Ayden Shankman
ECE-469
Prof Sable
Fall 2022

Othello AI Using Alpha-Beta Pruning

In this program I implemented minimax search with alpha-beta pruning in order to create an AI that can play Othello relatively well. I wrote all of the code in Golang due to its efficiency and intuitive syntax. The basic structure of the code includes 3 main data types. The first is a board, which contains information such as the location of each player's pieces as well as who's turn it is. The second is a player, which contains its own color and the opponent's color, if it is a computer or not, and the type of evaluation function it uses if it is a computer. The last type is a game, which stores the current board, each of the players, and the time limit for the computer to make a decision.

For the computer's search function, I combined iterative deepening and alpha-beta pruning. To make sure the search didn't surpass the allotted time limit for a turn, I implemented timers in the iterative deepening loop. During a loop, I calculated the time it took to do a search for one possible move (out of all possible moves for the current player) up to the current iterative deepening depth and multiplied it by the amount of moves it has left to calculate. For example, if the iterative deepening is up to max depth of 3, the user has 4 possible moves, it took 0.5 seconds to calculate the score of the first move, and the max time limit is 5 seconds, then I calculate (0.5)*(3 remaining moves) and check if it surpasses the time limit or not. If it does, I cancel the current search and return the best moves calculated from the last completed depth.

To evaluate the score for a given player given a board, I tried 2 methods. The first method was a static weights heuristic function that simply checked the spaces where a player's pieces were and added up the associated heuristic values (see Figure 1). I assigned very high weights to the four corners and very low weights to the spaces surrounding the corners and edges. This strategy worked fine against a low level player such as myself, but when I tried it against a computer like the one found at https://hewgill.com/othello/ or Professor Sables Othello AI it would lose.

The next evaluation method I tried was a dynamic heuristic evaluation function. To achieve this, I would determine the state of the game (early, middle or late) and change what should be weighted more. There were four evaluation functions used to calculate the total evaluation score: corner score (amount of corners player has compared to opponent), around-empty-corner score (amount of pieces player has around empty corners compared to opponent), mobility score (amount of moves player has compared to opponent), and piece-difference score (amount of total pieces player has compared to opponent). In the early game (<20 total pieces) I weighed the mobility and corner score heavily and the piece-difference score negatively. In the middle game (<50 total pieces) it was basically the same except I didn't take into account the piece difference score.. Finally, in the end game, I weighed the corner and piece difference score heavily and slightly lowered the mobility score weight. The final weights for each stage of the game can be seen in Figure 2. I tested the static evaluation against the dynamic one and the dynamic function consistently won. I then tested the dynamic function against the website mentioned earlier and it won at every difficulty. Finally, I tested it against Professor Sables AI and mine won whether it went first or second.

200	-30	30	10	10	30	-30	200
-30	-50	-10	-10	-10	-10	-50	-30
30	-10	20	5	5	20	-10	30
10	-10	5	5	5	5	-10	10
10	-10	5	5	5	5	-10	10
30	-10	20	5	5	20	-10	30
-30	-50	-10	-10	-10	-10	-50	-30
200	-30	30	10	10	30	-30	200

Figure 1: Static weights assigned to each position for the static evaluation function. Corners associated with the corner score evaluation function (green). Spaces that border corner squares used in the around-empty-corner evaluation function (red).

a) (20 * corner score) + (5 * around empty corner score) + (6 * mobility score) + (-1 * piece difference score)
b) (20 * corner score) + (5 * around empty corner score) + (6 * mobility score) + (0 * piece difference score)
c) (20 * corner score) + (5 * around empty corner score) + (4 * mobility score) + (10 * piece difference score

Figure 2: Evaluation scores during early game (a), middle game (b), and late game (c).