In the name of Allah

Othello Game Implementation Documentation

Overview of the Game Rules

Othello, also known as Reversi, is a two-player board game played on an 8x8 grid. The game starts with four discs (pieces) placed in the center of the board in a specific pattern. Players take turns placing their discs on the board with the goal of capturing their opponent's discs. A player captures discs by trapping one or more of the opponent's discs between two of their own discs, either horizontally, vertically, or diagonally. The game ends when neither player can make a valid move, and the player with the most discs on the board wins.

Game Rules:

1. The game is played on an 8x8 board.
2. Players take turns placing their discs on the board.
3. A move is valid if it captures at least one of the opponent's discs.
4. Discs are captured when they are trapped between two of the current player's discs.
5. The game ends when neither player can make a valid move.
6. The player with the most discs on the board at the end of the game wins.

Code Implementation

The code provided is an implementation of the Othello game using Python and **Pygame library** for the graphical user interface. The main components of the implementation are as follows:

1. **Initialization**:
    - The game board is initialized with a starting configuration using the *init_board* function. This sets up the initial four discs in the center of the board.
2. **Game Loop**:
    - The *main_menu* function displays the difficulty selection menu and waits for user input to start the game.
    - The *play_game* function contains the main game loop, alternating between the human player and the AI until the game is over.
3. **Drawing the Board**:

    - The *draw_board* function handles rendering the game board and pieces using Pygame.
    - The *draw_menu* and *draw_game_over_menu* functions display the main menu and game over menu, respectively.
4. **Game Logic**:
    - The *is_valid_move*, *check_direction*, and *get_valid_moves* functions determine the validity of moves.
    - The *apply_move* function updates the board state by placing a disc and flipping the opponent's discs.
    - The *is_game_over* and *get_winner* functions determine if the game is over and who the winner is.
5. **AI Implementation**:
    - The AI uses the Minimax algorithm with alpha-beta pruning to decide its moves. This is implemented in the *minimax* function.

Explanation of the Minimax Algorithm

The Minimax algorithm is a decision-making algorithm used in two-player and zero-sum games to determine the optimal move for a player. It operates by simulating all possible moves and their outcomes, assuming that both players play optimally. The goal is to maximize the player's advantage while minimizing the opponent's advantage.

Minimax Algorithm:

1. **Maximizing Player**: The algorithm maximizes the score for the current player.
2. **Minimizing Player**: The algorithm minimizes the score for the opponent.
3. **Recursion**: The algorithm recursively evaluates all possible moves to a specified depth.
4. **Evaluation Function**: The algorithm uses an evaluation function to score the board state.

Implementation in the Code:

- The minimax function takes the current board state, depth, alpha, beta, and the player as inputs (difficulty level as depth).
- It generates a list of valid moves for the current player using the *get_valid_moves* function.
- If the depth is 0 or there are no valid moves, the function returns the evaluated score of the board.

- For each valid move, the function applies the move to a copy of the board and recursively calls itself with the updated board and switched player.
- Alpha-beta pruning is used to eliminate (prune) branches of the search tree that do not need to be explored, improving efficiency.

Alpha-Beta Pruning:

- Alpha represents the maximum score that the maximizing player is assured of.
- Beta represents the minimum score that the minimizing player is assured of.
- If the current node's beta score is less than or equal with the current node's alpha score ($\alpha \geq \beta$), the branch is pruned.

```
for move in valid_moves:
    ...
    if beta <= alpha:
        break
```

The *minimax* function recursively evaluates the board states, switching between maximizing and minimizing players, and uses alpha-beta pruning to optimize the search process.

Conclusion

The provided code implements the Othello game using Python and Pygame library, with an AI opponent that uses the Minimax algorithm with alpha-beta pruning to make optimal moves. The documentation covers the game rules, code implementation details, and an explanation of the Minimax algorithm and its optimization through alpha-beta pruning. This should provide a comprehensive understanding of how the game works and how the AI makes decisions.

Good luck! ☺

Seyed Mahdi Mahdavi Mortazavi - 40030490