Gomoku: A Look at Alpha-Beta Pruning

Elijah Anderson-Justis,
Samuel Kuhlwein,
&
Andres Rivero
Southern Oregon University
andersoe4@sou.edu

INTRODUCTION

Gomoku is a game played on an 11x11 in which each player places pieces. If a player gets 5 pieces in a row they win. Our task was to create an AI that would run against AI from other groups. One of the constraints for the project was that we implement the minimax search with AB Pruning. The other constraint was that each move had a 2 second limit placed on it. If the timer ended without a move being played, the AI would forfeit the match. Although the server is written in Racket, our group chose to write the AI in Java because it was more familiar and had some useful features built into it.

WORKING ON THE PROBLEM

The first step we took in creating the AI was to make sure a connection to the Gomoku server could be established. To speed up the process we ported the random player code, that the professor provided, from Racket to Java. This process was straightforward and painless. Once we knew we could send and receive data we started on the real code. To begin with our AI needed a way to keep track of the board and the changes that were being made to it.

For this task, we decided on creating a board class. We used this approach so that we could have a board that holds the location of the pieces as well as a board that holds the weights used in the search algorithm. Each time a move is made the both boards are updated.

WEIGHING OUR OPTIONS

Because the minimax search uses weights to decide the optimal move we had to figure out a way to weight each possible move. If weighted poorly then the outcome of the minimax search would not be optimal. Our approach was to look at where pieces had already been place. With this information, our program adds one to the weight of the surrounding locations or “moves” if the piece is ours, and the program subtracts one from each area that is close to the opponents move. In this way, the positive numbers represent moves that will benefit our AI. Negative numbers represent moves that will work against the opponent AI. Once the weights are applied to each location on the board, the program dives into the minimax search.

SEARCHING FOR AN ANSWER

The minimax implementation that we used is dependent on the Tree class that comes with Java. By using the default nodes as a base, it allowed for a quicker development time. For this to work however, there were a few extra variables that the nodes had to keep track of. For this task a point class was created. This class holds the row and column position of the move on the board as well as the weight that corresponds with the location. This allowed us to attach the separate point objects onto the nodes. Once we had the possible moves and their weights attached to the tree we could start the AB Pruning process.

PRUNING THE TREE

Our AI was doing good with the minimax search, but it needed some optimization because we only had 2 seconds to make a move or we would forfeit. To speed up the search process we implemented AB Pruning on top of the original minimax sort. Because we knew ahead of time that the board was going to be 9x9, we knew our AI would only need to keep track of at most 81 moves. This made it simple to design the structure of the tree. The tree that our AI was working with was a 4-level tree with each child node holding 3 more nodes or leaves. We also decided that there wouldn’t be a reason to dynamically change the size of the tree because we were implementing AB Pruning. All we had to do was add values at the end of the tree that would be sure to be pruned. Of course, we also added heuristics to increase the performance.

EXPRESSING OURSELVES

To implement our heuristics, we needed to figure out a quick way of looking at possible patterns on the board. That is where the use of string expressions came in. Thanks to the fact that the board is made up of character strings we could match certain patterns against string expressions. One of the first expressions that we implemented was the 4 in a row. If there were 4 pieces in a row our AI would work with that area first.

CONCLUSION

Throughout this project the biggest challenge that we ran into was wrapping our heads around the AB Pruning algorithm and the way to distribute weights to each place. After getting through these parts it was simple to implement our ideas. However, although we tested the algorithms, we must wait for the tournament to see how good our AI works in a real-world sense.