Gabriel Ferreira
Jin Tian
572 - Principle of Artificial Intelligence
March 19, 2024

Lab 2 - Gomoku Game Report

6. Compare the effect of increasing search depth (come up with a method to demonstrate your point)

In order to compare the effects of increasing the search depth of our alpha-beta agent, I used some measure game performance across different metrics, along with the concept and principles learned in class.

The bellow bar chart presents the performance metrics of an alpha-beta pruning algorithm playing against an opponent making random moves. The metrics are recorded for the algorithm operating at search depths of 1 and 2. Each bar represents a specific performance metric at a given depth level.

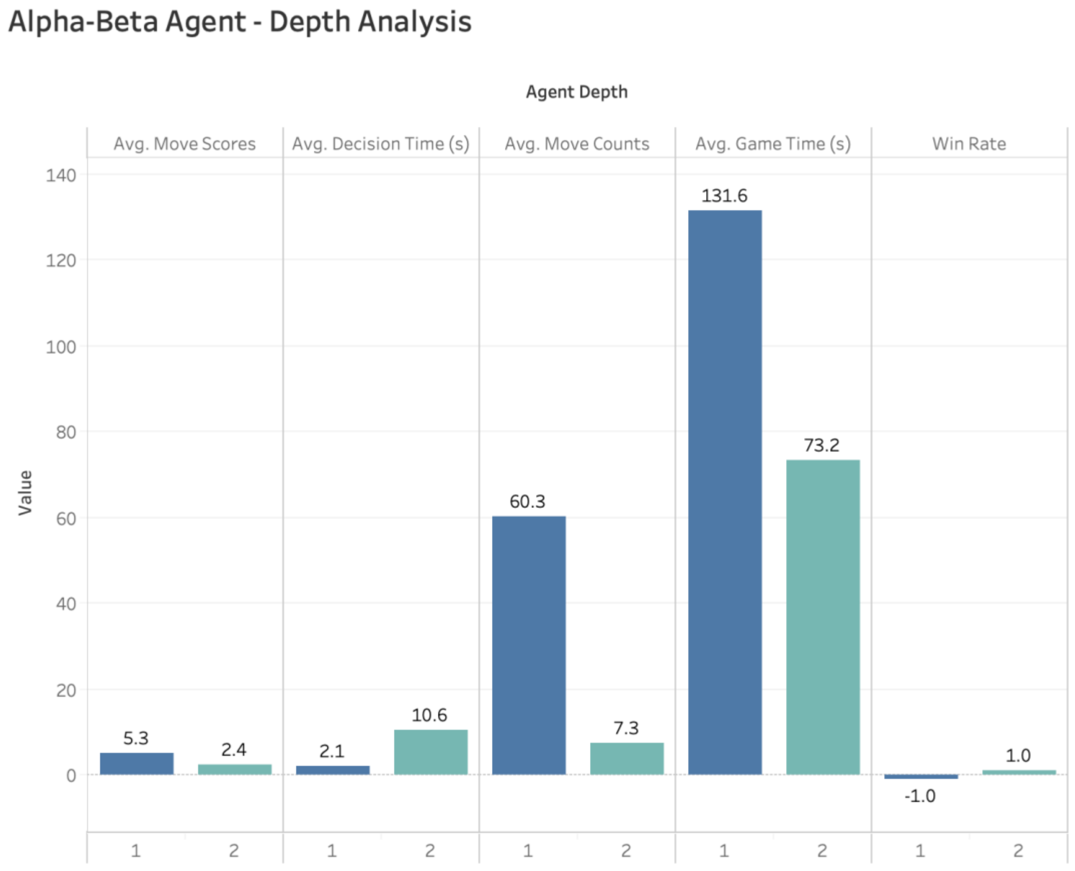

Figure 1. Average Performance Metrics of Alpha-Beta Pruning Algorithm at Different Search Depths Against a Random Opponent

Let’s do a short analysis of each of these metrics.

Average Move Score

This metric represents the average score per move per game at each depth. We can see that at depth 1, the agent’s moves score way higher than the same at depth 2. This only makes sense when we understand the alpha-beta pruning assumes the opponent is playing optimal.

At depth 1, the agent only considers the immediate next move and its score. This can lead to choosing moves that appear favorable in isolation, potentially overlooking the opponent's response.

When the search depth increases to 2, the agent employs alpha-beta pruning to improve efficiency while exploring possible opponent responses. This strategic consideration might lead the agent to choose moves that sacrifice some immediate positional advantage in favor of moves that are more robust against the opponent's best counter.

In other words, the agent is giving up some immediate high-scoring options (depth 1) to make moves that are more strategically sound against a potentially strong opponent (depth 2). This is a trade-off between immediate gains and long-term security.

Average Decision Time in Seconds

I believe the bar chart for this metric is pretty straight-forward and very intuitive, showing that decision time increases significantly as search depth increases. It also allows us to better understand the trade-off in computational complexity associated with deeper search levels.

Thus, it’s reasonable to think that deeper searches allow for more strategic moves but come at the cost of increased computation time. Finding the optimal depth for agent depends on the desired balance between performance and real-time playability.

Average Move Counts

This metric reflects the average number of moves played by the agent in each game at a specific search depth. This is a clear manifestation of the behavior explained in the analysis of **Average Move Score** given the agent is much more effective at depth 2.

Average Game Time in Seconds

This metric reflects the combined effect of decision time and move count. While depth 2 has a longer decision time, it’s much more effective having significantly lower number of moves played, which leads to a shorter overall game time when compared to depth 1.

Win Rate

The agent's performance at Depth 1 highlights a limitation in concluding games successfully. Its strategy involved placing four pieces in a row and then skipping a spot resulted in delaying or in most scenarios preventing a win.

Contrastingly, at Depth 2, the agent exhibited markedly improved effectiveness. The deeper search enabled the agent to execute more decisive plays, leading to 100% winning rate against a player playing randomly.

This distinction highlights the critical impact of search depth on the agent's ability to secure victories.

7. Implement at least two evaluation functions with varying quality. Compare the effect of improving the evaluation function.

In order to compare the effects of improving the evaluation function used by our alpha-beta agent, I used the same metrics used to evaluate the agent at different depths. However, at this time not only the random opponent was fixed but the depth as well.

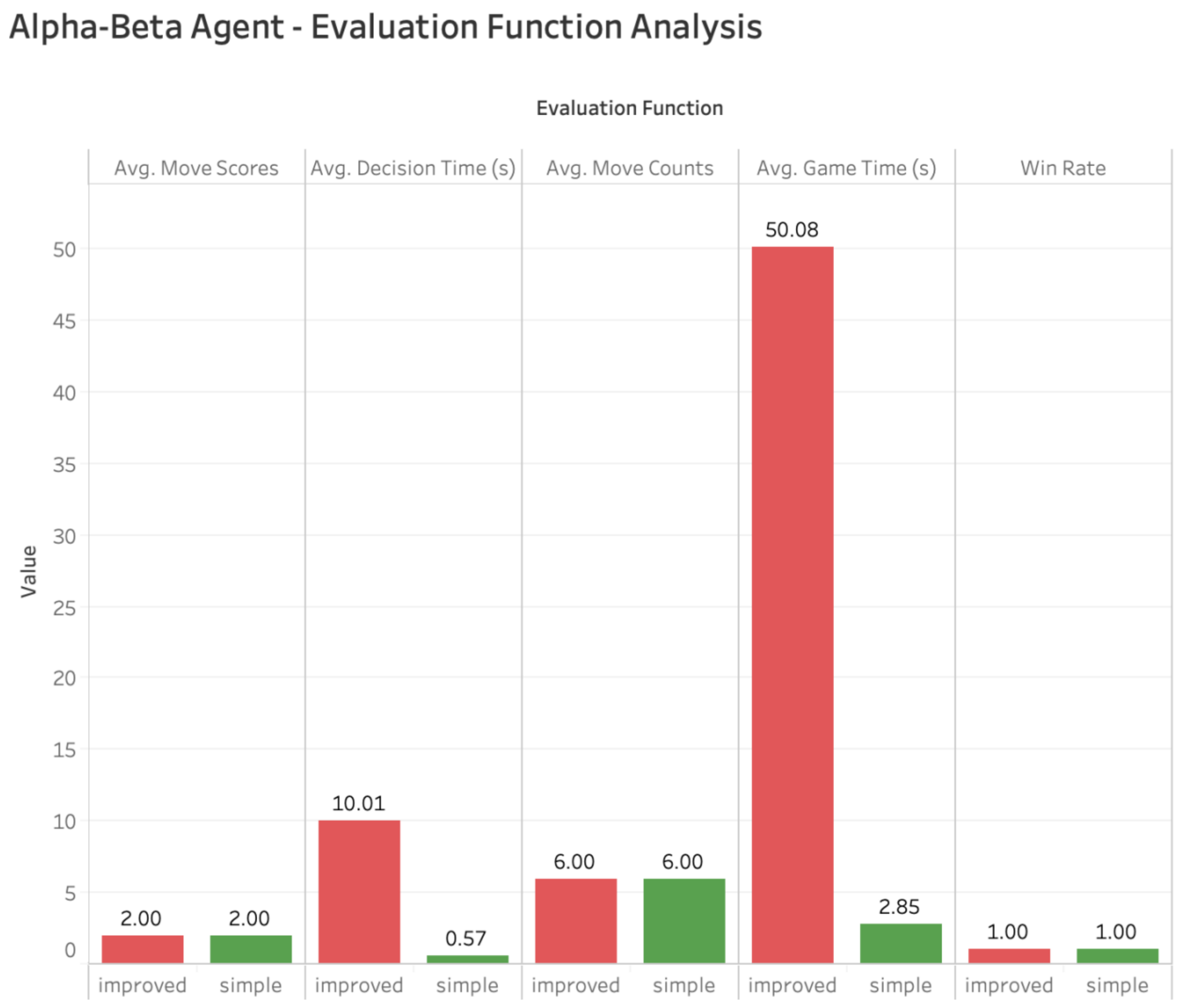

Figure 2. Average Performance Metrics of Alpha-Beta Pruning Algorithm Using Different Evaluation Functions Against a Random Opponent

The **Simple Evaluation Function** simply compares the number of stones each player has on the board, with a higher score for having more stones than the opponent.

The **Improved Evaluation Function** is more nuanced, accounting for specific patterns like open threes, closed fours, and straight fives which are indicative of potential wins.

The insights from the analysis of these metrics highlights that complexity in a heuristic does not guarantee superior gameplay. The graph indicates that the improved evaluation function did not improve win rates or move quality. However, it noticeably increased the computational demand.

This finding serves as a reminder of the importance of equilibrium between high-quality decision-making and computational speed. This is especially true in expansive gaming environments (15 x 15), where additional computations can increase complexity exponentially.

Therefore, the goal is to achieve an equilibrium that allows high-quality decision-making without incurring exponential computational costs.

README

Lab 2: Gomoku Game Implementation and Analysis

This repository contains a Python implementation of the Gomoku game (also known as Five in a Row), along with utilities for playing the game, implementing various player strategies, and analyzing gameplay. Below is an overview of each file and its role in the project.

Overview

game.py: Defines the core game logic for Gomoku, including the board representation, game state, and rules for making moves. It provides the foundational classes and methods used by other components.

utils.py: Provides utility functions and helpers that support game setup, execution, and analysis.

players.py: Contains implementations of different player types, including human player, AI players using the Alpha-Beta pruning algorithm, and random players. **Note the human player implementation will provide a list of all legal moves at each play as asked in the instructions.**

play_gomoku.py: A script that sets up and runs a Gomoku game instance. Use this script to play Gomoku using the command line.

play_analysis.ipynb: A Jupyter notebook for analyzing different depths and evaluation functions as requested by the lab intrunctions. It includes code for running simulations, collecting gameplay metrics (llike win rates, decision times, etc).

play_analysis_eval_func.ipynb: A Jupyter notebook for analyzing differentevaluation functions as requested by the lab intrunctions. It includes code for running simulations, collecting gameplay metrics (like win rates, decision times, etc).

Playing the Game

Run the **play_gomoku.py** script to start a game. You can modify this script to set up specific player configurations (e.g. different depths) or setting two AI players to play against each other.

Acknowledgments

- [GitHub Pages] https://github.com/aimacode