Project 2: AI agents for Gomoku

Truong G. Bao
CECS
VinUniversity
21bao.tg@vinuni.edu.vn

1 Introduction

Gomoku, also known as Five in a Row, is a strategic board game where the objective is to be the first player to align five of their stones consecutively on a grid.

In this project, various AI agents were developed to play Gomoku, each employing distinct methodologies. For the first task, a reflex agent was implemented, whose actions are based on the current state of the board. In the second task, a heuristic evaluation function was developed, building upon the reflex agent's foundation to guide the agent's moves more effectively. Given the vast search space, approximate Q-learning was utilized, incorporating domain-knowledge features to enhance the learning process. Additionally, the Monte Carlo Tree Search (MCTS) algorithm was optimized using the reflex agent from the first task and the alpha-beta algorithm from the second task.

The results of this project demonstrate significant performance improvements across all implemented algorithms, successfully meeting nearly all project requirements.

2 Methodology

2.1 Question 1: Reflex Agent

Inspired by Sun Tzu's quote, "If you know the enemy and know yourself, you need not fear the result of a hundred battles," from The Art of War, I decided to analyze the strategy of the intermediate agent before implementing the reflex agent. It became evident that the intermediate agent itself was a reflex agent. The strategy of the intermediate agent is as follows:

- Execute a winning move.
- Block the opponent's winning move.
- Create a straight four.
- Block the opponent's straight four.
- Construct moves with the most threats.

With a slight adjustment, incorporating the "control center" strategy, my reflex agent was enhanced to outperform the intermediate agent and even defeat the master agent. Two following codes are in my reflex.py file, line 35, line 61, respectively.

```
distance_to_center = abs(move[0] - center[0]) + abs(move[1] - center[1])
```

```
center = (game.size // 2, game.size // 2)
```

2.2 Question 2: Minimax (with Alpha-Beta Pruning) Player

In this question, I evaluate the current board given the current player as follows:

- If the game is over, give the terminate score for that state (10000000 * (number of emptycells + 1), or -10000000 * (number of emptycells + 1)).
- Check all the "threats" of the current and opponent players. There are three types of threats (Threat 1: NONE X X X X None, Threat 2: None X X None X None, Threat 3: O/None None X X X None None/O)). Assign each threat to the hand-designed score.
- Check the number of "surrounded aligns" for the current and opponent players.
- Check the "control center" density of each player.
- the difference between the current and the opponent player is the score of the board.

In the implementation of the promising next moves function, an important observation was made: the next move in the game is unlikely to occur far from the existing moves on the board. This insight significantly reduces the search space and enhances the algorithm's efficiency.

2.3 Question 3: Approximate Q-Learning Player

The updated rules are based on the Tabular Q-Learning.

The features are chosen as follows:

- Number Of unblocked 2/3/4/5 of the current player.
- Number Of unblocked 2/3/4 of the opponent.

Another strategy to achieve better performance with a small number of episodes when it comes to dueling with other agents is the need to specify the initial weight of each feature. This can be done by using domain knowledge, trial, and error.

2.4 Question 4: Monte Carlo Tree Search (MCTS) Player

Identical to the agent for tictactoe.

2.5 Question 5: Better MCTS Player

Almost similar to the agent in question 4 with two optimiations:

- Hybrid MCTS and Minimax: Combine MCTS with a depth-limited minimax search to improve the evaluation of leaf nodes.
- Better Rollout policy: Use the reflex agent implemented in question 1 choose the "good move" that maximize winning rate to achieve the better rollout policy.

3 Results

3.1 Versus other agents

Except for the Better MCTS (15x15 board), all the results are conducted in the settings: Board size: 8x8, number of games: 10, and the weights for qplayer are from the file 8x8-200.pkl. Table 1 shows the results of all players

3.2 Evaluation

Table 2 shows the results for the evaluation of each agent.

4 Conclusion

The Gomoku AI project demonstrated significant performance improvements through various AI methodologies. The reflex agent provided a solid foundation, while the heuristic evaluation function enhanced decision-making by incorporating domain-specific knowledge. Approximate Q-learning efficiently navigated the search space with domain-knowledge features, and the Monte Carlo Tree

Player 1	Player 2	Player 1 Wins	Player 2 Wins	Draws
Reflex Player	Beginner Player	10/10	0/10	0/10
Reflex Player	Intermediate Player	4/10	0/10	6/10
AlphaBeta Player	Beginner Player	10/10	0/10	0/10
AlphaBeta Player	Intermediate Player	4/10	3/10	3/10
AlphaBeta Player	Master Player	8/10	0/10	2/10
Approximate Q-Learning Player	Beginner Player	10/10	0/10	0/10
Approximate Q-Learning Player	Intermediate Player	3/10	0/10	7/10
Approximate Q-Learning Player	Master Player	3/10	0/10	7/10
Approximate Q-Learning Player	Advanced Player	0/10	0/10	10/10
MCTS (2000 simulations)	Beginner	8/10	2/10	0/10
Better MCTS (40000 simulations)	Intermediate	TimeOut	TimeOut	TimeOut
Better MCTS (40000 simulations)	Advanced	TimeOut	TimeOut	TimeOut
Better MCTS (40000 simulations)	Master	TimeOut	TimeOut	TimeOut

Table 1: Performance statistics of Reflex Player, AlphaBeta Player, and Approximate Q-Learning Player, MCTS Player, and Better MCTS Player against various opponents on an 8x8 board over 10 games.

Agent	Evaluation Score
Reflex	61/70
Alphabeta	65/70
Approximate Q-learning	46/70
MCTS (10000 simulations)	66/70
MCTS (1000 simulations)	41/70
Better MCTS (10000 simulations)	37/70

Table 2: Agent Evaluation Scores

Search (MCTS) algorithm, optimized with the reflex agent and alpha-beta algorithm, showed notable gains.

This project has encountered various challenges related to domain knowledge, including the design of a suitable heuristic function score for the minimax algorithm, the selection of feature sets for approximate Q-learning, and the creation of strategies for a reflex agent. Moreover, the number of features doesn't align with the performance of an agent, and the number of episode isn't significant if the set of features is not good enough. I personally trained the model on the 2000 epochs, 16 hours of training, 10 features. And still get the 30/70 evaluation score, my model even didn't beat the intermediate player (You could try to load the weight 8x8-2000k.pkl to see the result)

Several significant insights emerged during the course of this project regarding the intricacies of working with and learning about AI. For instance, the initial weights assigned to features in approximate Q-learning significantly influence the convergence speed to the optimal policy. Specifically, when the number of episodes is limited, suboptimal initial weights can impede the attainment of a robust policy. This highlights the critical role of optimization in AI.

In the context of the Gomoku game, the vast search space presents considerable challenges. The Minimax agent experiences prohibitive action times, and Monte Carlo Tree Search (MCTS) requires a substantial number of simulations to achieve satisfactory performance. Additionally, tabular Q-learning struggles to capture all Q-values within the state table. Consequently, optimized algorithms are essential to address the inefficiencies inherent in traditional approaches.