Connect Four Bot Analysis

3 Bytess

November 22, 2025

Medium Bot Logic and Time Complexity Analysis

Overview

The Medium Bot in this Connect Four implementation is designed to play more intelligently than the Easy Bot. Its strategy consists of four prioritized steps:

1. Attempt to win immediately if a winning move is available.
2. Block the opponent's immediate winning move.
3. Choose a strategic column, avoiding moves that give the opponent an immediate win.
4. If none of the above, select a random valid column as a fallback.

Each step relies on helper functions to evaluate the board and make decisions.

Key Functions and Their Roles

1. `addValue(board, col, player)`

This function attempts to place a piece for `player` in the specified column.

- Returns the row index of placement, or -1 if the column is full.
- Complexity: $O(\text{rows})$ because it may need to iterate from the bottom row to the top to find the first empty cell.
- Used in all stages of the Medium Bot strategy to simulate moves.

2. `checkDirectionFrom(board, player, row, col, dRow, dCol)`

Checks for four consecutive pieces of the same player in a specific direction starting from the last placed piece.

- Only sequences including the last move are checked.
- Complexity: $O(1)$ because the maximum sequence length is 4, independent of the board size.
- Used by `checkWin` to determine if a move leads to victory.

3. `checkWin(board, player, lastRow, lastCol)`

Checks for a win by calling `checkDirectionFrom` in four directions: horizontal, vertical, diagonal down-right, and diagonal down-left.

- Complexity: $O(1)$ per call.
- Localized checking avoids scanning the entire board.

4. `wouldGiveOpponentWin(col)`

Evaluates whether placing a bot piece in a column gives the opponent an immediate winning opportunity.

- Temporarily places a bot piece in the column.
- Simulates the opponent placing a piece in every column and checks if it results in a win.
- Undoes all temporary moves.
- Complexity: $O(\text{cols} \times \text{rows})$.

Medium Bot Strategy Steps and Complexity

Step 1: Immediate Win

- Tries each column, places a bot piece, and checks if it leads to a win.
- Complexity: $O(\text{cols} \times \text{rows})$.

Step 2: Block Opponent Win

- Simulates placing an opponent piece in each column and checks if it results in a win.
- Complexity: $O(\text{cols} \times \text{rows})$.

Step 3: Strategic Move

- Iterates through a center-preferred column order: $\{3, 2, 4, 1, 5, 0, 6\}$.
- Temporarily places a bot piece in each column and calls `wouldGiveOpponentWin`.
- Complexity: $O(\text{cols} \times (\text{cols} \times \text{rows})) = O(\text{cols}^2 \times \text{rows})$.

Step 4: Fallback Random Move

- Randomly selects a valid column if no other strategy applies.
- Complexity: $O(\text{rows})$ in the worst case.

Total Complexity per Bot Move

$$O(\text{cols}^2 \times \text{rows})$$

For the standard board (6 rows $\times$ 7 columns):

$$O(7^2 \cdot 6) = O(294)$$

Hard Bot Logic and Time Complexity Analysis

Overview

The Hard Bot uses a Minimax algorithm with Alpha-Beta pruning and Transposition Tables to evaluate the best move up to a search depth of $d = 11$. Its decision process consists of:

1. Check for an immediate winning move.
2. Block opponent's immediate win.
3. Use Minimax with Alpha-Beta pruning to select the optimal move.
4. Fallback strategies (center-preferred or random column) if Minimax fails.

Time Complexity with Alpha-Beta Pruning

- Branching factor: $b \leq 7$ (number of valid columns).
- Search depth: $d = 11$.
- Without pruning: $O(b^d)$.
- With Alpha-Beta pruning and optimal move ordering, the number of nodes explored in the best case is reduced to approximately $O(b^{d/2})$.
- Each node evaluation involves:
  - Placing a piece: $O(\text{rows})$
  - Evaluating the board (heuristic scoring): $O(\text{rows} \cdot \text{cols})$
  - Undoing a move: $O(1)$
- Therefore, the **effective worst-case time complexity with Alpha-Beta pruning** is:

$$O(b^{d/2} \cdot (\text{rows} \cdot \text{cols}))$$

For the standard board (6 rows $\times$ 7 columns) and $b = 7$, $d = 11$:

$$O(7^{5.5} \cdot 42) \approx O(16807 \cdot 42) = O(705,894)$$

Conclusion

The Hard Bot efficiently searches the game tree using Alpha-Beta pruning, reducing the effective complexity from $O(b^d)$ to $O(b^{d/2})$.