CS6380 Artificial Intelligence Assignment 2

Vedant Saboo, CS19B074

November 28, 2021

Abstract

Primarily as a report, for the Assignment 2 in the course CS6380 Artificial Intelligence, on a bot formation for the game of Othello (AKA reversi) on a 8 by 8 board, with a move timer of 2 seconds. This document includes the briefs of the algorithm used for choosing the move for the bot. I have used a typical design for the alpha beta pruning, with an additional pruning based on time of execution (for safety purpose), and a specially designed evaluation function for horizon nodes, the details of which we will see further in the paper.
The bot will be used in the Desdemona framework. We will demonstrate the performance of this bot against the random bot already provided.

1 Problem Statement

Othello Game Playing Bot: Design an AI bot for Othello game, in 8 by 8 set up.
A bot being better relatively means doing better in multiple duals overall. We will consider the performance of the bot, however, against the random bot, although the intended purpose of this bot will be to use in the round robin competition - a part of the assignment.

2 Algorithm Implementation and Development

The Algorithm is implemented basically as a **alpha beta pruning** framework, with added time controlled pruning. Since bots are severely penalised for exceeding the time limits (2 seconds), we, as a precautionary measure mark a cutoff time limit per move as 1.8 seconds. Once this mark is reached, all further nodes, not traversed, will be pruned. This can lead to wrong decisions, but this is a required precaution.

2.1 Game tree

For each move, we construct a minmax game tree, nodes of which correspond to board configurations, either the node being min node or max node. **alpha beta pruning** algorithm is used to traverse the tree and finding the best node. The depth of the tree is interesting - it has a **variable depth** based on the progress of the game. In the opening and mid-game stages, we have a low depth of 5, while in the end-game stage, we have the depth of 8. This variation is there because in mid-game stages, there can be a lot of moves and thus a huge game tree - which might not be possible to explore it all in mere two seconds. In the end game stages, however, moves possible are limited, thus high depth can be afforded. Besides, it is this period when we can be more sure that the evaluation function makes a good evaluation of horizon nodes - since they are closer to the finished game configuration.

2.2 Evaluation Function

Evaluation function we use is a hybrid and complex one. It has two different functions based on progress - while in the most of the game, the function `getHScore()` is used, function `endGameScore()` is used for the end-game stage.
`getHScore()` consists of weighted additions of the following evaluation schemes:

	Black	Red	Result
1	47	17	Win
2	50	14	Win
3	44	20	Win
4	53	11	Win
5	51	13	Win
6	55	9	Win
7	51	13	Win
8	50	14	Win
9	49	15	Win
10	48	16	Win
11	44	20	Win
12	52	12	Win
13	48	16	Win
14	50	14	Win
15	59	5	Win
16	42	22	Win
17	53	11	Win
18	57	7	Win
19	51	13	Win
20	51	13	Win

Table 1: The Bot's performance as Black against random bot - 100 percent success on 20 runs

1. **Coin positions**: Specific squares are more important than others, and several squares should be avoided if they can. This portion is low on weightage.
2. **Corner possession**: Possessing corner gives a lot of control over most of the squares, and once possessed, it can't be undone. Hence it is the most important and highly weighted benchmark.
3. **Corner accessing**: squares right next to the corner are dangerous, as they may give the opponent the access to that corner. Hence, we try to avoid them by giving them negative weights.
4. **Front squares**: squares flanked by the corners completely cannot be captured back. These are called front squares. We try to possess as many of those.
5. **Mobility**: the number of moves available to the player to play is a good measure, because if the opponent has fewer moves to choose from, they will be forced to make a bad choice. We give this a high weightage.

`endGameScore()` uses disk count as the only measure to compare board configurations.

3 Test Results

See Table 1 to refer to the results obtained while our bot played as Black against the Random Bot. See Table 2 to refer to the results obtained while our bot played as Red against the Random Bot.
We can very well see that the bot has a consistently good performance against the random bot. For 20 runs in each color, it has registered 100 percent wins, and all with a high margin.

	Black	Red	Result
1	10	54	Win
2	11	53	Win
3	7	57	Win
4	8	56	Win
5	7	57	Win
6	4	60	Win
7	6	58	Win
8	12	52	Win
9	13	51	Win
10	16	48	Win
11	13	51	Win
12	10	54	Win
13	7	57	Win
14	7	57	Win
15	10	54	Win
16	11	53	Win
17	5	59	Win
18	12	52	Win
19	4	60	Win
20	10	54	Win

Table 2: The Bot's performance as Red against random bot - 100 percent success on 20 runs