Artificial Intelligence
Othello Project – Iago

Implementation

Iago is an Othello playing program written in C++ which uses Alpha Beta Pruning combined with a depth first search to decide on the optimal move given a board state. The data structure which I used to describe a board state is the Board class. This class stores information about the state such as a multidimensional array that describes the board positions, the player whose turn it is, an array of the player's legal moves, black's score, white's score, the Heuristic and Search options, the opponent's previous move, an array that represents the best path of the previous search, and various other options and flags.

Most of the computation in this program is done within the member functions of the Board class. This is convenient because all of the variables which need to be accessed are already within the scope of the functions. The Square class represents an action or legal move. In addition to position, the Square class also stores the pieces which are flipped if that action is taken in the current Board state. The search was implemented with two functions, TreeSearch() and AlphaBetax(). TreeSearch generates the depth 1 states and calls the AlphaBetax function with the minimizing option for each state. TreeSearch then sorts the moves by utility and the move with the highest utility is chosen. The UtilitySort function also takes care of randomizing moves with the same utility. Because the search uses iterative deepening, AlphaBetax is able to pass the path to the best terminal node up the tree in much the same way as the utility. This path is then used to traverse the tree in the next search to depth+1 and hence prune more nodes from the tree. This strategy increases the speed of a search by 2 to 10 times and will occasionally search 1 ply deeper than regular AlphaBeta.

The Heuristic is zero sum and takes into account 6 metrics: board position, mobility, corner occupancy, corner closeness, parity and stability. The weights to these metrics change based on the turn number so that opening strategy, middle game, and endgame are distinct. Board position assigns a value to each square and adds that value to a player's score; mobility will attempt to minimize the opponent's legal moves; corner occupancy is self-explanatory; corner closeness prevents the player from playing next to an empty corner; parity keeps track of who will have the last move if the game is played until the board is filled; lastly, stability classifies disks into the categories of stable, semi-stable, and unstable and weights the disks accordingly.

Features

To play a game, compile with make (preferably on a linux machine) then run the executable with the command "./iago". This will start a game with the default options (5 second time limit player vs computer) . Options are specified at runtime and in any order. With the options, you can choose pvp "player vs player", cvc "computer vs computer", pvc "player(black) vs computer(white)", or cvp "computer(black) vs player(white)". 'r' can also replace 'c' or 'p' for a random AI. You can also specify the time limitation for the computer (in seconds). Lastly you can load a board with the format provided on the class website and optionally specify a time limit on the last line of the file. I have provided a sample file BoardLoad.txt for reference.

Examples:
```
./iago pvc 10              // player is black computer is white computer takes 10s to move
./iago cvp BoardLoad.txt   //computer(black) vs player(white)  load BoardLoad.txt
./iago cvr 0.01            // computer(black) vs randomAI(white)  0.01s to move
```