California Polytechnic State University, Pomona
CS420 Section 1
Artificial Intelligence

Jacob Lepere & Tho Nguyen
Team Name: ALL H**ai**L

Project 3: 4-in-a-line

03/06/2018

Introduction

The goal of this project was to create an AI program for playing a modified connect 4 game to compete against other teams of students in a friendly competition. Other than the requirement to use the MiniMax algorithm with alpha-beta pruning, implementation was very open ended. This report will document the approaches taken by the team to maximize performance on competition day.

Approach

There are several components of MiniMax that can be optimized to maximize game play performance. These include depth limitation, successor ordering, board evaluation and hashing.

First, iterative deepening was chosen as the skeleton for our MiniMax algorithm. For the current board configuration, the algorithm will find the optimal move at max depth 1. If the time limit has yet to be succeeded, the optimal play at max depth 2 will be returned. This continues until the time limit has been reached. When this happens, a play is returned from the search given the current max depth. This value is then discarded because it will not have completed its search and could very likely return a non optimal solution. Therefore, the move from the previous max search depth is taken.

To maximize pruning of the search tree, a 'blooming' function was chosen with respect to the opponents previous move. The ordering of the returned moves is illustrated below, where X represents the opponent's previous play location.

5	4	3	4	5
4	2	1	2	4
3	1	X	1	3
4	2	1	2	4
5	4	3	4	5

The function gives precedence by blooming outwards from the previous play position. Within each bloom level, the locations with the same row or column are returned first, followed by the fill-in values, and finally the locations along the diagonals. This attempts to give precedence to moves that are believed to result in higher evaluation values. This coupled with alpha-beta pruning will lead to a smaller search tree. Furthermore, the possible moves were then sorted based on the heuristics that will be discussed later in this report. The best 7 or so moves were given precedence, while the rest of the moves were left as is.

The heuristic chosen was clean and simple. Rather than devise a complicated and time expensive algorithm to compute the evaluation of each board, we chose to check for three key configurations. Those were:

- Winner/Loser
- Killer move
- One away from a win

Not only are these relatively inexpensive to find, but we can create a simple hierarchy that weights each of the three accordingly. A winner/loser board was given ±100, killer move ± 99 and one away ±98. The values were chosen with the option in mind to include more configurations in the hierarchy, which was not done for efficiency purposes.

Finally, many boards could be reached from different plays during the search. Rather than computing the evaluation again, a hash table was implemented to quickly return already visited board evaluation result. For efficient hashing, we implemented the Zobrist Hash. The Zobrist Hash initiated by assigning a random value to all possible moves that could occur on the board, for both players. Then to calculate the hash, the Zobrist function scanned the board looking for non-empty spaces. At each non-empty space, the assigned value for that piece would be returned and XORed with the current hash value (initially set to 0). Although difficult to explain, this technique was by far the most efficient when compared to the other hash functions we tested.

Conclusion

The approaches chosen for the project were made to maximize our believed chances of winning. Because we were only able to test the algorithm against ourselves or other algorithm designs we implemented, it is hard to judge the true quality of our program. However, we are bound to receive confirmation, whether positive or negative, about our programs quality when we compete against our peers on competition day.

Regardless of the outcome, there are improvements we could have made, or at least tested, given more time. These include symmetry checking to eliminate search redundancies. Symmetry checking was tested, but it ended up slowing down the search. With more time, it is believed that a more efficient symmetry testing technique could have been implemented.

All in all, the project was extremely beneficial to our understanding of the MiniMax algorithm and the different techniques available to maximize efficiency.