CS 4200 - Project 3

Noah Reef

Winter 2023

1 Code Outline

For the outline of the program, I implemented a few auxiliary functions such as: `create_board` which creates the 8×8 `numpy array` that represents the board, `print_board` which prints out the current board, `x_moves` which returns the moves of the X player on the board, `o_moves` which returns the moves of the O player on the board, `valid_move` which checks if a given move is a "valid" move, and `game_over` which checks to see if a board state is a terminal board state.

The `heuristic` function takes in a given board state and computes the heuristic using the following formula:

$$h = 100X_3 + 10X_2 + X_1 - 100O_3 - 10O_2 - O_1$$

where,

$$\begin{aligned}
X_3 &: \text{The number of X's that are 3 in a row} \\
X_2 &: \text{The number of X's that are 2 in a row} \\
X_1 &: \text{The number of X's that are 1 in a row} \\
O_3 &: \text{The number of O's that are 3 in a row} \\
O_2 &: \text{The number of O's that are 2 in a row} \\
O_1 &: \text{The number of O's that are 1 in a row}
\end{aligned}$$

for the `minimax` function I implemented the $\alpha - \beta$ pruning method with a limit depth search of $d = 2$. It's a recursive implementation that passes along the current depth, current board, α, β, and whether it is the maximizing players turn to each recursive call for each successive child node.

```python
def minimax(board, depth, alpha, beta, maximizing_player):

    '''
    Summary: Use the minimax algorithm to find the best move for a player
    Input: 8x8 numpy array, integer, integer, integer, boolean
    Output: Tuple
    '''

    # Check if the game is over or the maximum depth has been reached
    if game_over(board) or depth == 2:
        return (heuristic(board), None)

    else:
        # If the player is maximizing
        if maximizing_player:
            max_eval = float("-inf")
            best_move = None

            # Iterate through all possible moves
            for move in possible_moves(board):
                board[move[0]][move[1]] = 1
                eval = minimax(board, depth + 1, alpha, beta, False)[0]
                board[move[0]][move[1]] = -1
                if eval > max_eval:
                    max_eval = eval
                    best_move = move
                alpha = max(alpha, eval)
                if beta <= alpha:
                    break

            return (max_eval, best_move)

        # If the player is minimizing
        else:
            min_eval = float("inf")
            best_move = None

            # Iterate through all possible moves
            for move in possible_moves(board):
                board[move[0]][move[1]] = 0
                eval = minimax(board, depth + 1, alpha, beta, True)[0]
                board[move[0]][move[1]] = -1
                if eval < min_eval:
                    min_eval = eval
                    best_move = move
                beta = min(beta, eval)
                if beta <= alpha:
                    break

            return (min_eval, best_move)
```

Figure 1: Code sample of `minimax` implementation

the children of the current root node is generated by the `possible_moves` function. As we pass our function calls back up the stack each recursive call of `minimax` is responsible for updating the α and β parameters when necessary.

2 Output

```
Computer move: g4 in 0.29433131217956543 seconds
  1 2 3 4 5 6 7 8
A - - - - - - - -
B - X - - - - - -
C - - - - - - - -
D - - - - - - - -
E - - - - - - - -
F - - - - - - - -
G X X X X - - - -
H O O X O O O X O

X wins
Move history:
1. O: b2
2. X: g1
3. O: h3
4. X: g2
5. O: g3
6. X: h7
7. O: g4
(base) nreef@Noahs-MacBook-Air-2 CS-4200-Artificial-Intelligence %
```

Figure 2: Example game of Computer Winning

```
Enter your move: e5
  1 2 3 4 5 6 7 8
A - - - - - - - -
B - X - - - - - -
C - - - X X - - -
D - - - X O - - O
E - - - - O - X X
F - - - - O X O O
G - - - - O - - -
H - - - - - - - -

O wins
Move history:
1. O: b2
2. X: c5
3. O: d4
4. X: c4
5. O: e7
6. X: e8
7. O: f6
(base) nreef@Noahs-MacBook-Air-2 CS-4200-Artificial-Intelligence %
```

Figure 3: Example Game of Human Winning