Sierra Schwellenbach

7799323

MP2 – Reversi AI

Architecture: My code is split up into 3 different classes: Reversi.java, ReversiBoard.java, and ReversiPlayer.java. Each of these have their own defined behavior. Reversi.java contains the main function and handles input, printing, and flow of the game. It also controls whose turn it is, validates moves, etc. ReversiPlayer.java contains the AI to play Reversi which is called on the computers turn. It has a function which launches minimax and handles the result. This is where most of the algorithms and complexity occurs including minimax, alpha beta pruning, etc…. Finally, ReversiBoard.java does all of the "housekeeping" and helper functions to do with the board. It prints the board and maintains the state of the board. It also contains several functions which are called throughout the program such as getScore() and getFillFactor(). Most notably (at least as far as the AI component), this class contains my function which calculates the relative value of the board which the minimax relies on.

Search: My search algorithm is a basic minimax algorithm of depth 5 with Alpha-Beta pruning. The algorithm follows the standard model of the minimax and Alpha-Beta ideas and only differs from them in a few implementation details and minor optimization decisions I made. Where I really expanded on the basis of this algorithm was with my evaluation function. The heuristic I used to evaluate the board added emphasis to the corner slots. In addition, it adds weight to edges which are adjacent to corners already taken by the player. In this way, it gives extra weight to the edges which are "unflippable" but does not give excess weight to those edges which can be flipped back over. I weight every corner as 4 times the value of a normal piece and every edge that is unflappable as 2 times as valuable. If the player takes every single edge on a side, the algorithm will weight each of the edge pieces as 3 times as valuable. This takes into account the fact that having an entirely unflappable edge is extremely powerful.

Challenges: A lot of the challenges that I faced in this project were to do with getting minimax and alpha-beta pruning to work correctly. I understood how to do them in theory, but when I coded them there were a lot of little bugs which were difficult

to find. Also, this problem is inherently difficult to test as it is hard to tell if your algorithm is working 100% correctly or if it is simply getting lucky or working mostly right. To overcome this challenge, I played my algorithm against a deterministic opponent to allow me to see my improvement and on several different board sizes to give me multiple data points. Also, I played my AI myself a lot and, having a decent amount of experience with playing Reversi, I was able to recognize when it made a really stupid move and track down why.

Weaknesses: One of the biggest weaknesses of my algorithm is that it is of fixed depth 5. This works well and I've had a reasonable amount of success with it (especially with added heuristics), but I would have really liked to add at least iterative deepening to allow the AI to take full advantage of its allotted time. My program would also benefit from some additional tuning to the evaluation function. There are several strategies which I notice myself using when I play Reversi which I believe my AI would benefit from using, but I did not have time to implement all of them.