Awale Playing MiniMax

Nathan AMOUSSOU, Ismail EL AMRANI

December 2024

The Evaluation Function The evaluation function in the AI assesses the quality of a game state based on multiple factors:

- *Score Difference:* The difference in scores between the current player and the opponent (as the `MinimaxAgent6` attribute `SCORE_WEIGHT`).
- *Board Control:* The total seeds under the current player's control versus the opponent's (as the `MinimaxAgent6` attribute `CONTROL_WEIGHT`).
- *Capture Potential:* Positions with seeds that are likely to be captured (e.g., totals of 2 or 3 seeds) are identified, and the player's advantage in these positions is calculated (as the `MinimaxAgent6` attribute `CAPTURE_WEIGHT`).
- *Mobility:* The number of holes the player can play from, representing the freedom to make strategic moves (as the `MinimaxAgent6` attribute `MOBILITY_WEIGHT`).
- *Weighted Combination:* The factors above are combined using pre-determined weights for a final evaluation score.

We chose this evaluation function because we considered that factors like mobility and capture potential directly correlate to winning strategies in Awale. We also empirically tested the weights to maximize performance during gameplay analysis.

The AI Algorithm The AI implements an iterative deepening Minimax algorithm with several enhancements:

- *Alpha-Beta Pruning:* Reduces the search space by cutting branches that won't affect the final decision (implemented in `MinimaxAgent6_4.minimax`).
- *Transposition Table:* Stores previously computed states to avoid redundant calculations.
- *Move Ordering:* Prioritizes moves likely to produce better results, improving pruning efficiency (implemented in `MinimaxAgent6_4.minimax`, `.get_move` and `._order_move`).
- *Iterative Deepening:* Begins with shallow searches and increases depth incrementally, ensuring a valid move is always available within the time limit (implemented in `MinimaxAgent6_4.minimax`).
- *Principal Variation:* Explores the most promising move first, based on previous iterations, for faster convergence (implemented in `MinimaxAgent6_4.minimax`).
- *Vectorized Evaluation:* Uses NumPy to speed up state evaluation, critical in a performance-limited language like Python.

Why did we chose this algorithm? Minimax with Alpha-Beta pruning was chosen because it is a well-established method for two-player games like Awale, offering optimal decision-making by pruning irrelevant branches to reduce complexity. This algorithm was explicitly taught in class by the professor, making it a natural choice to align with course objectives. It also allowed us to incorporate advanced features like move ordering and iterative deepening. However, Python's performance limitations made it difficult to fully leverage Minimax's potential. Alternatives like Monte Carlo Tree Search (MCTS) could have utilized parallel processing and GPUs more effectively for deeper exploration.

Lessons Learned While Minimax was appropriate, Python's slowness limited the depth of exploration and overall competitiveness. Using a compiled language like C++ or Java would have enabled faster computations and deeper searches, significantly improving performance.