m)){
                                Board b = new Board
(parent,parent.moves_b.get(i));
                                //for(int k=0; k<parent.children_b.size();
k++){

//if(parent.children_b.get(k).equals(this)){
                                    if(b.equals(this)){
                                        mp++;
                                        Move m2= new Move(4,j,5,j+1,'p');
                                        moves_w.add(m2);
                                    }
                                //}
                            }
                        }
                    }
                }
            }
        }
        else{ //black
            for(int j=0; j<8; j++){
                if(B[3][j]==-1){
                    if(j!=0 && B[3][j-1]==1){
                        Move m= new Move(1,j-1,3,j-1,1);
                        for(int i=0; i<parent.moves_w.size(); i++){
                            if(parent.moves_w.get(i).equals(m)){
```

```
                                Board b = new Board
(parent,parent.moves_w.get(i));
                                //for(int k=0; k<parent.children_w.size();
k++){

//if(parent.children_w.get(k).equals(this)){
                                    if(b.equals(this)){
                                        mp++;
                                        Move m2= new Move(3,j,2,j-1,'p');
                                        moves_b.add(m2);
                                    }
                                //}
                            }
                        }
                        if(j!=7 && B[3][j+1]==1){
                            Move m= new Move(1,j+1,3,j+1,1);
                            for(int i=0; i<parent.moves_w.size(); i++){
                                if(parent.moves_w.get(i).equals(m)){
                                Board b = new Board
(parent,parent.moves_w.get(i));
                                //for(int k=0; k<parent.children_w.size();
k++){

//if(parent.children_w.get(k).equals(this)){
                                    if(b.equals(this)){
                                        mp++;
                                        Move m2= new Move(3,j,2,j+1,'p');
                                        moves_b.add(m2);
                                    }
                                //}
                            }
                        }
                    }
                }
            }
        }
        return mp;
    }

    int mobPawn(int side){ //calculate pawn mobility & generate pawn moves
        int mp=0;
        //mp += enPassant(side); //find en passant moves if possible
        if(side==1){ //white
            for(int i=0; i<8; i++){
                for(int j=0; j<8; j++){
                    if(B[i][j]==1){
                        if(i!=7 && B[i+1][j]==0) {
                            mp++;
                            Move m = new Move(i,j,i+1,j,1);
                            moves_w.add(m);
                        }
                        if(i==1 && B[i+1][j]==0 && B[i+2][j]==0) {
                            mp++;
                            Move m = new Move(i,j,i+2,j,1);
                            moves_w.add(m);
                        }
                        if(i!=7 && j!=7 && B[i+1][j+1]<0) {
                            mp++;
                            Move m = new Move(i,j,i+1,j+1,1);
                            moves_w.add(m);
                        }
                        if(i!=7 && j!=0 && B[i+1][j-1]<0) {
                            mp++;
                            Move m = new Move(i,j,i+1,j-1,1);
                            moves_w.add(m);
                        }
                    }
```

```
                }
            }
        }
        else if(side==-1){ //black
            for(int i=0; i<8; i++){
                for(int j=0; j<8; j++){
                    if(B[i][j]==-1){
                        if(i!=0 && B[i-1][j]==0) {
                            mp++;
                            Move m = new Move(i,j,i-1,j,-1);
                            moves_b.add(m);
                        }
                        if(i==6 && B[i-1][j]==0 && B[i-2][j]==0) {
                            mp++;
                            Move m = new Move(i,j,i-2,j,-1);
                            moves_b.add(m);
                        }
                        if(i!=0 && j!=7 && B[i-1][j+1]>0) {
                            mp++;
                            Move m = new Move(i,j,i-1,j+1,-1);
                            moves_b.add(m);
                        }
                        if(i!=0 && j!=0 && B[i-1][j-1]>0) {
                            mp++;
                            Move m = new Move(i,j,i-1,j-1,-1);
                            moves_b.add(m);
                        }
                    }
                }
            }
        }
        return mp;
    }


    int mobKnight(int side){ //calculate knight mobility & generate knight
moves
        int mn=0;
        for(int i=0; i<8; i++){
            for(int j=0; j<8; j++){
                if(side==1 && B[i][j]==3){ //white
                    if(i<7 && j<6 && B[i+1][j+2]<=0) {mn++;Move m=new
Move(i,j,i+1,j+2,3);moves_w.add(m);}
                    if(i<7 && j>1 && B[i+1][j-2]<=0) {mn++;Move m=new
Move(i,j,i+1,j-2,3);moves_w.add(m);}
                    if(i>0 && j<6 && B[i-1][j+2]<=0) {mn++;Move m=new
Move(i,j,i-1,j+2,3);moves_w.add(m);}
                    if(i>0 && j>1 && B[i-1][j-2]<=0) {mn++;Move m=new
Move(i,j,i-1,j-2,3);moves_w.add(m);}
                    if(i<6 && j<7 && B[i+2][j+1]<=0) {mn++;Move m=new
Move(i,j,i+2,j+1,3);moves_w.add(m);}
                    if(i<6 && j>0 && B[i+2][j-1]<=0) {mn++;Move m=new
Move(i,j,i+2,j-1,3);moves_w.add(m);}
                    if(i>1 && j<7 && B[i-2][j+1]<=0) {mn++;Move m=new
Move(i,j,i-2,j+1,3);moves_w.add(m);}
                    if(i>1 && j>0 && B[i-2][j-1]<=0) {mn++;Move m=new
Move(i,j,i-2,j-1,3);moves_w.add(m);}
                }
                else if(side==-1 && B[i][j]==-3){ //black
                    if(i<7 && j<6 && B[i+1][j+2]>=0) {mn++;Move m=new
Move(i,j,i+1,j+2,-3);moves_b.add(m);}
                    if(i<7 && j>1 && B[i+1][j-2]>=0) {mn++;Move m=new
Move(i,j,i+1,j-2,-3);moves_b.add(m);}
                    if(i>0 && j<6 && B[i-1][j+2]>=0) {mn++;Move m=new
Move(i,j,i-1,j+2,-3);moves_b.add(m);}
                    if(i>0 && j>1 && B[i-1][j-2]>=0) {mn++;Move m=new
Move(i,j,i-1,j-2,-3);moves_b.add(m);}
```

```
                    if(i<6 && j<7 && B[i+2][j+1]>=0) {mn++;Move m=new
Move(i,j,i+2,j+1,-3);moves_b.add(m);}
                    if(i<6 && j>0 && B[i+2][j-1]>=0) {mn++;Move m=new
Move(i,j,i+2,j-1,-3);moves_b.add(m);}
                    if(i>1 && j<7 && B[i-2][j+1]>=0) {mn++;Move m=new
Move(i,j,i-2,j+1,-3);moves_b.add(m);}
                    if(i>1 && j>0 && B[i-2][j-1]>=0) {mn++;Move m=new
Move(i,j,i-2,j-1,-3);moves_b.add(m);}
                }
            }
        }
        return mn;
    }

    int mobBishop(int side){ //calculate bishop mobility & generate bishop
moves
        int mb=0;
        for(int i=0; i<8; i++){
            for(int j=0; j<8; j++){
                if(side==1 && B[i][j]==4){ //white
                    int k=Math.min(7-i,7-j);
                    for(int l=1; l<=k; l++){
                        if(B[i+l][j+l]==0) {mb++;Move m=new
Move(i,j,i+l,j+l,4);moves_w.add(m);}
                        else if(B[i+l][j+l]<0) {mb++;Move m=new
Move(i,j,i+l,j+l,4);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    k=Math.min(i,j);
                    for(int l=1; l<=k; l++){
                        if(B[i-l][j-l]==0) {mb++;Move m=new Move(i,j,i-l,j-
l,4);moves_w.add(m);}
                        else if(B[i-l][j-l]<0) {mb++;Move m=new Move(i,j,i-
l,j-l,4);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    k=Math.min(7-i,j);
                    for(int l=1; l<=k; l++){
                        if(B[i+l][j-l]==0) {mb++;Move m=new Move(i,j,i+l,j-
l,4);moves_w.add(m);}
                        else if(B[i+l][j-l]<0) {mb++;Move m=new
Move(i,j,i+l,j-l,4);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    k=Math.min(i,7-j);
                    for(int l=1; l<=k; l++){
                        if(B[i-l][j+l]==0) {mb++;Move m=new Move(i,j,i-
l,j+l,4);moves_w.add(m);}
                        else if(B[i-l][j+l]<0) {mb++;Move m=new Move(i,j,i-
l,j+l,4);moves_w.add(m); l=k;}
                        else l=k;
                    }
                }
                else if(side==-1 && B[i][j]==-4){ //black
                    int k=Math.min(7-i,7-j);
                    for(int l=1; l<=k; l++){
                        if(B[i+l][j+l]==0) {mb++;Move m=new
Move(i,j,i+l,j+l,-4);moves_b.add(m);}
                        else if(B[i+l][j+l]>0) {mb++;Move m=new
Move(i,j,i+l,j+l,-4);moves_b.add(m); l=k;}
                        else l=k;
                    }
                    k=Math.min(i,j);
                    for(int l=1; l<=k; l++){
                        if(B[i-l][j-l]==0) {mb++;Move m=new Move(i,j,i-l,j-
l,-4);moves_b.add(m);}
                        else if(B[i-l][j-l]>0) {mb++;Move m=new Move(i,j,i-
l,j-l,-4);moves_b.add(m); l=k;}
```

```
                        else l=k;
                    }
                    k=Math.min(7-i,j);
                    for(int l=1; l<=k; l++){
                        if(B[i+l][j-l]==0) {mb++;Move m=new Move(i,j,i+l,j-
l,-4);moves_b.add(m);}
                        else if(B[i+l][j-l]>0) {mb++;Move m=new
Move(i,j,i+l,j-l,-4);moves_b.add(m); l=k;}
                        else l=k;
                    }
                    k=Math.min(i,7-j);
                    for(int l=1; l<=k; l++){
                        if(B[i-l][j+l]==0) {mb++;Move m=new Move(i,j,i-
l,j+l,-4);moves_b.add(m);}
                        else if(B[i-l][j+l]>0) {mb++;Move m=new Move(i,j,i-
l,j+l,-4);moves_b.add(m); l=k;}
                        else l=k;
                    }
                }
            }
        }
        return mb;
    }


    int mobRook(int side){ //calculate rook mobility & generate rook moves
        int mr=0;
        for(int i=0; i<8; i++){
            for(int j=0; j<8; j++){
                if(side==1 && B[i][j]==2){ //white
                    int k=7-i;
                    for(int l=1; l<=k; l++){
                        if(B[i+l][j]==0) {mr++;Move m=new
Move(i,j,i+l,j,2);moves_w.add(m);}
                        else if(B[i+l][j]<0) {mr++;Move m=new
Move(i,j,i+l,j,2);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    k=i;
                    for(int l=1; l<=k; l++){
                        if(B[i-l][j]==0) {mr++;Move m=new Move(i,j,i-
l,j,2);moves_w.add(m);}
                        else if(B[i-l][j]<0) {mr++;Move m=new Move(i,j,i-
l,j,2);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    k=7-j;
                    for(int l=1; l<=k; l++){
                        if(B[i][j+l]==0) {mr++;Move m=new
Move(i,j,i,j+l,2);moves_w.add(m);}
                        else if(B[i][j+l]<0) {mr++;Move m=new
Move(i,j,i,j+l,2);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    k=j;
                    for(int l=1; l<=k; l++){
                        if(B[i][j-l]==0) {mr++;Move m=new Move(i,j,i,j-
l,2);moves_w.add(m);}
                        else if(B[i][j-l]<0) {mr++;Move m=new Move(i,j,i,j-
l,2);moves_w.add(m); l=k;}
                        else l=k;
                    }
                }
                else if(side==-1 && B[i][j]==-2){ //black
                    int k=7-i;
                    for(int l=1; l<=k; l++){
                        if(B[i+l][j]==0) {mr++;Move m=new Move(i,j,i+l,j,-
2);moves_b.add(m);}
```

```
                        else if(B[i+l][j]>0) {mr++;Move m=new
Move(i,j,i+l,j,-2);moves_b.add(m); l=k;}
                        else l=k;
                    }
                    k=i;
                    for(int l=1; l<=k; l++){
                        if(B[i-l][j]==0) {mr++;Move m=new Move(i,j,i-l,j,-
2);moves_b.add(m);}
                        else if(B[i-l][j]>0) {mr++;Move m=new Move(i,j,i-
l,j,-2);moves_b.add(m); l=k;}
                        else l=k;
                    }
                    k=7-j;
                    for(int l=1; l<=k; l++){
                        if(B[i][j+l]==0) {mr++;Move m=new Move(i,j,i,j+l,-
2);moves_b.add(m);}
                        else if(B[i][j+l]>0) {mr++;Move m=new
Move(i,j,i,j+l,-2);moves_b.add(m); l=k;}
                        else l=k;
                    }
                    k=j;
                    for(int l=1; l<=k; l++){
                        if(B[i][j-l]==0) {mr++;Move m=new Move(i,j,i,j-l,-
2);moves_b.add(m);}
                        else if(B[i][j-l]>0) {mr++;Move m=new Move(i,j,i,j-
l,-2);moves_b.add(m); l=k;}
                        else l=k;
                    }
                }
            }
        }
        return mr;
    }


    int mobQueen(int side){ //calculate queen mobility & generate queen
moves
        int mq=0;
        for(int i=0; i<8; i++){
            for(int j=0; j<8; j++){
                if(side==1 && B[i][j]==5){ //white
                    //rook moves
                    int k=7-i;
                    for(int l=1; l<=k; l++){
                        if(B[i+l][j]==0) {mq++;Move m=new
Move(i,j,i+l,j,5);moves_w.add(m);}
                        else if(B[i+l][j]<0) {mq++;Move m=new
Move(i,j,i+l,j,5);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    k=i;
                    for(int l=1; l<=k; l++){
                        if(B[i-l][j]==0) {mq++;Move m=new Move(i,j,i-
l,j,5);moves_w.add(m);}
                        else if(B[i-l][j]<0) {mq++;Move m=new Move(i,j,i-
l,j,5);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    k=7-j;
                    for(int l=1; l<=k; l++){
                        if(B[i][j+l]==0) {mq++;Move m=new
Move(i,j,i,j+l,5);moves_w.add(m);}
                        else if(B[i][j+l]<0) {mq++;Move m=new
Move(i,j,i,j+l,5);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    k=j;
                    for(int l=1; l<=k; l++){
```

```
                        if(B[i][j-l]==0) {mq++;Move m=new Move(i,j,i,j-
l,5);moves_w.add(m);}
                        else if(B[i][j-l]<0) {mq++;Move m=new Move(i,j,i,j-
l,5);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    //bishop moves
                    k=Math.min(7-i,7-j);
                    for(int l=1; l<=k; l++){
                        if(B[i+l][j+l]==0) {mq++;Move m=new
Move(i,j,i+l,j+l,5);moves_w.add(m);}
                        else if(B[i+l][j+l]<0) {mq++;Move m=new
Move(i,j,i+l,j+l,5);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    k=Math.min(i,j);
                    for(int l=1; l<=k; l++){
                        if(B[i-l][j-l]==0) {mq++;Move m=new Move(i,j,i-l,j-
l,5);moves_w.add(m);}
                        else if(B[i-l][j-l]<0) {mq++;Move m=new Move(i,j,i-
l,j-l,5);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    k=Math.min(7-i,j);
                    for(int l=1; l<=k; l++){
                        if(B[i+l][j-l]==0) {mq++;Move m=new Move(i,j,i+l,j-
l,5);moves_w.add(m);}
                        else if(B[i+l][j-l]<0) {mq++;Move m=new
Move(i,j,i+l,j-l,5);moves_w.add(m); l=k;}
                        else l=k;
                    }
                    k=Math.min(i,7-j);
                    for(int l=1; l<=k; l++){
                        if(B[i-l][j+l]==0) {mq++;Move m=new Move(i,j,i-
l,j+l,5);moves_w.add(m);}
                        else if(B[i-l][j+l]<0) {mq++;Move m=new Move(i,j,i-
l,j+l,5);moves_w.add(m); l=k;}
                        else l=k;
                    }
                }
                else if(side==-1 && B[i][j]==-5){ //black
                    //rook moves
                    int k=7-i;
                    for(int l=1; l<=k; l++){
                        if(B[i+l][j]==0) {mq++;Move m=new Move(i,j,i+l,j,-
5);moves_b.add(m);}
                        else if(B[i+l][j]>0) {mq++;Move m=new
Move(i,j,i+l,j,-5);moves_b.add(m); l=k;}
                        else l=k;
                    }
                    k=i;
                    for(int l=1; l<=k; l++){
                        if(B[i-l][j]==0) {mq++;Move m=new Move(i,j,i-l,j,-
5);moves_b.add(m);}
                        else if(B[i-l][j]>0) {mq++;Move m=new Move(i,j,i-
l,j,-5);moves_b.add(m); l=k;}
                        else l=k;
                    }
                    k=7-j;
                    for(int l=1; l<=k; l++){
                        if(B[i][j+l]==0) {mq++;Move m=new Move(i,j,i,j+l,-
5);moves_b.add(m);}
                        else if(B[i][j+l]>0) {mq++;Move m=new
Move(i,j,i,j+l,-5);moves_b.add(m); l=k;}
                        else l=k;
                    }
                    k=j;
                    for(int l=1; l<=k; l++){
```

```
                        if(B[i][j-l]==0) {mq++;Move m=new Move(i,j,i,j-l,-
5);moves_b.add(m);}
                        else if(B[i][j-l]>0) {mq++;Move m=new Move(i,j,i,j-
l,-5);moves_b.add(m); l=k;}
                        else l=k;
                    }
                    //bishop moves
                    k=Math.min(7-i,7-j);
                    for(int l=1; l<=k; l++){
                        if(B[i+l][j+l]==0) {mq++;Move m=new
Move(i,j,i+l,j+l,-5);moves_b.add(m);}
                        else if(B[i+l][j+l]>0) {mq++;Move m=new
Move(i,j,i+l,j+l,-5);moves_b.add(m); l=k;}
                        else l=k;
                    }
                    k=Math.min(i,j);
                    for(int l=1; l<=k; l++){
                        if(B[i-l][j-l]==0) {mq++;Move m=new Move(i,j,i-l,j-
l,-5);moves_b.add(m);}
                        else if(B[i-l][j-l]>0) {mq++;Move m=new Move(i,j,i-
l,j-l,-5);moves_b.add(m); l=k;}
                        else l=k;
                    }
                    k=Math.min(7-i,j);
                    for(int l=1; l<=k; l++){
                        if(B[i+l][j-l]==0) {mq++;Move m=new Move(i,j,i+l,j-
l,-5);moves_b.add(m);}
                        else if(B[i+l][j-l]>0) {mq++;Move m=new
Move(i,j,i+l,j-l,-5);moves_b.add(m); l=k;}
                        else l=k;
                    }
                    k=Math.min(i,7-j);
                    for(int l=1; l<=k; l++){
                        if(B[i-l][j+l]==0) {mq++;Move m=new Move(i,j,i-
l,j+l,-5);moves_b.add(m);}
                        else if(B[i-l][j+l]>0) {mq++;Move m=new Move(i,j,i-
l,j+l,-5);moves_b.add(m); l=k;}
                        else l=k;
                    }
                }
            }
        }
        return mq;
    }

    int mobKing(int side){ //calculate king mobility & generate king moves
        int mk=0;
        for(int i=0; i<8; i++){
            for(int j=0; j<8; j++){
                if(side==1 && B[i][j]==6){ //white
                    if(i<7 && B[i+1][j]<=0 && !isUnderAttack(i+1,j,side))
{mk++;Move m=new Move(i,j,i+1,j,6);moves_w.add(m);}
                    if(i>0 && B[i-1][j]<=0 && !isUnderAttack(i-1,j,side))
{mk++;Move m=new Move(i,j,i-1,j,6);moves_w.add(m);}
                    if(j<7 && B[i][j+1]<=0 && !isUnderAttack(i,j+1,side))
{mk++;Move m=new Move(i,j,i,j+1,6);moves_w.add(m);}
                    if(j>0 && B[i][j-1]<=0 && !isUnderAttack(i,j-1,side))
{mk++;Move m=new Move(i,j,i,j-1,6);moves_w.add(m);}
                    if(i<7 && j<7 && B[i+1][j+1]<=0
&& !isUnderAttack(i+1,j+1,side)) {mk++;Move m=new
Move(i,j,i+1,j+1,6);moves_w.add(m);}
                    if(i<7 && j>0 && B[i+1][j-1]<=0 && !isUnderAttack(i+1,j-
1,side)) {mk++;Move m=new Move(i,j,i+1,j-1,6);moves_w.add(m);}
                    if(i>0 && j<7 && B[i-1][j+1]<=0 && !isUnderAttack(i-
1,j+1,side)) {mk++;Move m=new Move(i,j,i-1,j+1,6);moves_w.add(m);}
                    if(i>0 && j>0 && B[i-1][j-1]<=0 && !isUnderAttack(i-1,j-
1,side)) {mk++;Move m=new Move(i,j,i-1,j-1,6);moves_w.add(m);}
                }
```

```
                else if(side==-1 && B[i][j]==-6){ //black
                    if(i<7 && B[i+1][j]>=0 && !isUnderAttack(i+1,j,side))
{mk++;Move m=new Move(i,j,i+1,j,-6);moves_b.add(m);}
                    if(i>0 && B[i-1][j]>=0 && !isUnderAttack(i-1,j,side))
{mk++;Move m=new Move(i,j,i-1,j,-6);moves_b.add(m);}
                    if(j<7 && B[i][j+1]>=0 && !isUnderAttack(i,j+1,side))
{mk++;Move m=new Move(i,j,i,j+1,-6);moves_b.add(m);}
                    if(j>0 && B[i][j-1]>=0 && !isUnderAttack(i,j-1,side))
{mk++;Move m=new Move(i,j,i,j-1,-6);moves_b.add(m);}
                    if(i<7 && j<7 && B[i+1][j+1]>=0
&& !isUnderAttack(i+1,j+1,side)) {mk++;Move m=new Move(i,j,i+1,j+1,-
6);moves_b.add(m);}
                    if(i<7 && j>0 && B[i+1][j-1]>=0 && !isUnderAttack(i+1,j-
1,side)) {mk++;Move m=new Move(i,j,i+1,j-1,-6);moves_b.add(m);}
                    if(i>0 && j<7 && B[i-1][j+1]>=0 && !isUnderAttack(i-
1,j+1,side)) {mk++;Move m=new Move(i,j,i-1,j+1,-6);moves_b.add(m);}
                    if(i>0 && j>0 && B[i-1][j-1]>=0 && !isUnderAttack(i-1,j-
1,side)) {mk++;Move m=new Move(i,j,i-1,j-1,-6);moves_b.add(m);}
                }
            }
        }
        //castling moves
        if(side==1){ //white
            if(isCastling(1,0)){ //queenside
                mk++;Move m=new Move(0,4,0,2,0);moves_w.add(m);
            }
            if(isCastling(1,1)){ //kingside
                mk++;Move m=new Move(0,4,0,6,0);moves_w.add(m);
            }
        }
        else if(side==-1){ //black
            if(isCastling(-1,0)){ //queenside
                mk++;Move m=new Move(7,4,7,2,0);moves_b.add(m);
            }
            if(isCastling(-1,1)){ //kingside
                mk++;Move m=new Move(7,4,7,6,0);moves_b.add(m);
            }
        }
        return mk;
    }


    int mobility(int side){ //calculate total mobilities & generate all
possible moves of board
        int m=0;
        if(side==1){ //white
            moves_w.clear();
            m = mobPawn(1) + mobKnight(1) + mobBishop(1) + mobRook(1) +
mobQueen(1) + mobKing(1);
            mobility_w=m;
        }
        else if(side==-1){ //black
            moves_b.clear();
            m = mobPawn(-1) + mobKnight(-1) + mobBishop(-1) + mobRook(-1) +
mobQueen(-1) + mobKing(-1);
            mobility_b=m;
        }
        return m;
    }


    public void printMoves(){ //list all possible moves of board
        //this.generateChildren();
        System.out.println("Moves for white(+):");
        for(int i=0; i<this.moves_w.size(); i++){
            System.out.print("Move "+(i+1)+": ");
            Board b = new Board(this,moves_w.get(i));
            System.out.print(" with value :"+b.value+" ");
```

```
            moves_w.get(i).printMove();
        }
        System.out.println("Moves for black(-):");
        for(int i=0; i<this.moves_b.size(); i++){
            System.out.print("Move "+(i+1)+": ");
            Board b = new Board(this,moves_b.get(i));
            System.out.print(" with value :"+b.value+" ");
            moves_b.get(i).printMove();
        }
    }

    public void set_mobility(){ //re-calculate mobilities
        moves_w.clear();
        moves_b.clear();
        mobility_w=mobPawn(1) + mobKnight(1) + mobBishop(1) + mobRook(1) +
mobQueen(1) + mobKing(1);
        mobility_b=mobPawn(-1) + mobKnight(-1) + mobBishop(-1) + mobRook(-1)
+ mobQueen(-1) + mobKing(-1);
    }

    public boolean isUnderAttack(int x, int y, int side){ //cell(x,y) is in
danger by opponent side pieces
        if(side==1){ //white
            for(int i=0; i<this.moves_b.size(); i++){ // all threats other
than pawn
                if(x==moves_b.get(i).to_x && y==moves_b.get(i).to_y &&
moves_b.get(i).piece!=-1) return true;
            }
            for(int i=0; i<8; i++){
                for(int j=0; j<8; j++){
                    if(B[i][j]==-1){ //pawn threat check
                        if(i!=0 && j!=0 && x==i-1 && y==j-1 ) return true;
                        if(i!=0 && j!=7 && x==i-1 && y==j+1 ) return true;
                    }
                    else if(B[i][j]==-6){ //king threat check
                        if(i!=0 && j!=0 && x==i-1 && y==j-1 ) return true;
                        if(i!=0 && x==i-1 && y==j ) return true;
                        if(i!=0 && j!=7 && x==i-1 && y==j+1 ) return true;
                        if(j!=0 && x==i && y==j-1 ) return true;
                        if(j!=7 && x==i && y==j+1 ) return true;
                        if(i!=7 && j!=0 && x==i+1 && y==j-1 ) return true;
                        if(i!=7 && x==i+1 && y==j ) return true;
                        if(i!=7 && j!=7 && x==i+1 && y==j+1 ) return true;
                    }
                }
            }
        }
        else if (side==-1){ //black
            for(int i=0; i<moves_w.size(); i++){ //all threats
                if(x==moves_w.get(i).to_x && y==moves_w.get(i).to_y &&
moves_w.get(i).piece!=1) return true;
            }
            for(int i=0; i<8; i++){
                for(int j=0; j<8; j++){
                    if(B[i][j]==1){ //pawn threats
                        if(i!=7 && j!=0 && x==i+1 && y==j-1 ) return true;
                        if(i!=7 && j!=7 && x==i+1 && y==j+1 ) return true;
                    }
                    else if(B[i][j]==6){ // king threats
                        if(i!=0 && j!=0 && x==i-1 && y==j-1 ) return true;
                        if(i!=0 && x==i-1 && y==j ) return true;
                        if(i!=0 && j!=7 && x==i-1 && y==j+1 ) return true;
                        if(j!=0 && x==i && y==j-1 ) return true;
                        if(j!=7 && x==i && y==j+1 ) return true;
                        if(i!=7 && j!=0 && x==i+1 && y==j-1 ) return true;
                        if(i!=7 && x==i+1 && y==j ) return true;
                        if(i!=7 && j!=7 && x==i+1 && y==j+1 ) return true;
                    }
```

```
                }
            }
        }
        return false;
    }

    public boolean isCastling(int side, int type){ //given board is ok for
castling
        //type=0: castling on queenside
        //type=1: castling on kingside
        if(side==1){ //white
            if(type==0 && B[0][0]==2 && B[0][4]==6 && B[0][1]==0 &&
B[0][2]==0 && B[0][3]==0 //no pieces between
                    && devArr[0][0]==0 && devArr[0][4]==0){ //not moved
before
                for(int i=2; i<=4; i++){
                    if(isUnderAttack(0,i,1)) return false; //king safety
                }
                return true;
            }
            else if(type==1 && B[0][7]==2 && B[0][4]==6 && B[0][5]==0 &&
B[0][6]==0
                    && devArr[0][7]==0 && devArr[0][4]==0){
                for(int i=4; i<=6; i++){
                    if(isUnderAttack(0,i,1)) return false;
                }
                return true;
            }
        }
        else if (side==-1){ //black
            if(type==0 && B[7][0]==-2 && B[7][4]==-6 && B[7][1]==0 &&
B[7][2]==0 && B[7][3]==0
                    && devArr[3][0]==0 && devArr[3][4]==0){
                for(int i=2; i<=4; i++){
                    if(isUnderAttack(7,i,-1)) return false;
                }
                return true;
            }
            else if(type==1 && B[7][7]==-2 && B[7][4]==-6 && B[7][5]==0 &&
B[7][6]==0
                    && devArr[3][7]==0 && devArr[3][4]==0){
                for(int i=4; i<=6; i++){
                    if(isUnderAttack(7,i,-1)) return false;
                }
                return true;
            }
        }
        return false;
    }



    public void printBoard(){ //print board with piece representer letters
        char[]  A = new char[7];
        A[0]='*';
        A[1]='P';
        A[2]='R';
        A[3]='N';
        A[4]='B';
        A[5]='Q';
        A[6]='K';
        System.out.println("    a  b  c  d  e  f  g  h");
        for(int i=0; i<8; i++){
            System.out.print((i+1)+": ");
            for(int j=0; j<8; j++){
                if(B[i][j]>0) System.out.print("+"+A[B[i][j]]+" ");
                if(B[i][j]<0) System.out.print("-"+A[Math.abs(B[i][j])]+"
");
```

```
                if(B[i][j]==0) System.out.print(" "+A[B[i][j]]+" ");
            }
            System.out.println();
        }
    }

    public void generateChildren(){ //generate all possible child
nodes(boards)
        children_w.clear();
        children_b.clear();
        for(int i=0; i<moves_w.size(); i++){
            Board b = new Board(this,moves_w.get(i));
            if(b.isInCheck(1))
                {moves_w.remove(i); i--;}
            else
                children_w.add(b);
        }
        for(int i=0; i<moves_b.size(); i++){
            Board b = new Board(this,moves_b.get(i));
            if(b.isInCheck(-1))
                {moves_b.remove(i); i--;}
            else
                children_b.add(b);
        }
        mobility_w=children_w.size();
        mobility_b=children_b.size();
        value_w=eval(1);
        value_b=eval(-1);
        if(Main.sideC==1) value=value_w-value_b;
        else value=value_b-value_w;
    }

    public boolean moveIsValid(Move m){ //given move is possible on this
board
        if(m.piece<=0){ // black's move
            for(int i=0; i<moves_b.size(); i++){
                if(moves_b.get(i).equals(m)) return true;
            }
        }
        if(m.piece>=0){ // white's move
            for(int i=0; i<moves_w.size(); i++){
                if(moves_w.get(i).equals(m)) return true;
            }
        }
        return false;
    }

    double eval(int side){ //evaluation function
        double val=0;
        if(side==1){ //white
            val = material_w*Main.Wmat/40 + mobility_w*Main.Wmob/50 +
dev_w*Main.Wdev/16 + center_w*Main.Wcen/20;
        }
        else{ //black
            val = material_b*Main.Wmat/40 + mobility_b*Main.Wmob/50 +
dev_b*Main.Wdev/16 + center_b*Main.Wcen/20;
        }
        return val;
    }

    boolean equals(Board b){ //check if two board configurations are the
same
        for(int i=0; i<8; i++){
            for(int j=0; j<8; j++){
                if(this.B[i][j]!=b.B[i][j]) return false;
            }
        }
        return true;
```

```
    }


    void copy(Board b){ //copy board matrix and development values
        for(int i=0; i<8; i++){
            for(int j=0; j<8; j++){
                this.B[i][j]=b.B[i][j];
                if(i<4) this.devArr[i][j]=b.devArr[i][j];
            }
        }
        this.parent=b.parent;
    }

    void loadBoard(){ //load board values after copy
        dev_w=dev(1);
        dev_b=dev(-1);
        material_w=material(1);
        material_b=material(-1);
        center_w=centeral(1);
        center_b=centeral(-1);

        this.set_mobility();
        this.generateChildren();
    }


    public boolean isInCheck(int side){ //control if the board is in check
for one side
        if(side==1){ //white
            for(int i=0; i<8; i++){
                for(int j=0; j<8; j++){
                    if(B[i][j]==6){
                        if(isUnderAttack(i,j,side)) return true;
                    }
                }
            }
        }
        else if(side==-1){ //black
            for(int i=0; i<8; i++){
                for(int j=0; j<8; j++){
                    if(B[i][j]==-6){
                        if(isUnderAttack(i,j,side)) return true;
                    }
                }
            }
        }
        return false;
    }

    public boolean isDraw(){ //check if the current board is a draw
condition
        int b;
        int nw=0, nb=0;
        int bw=0, bb=0;
        int x1=0,x2=0,y1=0,y2=0;
        for(int i=0; i<8; i++){
            for(int j=0; j<8; j++){
                b=B[i][j];
                if(b==1 || b==-1 || b==2 || b==-2 || b==5 || b==-5) return
false;
                else if(b==3) nw++;
                else if(b==4) {bw++; x1=i; y1=j;}
                else if(b==-3) nb++;
                else if(b==-4) {bb++; x2=i; y2=j;}
            }
        }
        if(nw==0 && nb==0 && bw==0 && bb==0) return true; // king vs king
```

```
        else if(nw==0 && nb==0 && bw==1 && bb==0) return true; // king vs
king&bishop
        else if(nw==0 && nb==0 && bw==0 && bb==1) return true;
        else if(nw==1 && nb==0 && bw==0 && bb==0) return true; // king vs
king&knight
        else if(nw==0 && nb==1 && bw==0 && bb==0) return true;
        else if(nw==0 && nb==0 && bw==1 && bb==1){ // king&bishop vs
king&bishop
            int[][] A= new int[][]
            {{0,1,0,1,0,1,0,1},
             {1,0,1,0,1,0,1,0},
             {0,1,0,1,0,1,0,1},
             {1,0,1,0,1,0,1,0},
             {0,1,0,1,0,1,0,1},
             {1,0,1,0,1,0,1,0},
             {0,1,0,1,0,1,0,1},
             {1,0,1,0,1,0,1,0}};
            if(A[x1][y1]==A[x2][y2]) return true; //both bishops on
diagonals of the same colour
        }

        return false;
    }

    public boolean isCheckMate2(int side){
        if(side==1 && children_w.size()==0) return true;
        else if(side==-1 && children_b.size()==0) return true;
        return false;
    }

    public boolean isCheckMate(int side){ //control if the game ends
        if(isInCheck(side) && side==1){ //white is in check
            for(int i=0; i<children_w.size(); i++){
                if(!children_w.get(i).isInCheck(side)) return false;
            }
            return true;
        }
        else if(isInCheck(side) && side==-1){ //black is in check
            for(int i=0; i<children_b.size(); i++){
                if(!children_b.get(i).isInCheck(side)) return false;
            }
            return true;
        }
        return false;
    }

    double Max(){
        double m=Main.NEG;
        if(Main.sideC==1){ //computer white
            for(int i=0; i<children_w.size(); i++){
                if(children_w.get(i).value>m) m=children_w.get(i).value;
            }
        }
        else{ //computer black
            for(int i=0; i<children_b.size(); i++){
                if(children_b.get(i).value>m) m=children_b.get(i).value;
            }
        }
        return m;
    }

    double Min(){
        double m=Main.POS;
        if(Main.sideC==1){ //computer white
            for(int i=0; i<children_b.size(); i++){
                if(children_b.get(i).value<m) m=children_b.get(i).value;
            }
        }
```

```
        else{ //computer black
            for(int i=0; i<children_w.size(); i++){
                if(children_w.get(i).value<m) m=children_w.get(i).value;
            }
        }
        return m;
    }


    Board Play(){
        Board best_move= new Board();
        double[] max=new double[2];
        int index=0;

        max=Minimax_AB(Main.sideC,Main.sideC,this,Main.NEG,Main.POS);
        index=(int)max[1];
        System.out.println("BEST MOVE IS FOUND AT CHILD NO: "+(index+1));
        if(Main.sideC==1){
           best_move.copy(children_w.get(index));
           best_move.loadBoard();
        }
        else{
           best_move.copy(children_b.get(index));
           best_move.loadBoard();
        }

        return best_move;
    }


  double[] Minimax_AB(int side, int turn, Board current,double alpha, double
beta){

       double val=0;
       double score=0;
       double index=0;
       int maxDepth;
       double[] arr=new double[2];
       maxDepth=Main.maxDepth/2;


       if(Main.depth==maxDepth)
           {
             val= current.value;
             arr[0]=val;
             arr[1]=index;
             return arr;
           }
       else{

           current.generateChildren();
           Main.depth++;
           if(side==1){//computer white and we will maximize children_w
               if(turn==side){ //max node
                   for(int i=0; i<current.children_w.size();i++){
                       arr=Minimax_AB(side,(turn*(-
1)),current.children_w.get(i),alpha,beta); //call for every child, computer
white,white's turn
                       score=arr[0];
                       if(score>alpha){
                           alpha=score;
                           index=i;
                           }
                       if(alpha>=beta){
                           val=alpha;
                           Main.depth--;
                           arr[0]=val;
                           arr[1]=index;
```

```
                            return arr;


                        }
                    }
                    val=alpha;
                    Main.depth--;
                    arr[0]=val;
                    arr[1]=index;
                    return arr;
                }
                else{// for every child that is black's turn (min node)
                    for(int i=0; i<current.children_b.size();i++){
                        arr=Minimax_AB(side,(turn*(-
1)),current.children_b.get(i),alpha,beta);
                        score=arr[0];
                        if(score<beta){
                            beta=score;
                            index=i;
                            }
                        if(alpha>=beta){
                            val=beta;
                            Main.depth--;
                            arr[0]=val;
                            arr[1]=index;
                            return arr;
                        }
                    }
                    val=beta;
                    Main.depth--;
                    arr[0]=val;
                    arr[1]=index;
                    return arr;
                }

            }
            else{//computer black and we will maximize children_b

                  if(turn==side){ //max node
                    for(int i=0; i<current.children_b.size();i++){
                        arr=Minimax_AB(side,(turn*(-
1)),current.children_b.get(i),alpha,beta); //call for every child, computer
black,black's turn
                        score=arr[0];
                        if(score>alpha){
                            alpha=score;
                            index=i;
                            }
                        if(alpha>=beta){
                            val=alpha;
                            Main.depth--;
                            arr[0]=val;
                            arr[1]=index;
                            return arr;
                        }
                    }
                    val=alpha;
                    Main.depth--;
                    arr[0]=val;
                    arr[1]=index;
                    return arr;
                }
                else{// for every child that is black's turn (min node)
                    for(int i=0; i<current.children_w.size();i++){
                        arr=Minimax_AB(side,(turn*(-
1)),current.children_w.get(i),alpha,beta);
                        score=arr[0];
                        if(score<beta){
```

```
                            beta=score;
                            index=i;
                            }
                        if(alpha>=beta){
                            val=beta;
                            Main.depth--;
                            arr[0]=val;
                            arr[1]=index;
                            return arr;
                        }
                    }
                    val=beta;
                    Main.depth--;
                    arr[0]=val;
                    arr[1]=index;
                    return arr;
                }
            }

        }

    }
}
```

Move.java:

```
package my_chess_pkg;


/**
 * @author eda & selin
 */
public class Move {
    int from_x; //source
    int from_y;
    int to_x; //destination
    int to_y;
    int piece; //type of moved piece
    /*
     + for white
     - for black
     1: pawn
     2: rook
     3: kngiht
     4: bishop
     5: queen
```

```
     6: king
     0: castling move
     'p': en passant move
     */

    public Move(){
        from_x=0;
        from_y=0;
        to_x=0;
        to_y=0;
        piece=0;
    }

    public Move(int x, int y, int x2, int y2, int p){
        from_x=x;
        from_y=y;
        to_x=x2;
        to_y=y2;
        piece=p;
    }

    public boolean equals(Move m){
        if(from_x==m.from_x && from_y==m.from_y && to_x==m.to_x &&
to_y==m.to_y && piece==m.piece)
            return true;
        return false;
    }

    public void copy(Move m){
        from_x=m.from_x;
        from_y=m.from_y;
        to_x=m.to_x;
        to_y=m.to_y;
        piece=m.piece;
    }
```

```
    public Move(String input){ //create a move from user input

        if((input.length()>3 && input.charAt(3)=='0' &&
input.charAt(5)=='0')

                || (input.charAt(0)=='0' &&
input.charAt(2)=='0')){ //castling

            piece=0;

            if((input.length()>7 && input.charAt(7)=='0')

                    || (input.length()>4 &&
input.charAt(4)=='0')){ //queenside

                if(Main.sideC==1) {from_x=7; from_y=4; to_x=7; to_y=2;}
//black

                else {from_x=0; from_y=4; to_x=0; to_y=2;} //white

            }

            else{ //kingside

                if(Main.sideC==1) {from_x=7; from_y=4; to_x=7; to_y=6;}
//black

                else {from_x=0; from_y=4; to_x=0; to_y=6;} //white

            }

        }

        else{ //not castling

            if(input.length()>9){ //P: X ni-mj

        if(input.charAt(3)=='P' || input.charAt(3)=='p') piece=1;

        else if(input.charAt(3)=='R' || input.charAt(3)=='r') piece=2;

        else if(input.charAt(3)=='N' || input.charAt(3)=='n') piece=3;

        else if(input.charAt(3)=='B' || input.charAt(3)=='b') piece=4;

        else if(input.charAt(3)=='Q' || input.charAt(3)=='q') piece=5;

        else if(input.charAt(3)=='K' || input.charAt(3)=='k') piece=6;

        from_x=Character.digit(input.charAt(6),10)-1;

        if(input.charAt(5)=='A' || input.charAt(5)=='a') from_y=0;

        else if(input.charAt(5)=='B' || input.charAt(5)=='b') from_y=1;

        else if(input.charAt(5)=='C' || input.charAt(5)=='c') from_y=2;

        else if(input.charAt(5)=='D' || input.charAt(5)=='d') from_y=3;

        else if(input.charAt(5)=='E' || input.charAt(5)=='e') from_y=4;

        else if(input.charAt(5)=='F' || input.charAt(5)=='f') from_y=5;

        else if(input.charAt(5)=='G' || input.charAt(5)=='g') from_y=6;

        else if(input.charAt(5)=='H' || input.charAt(5)=='h') from_y=7;
```

```
to_x=Character.digit(input.charAt(9),10)-1;
if(input.charAt(8)=='A' || input.charAt(8)=='a') to_y=0;
else if(input.charAt(8)=='B' || input.charAt(8)=='b') to_y=1;
else if(input.charAt(8)=='C' || input.charAt(8)=='c') to_y=2;
else if(input.charAt(8)=='D' || input.charAt(8)=='d') to_y=3;
else if(input.charAt(8)=='E' || input.charAt(8)=='e') to_y=4;
else if(input.charAt(8)=='F' || input.charAt(8)=='f') to_y=5;
else if(input.charAt(8)=='G' || input.charAt(8)=='g') to_y=6;
else if(input.charAt(8)=='H' || input.charAt(8)=='h') to_y=7;
    }
    else if(input.length()>6){ //x ni-mj
if(input.charAt(0)=='P' || input.charAt(0)=='p') piece=1;
else if(input.charAt(0)=='R' || input.charAt(0)=='r') piece=2;
else if(input.charAt(0)=='N' || input.charAt(0)=='n') piece=3;
else if(input.charAt(0)=='B' || input.charAt(0)=='b') piece=4;
else if(input.charAt(0)=='Q' || input.charAt(0)=='q') piece=5;
else if(input.charAt(0)=='K' || input.charAt(0)=='k') piece=6;
from_x=Character.digit(input.charAt(3),10)-1;
if(input.charAt(2)=='A' || input.charAt(2)=='a') from_y=0;
else if(input.charAt(2)=='B' || input.charAt(2)=='b') from_y=1;
else if(input.charAt(2)=='C' || input.charAt(2)=='c') from_y=2;
else if(input.charAt(2)=='D' || input.charAt(2)=='d') from_y=3;
else if(input.charAt(2)=='E' || input.charAt(2)=='e') from_y=4;
else if(input.charAt(2)=='F' || input.charAt(2)=='f') from_y=5;
else if(input.charAt(2)=='G' || input.charAt(2)=='g') from_y=6;
else if(input.charAt(2)=='H' || input.charAt(2)=='h') from_y=7;
to_x=Character.digit(input.charAt(6),10)-1;
if(input.charAt(5)=='A' || input.charAt(5)=='a') to_y=0;
else if(input.charAt(5)=='B' || input.charAt(5)=='b') to_y=1;
else if(input.charAt(5)=='C' || input.charAt(5)=='c') to_y=2;
else if(input.charAt(5)=='D' || input.charAt(5)=='d') to_y=3;
else if(input.charAt(5)=='E' || input.charAt(5)=='e') to_y=4;
else if(input.charAt(5)=='F' || input.charAt(5)=='f') to_y=5;
else if(input.charAt(5)=='G' || input.charAt(5)=='g') to_y=6;
else if(input.charAt(5)=='H' || input.charAt(5)=='h') to_y=7;
```

```
            }
        if(Main.sideC==1) piece*=-1; //user is black(-)
        }
    }

    public void printMove(){ //print move as "<piece> from: <source> to:
<destination>"
        char[] con = new char[8];
        con[0]='a';
        con[1]='b';
        con[2]='c';
        con[3]='d';
        con[4]='e';
        con[5]='f';
        con[6]='g';
        con[7]='h';
        if(piece!=0){ //not castling
        if(piece==1 || piece==-1 )
            System.out.print("Pawn("+piece+")");
        else if(piece==2 || piece==-2 )
            System.out.print("Rook("+piece+")");
        else if(piece==3 || piece==-3 )
            System.out.print("Knight("+piece+")");
        else if(piece==4 || piece==-4 )
            System.out.print("Bishop("+piece+")");
        else if(piece==5 || piece==-5 )
            System.out.print("Queen("+piece+")");
        else if(piece==6 || piece==-6 )
            System.out.print("King("+piece+")");
        else if(piece=='p')
            System.out.print("En Passant move ");
        System.out.println(" from:
"+con[from_y]+(from_x+1)+"("+from_x+","+from_y+") to:
"+con[to_y]+(to_x+1)+"("+to_x+","+to_y+")");
        }
        else{ //castling
```

```
            if(from_x==0){ //white

                System.out.print("King(6) from:
"+con[from_y]+(from_x+1)+"("+from_x+","+from_y+") to:
"+con[to_y]+(to_x+1)+"("+to_x+","+to_y+") and Rook(2)");

                if(to_y==2) System.out.println(" from: a1(0,0) to:
d1(0,3)");

                if(to_y==6) System.out.println(" from: h1(0,7) to:
f1(0,5)");

            }

            else if(from_x==7){ //black

                System.out.print("King(-6) from:
"+con[from_y]+(from_x+1)+"("+from_x+","+from_y+") to:
"+con[to_y]+(to_x+1)+"("+to_x+","+to_y+") and Rook(-2)");

                if(to_y==2) System.out.println(" from: a8(7,0) to:
d8(7,3)");

                if(to_y==6) System.out.println(" from: h8(7,7) to:
f8(7,5)");

            }

        }

    }


}
```

Main.java:

```
package my_chess_pkg;
import java.io.*;

/**
 *
 * @author eda & selin
 */
public class Main {

    public static int sideC=-1; //side of computer: 1=white, -
1=black
    //evaluation function weights
    public static final double Wmat=100; //weight of material
    public static final double Wmob=10; //weight of mobility
    public static final double Wdev=5; //weight of development
    public static final double Wcen=1; //weight of centerality
    public static int count=0; //for draw
    public static int depth=0;
    public static final int maxDepth=4;
    public static final double NEG=-1000000000;
    public static final double POS=1000000000;
   /**
    * @param args the command line arguments
    */
   public static void main(String[] args) throws IOException{
       // TODO code application logic here
```

```
                int[][] B= new int[][]
                /*     a b c d e f g h    */
                /*1*/{{2,3,4,5,6,4,3,2}, //white
                /*2*/ {1,1,1,1,1,1,1,1},
                /*3*/ {0,0,0,0,0,0,0,0},
                /*4*/ {0,0,0,0,0,0,0,0},
                /*5*/ {0,0,0,0,0,0,0,0},
                /*6*/ {0,0,0,0,0,0,0,0},
                /*7*/ {-1,-1,-1,-1,-1,-1,-1,-1},
                /*8*/ {-2,-3,-4,-5,-6,-4,-3,-2}}; //black


                Board b = new Board();
                b.B=B;
                b.loadBoard();
                //b.side=1;
                System.out.println("Material of White:
"+b.material(1));
                System.out.println("Material of Black: "+b.material(-
1));
                //b.set_mobility();
                System.out.println("Mobility of White: "+b.mobility_w);
                System.out.println("Mobility of Black: "+b.mobility_b);

                b.printMoves();
                //b.printBoard();

                String input = ""; // Line read from standard in
                InputStreamReader converter = new
InputStreamReader(System.in);
                BufferedReader in = new BufferedReader(converter);

                Board prev = new Board();
                prev.B=B;
                //prev.B=B_test;
                //prev.B=B_draw;
                //prev.B=B_max2;
                prev.loadBoard();
                Board next = new Board();
                int index=0;
                Move bestmove = new Move();
                int Nply=0;

                while(true){
                System.out.println("Please choose your side: \nEnter
'w' for white,'b' for black: ");
                input = in.readLine();
                if(input.equalsIgnoreCase("w")) {sideC=-1; break;}
                else if(input.equalsIgnoreCase("b")) {sideC=1; break;}
                }

                System.out.println("Initial board: ");
                prev.printBoardInfo();
                prev.printBoard();

                if(sideC==1){
                    prev.copy(prev.Play());
                    prev.loadBoard();
                    depth=0; //set depth to zero for the next play tree
                    System.out.println("Current board after computer's
move: ");
                    prev.printBoardInfo();
                    prev.printBoard();
                }

                while(true){
                System.out.println("Enter your next move: ");
```

```
                input = in.readLine();
                if(input.equalsIgnoreCase("resign")){
                    System.out.println("Player is Resigned: \nCOMPUTER
WINS!!! GAME OVER!!!");
                    System.out.println("Number of plys: "+Nply);
                    break;
                }
                if(input.equalsIgnoreCase("draw")){
                    System.out.println("Player Proposed a DRAW!");
                    if(prev.value<0){
                        System.out.println("Computer accepted this
proposal: \nIt is a DRAW!!! GAME OVER!!!");
                        System.out.println("Number of plys: "+Nply);
                        break;
                    }
                    else{
                        System.out.println("Computer rejected this
proposal: \nPlease continue the game:");
                        input = in.readLine();
                        if(input.equalsIgnoreCase("resign")){
                            System.out.println("Player is Resigned:
\nCOMPUTER WINS!!! GAME OVER!!!");
                            System.out.println("Number of plys:
"+Nply);
                            break;
                        }

                    }

                }

                if(inputValid(input)){
                    System.out.println("Valid Input =)");
                    Move plmove = new Move(input); //player's move
                    plmove.printMove();
                    if(!prev.moveIsValid(plmove))
                        System.out.println("Invalid Move!");
                    else{ //move is valid
                        System.out.println("Valid Move =)");
                        Board current = new Board(prev,plmove);
                        //current.printBoardInfo();
                        current.generateChildren();
                        System.out.println("Current board info after
your move: ");
                        current.printBoardInfo();
                        System.out.println("Current board after your
move: ");
                        current.printBoard();

                        if(current.isInCheck(sideC))
                            System.out.println("Computer is in
CHECK!!!"); // C is in check by P
                        //current.printBoardInfo();
                        if(current.isCheckMate(sideC)
                                || (sideC==1 &&
current.children_w.size()==0 && current.isInCheck(sideC))
                                || (sideC==-1 &&
current.children_b.size()==0 && current.isInCheck(sideC))) {
                            System.out.println("Computer is in CHECK
MATE!!!");
                            System.out.println("PLAYER WINS!!!\nGAME
OVER!!!");
                            System.out.println("Number of plys:
"+Nply);
                            break;
                        } // C loses
                        else if(current.isDraw() || (sideC==1 &&
current.children_w.size()==0 && !current.isInCheck(sideC))
```

```
                                || (sideC==-1 &&
current.children_b.size()==0 && !current.isInCheck(sideC))){
                            System.out.println("It is a DRAW!!!\nGAME
OVER!!! ");
                            System.out.println("Number of plys:
"+Nply);
                            break;
                        }
                        count++;
                        if(count==50){
                            System.out.println("50 move limit is
reached. The game is over. It is a draw.");
                            System.out.println("Number of plys:
"+Nply);
                        }

                       // current.printBoardInfo();
                        current.printMoves();
                        next.copy(current.Play());
                        next.loadBoard();
                        depth=0; //set depth to zero for the next play
tree

                        if(sideC==1){ //computer is white
                            for(int i=0; i<current.children_w.size();
i++){

if(current.children_w.get(i).equals(next)){
                                    index=i; break;
                                }
                            } //find index of best child

                            System.out.println("BEST MOVE IS FOUND AT
CHILD NO: "+(index+1));
                            bestmove.copy(current.moves_w.get(index));
                            System.out.println("Computer's move is: ");
                            bestmove.printMove();
                            if(next.isInCheck(-1))
                                System.out.println("Player is in
CHECK!!!"); // P is in check by C
                            if(next.isCheckMate(-1) ||
(current.children_b.size()==0 && current.isInCheck(-1))) {
                                System.out.println("Player is in CHECK
MATE!!!");
                                System.out.println("COMPUTER
WINS!!!\nGAME OVER!!!");
                                System.out.println("Number of plys:
"+Nply);
                                break;
                            } // P loses
                            else if(current.isDraw() ||
current.children_b.size()==0 && !current.isInCheck(-1)){
                                System.out.println("It is a
DRAW!!!\nGAME OVER!!! ");
                                System.out.println("Number of plys:
"+Nply);
                                break;
                            }
                            count++;
                            if(count==50){
                                System.out.println("50 move limit is
reached. The game is over. It is a draw.");
                                System.out.println("Number of plys:
"+Nply);
                            }
                        }
                        else if(sideC==-1){ //computer is black
```

```
                            for(int i=0; i<current.children_b.size();
i++){

if(current.children_b.get(i).equals(next)){
                                    index=i; break;
                                }
                            } //find index of best child

                            System.out.println("BEST MOVE IS FOUND AT
CHILD NO: "+(index+1));
                            bestmove.copy(current.moves_b.get(index));
                            System.out.println("Computer's move is: ");
                            bestmove.printMove();
                            if(next.isInCheck(1))
                                System.out.println("Player is in
CHECK!!!"); // P is in check by C
                            if(next.isCheckMate(1) ||
(current.children_w.size()==0 && current.isInCheck(1))) {
                                System.out.println("Player is in CHECK
MATE!!!");
                                System.out.println("COMPUTER
WINS!!!\nGAME OVER!!!");
                                System.out.println("Number of plys:
"+Nply);
                                break;
                            } // P loses
                            else if(current.isDraw() ||
current.children_w.size()==0 && !current.isInCheck(1)){
                                System.out.println("It is a DRAW!!!\n
GAME OVER!!! ");
                                System.out.println("Number of plys:
"+Nply);
                                break;
                            }
                            count++;
                            if(count==50)
                                System.out.println("50 move limit is
reached. The game is over. It is a draw.");
                        }
                        System.out.println("Current board info after
computer's move: ");
                        next.printBoardInfo();
                        System.out.println("Computer's last move is:
");
                        bestmove.printMove();
                        System.out.println("Current board after
computer's move: ");
                        next.printBoard();
                        prev.copy(next);
                        prev.loadBoard();
                        Nply++;
                        System.out.println("Number of plys: "+Nply);
                    }
                }
                else System.out.println("Invalid Input!");
                }

          }

          static public boolean inputValid(String input){ //check the
validity of user input
               if(input.length()>=6 && input.charAt(0)=='P'){
                   if(input.charAt(3)=='0' && input.charAt(5)=='0')
return true; //castling
                   else if(input.length()>=10){
                    char p=input.charAt(3);
                    int fy=input.charAt(5);
                    int fx=input.charAt(6);
```

```
                    int ty=input.charAt(8);
                    int tx=input.charAt(9);
                    if(p=='P' || p=='R' || p=='N' || p=='B' || p=='Q'
|| p=='K' || p=='p' || p=='r' || p=='n' || p=='b' || p=='q' ||
p=='k'){
                        if(fy>='a' && fy<='h' && ty>='a' && ty<='h' &&
fx>='1' && fx<='8' && tx>='1' && tx<='8')
                            return true;
                    }
                   }
               }
               else if(input.length()>=3){
                   if(input.charAt(0)=='0' && input.charAt(2)=='0')
return true; //castling
                   if(input.length()>=6 && input.charAt(3)=='0' &&
input.charAt(5)=='0') return true; //castling
                   else if(input.length()>=7){
                    char p=input.charAt(0);
                    int fy=input.charAt(2);
                    int fx=input.charAt(3);
                    int ty=input.charAt(5);
                    int tx=input.charAt(6);
                    if(p=='P' || p=='R' || p=='N' || p=='B' || p=='Q'
|| p=='K' || p=='p' || p=='r' || p=='n' || p=='b' || p=='q' ||
p=='k'){
                        if(fy>='a' && fy<='h' && ty>='a' && ty<='h' &&
fx>='1' && fx<='8' && tx>='1' && tx<='8')
                            return true;
                    }
                   }
               }
               return false;
           }

       }
```

APPENDIX B: PROJECT PROGRESS CHART (Before Implementation)

Project Progress Chart				
Literature Survey				
Date	Student Name	Hours worked	Progress (this week)	Plan (next week)
09.03.2011	Eda	8	Study of the existing work in the literature in order to familiarize with the topic is started. 8 conference/journal papers and lecture materials assigned by the instructor are read.	Prepare the first document PaperList which keeps a list of the papers we have read. Start preparing an abstract/summary for each paper to be put in the document PaperSummary.
	Selin	8	Study of the existing work in the literature in order to familiarize with the topic is started. 8 conference/journal papers and lecture materials assigned by the instructor are read.	Prepare the first document PaperList which keeps a list of the papers we have read. Start preparing an abstract/summary for each paper to be put in the document PaperSummary.
17.03.2011	Eda	7	The first document PaperList which keeps a list of the papers we have read, along with their basic information, is prepared. Preparing an abstract of 1-2 page for each paper is started. 3/8 papers summaries are finished and corrected.	Finish preparing an abstract/summary for all 8 papers, and prepare PaperSummary document.
	Selin	7	The first document PaperList which keeps a list of the papers we have read, along with their basic information, is prepared. Preparing an abstract of 1-2 page for each paper is started. 3/8 papers summaries are finished and corrected.	Finish preparing an abstract/summary for all 8 papers, and prepare PaperSummary document.
24.03.2011	Eda	6	Preparing an abstract of 1-2 page for the rest 5/8 papers is finished. All summaries are corrected and put in the document "PaperSummary".	Start preparing a literature survey report "LiteratureSurvey", about 10 pages long, to combine all the information gathered.
	Selin	6	Preparing an abstract of 1-2 page for the rest 5/8 papers is finished. All summaries are corrected and put in the document "PaperSummary".	Start preparing a literature survey report "LiteratureSurvey", about 10 pages long, to combine all the information gathered.
31.03.2011	Eda	6	Preparation of a literature survey report is started, to combine all the information we gathered so far.	As a final work of the literature survey phase, finish preparing the literature survey report LiteratureSurvey.
	Selin	6	Preparation of a literature survey report is started, to combine all the information we gathered so far.	As a final work of the literature survey phase, finish preparing the literature survey report LiteratureSurvey.
...				
...				
Proposal				
Date	Student Name	Hours worked	Progress (this week)	Plan (next week)

07.04.2011	Eda	5	In the light of all the papers we read, the preparation of the Project Proposal Document is started.	Extract all information from the literature survey phase and decide which ones to include in the Proposal Document.
	Selin	5	In the light of all the papers we read, the preparation of the Project Proposal Document is started.	Extract all information from the literature survey phase and decide which ones to include in the Proposal Document.
14.04.2011	Eda	5	Project Overview section, which explains the project as a whole, of the Project Proposal Document is discussed and finished.	Revise Project Overview if necessary. Start preparing Project Methodology section of Project Proposal.
	Selin	5	Project Overview section, which explains the project as a whole, of the Project Proposal Document is discussed and finished.	Revise Project Overview if necessary. Start preparing Project Methodology section of Project Proposal.
21.04.2011	Eda	6	Project Methodology, which explains all the details of what will be done and how it will be done, and Success Criteria sections are discussed and prepared. Complete version of Project Proposal Document is finished.	Discussing on the Project Proposal Document with the instructor and revising it accordingly.
	Selin	6	Project Methodology, which explains all the details of what will be done and how it will be done, and Success Criteria sections are discussed and prepared. Complete version of Project Proposal Document is finished.	Discussing on the Project Proposal Document with the instructor and revising it accordingly.
...				
...				

REFERENCES NOT CITED

- Campbell, M. , Hoane, A.J. & Hsu,F. Deep Blue, August 1, 2011

- Campbell, M. Communications of The ACM: Knowledge Discovery in Deep Blue, November 1999/ Vol.42, No.11

- M. Chaves, Escuela de Fsica, Universidad de Costa Rica, Chess Pure Strategies are Probably Chaotic August 21, 1998

- Boškovi´c, B., Greiner, S., Brest, J. & Žumer, V. 2006 IEEE Congress on Evolutionary Computation: A Differential Evolution for the Tuning of a Chess Evaluation Function, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada, July 16-21, 2006

- Poranen, T. ,Department of Computer Sciences, University of Tampere, Chess Algorithms

- Arbiser, A. Dept. of Computer Science, FCEyN, University of Buenos Aires**,** Towards the Unification of Intuitive and Formal Game Concepts with Applications to Computer Chess Proceedings of DIGRA 2005 Conference: Changing Views- Worlds in Play.

- Hallam, N., Poh, H. S. & Kendall, G. Using an Evolutionary Algorithm for the Tuning of a Chess Evaluation Function Based on a Dynamic Boundary Strategy, 1-4244-0023-6/06 - 2006 IEEE

- Fogel, D. B., Fellow, IEEE, Hays ,T. J. ,Hahn, S. L. & Quon, J. A Self-Learning Evolutionary Chess Program, PROCEEDINGS OF THE IEEE, VOL. 92, NO. 12, DECEMBER 2004

- Schaeffer,J. The Games Computers (and People) Play May 10,2000