

# Chess Project - AI - COSC 3P71

Elya Denysova 6667596  
Shahrear Chowdhury, 6605273

## Introduction

The goal of the final project was to demonstrate an adversary AI algorithm by implementing it in a chess game. The Minimax algorithm was used to evaluate possible moves in a chess game, assign evaluation for each possible strategy, and based on that score choose the 'best' move. Alpha-Beta pruning was used to cut off some of the branches in a recursive tree to improve efficiency of the algorithm.

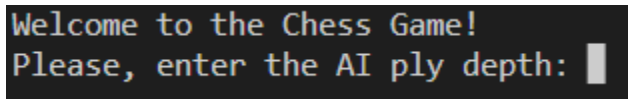
The interface is terminal input/output. Users can select to play with another human player or against AI. There is an option to start with any board configuration. There is also a ply parameter to be selected by the user which determines the tree depth for the minimax algorithm.

## Compile & Run instructions

Tools: Java (8+), VSCode

1. Compile the source file(click on App.java to see compile button). (Tested on VsCode with java io, util and generic packages).
2. For command line:
  - Navigate inside src folder
  - Compile: `javac *.java`
  - Run: `java App`
3. The user is asked to input the ply depth for AI moves, which is required for the Alpha-Beta pruning search algorithm.

**Note:** the best ply depths have been 1 to 4 so far.



```
Welcome to the Chess Game!  
Please, enter the AI ply depth: █
```

4. The user is then asked if they want to build a board in their own way. If the user selects "yes", then they are asked where to place which pieces.

```

Welcome to the Chess Game!
Please, enter the AI ply depth: 3
Do you want to build your own board?
1: Yes 0: No
1
Building Chess Board...

Note: A chess board must have atleast both player's king pieces. Pieces will be asked to be entered
for both players. Selecting exit (asked for both players) will quit adding anymore pieces and star
t the game.

Firstly, make your board. Exit to quit and build oponent's board.

Do you want to place your R0?
1: Yes 2: No 3: Exit

```

### Note:

In the program a typical chess board looks like this:

```

A typical chess board:
R0  N0  B0  Q0  K0  B1  N1  R1
P0  P1  P2  P3  P4  P5  P6  P7
-   -   -   -   -   -   -   -
-   -   -   -   -   -   -   -
-   -   -   -   -   -   -   -
-   -   -   -   -   -   -   -
P8  P9  P10 P11 P12 P13 P14 P15
R2  N2  B2  Q1  K1  B3  N3  R3

```

Green and red colored pieces represent player-0 and player-1 or AI (enemy) respectively. Here, “R” = Rook; “N” = Knight; “B” = Bishop, “Q” = Queen, “K” = King, “P” = Pawn, “-” = Empty; 0-15 = Number of pieces of a particular piece. A board can have at least 2 kings (player’s and opponent’s) to build a board manually. A player can place any pieces anywhere inputting the position X and Y. The board itself is a 2D array (Piece[][]), where each position represents a piece (Eg., Piece[0][0] represents “R0” for the above board).

5. The user is then asked to select if they want to play Human vs Human or Human vs Computer (AI). Any other numbers, exits the program.

```

Building Chess Board...
R0  N0  B0  Q0  K0  B1  N1  R1
P0  P1  P2  P3  P4  P5  P6  P7
-   -   -   -   -   -   -   -
-   -   -   -   -   -   -   -
-   -   -   -   -   -   -   -
-   -   -   -   -   -   -   -
P8  P9  P10 P11 P12 P13 P14 P15
R2  N2  B2  Q1  K1  B3  N3  R3

Please choose whether do you want to play with human or computer
0 : Human Vs. Human   1 : Human Vs. Computer   Any Other Number : Exit

```

- The game is now ready to be played. For Human vs Human, the program asks to select which available piece the player (player-0 or player-1) wants to move. During Human vs AI, only player-0 needs to input, AI does its moves automatically.

```

Player-0 please choose which piece you want to move-

Your available pieces are:
0 : K   1 : Q   2 : B   3 : N   4 : R   5 : P

```

- After selecting a desired piece to be moved, the player's are asked to choose available pieces in that name. Such as below,

```

Player-1 please choose which piece you want to move-

Your available pieces are:
0 : K   1 : Q   2 : B   3 : N   4 : R   5 : P
5

Please select one from the available pieces in a piece:
8 : P8   9 : P9  10 : P10  11 : P11  12 : P12  13 : P13  14 : P14  15 : P15
10

```

Here we can see that, after selecting Pawn, all the available pieces of that piece are displayed for Player-1. Then, the Player-1 chooses the desired piece that they want to move and all the available directions for that particular piece and maximum steps are displayed.

**Note:** "Available special moves" are considered as Enpassant, Piece promotion or King castling. All other moves are "Available moves".

```
Available special moves:
```

```
1 : Top
```

```
Please select available P directions:
```

```
1
```

```
Please select available steps from available total for the piece:
```

```
Total available steps in the selected direction : 2
```

```
2
```

8. En passant: En Passant moves are available when two opposite pawns are crossing each other, the player whose move is current is able to eat the opponent's pawn and move 1 step ahead. For instance, clearly for Player-0 an En Passant move is available and when the user selects to perform en passant, it is immediately performed.

```
Please see available P directions:
```

```
R N B K - B N R
```

```
P P P - P P P P
```

```
- - - Q - - - -
```

```
- - P P - - - -
```

```
Q - - x - - - -
```

```
- - - - - - - -
```

```
P P - P P P P P
```

```
R N B - K B N R
```

```
Available special moves:
```

```
19 : enPassantRight 1 : Top
```

```
Please select available P directions:
```

```
1
```

```
Performing enpassant.
```

```
R0 N0 B0 K0 - B1 N1 R1
```

```
P0 P1 P2 - P4 P5 P6 P7
```

```
- - - Q0 - - - -
```

```
- - - - - - - -
```

```
Q1 - P3 - - - - -
```

```
- - - - - - - -
```

```
P8 P9 - P11 P12 P13 P14 P15
```

```
R2 N2 B2 - K1 B3 N3 R3
```

```
Player-1 please choose which piece you want to move-
```

9. Castling: It is a special move that interchanges the king and rook if they haven't moved yet. For instance, when the user selects the Castle Right special move, king and rook's position are interchanged.

Please see available K directions:

```
R N K - - B N R
P - P - P P P P
B P - Q - - - -
- - - - - - - -
Q - P - P - - -
- - - B - - - N
P P - P x P P P
R N B x K x - R
```

Available special moves:

17 : Castle Right

Available moves:

1 : Top 15 : left 16 : right

Please select available K directions:

17

Please select available steps from available total for the piece:  
Performing castling.

```
R0  N0  K0  -  -  B1  N1  R1
P0  -  P2  -  P4  P5  P6  P7
B0  P1  -  Q0  -  -  -  -
-  -  -  -  -  -  -  -
Q1  -  P3  -  P12 -  -  -
-  -  -  B3  -  -  -  N3
P8  P9  -  P11 -  P13 P14 P15
R2  N2  B2  -  -  R3  K1  -
```

Player-0 please choose which piece you want to move-

Your available pieces are:

0 : K 1 : Q 2 : B 3 : N 4 : R 5 : P

10. Piece Promotion: This happens when the opponent's pawn piece travels to the far end of the board. As an example, the "P3" being moved to the very end of the opponent's part, enables it to perform the piece promotion.

```

Available special moves:
21 : piecePromotion  22 : piecePromotionLeft  1 : Top  2 : diagonalLeftBottom
Please select available P directions:
21

Please select available steps from available total for the piece:
Performing piece promotion.
Please select the piece you want to promote.
0 : K  1 : Q  2 : B  3 : N  4 : R  5 : P
1
R0  N0  K0  -  -  B1  N1  R1
P0  -  P2  -  P4  P5  P6  P7
B0  P1  -  Q0  -  -  -  -
-  -  -  -  -  -  -  -
Q1  P9  -  -  P12  -  -  -
-  -  -  B3  -  -  -  N3
P8  B2  -  P11  -  P13  P14  P15
R2  N2  Q2  -  -  R3  K1  -

```

11. In case of checkmate the player on checkmate has to make a legal move to save the King.

```

R0  N0  B0  -  K0  B1  N1  R1
P0  P1  P2  -  P4  P5  P6  P7
-  -  -  Q0  -  -  -  -
-  -  -  P3  -  -  -  -
Q1  -  P10  -  -  -  -  -
-  -  -  -  -  -  -  -
P8  P9  -  P11  P12  P13  P14  P15
R2  N2  B2  -  K1  B3  N3  R3

0 its checkmate, please make a legal move to save your king.
Player-0 please choose which piece you want to move-

Your available pieces are:
0 : K  1 : Q  2 : B  3 : N  4 : R  5 : P

```

As an example from above, “Q1” check-mates opponent’s king and is asked to save the King. And, the example below, changing the king’s position saves the king and play continues.

```

Available moves:
2 : diagonalLeftBottom    15 : left
Please select available K directions:
15

Please select available steps from available total for the piece:
Total available steps in the selected direction : 1

```

The program ends when there are no moves available for a check mated player or stalemate (when no pieces are left except king).

## Code overview

Player turns: Player.moves function takes the board and playerName to perform the turn for a player. Here, the function has the 2D array board and 0 is Player-0 or 1 is Player-1 as necessary parameters to perform the next moves. Method “checkKinCanBeSaved” is used to find pieces that will help to save a checkmated king. “checkKingSafe” method helps to determine if the king is checkmated and any possible moves can be done as bare minimum to save the king. Otherwise, the game ends as the opponent to win is declared.

```

// check if the king is safe to move or else its checkmate and until user makes
// a move that save king
if (King.checkKingSafe(board.board, playerName: 0)) {
    if (Utils.checkKingCanBeSaved(board.board, playerName: 0)) {
        // checkmate
        while (King.checkKingSafe(board.board, playerName: 0)) {
            System.out.println(0 + " its checkmate, please make a legal move to save your king.");
            Player.moves(board, playerName: 0);
        }
    } else {
        System.out.println("Player-" + 1 + " has won the game.");
    }
} else {
    Player.moves(board, playerName: 0);
}

if (King.checkKingSafe(board.board, playerName: 1)) {
    if (Utils.checkKingCanBeSaved(board.board, playerName: 1)) {
        while (King.checkKingSafe(board.board, playerName: 1)) {
            System.out.println(1 + "its checkmate, please make a legal move to save your king.");
            Player.moves(board, playerName: 1);
        }
    } else {
        System.out.println("Player-" + 0 + " has won the game.");
    }
} else {
    Player.moves(board, playerName: 1);
}

```

## Implementation details (Heuristics)

```

/**
 * Returns the heuristic value for the board
 *
 * @param board
 * @return
 */
public static int calculateHeuristic(Piece[][] board) {

    return materialW * material(board, player: 2)
        + materialopponentW * material(board, player: 0)
        + availMovesW * availMoves(board, player: 2)
        + availMovesopponentW * availMoves(board, player: 0)
        + checkStatusW * checkStatus(board, player: 1)
        + checkStatusopponentW * checkStatus(board, player: 0)
        + Evaluator.evaluateBoard(board, playerName: 2)/1000
        + Evaluator.evaluateBoard(board, playerName: 0)/1000;
}

```

Minimax algorithm recursively generates successor states, in this case chess board configurations, evaluates them based on heuristics, and then chooses the strategy for the next move. Minimax algorithm selection is based on altering goals of AI and user, where AI attempts to get higher board evaluation and assumes that user tries to get lowest board evaluation.

For this project, evaluation consisted of linear function: the sum of few different heuristics multiplied by weight. The weight represents the importance of certain heuristics in decision making.

#### *Material heuristic*

Evaluates the board based on which pieces are available to the AI and opponent (human player). Each piece has a value: queen - 8, pawn - 1, bishop and knight - 3, rook - 5. It encourages AI to protect valuable pieces and to capture opponents' pieces.

#### *Available moves*

Corresponds to the amount of moves available on the next turn (assuming the same board configuration). It encourages AI to dominate the board and leave weaker moves to the opponent.

#### *Check status*

It analyzes if the king is safe in such configuration to avoid check/checkmate if this strategy was to be selected.

#### *Piece distribution*



Evaluates the position of different pieces on the board. Used a matrix, to determine better location for each piece and assign score accordingly. This is added in order to make AI advance their pieces forward with preference for certain locations that can give an advantage.

#### *Check status*

Analyses position of the king and its safety, basically checks for possible check or checkmate. It encourages strategies that won't lead to losing the game.

## **References**

1. Minimax algorithm and Alpha-Beta pruning: textbook pseudocode from S. Russel, P. Norvig, "Artificial Intelligence A Modern Approach" 4th edition, Pearson Education Inc., 221 River Street, Hoboken, NJ07030
2. Heuristic ideas: <https://github.com/lamesjim/Chess-AI>
3. Adam Berent's chess evaluation, [Chess Board Evaluation - Adam Berent Software & Hobbies](#)