Artificial Intelligence – Adversarial Game Search

Jacob Ochs
Westmont College
jochs@westmont.edu

INTRODUCTION

The GomokuSearch java class attempts to automatically play an informed game of Gomoku against one opponent. Some of the basic code/game implementation within the GomokuSearch java class has been inspired by/adapted from Dr. Wayne Iba's Racket Gomoku Client and Server which was provided in the deliverable specifications, to whom credit is due. In addition, some of the concepts behind the code was brainstormed in joint collaboration with Chris Betsill and Hunter McGusheon. For our purposes, Gomoku is played between two opponents on a square board, with the goal of the game being to place five of your own pieces in a consecutive order (that is, either horizontally, vertically, or diagonally) while simultaneously preventing your opponent from placing five of their pieces in consecutive order. In this project deliverable, the primary goals in order to play an intelligent game of Gomoku are to implement a search algorithm, AlphaBeta Search in particular, to decide which move has the maximum expected utility in the long term. Along with search, a heuristic function must be implemented to correctly evaluate each potential move and game state, and thus run the search algorithm on these evaluated moves. Thus, the end goal of this project deliverable is to have a fully functional, informed Gomoku-playing agent capable of implementing search and heuristics to result in the most intelligent move.

APPROACH

Initial Approaches and Application

Coming into this project, I had absolutely zero prior experience with the game of Gomoku, even being oblivious as to the structure and rules of the game. In order to learn about Gomoku and to begin to formulate strategies, I took to playing Gomoku online against computers as well as a few of my fellow classmates. This approach was beneficial as it gave me a starting place to begin with the project, as I began to notice patterns and situations which the program would need to be aware of as it plays the game.

After establishing the server-client relationship with both clients successfully connected, my Gomoku program first sends a random move on grid board to the server. Once the first random move is played, the initial evaluating method gets the row and column of this first move and check its immediate surroundings for any combinations of four, three, two and one piece in immediate conjunction in the current game state, respectively. If any of these are present, the method will build on these connecting pieces until there are five. If this row is eventually blocked by the opponent, another random move is given and the process is repeated recursively.

Heuristic and Search

For each of the possible game states described in the above "initial" method, the heuristic function will evaluate each possible move and assign a "weight" to these moves. The weights assigned are integers from 1-100, with a 100 being the best possible move in the given game state (i.e. resulting in a definite win). Once all potential moves have been evaluated and weighted, the search algorithm will go through each potential move, searching out all the possible leafs which this "parent move" may lead to if chosen. As the algorithm searches, it checks the weights and stores the single move with the highest weight value, which is the move to be chosen.

EXPERIMENTS

Throughout the development of this project, I was continually running the program in attempts to play games against the random client given to us for experimentation purposes. This proved to be invaluable in debugging and solving problems which I may have overlooked if I were not able to see them put into immediate action. In addition, once my Gomoku-playing client was capable of defeating the random playing client, I began to test it against a few of my classmates' programs to evaluate my program's strong and weak points in competition. I attempted an implementation of AlphaBeta Search using my heuristic, but was unable to get this useful functionality working correctly; in the end, I commented-out or did not call my search algorithm, as my basic method for educated game movements was able to win more games against the opponent than with my incomplete search algorithm.

RESULTS

Although my resulting product is by no means a completed implementation of this project's specifications, I am convinced that a few minor tweaks will enable the "basic" method to get five in conjunction in vertical and diagonal arrangement as well. In addition, once AlphaBeta Search is implemented correctly, the program will be able to project a more educated move for any arbitrary game state. Surprisingly, though, even though I tested my Gomoku client against other Gomoku clients which were using a more completed implementation of AlphaBeta Search, my program was able to win a few games, depending on the given game state.

CONCLUSION

This project deliverable goes to show the power of search in attempting to play a game against an educated/intelligent opponent. In similarity to the last project deliverable, the correct implementation of AlphaBeta would drastically increase the utility of moves chosen by the program. There is certainly more work to be done in the future, both on my own Gomoku-playing program and in the field of Artificial Intelligence as it relates to games such as Gomoku. In particular, given the recent events of Google's AlphaGo AI program defeating the world champion Go player Lee Sedol, this problem is obviously a relevant and exciting exploration into the nature of Artificial Intelligence.

REFERENCES

[1] Iba, Wayne. *Gomoku Server and Client.*