- Team Members:

1- Ahmeed Mohammed Ahmed Khattab

2- Mohammed Tamer Atef

3- Omar Adel Mohammed

Othello (Reversi) Pygame Project Documentation

1. Overview

This project is a fully interactive Othello/Reversi game using **Pygame**. It allows for:

- Human vs Human gameplay
- Human vs AI gameplay
- Dynamic scoring and winner display

The AI uses **Minimax with Alpha-Beta pruning** and positional heuristics.

2. File Structure

- **Main Game Loop (OthelloGame)**: Handles Pygame rendering, user input, AI turns, and game state.
- **Game Logic (OthelloLogic)**: Contains board setup, move validation, flipping logic, scoring, and game-over checks.
- **AI (OthelloAI)**: Implements Minimax algorithm with Alpha-Beta pruning, positional weighting, and mobility heuristics.

3. Constants

- **BOARD_SIZE**: 8 (standard Othello board)
- **CELL_SIZE**: 60 pixels
- **BOARD_OFFSET**: X/Y margin for rendering
- **COLORS**: Background, grid, pieces, valid move indicators, text
- **Players**: BLACK, WHITE, EMPTY

4. Core Game Logic (OthelloLogic)

Key Methods:

1. **initialize_board():** Sets up the 4 central starting pieces.
2. **get_opponent(player):** Returns the opposite player.
3. **is_valid_move(row, col, player):** Checks if a move is legal.
4. **make_move(row, col, player):** Applies move and flips captured discs.

5. **get_valid_moves(player):** Returns all valid moves.
6. **get_score():** Returns current piece counts.
7. **is_game_over():** Returns True if neither player has valid moves.
8. **get_winner():** Returns the winner or None for a tie.

Move Validation:

- Checks all 8 directions.
- Flips are only valid if opponent pieces are flanked by player's piece.

5. AI Player (OthelloAI)

Features:

- **Minimax Algorithm** with **Alpha-Beta pruning**
- **Heuristic Evaluation**:
    - Piece count differential
    - Positional weights (corners and edges are highly valued)
    - Mobility (number of valid moves)
- Depth configurable for difficulty.

Key Methods:

1. **evaluate_board(logic):** Evaluates board using combined heuristics.
2. **minimax(logic, depth, alpha, beta, maximizing):** Standard minimax with alpha-beta pruning.
3. **get_best_move(logic):** Returns optimal move for AI.

6. Game Rendering (OthelloGame)

- **Board Drawing**: Grid, pieces, valid move indicators.
- **UI Elements**: Score, messages (whose turn, AI thinking, game over).
- **Input Handling**: Mouse clicks for human moves, AI auto-moves, reset key R.
- **Game Loop**:
    1. Process events (clicks, quit, keypresses)
    2. Execute AI if needed
    3. Draw board and UI
    4. Update display and tick clock

7. Game Flow

1. Initialize board and AI.
2. Black starts.

3. Human clicks to make a move.
4. Check game state; if AI enabled and it's AI turn, AI chooses move.
5. Update board, flip pieces, switch player.
6. Repeat until game over.
7. Display final score and winner.

8. AI Difficulty

- Configurable depth (default 4) controls thinking depth and strength.
- Depth 1: very easy, fast
- Depth 6: strong, slower

9. Notes

- **Consecutive Passes**: Tracks passes when a player has no valid moves; game ends after 2 consecutive passes.
- **Heuristics**: Prioritizes corners and edges.
- **Extensibility**: Can easily enable 2-player mode (ai_enabled=False).

10. Controls

- Mouse click: Place piece (if valid).
- R key: Reset game.
- Close window: Quit game.

11. Libraries

- pygame: Rendering and event handling
- copy: Deep copying logic for AI
- sys: Exit handling
- typing: Type hints

12. Summary

This project demonstrates:

- Real-time board game with GUI using Pygame
- Human vs AI gameplay with strong AI logic
- Minimax with Alpha-Beta pruning for decision making
- Complete scoring, winner determination, and UI feedback