

CS-747 Assignment Report

Tanish Sahu - 23B1816

Task 1 Agent:

This agent uses a **Depth-2 Minimax** search to select its move.

1. How it Works:

- **Depth 1 (My Move):** The agent looks at a random sample of its own legal moves (about 40-50%) to save time.
- **Depth 2 (Opponent's Reply):** For each of its moves, it then simulates *all* of the opponent's possible replies.
- **Logic:** It assumes the opponent will always pick the reply that is worst for the agent (`min(vals)`). It then chooses its own move that leads to the "best-of-the-worst" outcome.

2. Evaluation & Heuristics:

- **Purely Material:** The agent only cares about the material count. It uses standard piece values (Pawn=1, Knight=3, Bishop=3, Rook=5, Queen=9, King=1000).
- **Blunder Pruning:** It immediately ignores any of its own moves that lose material (score ≤ -1) without searching any deeper.

3. Key Properties:

The agent is designed to be robust. By searching to depth 2, it avoids simple one-move traps (like capturing a pawn that allows its queen to be captured) that a depth-1 agent would fall for.

Task 2 Agent:

This agent uses a **Depth-4 Minimax** search, guided by a very aggressive "pawn-first" heuristic.

1. How it Works:

- **Heuristic 1 (Pawn Capture First):** Before starting its search, the agent checks if its single pawn has any legal 1-square diagonal (capture) moves. If it finds one, it **immediately plays that move** and skips the entire search.
- **Heuristic 2 (Minimax Search):** If no immediate pawn capture is found, the agent initiates a 4-ply (Depth-4) Minimax search:
 - **Depth 1 (My Move):** Iterates through all its moves.

- **Depth 2 (Opponent's Reply):** Finds the opponent's best reply (the `min` score).
- **Depth 3 (My Reply):** Finds its own best reply (the `max` score).
- **Depth 4 (Opponent's Reply):** Finds the opponent's final best reply (the `min` score) by evaluating the board at this depth.
- **Logic:** The agent assumes the opponent will play perfectly at each step (`min(vals)`). It then chooses its own move that leads to the best possible outcome after this 4-ply exchange.

2. Evaluation & Heuristics:

- **Evaluation (Flawed):** The agent's `eval_board` function has two parts:
 1. **Material Score:** It counts material, but the loop `range(4)` means it **only counts Pawns, Knights, Bishops, and Rooks**. It completely ignores the Queen.
 2. **Pawn Position Score:** It adds a small bonus (`+0.0` to `+0.4`) for pushing its single pawn toward the center/opponent's side.
- **Pruning:** The agent uses custom pruning logic at Depths 2 and 3 to skip analyzing branches that already look worse than a previously found move. It also has a "good enough" cutoff (`if best_val > 15: break`) to stop searching early if it finds a winning move (like capturing a queen, though its eval can't see it).

3. Key Properties:

This agent is much deeper than Task 1, but its strategy is dominated by the **"Pawn Capture First"** heuristic, making it extremely greedy and prone to falling into traps. Its evaluation function is also flawed as it is blind to the most valuable piece on the board (the Queen).

Task 3 Agent

This agent is a hard-coded, **5-ply (Depth-5) Minimax search** designed for high performance. Its core design principle is to go deep in the search tree by avoiding slow board copies and instead using a fast `make_move/unmake_move` pairing on a single board object.

1. How it Works:

- **Algorithm (Minimax):** The `move` function is a manually unrolled 5-ply search. It consists of 5 nested `for` loops that iterate through legal moves, simulating a "game" 5 moves deep.
 - **Depth 1 (My Move):** `max` (finds the best `val`)
 - **Depth 2 (Opponent's Reply):** `min` (finds the worst `val` for me)
 - **Depth 3 (My Reply):** `max`
 - **Depth 4 (Opponent's Reply):** `min`

- **Depth 5 (My Reply):** `max` (evaluates all final board states)
- **Performance (Fast):** Instead of using `board.copy()` at each step, the agent saves the board's "undo" information (`piece_at`, `captured_piece`, `old_ply_count`), calls `board.make_move()`, gets the score from the next level, and then immediately restores the board with `self.unmake_move()`. This is extremely fast.
- **Logic Flaw (Critical Bug):** The agent has a bug where if it finds an immediate **checkmate at Depth 1**, it calls `return` before calling `self.unmake_move()`. This wins the game but corrupts the main `chess_obj` board, which will cause errors in a tournament setting.

2. Evaluation & Heuristics:

The `eval_board` function is complex and has two modes:

1. **Standard Mode (Material):**
 - **Flaw:** The agent's `piece_values` array only has 5 elements, and the evaluation loop `range(5)` **completely ignores the King (piece 5)**. The agent calculates the king's material value as 0.
2. **"King-Hunter" Mode (Heuristic):**
 - This logic activates only when the agent is in a winning position (`material_score >= 8`). Its goal is to convert the material advantage into a checkmate and avoid "repeating move" draws.
 - **Pawn Bonus:** A small bonus is given for advancing its single pawn.
 - **King Mobility Penalty:** A bonus is given for restricting the opponent's king (fewer legal moves).
 - **King-to-Edge Bonus:** A bonus is given for forcing the opponent's king away from the center, as checkmates are easier on the edge of the board.
 - **Flaw:** The calculation for the board's "center" is incorrect (swaps row/column dimensions and uses 1-based indexing math).

3. Key Properties:

This is a powerful agent that searches 5 moves deep, making it strategically far superior to the Task 1 and 2 agents. Its "King-Hunter" logic shows it is designed to solve complex endgame problems. However, it is held back by several critical bugs, including a flawed evaluation (ignores the king) and a search bug (forgets to `unmake_move` on an immediate win).