AI vs Player Chess Game

Made by: Anurag Shah, B16CS034

Chinmay Garg, B16CS041

We have written a C++ program for a chess game which is to be played between an AI and a human player.

Major Features in the program are:

1. Move generation and validation
2. Position evaluation
3. Minimax algorithm
4. Alpha beta pruning
5. Difficulty adjustment

# 1. Move Generation and Validation

1. All possible and valid moves are stored in a vector named ‘possibleMoves’ in ‘ChessBoardClass.h’.
2. On reaching the end of the column the Pawn in automatically promoted to the Queen.
3. AI’s move is decided by the minimax and alpha beta pruning algorithm.

# 2. Position Evaluation

Evaluation function has two parts:

1. Piece value evaluation

Each piece is assigned some value according to their relative strength in the game. Values of the white pieces are in positive and that of black pieces are in negative.

|  |  |
| --- | --- |
| King | 9000 |
| Queen | 900 |
| Rook | 500 |
| Bishop | 300 |
| Knight | 300 |
| Pawn | 100 |

1. Piece position evaluation

Now the above evaluation in criterion in alone, is quite naïve as we count only the material that is on the board regardless of the situation.

Also, after testing, we found that if AI plays with only the above given evaluation criterion, it will never make an aggressive move and for first few moves it will keep repeating the move: right-most pawn moves one step, then queen-side rook moves one step forward and backward repetitively, unless opponent’s piece comes to close vicinity to any of its piece.

So, to improve this, we add to the evaluation a factor that takes in account the position of the pieces. For example, a knight on the center of the board is better (because it has more options and is thus more active) than a knight on the edge of the board.

Pawn:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| 10 | 10 | 20 | 30 | 30 | 20 | 10 | 10 |
| 05 | 05 | 10 | 25 | 25 | 10 | 05 | 05 |
| 00 | 00 | 00 | 20 | 20 | 00 | 00 | 00 |
| 05 | -05 | -10 | 00 | 00 | -10 | -05 | 05 |
| 05 | 01 | 10 | -20 | -20 | 10 | 10 | 05 |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

Knight:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| -50 | -40 | -30 | -30 | -30 | -30 | -40 | -50 |
| -40 | -20 | 00 | 00 | 00 | 00 | -20 | -40 |
| -30 | 00 | 10 | 15 | 15 | 10 | 00 | -30 |
| -30 | 05 | 15 | 20 | 20 | 15 | 05 | -30 |
| -30 | 00 | 15 | 20 | 20 | 15 | 00 | -30 |
| -30 | 05 | 10 | 15 | 15 | 10 | 05 | -30 |
| -40 | -20 | 00 | 05 | 05 | 00 | -20 | -40 |
| -50 | -40 | -30 | -30 | -30 | -30 | -40 | -50 |

Bishop:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| -20 | -10 | -10 | -10 | -10 | -10 | -10 | -20 |
| -10 | 00 | 00 | 00 | 00 | 00 | 00 | -10 |
| -10 | 00 | 05 | 10 | 10 | 05 | 00 | -10 |
| -10 | 05 | 05 | 10 | 10 | 05 | 05 | -10 |
| -10 | 00 | 10 | 10 | 10 | 10 | 00 | -10 |
| -10 | 10 | 10 | 10 | 10 | 10 | 10 | -10 |
| -10 | 05 | 00 | 00 | 00 | 00 | 05 | -10 |
| -20 | -10 | -10 | -10 | -10 | -10 | -10 | -20 |

Rook:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 05 | 10 | 10 | 10 | 10 | 10 | 10 | 05 |
| -05 | 00 | 00 | 00 | 00 | 00 | 00 | -05 |
| -05 | 00 | 00 | 00 | 00 | 00 | 00 | -05 |
| -05 | 00 | 00 | 00 | 00 | 00 | 00 | -05 |
| -05 | 00 | 00 | 00 | 00 | 00 | 00 | -05 |
| -05 | 00 | 00 | 00 | 00 | 00 | 00 | -05 |
| 00 | 00 | 00 | 05 | 05 | 00 | 00 | 00 |

Queen:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| -20 | -10 | -10 | -05 | -05 | -10 | -10 | -20 |
| -10 | 00 | 00 | 00 | 00 | 00 | 00 | -10 |
| -10 | 00 | 05 | 05 | 05 | 05 | 00 | -10 |
| -05 | 00 | 05 | 05 | 05 | 05 | 00 | -05 |
| 00 | 00 | 05 | 05 | 05 | 05 | 00 | -05 |
| -10 | 05 | 05 | 05 | 05 | 05 | 00 | -10 |
| -10 | 00 | 05 | 00 | 00 | 00 | 00 | -10 |
| -20 | -10 | -10 | -05 | -05 | -10 | -10 | -20 |

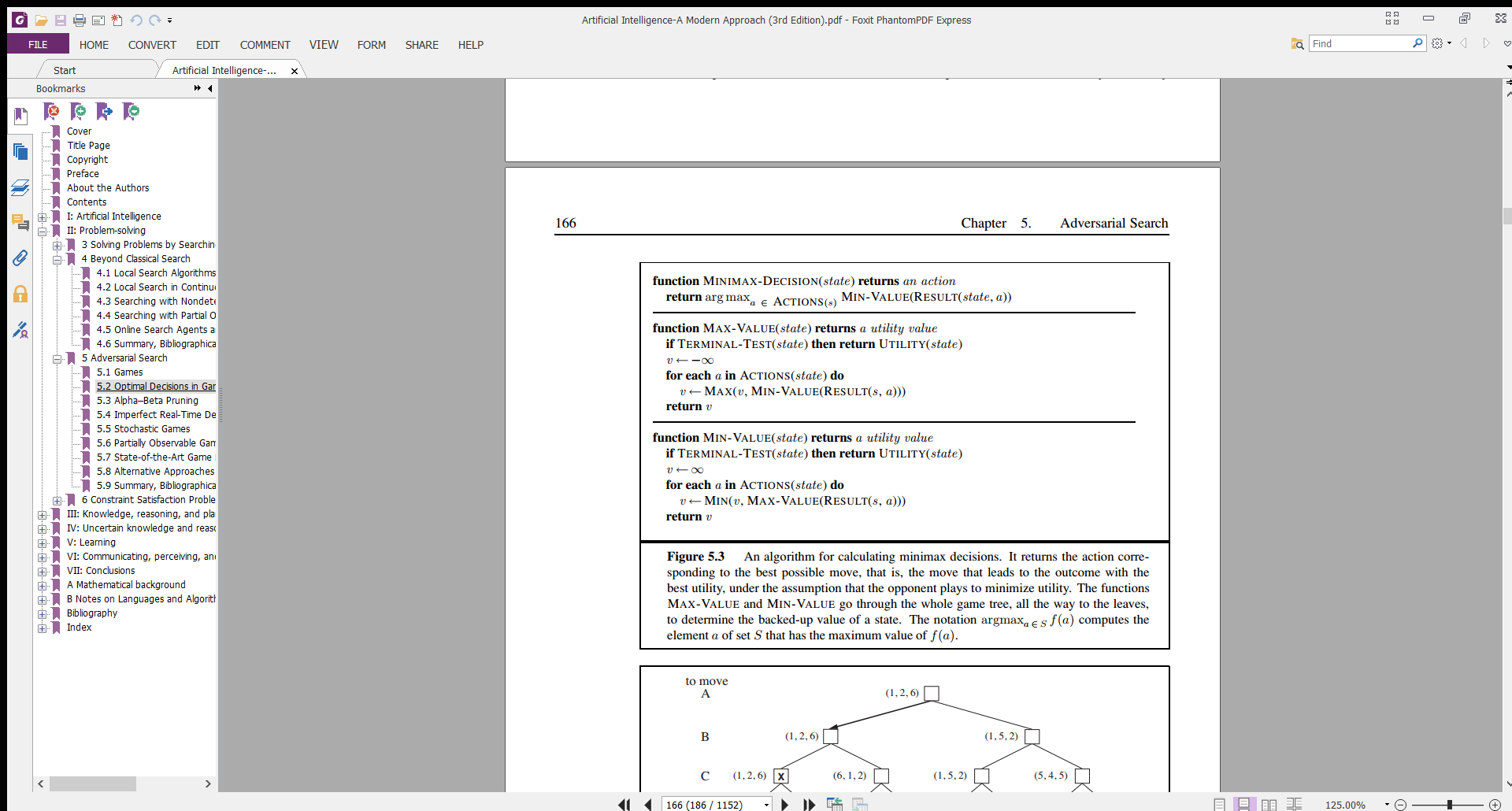
King:

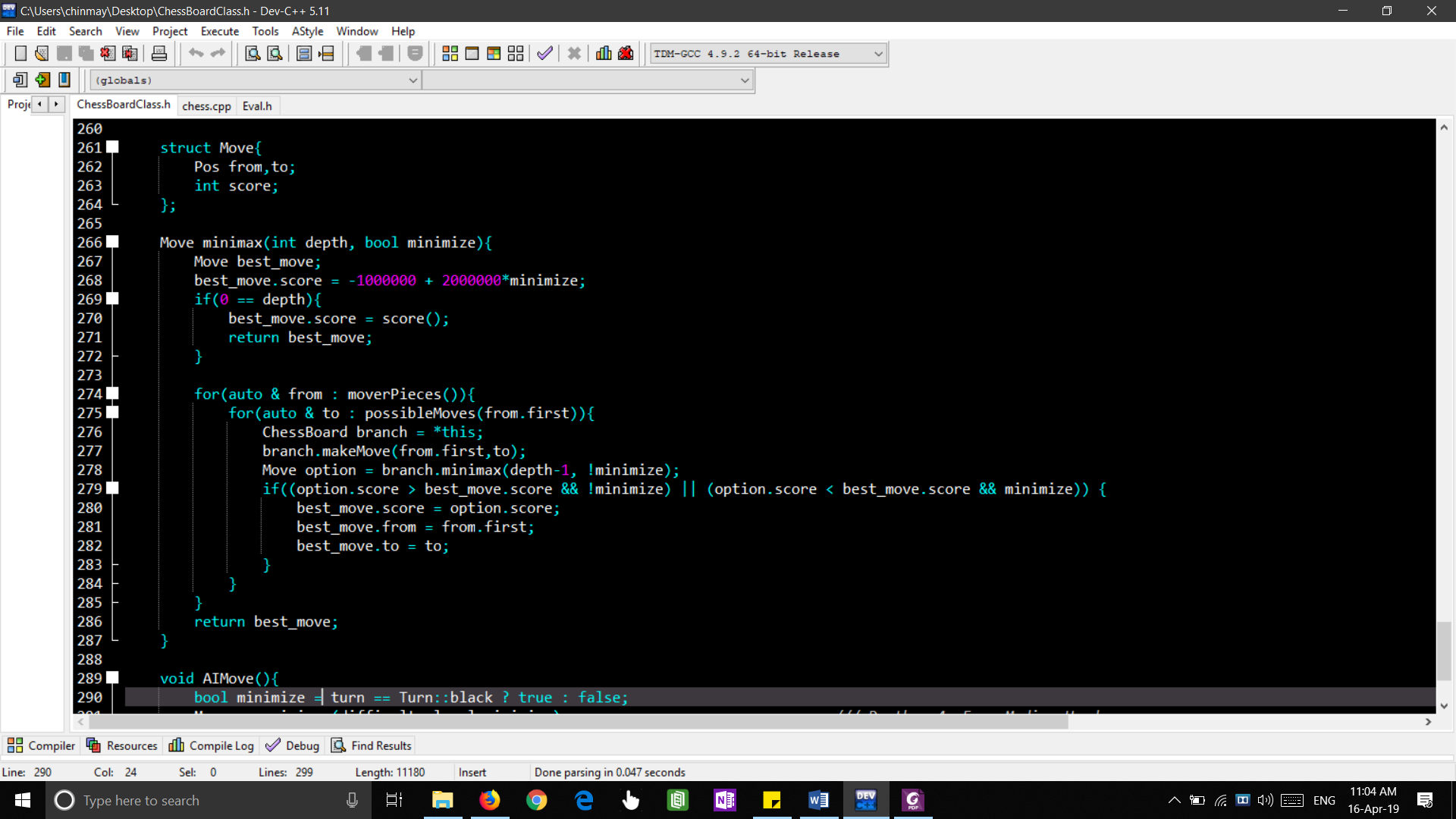
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| -30 | -40 | -40 | -50 | -50 | -40 | -40 | -30 |
| -30 | -40 | -40 | -50 | -50 | -40 | -40 | -30 |
| -30 | -40 | -40 | -50 | -50 | -40 | -40 | -30 |
| -30 | -40 | -40 | -50 | -50 | -40 | -40 | -30 |
| -20 | -30 | -30 | -40 | -40 | -30 | -30 | -20 |
| -10 | -20 | -20 | -20 | -20 | -20 | -20 | -10 |
| 20 | 20 | 00 | 00 | 00 | 00 | 20 | 20 |
| 20 | 30 | 10 | 00 | 00 | 10 | 30 | 20 |

# 3. Minimax Algorithm

We created a search tree from which the algorithm can chose the best move. This is done by using the Minimax algorithm.

In this algorithm, the recursive tree of all possible moves is explored to a given depth, and the score is evaluated at the ending “leaves” of the tree.

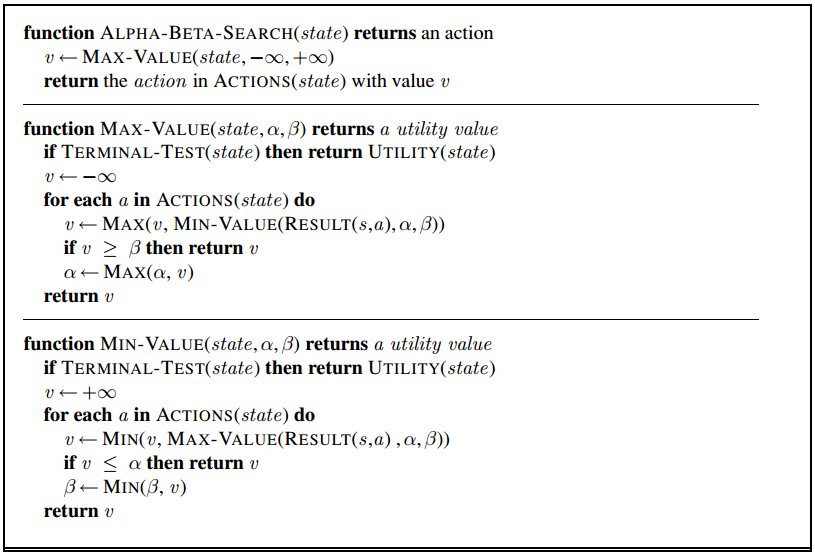


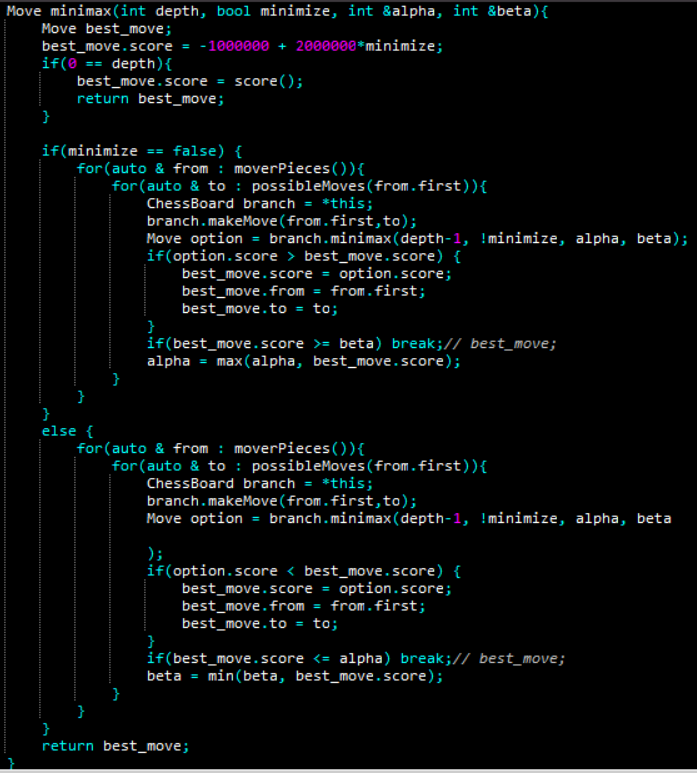


# 4. Alpha Beta Pruning

Alpha-beta pruning is an optimization method to the minimax algorithm that allows us to disregard some branches in the search tree. This helps us evaluate the minimax search tree much deeper, while using the same resources.

The alpha-beta pruning is based on the situation where we can stop evaluating a part of the search tree if we find a move that leads to a worse situation than a previously discovered move. The alpha-beta pruning does not influence the outcome of the minimax algorithm — it only makes it faster. The alpha-beta algorithm also is more efficient if we happen to visit firstthose paths that lead to good moves.





# 5. Difficulty Adjustment

We can set the difficulty of AI player by adjusting the depth of tree to search in MINIMAX algorithm. Difficulty can be set to three levels:

1. Easy (depth = 2)
2. Medium (depth = 4)
3. Hard (depth = 6)