Artificial Intelligence

Practice 2. Games

Jorge González Gómez

Pablo Almarza Marques

Part 1. Minimax + alpha-beta pruning

a. **Implementation details**
   i. **Which tests have been designed and applied to determine whether the implementation is correct?**

We checked the implementation with demo_tictactoe.py and demo_reversi.py, and we played this minimax with our heuristic on the reversi. In verbose mode, we saw that the values shown in the console made sense and, using our heuristic (we fiddled a bit with the .py files to use our own heuristic), our AI was able to beat us.

   ii. **Design: Data structures selected, functional decomposition, etc.**

We selected the same structures as MinimaxStrategy in order to have something to start working, as proposed by the practice statement. Then we just added the pruning part, that is, checking alpha and beta values and modifying them if necessary, following the pseudocode in the PDF provided by the teachers.

   iii. **Implementation**

We implemented the alpha-beta pruning with the pseudocode provided in the PDF, so, in the maximizing player, we check if the value of the successor is greater than beta, and if it is, we return the successor value, and if not, we compare alpha and the value and assign the max between both to alpha.
In the case of the minimizing player, we just reversed the roles of alpha and beta.

   iv. **Other relevant information**

We believed this practice was trivial since all we had to do was implement the proposed pseudocode step by step and, thus, we feel we did not have to give more implementation details than that in the previous points to not flood the PDF with meaningless proof for something we did not really build ourselves.
Both of us understand the algorithm and why Alpha-Beta pruning is faster than a regular minimax algorithm, which is what the practice is meant to teach us: learn the difference between an alpha-beta pruning minimax and a regular minimax (which we did without the need of further, useless details, really).

b. **Efficiency of alpha-beta pruning**

i. **Complete description of the evaluation protocol.**

To evaluate the efficiency of the alpha-beta pruning algorithm we considered the option of using “timeit” but we ended up using the old-school method of calling Python’s **`time.time()`** and doing a for loop in the file **`tournament_timing.py`** to run different tournaments.

To switch between different strategies we would have needed to modify the code of **`tournament.py`** or just implement our own, special tournament class which would have meant a lot of wasted time for no reason (and, as it is, we do not have much time to finish other subjects assignments and study for the partial exams at the same time). If we were to had done it, we would have just added a boolean in tournament so the “strategy” variable would be picked through a simple bool

For the time beign, we just changed the commented code back and forth to switch between variables. This worked for us, so if it is not broken why would we bother to fix it.

On the topic of which methods we used, the Heuristic1 as proposed in the practice, and we later used our own heuristic, using an 8x8 Reversi board in both cases (we had to comment and uncomment code here and there in **`tournament_timing.py`**).

Please note: the CPU used is a standard AMD Ryzen 5600X @ 4.6GHz, using the PyPy3 interpreter. Times may vary across computers and interpreters.

ii. Tables in which times with and without pruning are reported

Running it 6 times for each strategy, for depth 3 and depth 4, for the dummy Heuristic1

Depth 3	MinimaxAlphaBetaStrategy	MinimaxStrategy
Total time taken	67.71	142.68
Average time	11.28	23.78
Best	10.38	21.69
Worst	15.35	27.84
Depth 4	MinimaxAlphaBetaStrategy	MinimaxStrategy
Total time taken	151.44	764.28
Average time	25.24	127.38
Best	24.89	125.96
Worst	26.46	128.09

Running it 6 times for each strategy, for depth 3 and depth 4, for our heuristic

Depth 3	MinimaxAlphaBetaStrategy	MinimaxStrategy
Total time taken	518.31	1276.25
Average time	86.39	212.71
Best	82.79	208.34
Worst	101.83	215.44
Depth 4	MinimaxAlphaBetaStrategy	MinimaxStrategy
Total time taken	6464.43	63733.69
Average time	1077.40	10622.28
Best	1063.02	10450.33
Worst	1124.53	11148.54

iii. Computer independent measures of improvement.

These are the improvements based on percentages of time taken to execute the scripts:

	Regular Minimax	Minimax with Alpha Beta	Speed improvement
Depth 3, Trivial Heuristic	142.68	67.71	x2.15 faster
Depth 4, Trivial Heuristic	[redacted]	[redacted]	x5.04 faster
Depth 3, Our own Heuristic	1276.25	518.31	x2.46 faster
Depth 4, Our own Heuristic	63733.69	[redacted]	x9.86 faster

iv. Correct, clear, and complete analysis of the results.

We clearly see impressive improvements when using alpha-beta pruning when measuring the total times taken, specially when using our own heuristic: an improvement of up to 10 times faster over having to wait 17.7 hours for the regular minimax to complete 6 iterations.

Based on the data in the table in point iii., we can conclude there’s a bigger weight on the time taken based on the heuristic than on the strategy itself, but, either way, we also learned using alpha-beta pruning over a regular AI strategy without any pruning improves the execution time humongously for these cases where the heuristic takes a long time.

So, all in all, the results prove that a good heuristic is key if you want to do an AI based on heuristics but, if you can’t help it, using alpha-beta pruning may improve the performance of the AI algorithm from roughly x2 up to x10 times.

Part 2. Heuristic designs.

a. **Review of previous work on Reversi strategies, including references in APA format.**

Before starting thinking about the heuristic and how we will be coding it, we played against each other a couple of games in order to know which moves were good and which were bad, and we found that everytime you got a corner, the probability of to beat the opponent increased drastically, so the corners were crucial in this game. Then we researched information about the game and some mechanics and we found a report[1] , that talked about the basics of the game and some strategies that a player could follow. One of them was about giving weights to the squares, so the corners had the maximum weight, the squares surrounding the corner had the least weight (negative weight) and so on, that inspired us to do our first heuristic, a simple one but effective. Since that we just saw several strategies such as a greedy one (move where you get the most number of pieces) or even one where you tried to leave the opponent without moves.

b. Description of the design process:

i. How was the design process planned and realized?
We planned to make a simple heuristic, not too difficult to code based on the weights and then start from there. Then we tried to modify the weights, revert them, base the move we do depending on the moves will be available for the opponent or try to give the opponent so many moves to make him lose by timeout.

ii. Did you have a systematic procedure to evaluate the heuristics designed?
The way that we evaluated the heuristic was just playing them against each other and against the two other heuristics that were given.

iii. Did you take advantage of strategies developed by others? If these are publicly available, provide references in APA format; otherwise, include the name of the person who provided the information and give proper credit for the contribution as “private communication”.
We found a GitHub repository[2] where there were some strategies and one that we were finding that consisted of leaving the other player without moves.

c. Description of the final heuristic submitted.
Our final heuristic consists in leaving the opponent without moves, so if we achieve that it will mean that we will be continually making moves and so we will be able to take the squares in which our opponent is. Therefore our algorithm will win since it will “play alone”.

[1]Connelly, D. H. (2013, September 15). othello.py. dhconnelly’s GitHub page.
http://dhconnelly.com/paip-python/docs/paip/othello.html#section-63
[2]Arminkz, Reversi, (2018), GitHub repository,
https://github.com/arminkz/Reversi