# ASSIGNMENT 3

Adversarial Search

**Name: Kamil Ilyas**

**Roll No: 20i-2371**

**Section: SE(P)**

## Report

This is a Python version of a simple chess game where a human player can compete against a computer opponent who chooses its moves using a minimax algorithm with alpha-beta pruning. The computer player utilizes a straightforward evaluation function to assess the state of the board as the game is being played on a regular chessboard.

Initializing the chess board and setting constants like the win score and the maximum depth of the minimax algorithm are the first steps of the game.

Then, the alpha-beta pruning minimax method is defined. This algorithm identifies the best move for the current player by iteratively evaluating each possible move for a specific depth. Each board state is given a score using the evaluation function.

After that, the evaluation function is defined. Each piece on the board is given a value based on its nature and location. If the piece belongs to the human player, the score is positive; if it belongs to the computer player, the score is negative. The function also looks for conditions of checkmate and stalemate.

The actions of the computer and human players are then handled by two functions that are defined. A move in UCI notation must be entered by the human player, and it is then verified as genuine. The computer player chooses its move and puts it to the board using the minimax algorithm.

The primary game loop is finally defined. Until the game is over, the human and machine players switch off. When one player is checkmated or a draw condition is satisfied, the game is ended. The loop keeps going until a winning or drawing condition is satisfied if the game hasn't ended. The winner or draw is revealed after the game.

So, let's break the code into the following parts first I applied the minimax algorithm's alpha-beta pruning variation to determine the optimal move for a player in a specific chess position.

The function accepts the following five inputs:

chess is the board.

Current position is represented by a board object.
depth: An integer that indicates how deep the search should be conducted.
Alpha is a float that represents the maximal player's best-known value.
beta is a float that represents the minimising player's best-known value.
The boolean value "maximizing player" indicates whether or not the current player is maximising.

```
def minimax_alpha_beta(board, depth, alpha, beta, maximizing_player):
    if depth == 0 or board.is_game_over():
        return evaluate_board(board), None

    if maximizing_player:
        max_eval = -float('inf')
        best_move = None

        for move in board.legal_moves:
            board.push(move)
            eval = minimax_alpha_beta(board, depth - 1, alpha, beta, False)[0]
            board.pop()
            if eval > max_eval:

                max_eval = eval
                best_move = move
            alpha = max(alpha, eval)
            if beta <= alpha:
                break

        return max_eval, best_move

    else:
        min_eval = float('inf')
        best_move = None

        for move in board.legal_moves:
            board.push(move)
            eval = minimax_alpha_beta(board, depth - 1, alpha, beta, True)[0]
            board.pop()
            if eval < min_eval:
                min_eval = eval
                best_move = move
            beta = min(beta, eval)
            if beta <= alpha:
                break

        return min_eval, best_move
```

Then by adding up the values of each piece on the board, this function determines the score of a given chessboard. If checkmate occurs, the result is a high score for white and a low score for black. A score of 0 is given if the match is deadlocked.

```
def evaluate_board(board):
    if board.is_checkmate():
        if board.turn == chess.WHITE:
            return -WIN_SCORE
        else:
            return WIN_SCORE
    elif board.is_stalemate():
        return 0

    score = 0
    for square in chess.SQUARES:
        piece = board.piece_at(square)
        if piece is None:
            continue

        if piece.color == chess.WHITE:
            score += piece_value(piece)
        else:
            score -= piece_value(piece)
```

```
return score
```

After that piece_value function is used and this function retrieves the point value of a piece object from the Python Chess module using the standard chess point values. A pawn has a point value of 1, a knight or bishop has three, a rook has five, a queen has nine, and a king has no points (since it cannot be captured). It returns zero if the item is not recognised.

```
def piece_value(piece):
    if piece.piece_type == chess.PAWN:
        return 1
    elif piece.piece_type == chess.KNIGHT:
        return 3
    elif piece.piece_type == chess.BISHOP:
        return 3
    elif piece.piece_type == chess.ROOK:
        return 5
    elif piece.piece_type == chess.QUEEN:
        return 9
    else:
        return 0
```

The UCI (Universal Chess Interface) format entry form is presented to the user via the function human move(), which then verifies the move and, if it is valid, pushes it to the game board. The user is prompted to enter a legitimate move until one is entered if the move is invalid. It also prompts the user to enter a valid input if the user input is not in the proper format.

```
def human_move():
    while True:
        try:
            move = input("Your move: ")
            move = chess.Move.from_uci(move)
            if move in board.legal_moves:
                board.push(move)
                print("You played:", move)
                break
            else:
                print("Invalid move. Please try again.")
        except:
            print("Invalid input. Please try again.")
```

Using the minimax method with alpha-beta pruning, this function chooses the optimal move out of all legitimate movements that can be made. It begins by setting the depth, alpha, and beta values to their beginning values. After that, it iteratively applies each of the moves that are permitted on the board and uses the minimax algorithm to determine the board's position. The optimal move for the computer to play is then determined by choosing the move with the highest score. The best move is then printed and pushed into the board.

```
def computer_move():
    depth = 3
    best_move = chess.Move.null()
    alpha = float('-inf')
    beta = float('inf')
    for move in board.legal_moves:
        board.push(move)
        score = minimax(depth - 1, alpha, beta, False)
        board.pop()
        if score > alpha:
            alpha = score
            best_move = move
    board.push(best_move)
    print("Computer played:", best_move)
```

Then function for checking if the game is over, draw and is it checkmate is used. After that play game function is used which is use to run the game and then the game is started.

```
def play_game():
    print("Welcome to Chess!")
    print(board)

    while not game_over():
        human_move()
        print(board)

        if is_checkmate():
            print("Checkmate! You win!")
            return

        if is_draw():
            print("Draw!")
            return

        computer_move()
        print(board)

        if is_checkmate():
            print("Checkmate! Computer wins!")
            return

        if is_draw():
            print("Draw!")
            return

board = chess.Board()
play_game()
```