ARTIFICIAL INTELLIGENCE

Assignment 3

MinMax Algorithm

Report

Harris Aamir

20i-0943

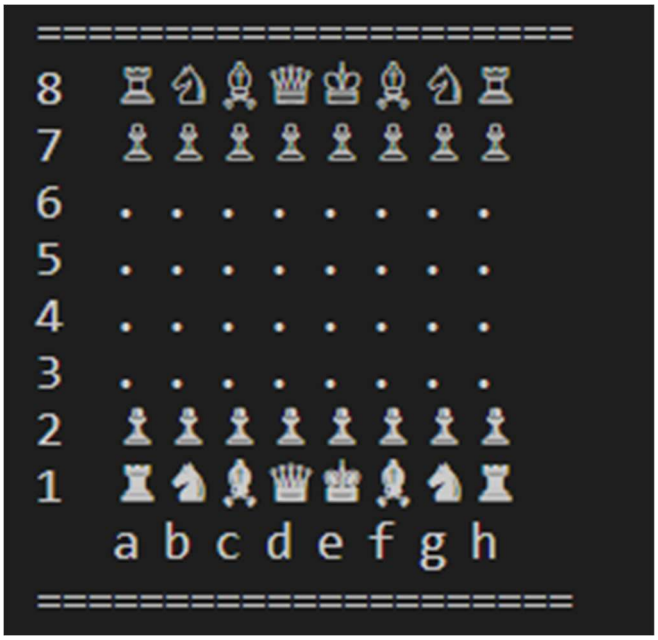
SE-S

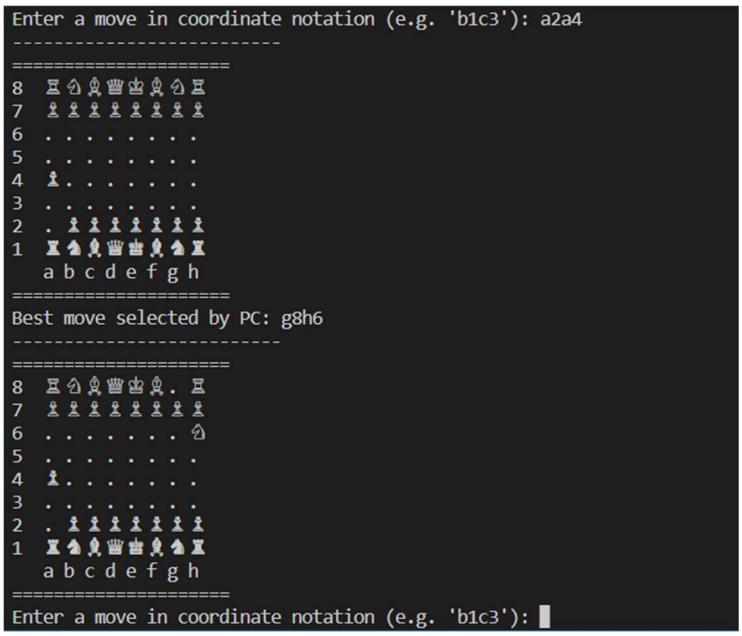
Code Explanation

The chess game was implemented in the assignment using the python-chess library. The library allowed multiple utilities such as board, move of piece, recording moves, checking for legal and illegal moves and many others. The following code snippet shows code of the game being executed. The while loop is executed till the game is not over.

```
while not board.is_game_over():
    printBoard(board)
    if board.turn == chess.WHITE:
        try:
            moveInput = input("Enter a move in coordinate notation (e.g. 'b1c3'): ")
            userMove = chess.Move.from_uci(moveInput)
            if chess.Move.from_uci(moveInput) in board.legal_moves:
             board.push_uci(moveInput)
            else:
              print("Illegal move. Try again.")
        except ValueError:
          print("Invalid move notaion, try again.")
    else:
        aiMove = moveByAI(board)
        board.push(aiMove)
        aiMoves.append(aiMove)
        print("Best move selected by PC:", aiMove)
    if board.is_checkmate():
        print("Checkmate! Game over.")
        break
    print("-------------------------")
```

First move is played by white who is player and second move is played by Ai who is black. The user inputs the moves in coordinates notation that is mentioning the current position of the piece followed by final position. The AI move is generated from the function moveByAI which takes current board as argument and returns the best possible move that can be played. The screenshot below shows the output of the code. Initially the board is displayed on terminal and user input is required. The labels are also added to make inputs easier.





The inputs a2a4 means that piece at a2 shall be placed to a4. After the user makes his move, Ai evaluates the current board and makes the best move. Each move is made on the board and its value is calculated. Next 4 moves are seen and the one with lowest damage is played. The following code shows the minmax function which is called recursively in to find the best move. If we increase the number of moves to be seen ahead, the computational time increase exponentially for example, it takes more than 10 minutes to find best move if we look at next 20 moves.

The following code snippet shows the evaluation function which calculates value of current board. The board is evaluated on value preassigned to every piece based on the value they keep in game. The value of each user piece is added in score and returned. And for Ai each piece value is subtracted.

```
piece_values = {
    chess.PAWN: 5,
    chess.KNIGHT: 15,
    chess.BISHOP: 15,
    chess.ROOK: 25,
    chess.QUEEN: 50,
    chess.KING: 1
}

def evaluate_board(board):
    score = 0
    for square in chess.SQUARES:
        piece = board.piece_at(square)
        if piece is not None:
            val = piece_values[piece.piece_type]
            if piece.color == chess.WHITE:
                score += val
            else:
                score -= val
    return score
```

The following code snippet shows the minMax algorithm. If the depth is 0 or game is over, the function returns the value of board (recursion terminating condition) If the player is maximizing, means that this is opponent who wants to get max value of board. The loop checks for every legal move and stored into alpha.. This function is called recursively for next player to check the state of game. The score is returned for both players which is compared to previous score and saved in alpha or beta respectively. If beta is greater then alpha, pruning occurs which is checked in checkPruning function. Loop is break when pruning occur thus no further legal move is checked and score is returned.

```
48  def checkPruning(beta,alpha):
49      if beta <= alpha:
50          return True
51      else: return False
52
53  def minMax(board, depth, alpha, beta, player):
54      if depth == 0 or board.is_game_over():
55          return evaluate_board(board)
56
57      if player=="maximizingPlayer":
58          maxScore = -1000000
59          for move in board.legal_moves:
60              board.push(move)
61              score = minMax(board, depth - 1, alpha, beta, "minimizingPlayer")
62              board.pop()
63              maxScore = max(maxScore, score)
64              alpha = max(alpha, score)
65              if checkPruning(beta,alpha):
66                  break
67          return maxScore
68      else:
69          minScore = 1000000 #represents pos infinty
70          for move in board.legal_moves:
71              board.push(move)
72              score = minMax(board, depth - 1, alpha, beta, "maximizingPlayer")
73              board.pop()
74              minScore = min(minScore, score)
75              beta = min(beta, score)
76              if checkPruning(beta,alpha):
77                  break
78          return minScore
```

The following code snippet shows the function which loops through all legal moves and find the best move from them using minmax algorithm and alpha beta pruning. Initially alpha and beta both are negative infinity and positive infinity respectively. And since the Ai cannot make same move multiple times, an array is maintained that contains all moves by Ai. If the current move is in that array, loop is iterated and board is evaluated on next move. 4 shows the number of moves that Ai must consider to select its best move.

```
81  def moveByAI(board):
82      bestScore = -1000000 #represents neg infinty
83      bestMove = None
84      # compare every legal move
85      for move in board.legal_moves:
86          if move in aiMoves: continue
87          board.push(move)
88          score = minMax(board, 4, -1000000, 1000000, "minimizingPlayer")
89          board.pop()
90          if score > bestScore:
91              bestScore = score
92              bestMove = move
93      return bestMove
```

←THE END→