भारतीय सूचना प्रौद्योगिकी संस्थान गुवाहाटी
Indian Institute of Information Technology Guwahati

CS236 - Artificial Intelligence Lab
Assignment - 11

Problem Statement

In this assignment, you are given a file named chess_engine.py and a folder containing chess piece images. The provided code implements a graphical chess game using Pygame and the python-chess library. Currently, the AI makes random moves. Your task is to modify the AI so that it chooses moves using the **alpha-beta pruning** algorithm along with a **heuristic evaluation** function. The modified program must:

1. Allow the human player to choose a color (White or Black) at the start.
2. Display the chess board and pieces graphically using Pygame.
3. Replace the current random move selection with an AI that:
    i. Uses alpha-beta pruning to search the game tree up to a specified depth.
    ii. Uses a heuristic evaluation function to assign scores to board states. The evaluation should at a minimum consider material balance (i.e., assign weights to pawns, knights, bishops, rooks, queens, and kings) and may include basic positional considerations.
4. Alternate moves between the human and the AI until the game reaches a terminal state (checkmate, stalemate, or draw).

Algorithm Specifications

1. **Board and Game Representation:**
    i. The provided code uses the python-chess library for board representation and legal move generation.
    ii. The graphical interface is implemented using Pygame, with images provided in the images folder.
2. **Alpha-Beta Pruning:**
    i. Modify the AI move selection to implement the minimax algorithm enhanced with alpha-beta pruning.
    ii. The search should explore possible moves up to a predefined depth (cut-off level). You may choose a reasonable depth based on performance constraints.
3. **Heuristic Evaluation Function:**

  i. Develop a heuristic function that returns a numerical score for a given board state.
  ii. The evaluation should at a minimum consider the material balance (e.g., assign weights for each type of piece). Additional factors, such as basic piece positioning, may be included to improve performance.

4. **Function Specification:**

   i. Implement a function `get_best_move(board, depth, alpha, beta)` that uses the alpha-beta pruning algorithm to determine the best move for the AI.

   ii. Ensure that the function correctly handles the switching of maximizing and minimizing roles depending on the current turn.

   iii. You may assume that the board is not in a terminal state when this function is called.

5. **Game Flow:**

   i. On the human turn, accept moves via mouse clicks as implemented in the provided code.

   ii. On the AI turn, call your `get_best_move` function to compute and execute the move.

   iii. Update the display after each move and continue until the game ends.

6. **Notes:**

   i. You are allowed to modify only the AI move generation part of the provided code.

   ii. Avoid rewriting the entire game engine; focus on integrating the alpha-beta pruning and heuristic evaluation into the AI.

   iii. There may be multiple valid implementations; any correct and efficient approach will be accepted.

Documentation and Move Representation

The provided game engine is built on top of the `python-chess` library. For a comprehensive list of available functions and classes, please refer to the official documentation:

`https://python-chess.readthedocs.io/en/latest/`

Some key components include:

1. `chess.Board()` – Represents the current state of the chess game. It can generate legal moves, detect checkmate, stalemate, etc.

2. `chess.Move` – Represents a move in the game. Moves are typically created using UCI notation.

3. `board.legal_moves` – Returns a generator of all legal moves in the current board state.

4. `board.push(move)` – Executes a given move on the board.

Move Representation: Moves in this system use Universal Chess Interface (UCI) notation. In this format:

1. The board is labeled with files from `a` to `h` and ranks from `1` to `8`.
2. A move is represented as a 4-character string, e.g., `h2h3`:
   i. The first two characters, `h2`, represent the starting square.
   ii. The last two characters, `h3`, represent the destination square.
3. For example, `e2e4` means the piece on `e2` moves to `e4`.

Universal Chess Interface (UCI) Documentation

The Universal Chess Interface (UCI) is a standardized protocol used for communication between chess engines and graphical user interfaces (GUIs). In our assignment, moves are represented in UCI notation. Here are some key points:

1. **Move Notation:** A move in UCI notation is expressed as a 4- or 5-character string:
   i. The first two characters indicate the starting square (file and rank). For example, `e2` refers to the square at file 'e' and rank '2'.
   ii. The next two characters indicate the destination square. For example, `e4` indicates that the piece moves to file 'e' and rank '4'.
   iii. For pawn promotion, a fifth character is appended to indicate the promotion piece (e.g., `q` for queen, `r` for rook, `b` for bishop, `n` for knight). For example, `e7e8q` means a pawn moves from `e7` to `e8` and promotes to a queen.
2. **Example:** `h2h3` means that the piece on square `h2` is moved to square `h3`.
3. **Additional UCI Commands:** While this assignment focuses on move representation, the UCI protocol also defines commands for engine initialization, option settings, and status updates. For a complete list of commands and their specifications, please refer to the official UCI documentation:

   `http://wbec-ridderkerk.nl/html/UCIProtocol.html`

4. **Usage in `python-chess`:** The provided game engine uses the `python-chess` library, which handles UCI-formatted moves internally. You can use `chess.Move.from_uci(move_str)` to create a move object from a UCI string and `move.uci()` to convert a move object back to its string representation.