Hrishikesha H Kyathsandra

rishihk@iastate.edu

March 29th 2024

COMS 472: Principles of Artificial Intelligence

Lab 2 Report

Index

Introduction

In this project I have implemented an AI agent capable of playing Gomoku (Five-in-a-Row), a strategy board game played on a 15x15 grid. The agent uses an alpha-beta pruning algorithm to determine the optimal move based on the current game state, and given evaluation function.

Game Rules

1. Players alternate turns placing a stone of their color (white or black) on an empty intersection on the board.
2. Black plays first, with the first stone placed in the center of the board.
3. The second player's first stone may be placed anywhere.
4. The first player's second stone must be placed at least three intersections away from the first stone.
5. The winner is the first to form an unbroken line of five stones horizontally, vertically, or diagonally.
6. The game ends in a draw if the board is filled without any player forming a line of 5 stones.

Components

- Alpha-Beta Agent: Utilizes the alpha-beta pruning algorithm to efficiently determine optimal moves by evaluating the game state up to a certain depth.
- Evaluation Functions: Two evaluation functions are implemented to assess the game state's favorability towards the agent. These functions consider factors like potential unbroken lines of stones and blocking opponent moves.
- Move Generator: Generates a list of legal moves from the current state, ensuring that moves adhere to Gomoku's rules.
- Game State Management (GUI): Manages the board's state, including stone placement and checking for game-ending conditions.

How to compile and run the program

- Extract the submitted source code zip file.
- Go to the root of the project through the command line.
- Run the command "**python3 main.py"** to start the game.
- A welcome message followed by the rules of the game and the initial game board will be displayed in your console.
- User Input: Enter your move as two integers representing the 0-indexed row and column coordinates on the 15x15 game board. The board will be updated accordingly and displayed
- AI move: After each user move, the AI agent will calculate its move and display the chosen move along with time taken. The board will be updated accordingly and displayed
- The game ends when either player wins by connecting five stones in a row or when the board is completely filled resulting in a draw, or when either player quits/resigns.
- You can choose which evaluation function you would like to play against, by using it inside the alpha_beta_with_cutoff inside ai_agent.py.

Search Depth Analysis

When I made the AI look more moves ahead in the tree by increasing the search depth, two things happened: it got better at the game, but it also took longer to decide on moves.

- Decision Quality: Making the AI look further into possible future moves made its decisions better. It started to recognize smarter strategies and important moves that could block the opponent from winning. These smarter plays were especially noticeable at deeper levels, where the AI could plan several moves ahead. However, after reaching a certain depth, making the AI deeper didn't really make it much smarter because it was already finding the best moves it could within the rules of the game and what it knew from its evaluation function.

- Computation Time: The main downside of increasing the search depth was that the AI took a lot longer to decide what to do. Each extra level meant the AI had to consider way more possible moves, which slowed down its decision-making. This was a big challenge, especially in a game like Gomoku where there are a lot of possible moves at each turn. I tried to speed things up by having the AI look at the most promising moves first, but there was still a trade-off between making the AI smarter and keeping it fast enough to be practical.

The method I used to demonstrate the point above was through using time stamps to record exactly how long the AI took to decide on each move. This provided me with clear data on the relationship between search depth and decision time as mentioned above. Along with time, I also tracked the AI's win rate, and how many best moves it was making which allowed me to access its decision quality directly.

Evaluation Functions Analysis

The way the AI decides if a move is good or not is most important. I looked at two different methods for doing this.

- Immediate Wins or Blocks (score_immediate_moves_e1): This method was all about looking for quick wins or moves that stop the opponent from winning right away. It was fast, which meant the AI could think ahead more moves in less time. It was straightforward, and was surprisingly very quick and was able to successfully block opponents wins and was able to win when the opponent blundered. This function quickly checks the entire board for spots where placing a stone could immediately win the game or block the opponent from winning. It assigns high scores to winning moves and penalties to spots where the opponent must be blocked to prevent their win.

- Thinking Ahead (defensive_priority_e2): This method was more about being careful and setting up for future moves. It took into account patterns that could lead to winning later on. This made the AI smarter, especially in tricky situations, but it also made it slower because it had to think about more complex things. This evaluation function prioritized defense over offense and showed very good results in blocking opponents wins.

I had the AI play using these two methods to see which was better.

The more thoughtful, defensive method often led to smarter defense, showing that taking a bit more time to decide on a move could pay off in a tougher game. However, it also meant that each move took longer, which could be a problem if you're trying to make the AI work in real time.

The first evaluation function which prioritized both attack and defense equally demonstrated quicker speeds in decision time, and made decent quality moves in both offense and defense.

Improving the Evaluation Functions

Improving Immediate Wins or Blocks (score_immediate_moves_e1):

- Improvement: This function could be enhanced by adding a layer that considers not just immediate wins or blocks but also moves that create double threats (where two winning moves are set up simultaneously). This improvement would make the AI smarter by forcing opponents to respond to multiple threats, increasing the chances of winning, and also able to block double threats by the opponent.

- Effect: With this enhancement, the AI would not only react to immediate game states but also proactively create situations where it's more likely to win in the next few moves, making the AI more aggressive, strategic and defensive by recognizing opponents double threats and blocking them.

Enhancing Thinking Ahead (defensive_priority_e2):

- Improvement: A significant improvement would be incorporating a dynamic scoring system that adjusts the importance of certain patterns based on the game's progress. Early in the game, the function could prioritize board control and central dominance, while later in the game, it could focus on immediate threats and opportunities.

- Effect: This would make the AI's decision-making process more adaptable to different stages of the game. Early on, it would work on establishing a strong position, and as the game progresses, it would switch to aggressively pursuing wins or defending against losses. This adaptability would make the AI tougher to predict and more effective throughout the game.

Conclusions

Improving search depth made the AI smarter but slower, showing the need to balance between thinking ahead and decision time. Tweaking evaluation functions, like adding threat recognition and dynamic scoring, significantly boosted the AI's strategic play, making it both more aggressive and adaptable. These enhancements together would lead to a more efficient Gomoku AI.

References

Aimacode, GitHub. https://github.com/aimacode