AI Project Report

Luu Nguyen Chi Duc – V202200664

1. Introduction

This AI project is a Gomoku project, a popular game which is similar to Tic Tac Toe but with a bigger board and the aim of connecting five stones in a row. The project idea was to construct AI players with different strategies (AI agents) and see how effectively they could play.

The report included a different AI theory such as including reflex agents, heuristic evaluation functions, and Monte Carlo Tree Search (MCTS) improvements. I discovered how effective each strategy was by testing these AI gamers against various opponents and also research to realize that a winning strategy exists.

2. Methodology

Reflex Agent

The class GMK_Reflex initiates a reflex agent in Gomoku, associating it with a letter to represent the player. After that, this will pick the best move depending on the score obtained from potential moves into empty cells. It will return a desirability depending on the player and opponent's scores. All this happens by examining the position that each stone is sitting on the board.

The setPosition method assesses a stone along four winning directions. It calls the function evaluate_direction for computing the score in one direction. Using this length of the sequence of rocks, get_score calculates its score concerning whether it is blocked or not. The weights are adjusted accordingly on how well the algorithm performs using feature extraction and Q-learning algorithms.

Alphabeta Agent

The minimax method uses a recursive search to evaluate player moves and opponent losses, optimizing the search with Alpha-Beta pruning to reduce computation time.

The evaluation method returns a heuristic score to evaluate the advantage of a player in a board configuration. It considers patterns of stones on the board, adjusting scores depending on whether they exist and how well they are placed. All of this is done using a weighted scoring system to differentiate threats from promising positions belonging to the player and the opponent. Five stones in a row, four with one empty space, four blocked at one end, three with two empty spaces, three blocked at one end, two with two empty spaces. The pattern scores were tested to get the highest evaluation score possible.

The method PromisingNextMoves generates an ordered list of all the possible moves according to their heuristic score, giving priority to what is most likely to offer an advantage on the board. It works by analyzing the influence from surrounding stones and sorting potential moves in descending order of their total influence.

Approximate Q-Learning Agent

The method uses SimpleExtractor to identify game state features, estimate Q-values, and train a player using a Q-learning algorithm with epsilon-gredy exploration strategy, enhancing learning and decision-making capabilities.

The SimpleExtractor class extracts features from the current game state to evaluate the strength or importance of certain positions on the board. It counts open threes and fours, unblockable fours and fives, and player biases. The extractor initializes with methods to compute these features and returns a dictionary with key values.

These features are crucial for Approximate Q-Learning algorithms to guide agents towards better decisions over time. The SimpleExtractor enhances the player's ability to assess and prioritize moves during gameplay, improving AI performance through iterative gameplay and learning cycles.

Better Agent

The GMK_BetterMCTS player enhances the MCTS algorithm with several advanced techniques through advanced techniques, including heuristic-based rollouts, hybrid MCTS and alphabeta search, better rollout policy, progressive widening, pattern search, memoization, and improved node selection. All these methods are aimed at ensuring the enhancement of the quality of simulations and search space optimization with competitive performance gain. These were in place to enhance strategic depth, efficiency, and competitive performance in Gomoku.

3. Evaluation

Here are some important results when testing and evaluating different agents and players:

Reflex player vs Beginner: 40W-10L-0D. Eval: 50/70.

Alphabeta player vs Intermediate: 9W-0L-1D. Eval 66/70.

Alphabeta player vs Master: 8W-2L-0D.

Naive MTCS player (2000 sim) vs Intermediate 8x8: 3W-0L-7D. Eval 1000 sims: 40/70.

Better MTCS player with 1000 sim Eval: 47/70.

Approximate Q-Learning player with 150 num eps Eval: 65/70.

4. Conclusion

One of the discoveries was how effectively heuristic-based rollouts and the combination of MCTS and minimax search increased the agent's performance. I also discovered that reflex agent performed effectively in easy situations, but more complex techniques were required to defeat more difficult opponents.

I also encountered several problems, such as dealing with the vast number of potential actions and the agent should have good running time. To address these issues, I applied approaches such as progressive broadening, threat-based search, and game state storage.

In the future, I may investigate applying deep learning to improve the MCTS, as was done in AlphaGo Zero. We may also try various board sizes to evaluate how well the AI adjusts. There is always opportunity for development, such as improving the heuristic functions and incorporating more powerful machine learning models.