Andrew Hale, Patrick Wilson, Spencer Seeger

BYU CS 470 - Artificial Intelligence

Dr. Crandall

Reversi Lab

Pruning Function

When starting to write the AI, the code was simply a minimax algorithm with no alpha-beta pruning. Without the pruning, it took about 30 seconds to calculate down to a depth of 5. Anything more than 5 took too long for the AI to play in a reasonable time frame. The pruning function takes into account the best scores that are available at each state of the game at each depth. If the score of any particular branch is better than any other given branch, the AI simply chooses that branch and will not search down any more trees that are down the worse path. After implementing this, we were able to travel to a depth of 10 without having an issue with time.

Evaluation Function

Our evaluation function took into account overall score, numbers of moves per player, and weighted scores of each square. Currently, our evaluation wins about 90% of the time against a Random player. The reason that it does not win all of the time is because of the randomness involved. The minimax theory assumes that the opponent is going to play rationally. However, when an opponent plays randomly, it limits the effectiveness of a minimax algorithm.

Initially, the evaluation function only took into account a way to maximize pieces that any given player had on the board. However, this was found to not be a good estimator of who would eventually win at the end of the game. This lead us to finding better heuristics. These included checking how much each state would allow a certain player to move, as well as

weighted scores of each square. Overall, this helped our AI win more consistently against a random opponent.

In the future, we wish to be able to implement a fast method of counting “stable” squares on the board. Stable squares are squares that can no longer be changed regardless of any placement of future pieces. To do this currently, it would take far too long to calculate for each state. Thus, this function is left out for now, but if are able to find a faster way to calculate, it will be reimplemented.

Time Report

Andrew - 11 hours.

Patrick - 3 hours.

Spencer - IDK, too many

Suggestions to improve lab

It would be nice to have a rough outline or pseudocode for a generic minimax algorithm with alpha-beta pruning, as well as an example heuristic/evaluation function. It would not have to be for this game, but a high level example implementation would be very useful.

Algorithm Review

- Creative Work - improve or hurt the performance of your algorithm
    - We assigned values to each space on the board. After a square was taken on the board, values were changed in the table to better represent the value of each board position. We also attempted to come up with a way to value amount of possible moves. This allowed the AI to determine how good particular spots were at any given time in the game.

- Did giving your algorithm more time to move improve its abilities?
    - Giving it a larger depth to search (and with that more time) did improve its performance against random.
- How deep in the tree could you search within X seconds?
    - With a depth of 7, it never takes longer than about 15 seconds. Anything more than that, and the AI starts taking too much time to make a move.
- Does your computer code beat you?
    - Yes, for the most part: it is rather difficult to beat.
- Videos
    - We have multiple videos made of the game, each pitting our AI against different players. The results varied, but can be seen in each specific video below.
        - Dr. Crandall's Easy AI versus our AI (our AI is the white player): https://drive.google.com/open?id=1soT3L8N1HabMYLujVDJhh37smvU7IzU0
        - A Human versus our AI (our AI is the white player): https://drive.google.com/open?id=1TUv4BhL3l7Vfwke9nTMmX-E8lGMB42un
        - A Random player versus our AI (our AI is the white player): https://drive.google.com/open?id=17KzDHINqx3r231fip5WRcYf0XX-vcNkY