CS 165A MP2 Report

Andrew Doan
3772365

Implementation

The minimax function is implemented by returning the move gotten from a call to the alphabeta function with the parameters: 4, -infinity, infinity, True, True, [0,0,0], horizontal, vertical, self.score_player2, self.score_player1. Depth 4 is being used even though depth 2 is just as effective against the test AI since the increased depth will likely help against smarter opponents. The next_possible_testMoves function is exactly the same as the given next_possible_moves function, but allows a vertical and horizontal state to be passed in so that the function only simulates a move, and does not actually affect the board state. The alphabeta function first checks if a move is terminal and there are no possible moves available, or that the depth is 0 and returns the score and move in a tuple. The returned score is used in my evaluation function to determine which move is the best move, as each call of alphabeta calls alphabeta on all possible moves for the current state and compares them to find either the highest or lowest score. The alphabeta function also keeps tracks of alphas and betas to prune if alpha>=beta. The evaluation strategy was implemented in alphabeta, but has been implemented abstractly in the evaluation function to allow the grader to easily understand it.

Evaluation Strategy in Minimax

The evaluation strategy is implemented in the alphabeta function, but an abstracted version of the strategy is implemented in the evaluate function definition even though it isn't used for the grader's convenience. Essentially, the alphabeta function where the heavy lifting of the move calculation is done returns the move done and the player2 score(me)-player1 score if that move is taken. When deciding which move to take, it takes the move with the highest score on player2's turn, and the lowest score on player1's turn.

Alpha Beta Pruning Parameter

Beta prunings occur when the beta values are 0, -1, -2, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, and 21 and the alpha values are 0, -1, 1, -2, 1, 2, 3, 4, 5, 6, 7, 8 ,9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, and 21. No alpha prunings were made. These values are also printed out in the terminal.

Time Taken By Algorithm to Make Move

With printing I found depth 2 to be the best performing depth since the moves are fast and only take about 2-3 seconds and the performance against the AI is acceptable, winning 8-28. Increasing depth increases the time by a little, with a depth of 3 taking around 4 seconds and winning with the same score, and by a depth of 4, the program takes around 5 seconds per turn, and wins with the same score as depth 2. After depth 4, the program takes exponentially long to run at depth 5, with around 50 seconds - 1 minute being used per turn with minimal performance gains against the test AI.