Artificial Othello Intelligence Using Minimax Searching

Aaron Whitehead, David Vaughan

CS 482: Artificial Intelligence
Instructor: Richard Kelley
December 18th, 2015

Department of Computer Science and Engineering
University of Nevada Reno

Introduction

Our project was to write an artificial intelligence which can competently play the game Othello (or Reversi). The strategy we used for implementing the AI was to run a minimax algorithm, looking forward multiple levels deep, to pick the expected optimal move. The main point of the minimax algorithm is to not just greedily choose what the best move for the user at any given point, but to actually minimize the maximum move your opponent can make. The goal of the game was to make an artificial intelligence that was "better than randomly guessing" and that seems to have been achieved.

Minimax Explanation

In order to give the Artificial Intelligence something to work on, first an othello engine was built. This was achieved using python, and was done using pretty straight forward board and game classes. By creating a core game loop in which moves from the computer were essentially coming out of a black box, which allowed us to have the black box take in several pieces of data (such as the board state) and spit out a move. Most of the work of this project happened inside of that black box, through the minimax algorithm. The algorithm works thusly. First, a list of all possible moves is created based on the given board. Next, a for loop is done in which all of the moves are performed on board copies. After each move is done, we recursively call on the new boards. This is done up to a given depth value in which we then simply return a heuristic (covered in a later section) score for the board. Then, by comparing each score to every other move, we can decide which move is best.

The implementation of the minimax algorithm we used is strongly related to the depth-first search algorithm we talked about in class. The algorithm recursively searches downward, branch by branch, through a search tree of possible moves and scores. The only difference from a typical DFS is the adversarial nature of the minimax. A variable that can be important in changing the outcome of the game is the depth to which the depth-first search runs. The code being submitted will be coded with a depth of 3, but the call can simply be changed to any other number desired, even though a larger depth with exponentially increase the runtime of the program.

The magic in the minimax algorithm is that we are not choosing to optimize our own score, but to minimize the maximum score of our opponent. This is achieved by simply allowing the board to be scored as if the opponent was playing, multiplying all of the values by -1, and choosing the maximum, ie the closest value to 0. This could have also been done using an absolute value system, although the scores would have had to been normalized above 0.

Heuristic Explanation

So the point of the heuristic function is to "score" each board passed into it. The point of the score is to show which boards are better than other boards. However this is not just a simple mathematical operation, as a heuristic needs to have some sort of logical deduction in order to decide what may or may not be better. In our heuristic board evaluation system, we prioritize three things with different weights. First, we have a simply piece count. If a move is going to cause you to have more tokens on the board than when you started, then it is a good thing. Each piece is worth 1 point. Next, we can say that having a piece in the corner of the board is a very strategic advantage, and each piece in the corner is worth 10 pieces. Finally, we consider the amount of moves that a move allows you to make, or the amount of freedom you allow yourself in the future. Each single move that is open is worth 5 pieces.

Conclusion

So in conclusion, the point of the project was to build an artificial intelligence that is better than random. While we truly think our minimax algorithm works, we can prove that no matter what it is indeed better than random. This can be seen by simply modifying the heuristic function and noting that the outcome of the game is different. This shows that the score is being taken into account when doing the search, meaning that it is not just randomly choosing moves. When you leave the heuristic function alone, and run the game multiple times, it will have the exact same outcome.