SMAI Assignment 3: Othello Bot Implementation

Submitted By:

- Vandita Lodha
- Suhani Jain

Objective:
The goal of this assignment was to design and implement a bot that plays the game of Othello efficiently. Given a board configuration and a turn, the bot should return a valid move within a strict time limit of 2 seconds per turn. The player with the highest number of coins at the end of the game is declared the winner.

---

1. Problem Understanding

Othello, also known as Reversi, is a strategy board game played on an 8x8 grid. Players take turns placing coins on the board such that at least one line of the opponent's coins is "sandwiched" between two of the player's coins. The objective is to maximize the number of your coins on the board by the end of the game.

Key Rules:

- A valid move must capture at least one opponent coin.
- If no valid move is available, the player forfeits their turn.
- The game ends when neither player has a valid move.

We referred to:

- How to Play Othello
- Wikipedia: Reversi/Othello

PROF

2. Bot Implementation Details

Framework Setup

The bot was developed using the provided "Desdemona" framework, which simulates the game environment. The `MyBot.cpp` file was modified to implement the decision-making logic for the bot. Other files and the framework were left untouched as per the assignment requirements.

---

3. Key Functions and Logic

a. `canMove`

- **Purpose:** Checks if a sequence of opponent coins can be flipped in a specific direction when placing a coin.

- **Logic:**
    - A move is valid in a direction if it starts with the opponent's coin(s) and ends with the player's coin.
    - Returns `true` if a valid sequence exists, `false` otherwise.

b. `isLegalMove`

- **Purpose:** Validates if placing a coin at a specific position is a legal move.
- **Logic:**
    - Uses `canMove` to evaluate all eight possible directions (horizontal, vertical, and diagonal).
    - Returns `true` if any direction is valid, `false` otherwise.

c. `numValidMoves`

- **Purpose:** Counts the total number of valid moves available for the current player.
- **Logic:**
    - Iterates through all cells on the board and calls `isLegalMove` to check validity.

d. `othelloBoardEvaluator`

- **Purpose:** Heuristically evaluates the current board state and assigns a score based on strategic metrics.
- **Metrics Used:**
    - **Piece Difference (`p`):** Compares the number of coins between the player and the opponent.
    - **Frontier Tiles (`f`):** Evaluates the number of unstable tiles adjacent to empty spaces.
    - **Corner Occupancy (`c`):** Rewards control of corners as they are stable positions.
    - **Corner Closeness (`l`):** Penalizes positions near unoccupied corners.
    - **Mobility (`m`):** Measures the number of valid moves available.
    - **Positional Weights (`d`):** Uses a predefined heuristic matrix to prioritize strategic positions on the board.

PROF

e. `testMyMove`

- **Purpose:** Implements the **Minimax algorithm with Alpha-Beta Pruning** to evaluate the outcome of a move.
- **Logic:**
    - Recursively simulates possible moves up to a depth of 6.
    - Alpha-beta pruning is applied to reduce unnecessary evaluations and improve efficiency.

f. `play`

- **Purpose:** The core decision-making function called by the game engine.
- **Logic:**
    - Retrieves valid moves using `getValidMoves`.
    - Sorts moves using the `compare` function, which uses `othelloBoardEvaluator`.
    - Applies the minimax algorithm to choose the optimal move based on the evaluation score.

4. Strategies Used

1. **Heuristic Evaluation:**
   The `othelloBoardEvaluator` function integrates multiple metrics, including mobility, corner control, and frontier stability, to guide the bot's decisions.

2. **Alpha-Beta Pruning:**
   This optimization to the minimax algorithm reduces the search space, allowing the bot to explore deeper move trees within the time constraints.

3. **Sorting Moves:**
   Valid moves are sorted based on their heuristic scores before being evaluated in depth, improving pruning efficiency.

5. Testing and Results

- **Testing:**
  The bot was tested against the provided `RandomBot` using the command:

  ```
  ./bin/Desdemona ./bots/MyBot.so ./bots/RandomBot.so
  ```

  - Game logs were analyzed to ensure correctness and optimal decision-making.
  - We kept modifying out bot and testing it with its older version to see if there has been an improvement.

- **Performance:**

  - The current bot was the best of all of our iterations.
  - Time usage was kept under the 2-second limit by efficient pruning and early exits in the minimax function.

PROF

6. Challenges and Improvements

Challenges

- **Time Constraints:** Ensuring the bot processes all possible moves within the 2-second limit.
- **Complexity:** Balancing depth of search with the evaluation of numerous game states.

Possible Improvements

- **Dynamic Depth Adjustment:** Adapting search depth based on the stage of the game.
- **Endgame Solver:** Implementing specialized logic for the late game when fewer moves are available.

References

1. Search Methods in Artificial Intelligence, Deepak Khemani
2. Minimax Algorithm with Alpha-Beta Pruning
3. Alpha-Beta vs Scout Algorithms for the Othello Game
4. Othello Game AI