A4 Report

Authors: Charugundla, Saimanasvi; Tran, Kristina

Agent Name: Kronos, the AI Tactician of the Grid Wars

Short Description: Kronos is a calculating strategist forged from pure logic. Designed to analyze, adapt, and annihilate opponents, it brings a competitive, dominant personality to the board, embodying an unstoppable force of precision and control.

Twin Feature

The twin agent differs from the basic version only in name and minor persona adjustments. While the main agent, Kronos, presents itself with dominance, the twin operates under a different nickname ("Nic2") and assumes a more passive presence. This feature is helpful for testing internal consistency by running the agent against a near-identical version of itself.

Minimax and Alpha-Beta Pruning Implementation

Our agent uses a recursive minimax algorithm to choose the optimal move in a game of K-in-a-Row. In this implementation, the minimax function explores the game tree to a specified depth (the "ply") and returns both the best move and its corresponding evaluation score. For terminal nodes or when the maximum depth is reached, the algorithm computes a static evaluation of the board state using a heuristic that measures the difference in consecutive tokens (that is, the longest sequence of tokens) for each player.

To improve search efficiency, we integrated alpha-beta pruning into the minimax function. The algorithm maintains two parameters during the search:

- **Alpha**: the highest evaluation score currently guaranteed along the path to the root for the maximizer (player X).
- **Beta**: the lowest evaluation score currently guaranteed for the minimizer (player O).

At each node in the game tree, if the algorithm finds that the current move's evaluation makes it impossible for the opponent to allow a better outcome (when beta is less than or equal to alpha), it prunes the remaining branches. This means that further moves in that branch are not explored because they cannot influence the final decision.

In addition, when not in autograding mode, we implemented a child ordering technique. Before recursively exploring moves in the minimax function, the agent orders the available moves by their static evaluation. By considering the most promising moves first, the chance of triggering

early alpha-beta cutoffs increases. This further reduces the number of nodes that need to be evaluated.

How Alpha-Beta Pruning Reduces Search Time

Alpha-beta pruning significantly reduces search time in a minimax algorithm by eliminating branches that do not affect the final decision. Without pruning, minimax examines every node in the game tree up to a given depth, which can be computationally expensive. With alpha-beta pruning:

- **Pruning Ineffective Branches:**
  When the algorithm determines that a move results in a score worse than a previously examined option (using the current alpha and beta values), it stops evaluating further moves in that branch. This eliminates the need to compute the static evaluation of many nodes that would never affect the outcome.
- **Improved Branching Factor:**
  In an ideal scenario with perfect move ordering, alpha-beta pruning can reduce the effective branching factor from b to roughly the square root of b. This exponential reduction in the number of nodes evaluated allows the agent to search deeper within the same time constraints.
- **Early Cutoffs Through Move Ordering:**
  Our implementation orders moves based on their static evaluations so that the most promising moves are examined first. This ordering increases the likelihood of early alpha-beta cutoffs and further reduces the number of evaluations needed.

In summary, alpha-beta pruning allows our agent to concentrate computational resources on the most relevant parts of the game tree. This results in faster decision-making and the ability to search deeper within the allotted time, which is critical in a real-time game setting.

Agent Persona Details

Kronos is designed as a powerful AI strategist with a commanding presence. Its persona embodies themes of control, strategy, and dominance. The agent’s tone is formal and calculated, reflecting its focus on winning at all costs.

Game-Responsive Dialogue Features

The agent's dialogue is designed to reflect its persona during gameplay. For example:

- After making a move, it announces the location with confidence: "I have made my move at (x, y). Your turn."
- It challenges opponents with assertive introductions, reinforcing its dominant character.

The utterances are kept relevant by referring to the current state of the game and maintaining a consistent persona throughout.

Opponent-Responsive Dialogue Features

The agent includes basic reactive dialogue to respond to the opponent's remarks. It scans the opponent's last statement for keywords and responds with tailored remarks that align with its dominant persona. For instance:

- If the opponent uses words like "lucky" or "random," the agent replies with: "Luck is but a fleeting illusion in the presence of pure strategy."
- If the opponent acknowledges a strong move, the agent responds: "Acknowledgment of my superiority is the first step toward understanding your inevitable defeat."

This feature enhances the immersion of the game by making the agent feel more interactive and aware of the opponent’s remarks.

Development of Dialog Capabilities

The dialog system was developed iteratively:

1. **Initial Phase:** Basic remarks were added for turn announcements and move confirmations.
2. **Personality Development:** Utterances were updated to reflect the agent's competitive nature.
3. **Game-State Relevance:** Remarks were refined to include the specific moves made, ensuring relevance to game progress.
4. **Opponent Response Integration:** Basic analysis of opponent remarks was introduced to allow tailored responses during gameplay.

Through this iterative process, the agent's persona became more consistent with its theme.

Extra Credit Implementations

1. Zobrist Hashing

Implemented for optimized move searches, Zobrist hashing assigns a unique hash to each board state, allowing quick retrieval of previously evaluated states. This reduces computational time by avoiding redundant evaluations.

- The system tracks:
    - Number of writes to the hash table
    - Number of read attempts -

- Number of successful retrievals

Advanced Response System: Implemented responses for specific in-game prompts.

‘Tell me how you did that’: Triggers an explanation of the agent's reasoning based on the current board state.

'What's your take on the game so far?': Provides a summary of the game's progress and a prediction based on the current evaluation of the board.

2. Static Evaluation-Based Move Ordering

For improved efficiency during search, our agent implements static evaluation-based move ordering. This feature simulates each available move, evaluates the resulting board state using the static evaluation function, and then sorts the moves accordingly.

- **Move Simulation:**
  Each potential move is applied to the current state to generate a new state.
- **Evaluation:**
  The static evaluation function computes a heuristic value for the new state.
- **Ordering:**
  - For the maximizing player (X), moves are sorted in descending order by their evaluation values.
  - For the minimizing player (O), moves are sorted in ascending order.

This ordering increases the chance of early alpha-beta cutoffs during the minimax search.