Semester Project

Alpha Beta Pruning in Python Chess Trees

This semester's project challenges you to apply your Data Structures course knowledge by implementing an advanced Alpha Beta pruning algorithm. The project encourages your creative problem-solving and expects you to independently acquire relevant missing information.

Please carefully read this document and promptly begin work on your project. You are expected to work in pairs of two. Select your partner by Friday 11:59 PM; after this deadline, partners will be assigned at random automatically.

Introduction

Alpha Beta Pruning is an optimization technique for the minimax algorithm. The algorithm reduces the number of nodes evaluated in a search tree by the minimax algorithm without affecting the final result. Alpha Beta pruning is especially effective in games like Chess, where the game tree can grow very large. The primary goal is to minimize the computation time, making the algorithm efficient in deciding the best move for a given chess position.

Examples of Alpha Beta Pruning Applications

- *Chess Engines:* Used to evaluate potential moves and decide the best strategy.
- *Video Games:* Used to improve the performance of AI opponents in strategic games.
- *Decision Making:* Used in software that requires optimization under constraints.

The practical applications of Alpha Beta pruning extend beyond just theoretical computer science and have been instrumental in the development of sophisticated artificial intelligence systems for competitive and strategic environments.

Alpha Beta pruning relies on two parameters: *Alpha* (the best already explored option along the path to the root for the maximizer) and *Beta* (the best already explored option along the path to the root for the minimizer). If the algorithm finds that a position is worse than previously examined positions, it will stop evaluating the position and move on, hence "pruning" the tree.

Input and Output

Your system is provided with chess board states as input in the Forsyth-Edwards Notation (FEN) format, a standard notation used to describe a particular board position of a chess game. The system is expected to analyze these inputs to determine the best next move using the Alpha Beta pruning algorithm. Additionally, you are encouraged to visualize the decision-making process of the algorithm by plotting potential moves, illustrating the pruning process, visualizing evaluated board states, or showing how the algorithm navigates through the game tree.

Input

Forsyth-Edwards Notation (FEN) is a compact way to describe a chess game position, capturing the placement of pieces on the board, the player to move, castling availability, en passant targets, halfmove clock, and fullmove number. This notation allows for a comprehensive description of any game state using a single line of text.

Pawns: ♙ p and P
Knights: ♘ n and N
Bishops: ♗ b and B
Rooks: ♖ r and R
Queens: ♕ q and Q
Kings:♔ k and K

Empty Squares: The number 1 through 8 represents the number of consecutive empty squares on the rank.

Special Symbols:
/: Indicates the end of a rank.
w or b: Indicates player turns.
KQkq: Castling availability.
-: Pawn has not moved two squares forward
0 1: halfmove clock and fullmove number

The following configuration shows the starting position of a chess game, with White moving first:

rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

The chess board is described from the 8th row down to the 1st row, with each row separated by a slash (/). Each piece is represented by a letter, with uppercase for White and lowercase for Black (e.g., 'K' for White King, 'q' for Black Queen). Empty squares are denoted by numbers from 1 to 8, representing the number of consecutive empty squares along a row. Active color is indicated by 'w' for White or 'b' for Black, signifying which player's turn it is. Castling availability is shown with a combination of 'K' (White can castle kingside), 'Q' (White can castle queenside), 'k' (Black can castle kingside), and 'q' (Black can castle queenside), or '-' if neither side can castle. The en passant target square is noted if there is a pawn in position to execute an en passant capture. Halfmove clock and fullmove number track the number of halfmoves since the last capture or pawn advance and the number of full moves in the game, respectively.

Output

Your system should output the best move for a given board state, determined by the Alpha Beta pruning algorithm. The output should include descriptive analytics that shed light on the decision-making process, such as the number of pruned nodes, depth of search, and evaluation scores of different moves:

1. **Best Move:** The primary output is the best move identified for the given board state. This move is the outcome of the Alpha Beta pruning algorithm's analysis, considering both the current board configuration and potential future moves. The move should be presented in Standard Algebraic Notation (SAN) and Universal Chess Interface (UCI).

2. **Descriptive Analytics (Optional):** Your system can provide a comprehensive analysis that offers insights into the algorithm's decision-making process. Such analysis includes:
    a. **Number of Pruned Nodes:** A count of the total nodes (potential moves and subsequent responses) that were not evaluated because the Alpha Beta pruning algorithm determined they could not affect the final decision.
    b. **Depth of Search:** The maximum depth reached during the search process before deciding on the best move. This depth indicates how many moves ahead the algorithm evaluated before making a decision.
    c. **Evaluation Scores of Different Moves:** For the top candidate moves considered during the algorithm's search, provide their evaluation scores. These scores reflect the perceived advantage or disadvantage resulting from a move.
    d. **Search Efficiency:** Information on how the Alpha Beta pruning impacted the search efficiency, possibly including metrics like the total number of nodes visited versus the number of pruned nodes and the time taken for the search.

3. **Visual Representations (Optional but Recommended):** Visual outputs significantly enhance understanding and engagement to complement the textual analysis. Possible visualizations include:
    a. **Game Tree Visualization:** Before and after the pruning process, to visually depict the search space reduction.
    b. **Heatmaps:** On the chess board, indicating the areas of focus or key pieces involved in the decision-making process.
    c. **Graphs:** Showing the evaluation score trends across different depths or branches of the game tree, illustrating how the decision was refined over the course of the search.

Please Notice:
Your system will be tested with a variety of starting chess positions in Forsyth-Edwards Notation. It is your responsibility to correctly represent these FEN inputs and efficiently implement the Alpha Beta pruning algorithm to determine the best possible move.

Application Programming Interface (API)

The Application Programming Interface (API) is the minimal contract you need to implement to enable communication between your application and the outside world. You are required to implement the following members in your application:

```python
class AlphaBetaChessTree:
    def __init__(self, fen):
        pass

    @staticmethod
    def get_supported_evaluations():
        pass

    def _apply_move(self, move, node, notation="SAN"):
        pass

    def _get_legal_moves(self, node, notation="SAN"):
        pass

    def get_best_next_move(self, node, depth, notation="SAN"):
        pass

    def _alpha_beta(self, node, depth, alpha, beta, maximizing_player):
        pass

    def _evaluate_position(self, node, depth):
        pass

    def _evaluate_board(self, board):
        pass

    def get_board_visualization(self, board):
        pass

    def visualize_decision_process(self, depth, move, notation="SAN"):
        pass

    def export_analysis(self):
        pass
```

Please see the provided class files for additional comments.

Hints, Material, and Requirements

Please follow the hints, material, and requirements below for implementing your project

Hints and Material

- You can view the python-chess library on GitHub.
- View this Colab Notebook for an introduction to python-chess.
- View and learn the rules and movement patterns of chess.
- Understand and apply Alpha-Beta Pruning.
- Understand the layout of a chessboard.
- Understand SAN and UCI notations.
- View an explanation about Forsyth–Edwards Notation (FEN).
- You can play Chess Online to understand openings and defenses.

Requirements

- You can use the python-chess library.
- You can not use any other import without explicit permission.
- Your project has to utilize alpha-beta pruning effectively.
- Visualizing pruning, trees, or results is optional.
- You have to use the provided API.
- You can not change the provided API.
- You can not change the provided file filenames.
- Your submission .zip file can not contain subfolders.
- Your submission .zip file needs to contain all the required resources.
- You are required to work with your assigned group member.
- You need to explain your submission. Gen AI submissions are not accepted.
- Only one submission per group is accepted.

Ideas and Tips for Your Implementation

- When implementing your project with the default chessboard FEN, you may have to retrieve a considerable depth before being able to make decisions in your tree. Instead, start with an advanced game FEN such as **`4k2r/6r1/8/8/8/8/3R4/R3K3 w Qk - 0 1`**
- You will need to evaluate moves for each turn to understand whether an attack or loss is acceptable. Try assigning values to individual chess pieces of the same type. In this context, Paws are worth less compared to your queen. After each turn, evaluate and compare the total points for black and white.
- You can create as many helper functions or method overloads as you want. Only the code within your AlphaBetaChessTree class is considered for testing.
- You can implement a main function e.g. to play chess against the computer or your group member, or to test the correct functionality of your implementation and algorithms.

Your Project

You will work in groups of 2 to design and implement an alpha-beta pruning chess tree using various data structures as covered in the Data Structures class. Your game tree should be capable of efficiently selecting the next best possible move given a defined game state.

You are required to work with your group partner.

Your Submission

You are required to submit a well-organized project zip file containing your source code, documentation, and resources to Gradescope. Your documentation should outline how your project works on the first page and the individual contributions per project member on the second page. Please upload at least the following files using the below file name:

- documentation.docx
- AlphaBetaChessTree.py
- TreeNode.py
- main.py

Only one team member needs to submit a solution to Gradescope.
Late submissions are not accepted, and time extensions are not granted.

Evaluation and Grading

This project aims to foster your creativity, critical thinking, and practical application of data structure concepts while allowing you to take ownership of your learning experience. Your project is evaluated and graded at the end of the semester through Gradescope.

A Gradescope assignment for submitting your solution will become available one week before the end of the semester. Please immediately start to work on your project and coordinate your work with your group members.