IT UNIVERSITY OF CPH

IT University of Copenhagen

Othello Bot Documentation

Introduction to Artificial Intelligence

Group Bob

2023

Introduction

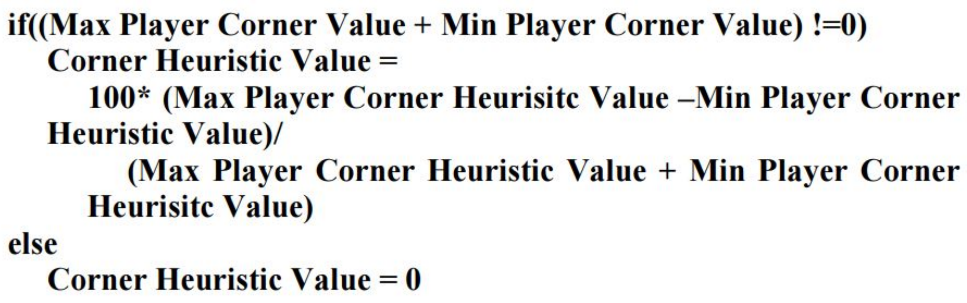
This report is written for the *Othello Project* from the course *Introduction to Artificial Intelligence* at The IT University Of Copenhagen. In this report, we will explain how our solution to the project is constructed with a main focus on the cut-off and evaluation function, as well as the search algorithm used to decide the best move.

Evaluation and Cut-off functions

The evaluation function uses three different heuristics to determine the best move. These heuristics are inspired by *An Analysis of Heuristics in Othello*, by *Sannidhanam and Annamalai (2015)*

Corners

The Corners heuristic prioritizes capturing the corners of the board as the name implies. It outputs a point value between -100-100 where 100 is the best move. It gives a corner value based on current corners captured and possible corners that could be captured this turn. It then compares the two players corner values with the formula in figure 1.1 from Sannidhanam and Annamalai (2015)



```
if((Max Player Corner Value + Min Player Corner Value) !=0)
    Corner Heuristic Value =
        100* (Max Player Corner Heurisitc Value –Min Player Corner
    Heuristic Value)/
            (Max Player Corner Heuristic Value + Min Player Corner
        Heurisitc Value)
else
    Corner Heuristic Value = 0
```

Figure 1.1: Corner Heuristic

This formula makes the heuristic prioritize having more corners relative to the opponent.

Moves

The Moves heuristic prioritises giving the opponent as few legal moves as possible. It does this by counting the legal moves the opponent has in a given state, then returning the negative of the count. The evaluation function takes this value and determines if it is a good move or not. The higher the value (closer to zero), the better. Thus, the more legal moves the opponent has, the bigger the negative number becomes, and the move is evaluated as worse than another move returning a higher value.

Tokens

The Tokens heuristic counts the number of tokens that each player has in the current game state and returns a value that depends on the difference in amount of tokens between the players. More specifically it returns the amount of tokens that the AI has subtracted by it's opponents tokens and then that is divided by the total amount of tokens on the board times 100: $(playerTokens - opponentTokens)/(playerTokens + opponentTokens) * 100$.

Combining heuristics

The above heuristics by themselves are somewhat effective, but where they really shine is in conjunction with each other. Our strategy for combining them is to use one heuristic, and if it results in the same value for two distinct actions, we apply the next heuristic. Specifically, we achieve this by using the fact that they all return a value in the range -100 to 100, so multiplying the first heuristic by 100 and adding the second heuristic gives us a new heuristic that is a combination of the two. In the case of combining three heuristics, we multiply the first heuristic by 10000, add the multiplication of the second heuristic by 100, and finally add the third heuristic.

Search Algorithm

Our bot uses a mini-max algorithm using alpha-beta pruning. This means that our bot starts at the root, and then recursively checks the different legal moves that exist in that given state. For choosing the max-value, it checks all the legal moves in a given state and gets the maximum utility found at each legal move. The same is true for the min-value function. If the algorithm at any given state, finds a utility that is less than α or greater than β, it will cut off the rest of the tree for that node, and not explore further, since it has found this path obsolete.

For the sake of getting different outcomes for our games, and to improve testing, we have decided to shuffle all the legal moves as the first thing each time our bot decides on a move. In other words, when we are at the root of our mini-max tree, we shuffle all the children, before going recurs down the tree. This makes it choose different moves between the ones with the same min-max value.

Notes on the code submission

Our AI is set to a cut-off depth of 10, which runs in less than 10 seconds per move for the majority of moves on a size 8 board. This is of course very dependent on the machine running the AI, so if something else is observed, we recommend just lowering it. We have put all our code in a single file that *should* just be drag and drop to test against other AIs. That file is OthelloAIBob.java, which contains

- Our AI class that just extends one of our AIs mentioned in the next bullet-point.
- A bunch of heuristics and AIs that use them. These are the AIs that we have tested against each other and we have just included them for completeness, they are not used.
- A BaseAI class that contains our actual implementation of the algorithm.
- A BetterGameState class, which is a copy of the GameState class with some modifications. Modifying GameState was discouraged, so our AI still implements the IOthelloAI interface and just converts it to a BetterGameState internally. It does two things better: legalMoves is an iterator meaning that checking if any exist is much faster because we don't need to materialize them all; and countTokens is calculated when we insert tokens.

Heuristic comparison

We have tested a selection of AIs/heuristics against each other—they are all included in our code submission. Our testing methodology is to battle the two 50 times and the result is then $blackWins/(blackWins + whiteWins)$ expressed as a percentage.

Because some heuristics are faster to compute than others, we ran them to different depths. In this case the cut-off depths we set as follows:

- Tokens: 8
- Corners: 7
- Moves: 6
- CornersTokens: 7
- CornersTokensMoves: 6
- CornersMovesTokens: 6

These could easily play single games at higher cut-off depths, but given that we do not have the budget to rent a server farm for a weekend, this will have to do for the benchmark. We'll just keep in mind that there's quite a high margin of error.

Black \White	DumAI	RandomAI	Tokens	Corners	Moves	Corners-Tokens	Corners-Tokens-Moves	Corners-Moves-Tokens
DumAI	100%	67%	10%	2%	33%	0%	0%	0%
RandomAI	30%	43%	10%	8%	16%	0%	0%	4%
Tokens	86%	85%	47%	14%	60%	10%	2%	0%
Corners	94%	90%	72%	52%	72%	17%	21%	4%
Moves	73%	72%	40%	22%	44%	16%	2%	4%
Corners-Tokens	100%	100%	90%	64%	92%	67%	47%	22%
Corners-Tokens-Moves	96%	100%	96%	82%	92%	63%	26%	31%
Corners-Moves-Tokens	98%	100%	92%	96%	84%	67%	65%	38%

Table 1.1: Down the left column is the black players and across the top row are the white players.

Some quick observations here is that

1. All heuristics beat DumAI
2. Corners is really good on its own.
3. Adding Tokens to Corners performs incredibly well.
4. Adding Moves on top is a little better.
5. Flipping those last two heuristics gives us CornersMovesTokens which beats everything.

To evaluate the importance of our cut-off depth, we also battled CornersMovesTokens with a cut-off of 6 against itself with a cut-off of 8. The result was that CornersMovesTokens-6 won 29% of the games when playing white, and 42% of the games when playing black.

References

Sannidhanam, Vaishnavi and Muthukaruppan Annamalai (2015). *An Analysis of Heuristics in Othello*. Available at https://courses.cs.washington.edu/courses/cse573/04au/Project/mini1/RUSSIA/Final_Paper.pdf. Accessed: 24. March 2023.