Chess

Mahesh Devalla

October 24, 2016

1 Introduction

Chess is a two-player strategy board game played on a board that contains 16 pieces of different kind, in 64 squares, where each of these pieces have different moves. This is typically played with humans but can also be played with computers with artificial intelligence. We will look into two different approaches among many approaches to play chess. Below are the two implementations used to play chess.

- Minimax Algorithm.
- Alpha Beta Pruning Algorithm.

1.1 Minimax Algorithm

Minimax algorithms is a decision based algorithm generally used in games to reduce the loss in a game. Generally this algorithm is used for two player games like *chess* where players take their turn to move their pieces.

1.2 Alpha Beta Pruning Algorithm

Alpha Beta Pruning Algorithm is a sophistiated search algorithm that is used to reduce the number of nodes that are calculated when compared in the search performed by the minimax algorithm.Alpha Beta won't consider a move if atleast one case is found that the move would be a bad move than the previously analyzed move in the search based upon the depth. The nodes may be similar when seen but Alpha Beta prunes the branches that play a significant role towards in winning the game.

2 Implementation changes done to the existing code

I changed the code a little to enhance the user experience in the existing stub provided in the assignment.

2.1 Checking for Check Mate

```
1          if(game.position.isMate())
           {
3            if(game.position.getToPlay()==0)
               log("Check Mate Game over : Black won the game");
5            else
               log("Check Mate Game over : White won the game.");
7          }

9          else if(game.position.isStaleMate() || game.position.isTerminal())
           {
11           log("Game cannot move further: Game has been drawn.");
           }
```

3 Implementation of Minimax Algorithm

The algorithm uses recusrion based upon the depth and then evaluates the nodes from the start state to the next state(nodes).

3.1 Basic Function Implementation

```
private Node miniMaxAlgorithm(Position pos) {
    short[] nextNodes = pos.getAllMoves();
    Node nextNodeSuccesor = null;
    // pseudo code from
    // https://en.wikipedia.org/wiki/Rules_of_chess#Illegal_position
    try {
      pos.doMove(nextNodes[0]);
      nextNodeSuccesor = new Node(nextNodes[0], calcMin(pos, 1));
      pos.undoMove();
    } catch (IllegalMoveException e) {
      // System.out.println("This is not a valid move");
      e.printStackTrace();

    }
    for (int i = 1; i < nextNodes.length; i++) {
      try {
        pos.doMove(nextNodes[i]);
        Node temp = new Node(nextNodes[i], calcMin(pos, 1));
        pos.undoMove();
        if (temp.getHueristic() >= nextNodeSuccesor.getHueristic()) {
          nextNodeSuccesor = temp;
        }
      } catch (IllegalMoveException e) {
        // System.out.println("This is not a valid move");
        e.printStackTrace();
      }
    }
    return nextNodeSuccesor;
  }
```

3.2 Results of Minimax when playing with Random AI

```
making move 6095
rnbqkbnr/pppppppp/8/8/7P/8/PPPPPPP1/RNBQKBNR b KQkq h3 0 1
making move 6647
rnbqkbnr/pppppppp1/8/7p/7P/8/PPPPPPP1/RNBQKBNR w KQkq h6 0 2
making move 5518
rnbqkbnr/pppppppp1/8/7p/7P/6P1/PPPPPP2/RNBQKBNR b KQkq - 0 2
making move 6841
r1bqkbnr/pppppppp1/2n5/7p/7P/6P1/PPPPPP2/RNBQKBNR w KQkq - 1 3
making move 5965
r1bqkbnr/pppppppp1/2n5/7p/5P1P/6P1/PPPPP3/RNBQKBNR b KQkq f3 0 3
making move 5866
r1bqkbnr/pppppppp1/8/7p/3n1P1P/6P1/PPPPP3/RNBQKBNR w KQkq - 1 4
making move 6493
r1bqkbnr/pppppppp1/8/5P1p/3n3P/6P1/PPPPP3/RNBQKBNR b KQkq - 0 4
```

```
making move 6452
r1bqkbnr/pppp1pp1/8/4pP1p/3n3P/6P1/PPPPP3/RNBQKBNR w KQkq e6 0 5
making move -5339
r1bqkbnr/pppp1pp1/4P3/7p/3n3P/6P1/PPPPP3/RNBQKBNR b KQkq - 0 5
making move 6899
r1bqkbnr/ppp2pp1/3pP3/7p/3n3P/6P1/PPPPP3/RNBQKBNR w KQkq - 0 6
making move -25236
r1bqkbnr/ppp2Pp1/3p4/7p/3n3P/6P1/PPPPP3/RNBQKBNR b KQkq - 0 6
making move 7420
r1bq1bnr/pppk1Pp1/3p4/7p/3n3P/6P1/PPPPP3/RNBQKBNR w KQ - 1 7
making move -20555
r1bq1bNr/pppk2p1/3p4/7p/3n3P/6P1/PPPPP3/RNBQKBNR b KQ - 0 7
making move 6587
r1b2bNr/pppk2p1/3p4/6qp/3n3P/6P1/PPPPP3/RNBQKBNR w KQ - 1 8
making move -26209
r1b2bNr/pppk2p1/3p4/6Pp/3n4/6P1/PPPPP3/RNBQKBNR b KQ - 0 8
making move 6235
r1b2bNr/pppk2p1/3p4/1n4Pp/8/6P1/PPPPP3/RNBQKBNR w KQ - 1 9
making move 5573
r1b2bNr/pppk2p1/3p4/1n4Pp/8/6PB/PPPPP3/RNBQK1NR b KQ - 2 9
making move 7923
....
```

3.3 Game play (White:Minimax Algorithm and Black: Random AI)

Due to the lack of space in this page the images moved to next page, please find the images in next page.

4 Implementation of Alpha Beta Algorithm

Alpha Beta Algorithm just uses the pruning technique to avoid the unnecessary moves(nodes) unlike the minmax search algorithm. In some ofthe cases it completely avoids the subtrees with bad node that don't contribute to the winning strategy.

4.1 Basic Function Implementation

```
  private Node alphaBetaAlgorithm(Position pos) throws IllegalMoveException {
    short nextNode = Short.MIN_VALUE;
    int nextNodeSuccesor = Integer.MIN_VALUE;
    for (short temp : pos.getAllMoves()) {
      pos.doMove(temp);
      int minOfAB = calcMin(pos, 1, Integer.MIN_VALUE, Integer.MAX_VALUE);
      if (minOfAB > nextNodeSuccesor) {
        nextNodeSuccesor = minOfAB;
        nextNode = temp;
      }
      pos.undoMove();
    }
    return new Node(nextNode, nextNodeSuccesor);
  }
```

4.2 Results of Alpha Beta when playing with Random AI

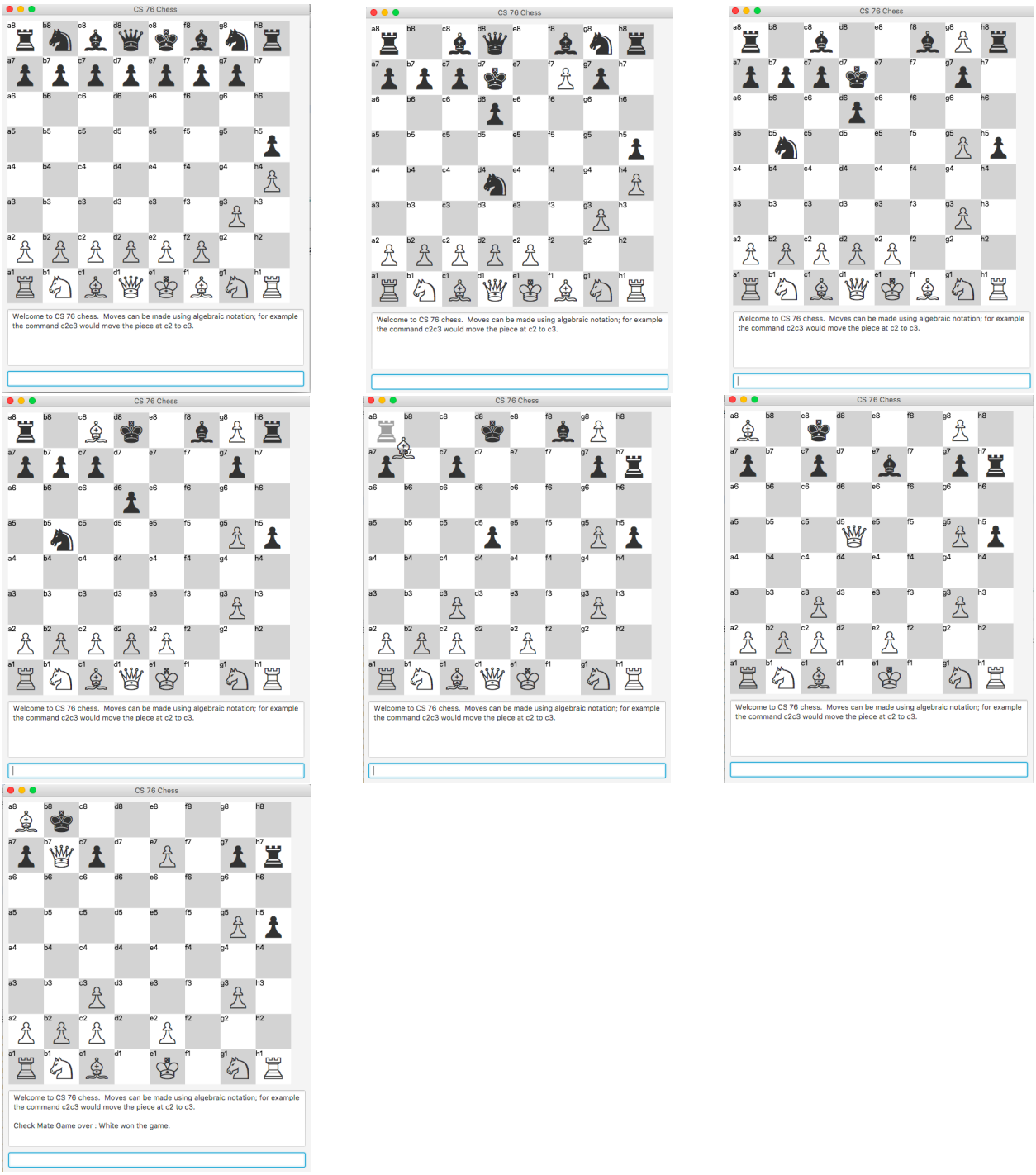

Figure 1: We can clearly see that MiniMax(White) clearly dominates RandomAI(Black).

```
making move 6095
rnbqkbnr/pppppppp/8/8/7P/8/PPPPPPP1/RNBQKBNR b KQkq h3 0 1
making move 6713
r1bqkbnr/pppppppp/n7/8/7P/8/PPPPPPP1/RNBQKBNR w KQkq - 1 2
making move 6623
r1bqkbnr/pppppppp/n7/7P/8/8/PPPPPPP1/RNBQKBNR b KQkq - 0 2
making move 5736
r1bqkbnr/pppppppp/8/7P/1n6/8/PPPPPPP1/RNBQKBNR w KQkq - 1 3
making move 5258
r1bqkbnr/pppppppp/8/7P/1n6/2P5/PP1PPPP1/RNBQKBNR b KQkq - 0 3
making move 6361
r1bqkbnr/pppppppp/8/3n3P/8/2P5/PP1PPPP1/RNBQKBNR w KQkq - 1 4
making move 5193
r1bqkbnr/pppppppp/8/3n3P/8/1PP5/P2PPPP1/RNBQKBNR b KQkq - 0 4
making move 6964
r1bqkbnr/pppp1ppp/4p3/3n3P/8/1PP5/P2PPPP1/RNBQKBNR w KQkq - 0 5
making move 5128
r1bqkbnr/pppp1ppp/4p3/3n3P/8/PPP5/3PPPP1/RNBQKBNR b KQkq - 0 5
making move 5987
r1bqkbnr/pppp1ppp/4p3/7P/5n2/PPP5/3PPPP1/RNBQKBNR w KQkq - 1 6
making move 5388
r1bqkbnr/pppp1ppp/4p3/7P/5n2/PPP1P3/3P1PP1/RNBQKBNR b KQkq - 0 6
making move 6365
r1bqkbnr/pppp1ppp/4p3/3n3P/8/PPP1P3/3P1PP1/RNBQKBNR w KQkq - 1 7
making move 5575
r1bqkbnr/pppp1ppp/4p3/3n3P/8/PPP1P2R/3P1PP1/RNBQKBN1 b Qkq - 2 7
making move 6755
r1bqkbnr/pppp1ppp/1n2p3/7P/8/PPP1P2R/3P1PP1/RNBQKBN1 w Qkq - 3 8
making move 4869
r1bqkbnr/pppp1ppp/1n2p3/7P/8/PPP1P2R/3PBPP1/RNBQK1N1 b Qkq - 4 8
making move 6377
r1bqkbnr/pppp1ppp/4p3/3n3P/8/PPP1P2R/3PBPP1/RNBQK1N1 w Qkq - 5 9
making move 6668
r1bqkbnr/pppp1ppp/B3p3/3n3P/8/PPP1P2R/3P1PP1/RNBQK1N1 b Qkq - 6 9
making move -26063
r1bqkbnr/p1pp1ppp/p3p3/3n3P/8/PPP1P2R/3P1PP1/RNBQK1N1 w Qkq - 0 10
making move 5648
r1bqkbnr/p1pp1ppp/p3p3/3n3P/P7/1PP1P2R/3P1PP1/RNBQK1N1 b Qkq - 0 10
making move 6755
r1bqkbnr/p1pp1ppp/pn2p3/7P/P7/1PP1P2R/3P1PP1/RNBQK1N1 w Qkq - 1 11
making move 6030
r1bqkbnr/p1pp1ppp/pn2p3/7P/P5P1/1PP1P2R/3P1P2/RNBQK1N1 b Qkq g3 0 11
making move 7290
r2qkbnr/pbpp1ppp/pn2p3/7P/P5P1/1PP1P2R/3P1P2/RNBQK1N1 w Qkq - 1 12

.....
```

4.3 Game play (White:Random AI and Black: Alpha Beta Algorithm)

Due to the lack of space in this page the images moved to next page, please find the images in next page.

5 Results when playing Alpha Beta against Minimax

```
making move 5121
rnbqkbnr/pppppppp/8/8/8/N7/PPPPPPPP/R1BQKBNR b KQkq - 1 1
making move 6647
rnbqkbnr/pppppppp1/8/7p/8/N7/PPPPPPPP/R1BQKBNR w KQkq h6 0 2
making move 5446
```

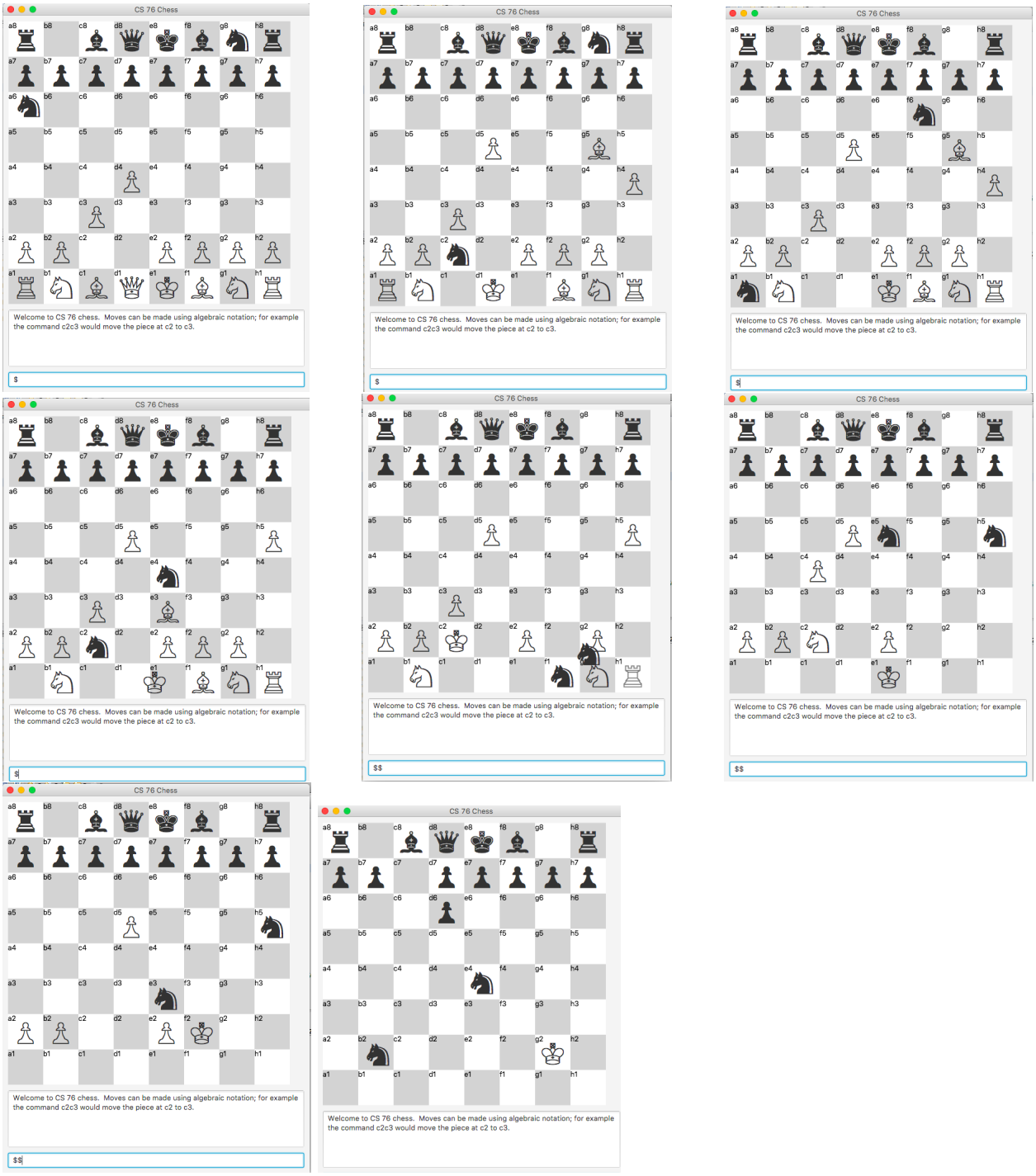

Figure 2: We can clearly see that Alpha Beta(Black) clearly dominates RandomAI(White).

```
rnbqkbnr/pppppppp1/8/7p/8/N4N2/PPPPPPPP/R1BQKB1R b KQkq - 1 2
making move 7094
rnbqkbnr/pppppp2/6p1/7p/8/N4N2/PPPPPPPP/R1BQKB1R w KQkq - 0 3
making move 4176
rnbqkbnr/pppppp2/6p1/7p/8/5N2/PPPPPPPP/RNBQKB1R b KQkq - 1 3
making move 6517
rnbqkbnr/ppppp3/6p1/5p1p/8/5N2/PPPPPPPP/RNBQKB1R w KQkq f6 0 4
making move 6101
rnbqkbnr/ppppp3/6p1/5p1p/7N/8/PPPPPPPP/RNBQKB1R b KQkq - 1 4
making move 6452
rnbqkbnr/pppp4/6p1/4pp1p/7N/8/PPPPPPPP/RNBQKB1R w KQkq e6 0 5
making move -25697
rnbqkbnr/pppp4/6N1/4pp1p/8/8/PPPPPPPP/RNBQKB1R b KQkq - 0 5
making move 7167
rnbqkbn1/pppp4/6Nr/4pp1p/8/8/PPPPPPPP/RNBQKB1R w KQq - 1 6
making move -24722
rnbqkNn1/pppp4/7r/4pp1p/8/8/PPPPPPPP/RNBQKB1R b KQq - 0 6
making move -24708
rnbq1kn1/pppp4/7r/4pp1p/8/8/PPPPPPPP/RNBQKB1R w KQ - 0 7
making move 5121
... ...
```

5.1 Game play (White: Alpha Beta Algorithm and Black: Minimax Algorithm)

Due to the lack of space in this page the images moved to next page, please find the images in next page.

6 Additional Work for Extra Credit

6.1 Opening book-bonus(Extra Credit)

In order to use the book.pgn provide, Please modify the value of book to *true* in AlphaBeta Algorithm. This will make good opening moves, which are already pre-configured.

6.2 Implementation

```
@Override
  public short getMove(Position position) throws Exception {
    alpha = position.getToPlay();
    if (runBook) {
      System.out.println("using the book.pgn for the initial moves in the algorithm as an
    extension");
      // code reference from http://www.cs.dartmouth.edu/~devin/cs76/04_chess/chess.html
      runBook=false;
      book = new Game[120];
      URL url = this.getClass().getResource("book.pgn");
      File f = new File(url.toURI());
      FileInputStream fis = new FileInputStream(f);
      PGNReader pgn = new PGNReader(fis, "book.pgn");
      for (int i = 0; i < 120; i++) {
        Game game = pgn.parseGame();
        game.gotoStart();
        book[i] = game;
      }
    return book(alpha);
    }

        private short book(int alpha) {
    int rand = new Random().nextInt(120);
```

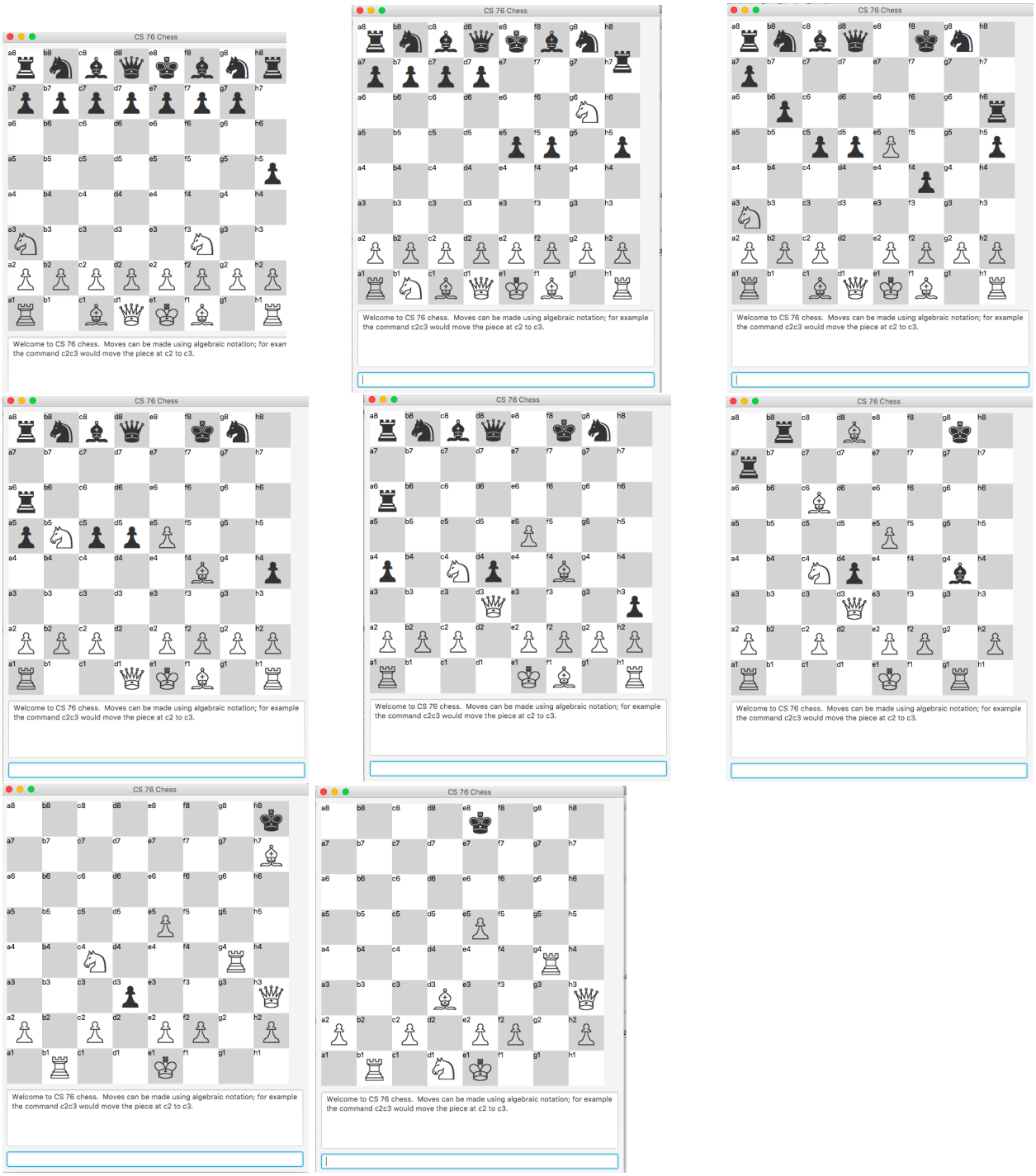

Figure 3: We can clearly see that Alpha Beta(White) clearly dominates Minimax(Black).

```
    if (alpha == 1)
      book[rand].goForward();
    return book[rand].getNextShortMove();
  }
```

6.3 Profiling-bonus(Extra Credit)

Generally chess will have finite number of moves or nodes(states in our case). If *computation is not a hurdle* we can solve chess. Solving a chess is finding a best method to *win always* or in the worst case to draw the match but cannot be lost. According to *Zermelo's theorem*, theoretically a method does exist to solve chess. If we consider a function f(position) we can assign the final state as a tuple *(-1, 0,1)* depending on who wins with the below function :

$$f(position) = \max_{p \to p'} -f(p')$$

Here $p \to p'$ denotes all the legal moves from position *p to p'*. The subtraction is placed because each player will have a turn at a time,which is the same as minimax algorithm. There are approximately 10^{43} positions, so it is very difficult to compute this.Our chess game in Artificial Intelligence uses the function which we are calling as a heuristic evaluation function that approximates the value of position. This function mainly concentrates in on finding the nodes(states) in a exponential manner in the search tree. Generally the depth we can go is 5-10. By employing some approximations and advance searches we can make this chess a better game. We can improve our chess AI performance in the following ways ways:

- Calculating the available nodes in a faster manner.
- Searching the fewer nodes by keeping a limit to get the similar answer to improve the future moves and in future games dynamically.

But the problem arises here is did we really made our chess AI stronger or did we compromise in terms of intelligence for the sake of faster performance? We have to make a log of statistics during move searches like:

- Time taken to complete the search.
- Calculating the number of nodes searched.

We know that our chess won't perform perfectly during the initial moves so we can create several games that are already won by professionals to start the game and in the middle of the game and if possible at the end to avoid draw. After computing this, perform same tests against among themselves to get a better and enhanced version to improve the speed by searching the nodes. By calculating the number of nodes and time taken we can definitely say that performance increased by which level(percentage).

6.3.1 Performance changes for chess AI

We can make a few tweaks for our chess AI to improve it's performance.

- We can make every class variable private and make methods as public to avoid unnecessary modification of class variables.
- The methods we use very often for every class can be designed into a class and make that class as final to avoid code changes to improve the performance.
- Trying to use primitive arrays instead of object arrays or lists.

- Avoiding arrays with two dimension and making it a single array and making that array as Byte instead of int to improve performance and seek time in terms of space.
- Always trying to use arrays, if not possible then going to stack and to list etc, because stacks are way better than lists.
- Returning from the methods(fail fast) i.e. handling the corner cases efficiently.
- Avoiding the late move reduction to perform better while searching the nodes.
- Keeping the track of Null Move Pruning, Check and Checkmate conditions.

6.4 Move reordering-bonus(Extra Credit)

Alpha Beta algorithm works well in the middle of the game and also at the start if we imply some techniques by using book.pgn file but at the end it still suffers horizon effects. For this algorithm to perform well we need to search the best node(best move) prior itself. This case is mainly true for principal variation nodes. The main idea here is to get closer to the minimal tree. There are three kinds of Move reordering techniques.

- PV-Move from the principal variation of the previous Iteration.
- Hash Move - stored move from Transposition Table, if available.
- Internal Iterative Deepening - if no hash move is available, likely only at PV-Nodes.

6.4.1 Principal variation

Priniciapl variation is a methodology to calculate the best moves in an sequential order and the end to finalize a best move that is to be played. This is simialr to iterative deeping search, which we have done in the previous assignments. After a certain depth is reached then to analyze the best move our algorithm will have an idea about Principal variation nodes about the current iteration to take advnatage of the previously analyzed best moves and considers the best among the best moves based upon the depth.

6.4.2 Principal variation Move

A Principal variation move is internally a part of Principal variation and so we can conclude that best move found in the last iteration of an Iterative deepening search with a specific depth. We can hash this move or we can store this in the transposition table for future usage to increase the algorithm efficiency.

6.4.3 HashMove using Transposition Table

HashMove is a special move calculated from the transposition table. This is a best move calculated from best moves of the principal variation node i.e. a pv move that is stopped at a particular depth. This move is given more importance and it is searched first. Actually this works on fail fast, where professor Devin Balkcom was mentioning in the class. We need apply some conditions to test this move if this move is best or not in some cases.

6.4.4 Implementation

```
class TranspositionTable {
    public int hueristic, versatility, depth;

    TranspositionTable(int hueristic, int versatility, int depth) {
      this.hueristic = hueristic;
      this.versatility = versatility;
      this.depth = depth;
```

```
    }

    public int getHueristic() {
      return hueristic;
    }

    public void setHueristic(int hueristic) {
      this.hueristic = hueristic;
    }

    public int getVersatility() {
      return versatility;
    }

    public void setVersatility(int versatility) {
      this.versatility = versatility;
    }

    public int getDepth() {
      return depth;
    }

    public void setDepth(int depth) {
      this.depth = depth;
    }

  }

    private int calcHueristic(Position pos) {
    if (pos.isTerminal()) {
      if (pos.isStaleMate()) {
        return 0;
      } else if (pos.isMate()) {
        return (pos.getToPlay() == alpha) ? Integer.MAX_VALUE : Integer.MIN_VALUE;
      } else {
        return 0;
      }
    } else {
      return nextNode(pos);
    }

  }
```

6.4.5 Internal Iterative Deepening:

This methodology is used when our alpha beta algorithm doesn't have a best move that is calculated from the previous searched from Principal variation or from the hashmove of the transposition table. This method is used to get the best move by first searching the current position to a decreased depth than the actual depth. According to by John J. Scott and Thomas Anantharaman reducing the depths at principal variation nodes with no good move in the transposition table is wastage and the heuristic functions search time increases by ten folds larger than the actual search time.

6.5 Improved cut-off test-bonus(Extra Credit)

6.5.1 Horizon Effect

Generally we limit a search algorithm with a given depth or else we may or may not find an optimal solution. But when we limit the depth we many encounter a negative event but good thing is we can postpone this

negative event. This occurs in our chess algorithm because we analyzed only partial game tree. This whole effect on any search algorithm(chess in our case) is called Horizon Effect.

According to *Tony Marsland* an unsolved problem in chess algorithms is delaying the moves that enables to the loss of material beyond the given depth, therefore the loss is hidden (Berliner, 1973 [3]).

According to *Gerald Tripard* if a problem is extended beyond the depth in a search algorithm, it may make a wrong choice of sacrifice to avoid loss.

So the *horizon effect* occurs when we postpone the event in order to avoid loss. For example this is obvious that it is good to save the queen by losing our four valuable pawns. This horizon effect is less visible in algorithms that has quiescence searches *(Hermann Kaindl, 1982 [4])*. The algorithms with programs with inadequate quiescence search suffer more horizon effects. Additionally extensions such as check and check mate extensions can be designed to reduce horizon effects.

6.5.2 Quiescence Search

Our chess AI performs a limited quiescence search at the end of the search and only moves fewer steps. The purpose of the search is to make moves such that are quiet where no wining moves are made and this is needed to avoid horizon effect. Stopping the search after a depth is reached and then calculating next moves is very risky. Quiescence searches are small and almost 50%*to*90% nodes are spent here.

In quiescence searches we need a metric to calculate and then return from the search if there are no captures currently available. At the start of quiescence search we create a lower bound called metric and by intuition we can say that there is atleast one move from here to beat the lower bound metric. Here there are two conditions

- fail fast: If the metric is is greater than beta we return the metric.
- fail safe: If the metric is less than beta then return the beta as lower bound.

In other case use recursion to continue search making this metric as lower bound to consider if any move can increase alpha. Below is the pseudo implementation of quiescence search.

```
int quiescence( int alpha, int beta ) {
//calculate function assigns the lower bound value for search.
    int metric = calculate();
    if( metric >= beta )
        return beta;
    if( alpha < metric )
        alpha = metric;

    while(calculating the every move and check conditions where metric is not valid)  {
        makeCapture();
        metric = metric-quiescence( -beta, -alpha );
        backTrack();

        if( metric >= beta )
            return beta;
        if( metric > alpha )
           alpha = metric;
    }
    return alpha;
}
```

6.5.3 Additional info

Some chess algorithms will consider check in this quiescence search because it may lead to end of the game and has to be resolved. The metric will become void in this case. Many of the chess algorithms stop this case by delta pruning techniques.

6.6 Related work section-bonus(Extra Credit)

On May11, 1997 IBM's Deep Blue beat the world chess champion after a six game match, which resulted in two wins for the Deep Blue and one win for the person playing against it and the remaining were draw matches. At that point of time it was a ground breaking move in computer science where the conflict raised between human and machine intelligence in a competition.

In Deep Blue[1], they improved the algorithm of the chess to an extensible level, which improved the performance by making use of complex evaluation function, which is known as Hill Climbing Approach. This evaluation function also makes estimate with the help of heuristic functions that decreases a lot of time from exact evaluations.

Along with these improvements they also suggested the use of good opening moves, which we used in this assignment as an extension. They moves are already pre-configured from the best chess playing people. The scientists made use of parallel system to calculate the game tree searches. As a result the the algorithm functions better than the existing algorithm. Additionally, the Deep Blue scientists used another technique called *Extended book* mechanism that includes the bonuses or penalties to the moves that were already played in the database to further enhance the algorithm. Among the moves the moves that are played by professional people will be preferred, similar to the priority queue. Similarly they make use of *End Game Databases* as well.

The scientists also accept that these pruning mechanisms would have improved their performance, and that parallel search efficiency could have been increased. Moreover genetic algorithm could also used to improve the end games because most of the current chess suffers from horizon effect at the end of the game. In this paper to improve the end game the authors tested the the King-Rook-King endgame, and found that genetic algorithm produced very good results when compared brute force or end game books. This genetic programming can be used to combine elementary chess patters defined by many chess professions. In this method i.e. Genetic Programming approach, we construct binary trees with operators as nodes and moves as leaves. Each tree is a chromosome, and we calculate Crossovers and Mutations to check the strategy of the game.

7 Additional References

- $https://en.wikipedia.org/wiki/Solving\_chess$
- $https://erikbern.com/2014/11/29/deep-learning-for-chess/$
- $https://chessprogramming.wikispaces.com/$
- $http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/$
- $https://www.cs.swarthmore.edu/\ meeden/cs63/f05/minimax.html$
- $https://www.cs.cornell.edu/courses/cs312/2002sp/lectures/rec21.htm$

References

[1] Murray Campbell, A Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1):57–83, 2002.