Project 3

Four in a Line

CS 420 - 01

Kristin Adachi

knadachi@cpp.edu

September 1, 2017

Project Specification

This project focused on using the alpha beta pruning algorithm to create an AI for a four in a line game. We had to create an 8x8 board where the player and the AI take turns placing a piece on any space. This process continues until either the player or the AI gets four of their pieces in a row. The user is also required to specify how long the AI is allowed to run its algorithm to search for and determine its next move. The game ends when a winner is found or if all spaces on the board are taken.

My Approach

I implemented my program using five different classes: `Board`, `Action`, `AlphaBeta`, `UI`, and `FourInALine`. A board is represented by an 8x8 two dimensional array where player pieces ("O") and AI pieces ("X") are placed on its empty spaces ("-"). The `Action` class is used to represent an individual action, which is a letter (A-H) followed by a number (1-8). The alpha beta pruning algorithm is implemented in my `AlphaBeta` class. Finally, the `UI` class handles all the user interface for the program and the `FourInALine` class contains the main method.

Implementation of the Alpha Beta Pruning Algorithm

As mentioned earlier, my `AlphaBeta` class handles my implementation of the alpha beta pruning algorithm. Because we were required to limit how long the algorithm could run, I also utilized iterative deepening search. As a result, this class keeps track of alpha, beta, and depth. Alpha and beta represent the current worst and best values respectively and are updated throughout the program. The depth is initialized as a very big number because the algorithm is allowed to continue searching for as long as the time the player permitted.

Alpha and beta values are updated by traversing the board and finding the minimum and maximum values. This continues until the cutoff test is satisfied. I implemented the cutoff test by keeping track of when the algorithm began searching. My `cutoff()` method is constantly checked and returns true when the algorithm's time is up. Because I didn't use a thread to implement the cutoff test, the accuracy of the timer is slightly off.

When the cutoff test is satisfied, the algorithm proceeds to use an evaluation function to update the “scores” on the board. These scores determine the potential of each empty space and helps determine which space the AI should place their piece. Each possible move for the AI is represented by a score that is found by checking three spaces away from the empty space and calculating its potential. Immediate diagonals are also checked, but this is mostly intended to break any possible score ties. For the AI’s defense, I implemented it to think in the player’s perspective. This means that it would look for a player win and essentially becomes a very annoying blocker. After the scores for each state are determined and compared, the alpha beta pruning algorithm returns the AI’s final decision.