**Report Homework 06**

Wenyu Yang

* + Questions specific to this assignment:
    - In solve\_opponent\_move(), why do we return the opposite (negative) of the score used in the base case of solve\_my\_move()?

Our Minimax algorithm evaluates all game states from the perspective of a single player (specified by the player parameter), where in solve\_my\_move(), a higher score benefits this player. However, in solve\_opponent\_move(), when it's the opponent's turn, any move leading to our (player's) win would be the worst choice for them (-1 points), while moves leading to their win would be their best choice (1 point), as the opponent will always strive to choose moves that minimize our potential score. This mirrors the real-world strategy where players make decisions to maximize their own chances of winning while minimizing their opponent's opportunities for success.

* + - Why is there no winner if both sides are playing optimally?

When both players use the Minimax algorithm to play optimally in Tic Tac Toe, there will be no winner because the game is a "solved" game with perfect strategy known.

Through our Minimax implementation, each player will always make the optimal move by: blocking any potential winning moves by the opponent, taking winning opportunities when available, and choosing strategically advantageous positions when no immediate threats or opportunities exist. This leads to a balanced game where neither player can force a winning position - if one player attempts to create a winning setup, the other player will always be able to identify and counter it through the Minimax algorithm's evaluation of all possible future game states. This is why when our code runs with both players using the Minimax algorithm, it consistently ends in a draw, demonstrating that Tic Tac Toe, when played perfectly, is fundamentally a balanced game where neither player can force a victory.

* + - This pseudocode includes alpha-beta pruning. What would be the pseudocode for implementing this same algorithm, but without alpha-beta pruning?

The key distinction lies in function signatures - the alpha-beta pruning version requires additional parameters (alpha and beta) that act as boundaries for the acceptable score range, while the basic version only needs the game state and player information. These alpha-beta parameters enable the pruning mechanism by keeping track of the best achievable scores for both players.

Alpha-beta pruning implements an intelligent search strategy that stops exploring branches once it determines they cannot yield better results than already found options. In contrast, the basic version must exhaustively examine every possible move sequence until game completion, regardless of whether those branches could potentially lead to better outcomes.

The efficiency gain from alpha-beta pruning comes from its ability to eliminate unnecessary branch explorations, potentially reducing the number of nodes evaluated significantly. Without pruning, the algorithm must traverse the entire game tree, examining every possible game state, which becomes exponentially more costly as the game complexity increases.

While both versions arrive at the same optimal moves, alpha-beta pruning represents a significant optimization over the basic minimax algorithm. The addition of alpha-beta parameters enables selective branch exploration, allowing the algorithm to intelligently skip portions of the game tree that cannot influence the final decision.

* + The usual questions:
    - How long did this assignment take you? (1 sentence)

I worked on this assignment for 3 days.

* + - Whom did you work with, and how? (1 sentence each)
      * Discussing the assignment with others is encouraged, as long as you don’t share the code.

I worked on this assignment by myself, reviewing lecture materials.

* + - Which resources did you use? (1 sentence each)
      * For each, please list the URL and a brief description of how it was useful.

I discussed with my classmate and I primarily used the course slides as a resource for this assignment.

<https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/>

* + - A few sentences about:
      * What was the most difficult part of the assignment?

The most challenging part was correctly implementing alpha-beta pruning in a way that maintained game state integrity while ensuring proper score propagation between solve\_my\_move() and solve\_opponent\_move() methods.

* + - * What was the most rewarding part of the assignment?

The most satisfying moment was seeing the AI make perfect decisions and consistently achieve draws when playing against itself, proving that the minimax algorithm was implemented correctly.

* + - * What did you learn doing the assignment?

I gained a deep practical understanding of how minimax algorithm works in real game scenarios and learned the importance of state management in recursive game tree algorithms.

* + - * Constructive and actionable suggestions for improving assignments, office hours, and class time are always welcome.

It would be helpful to have more intermediate checkpoints or unit tests to verify each component (like GameState, Move classes) individually before implementing the complete minimax algorithm with pruning.