ASSIGNMENT 2

REVERSI

Miguel Álvarez Valiente & Rodrigo Juez

Escuela Politécnica Superior

Computer Science Engineering

1. Documentation of minimax + alpha-beta pruning

a. Implementation details

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

There is an option to set verbose=2, but the output is so large that we found it really difficult to read it, that is why we used tools to count the lines and we saw that roughly it was expanding half of the nodes. We also run different tests from different starting states, and checked that the results were the same with and without pruning.

We also run the tournament with and without pruning and made a diff between the two outputs (the command is from PowerShell because we're using Windows but it works the same as *diff*):

```
PS C:\Users\Rodrigo\Desktop\IA_2-master> compare-object (get-content .\proof\minimax.txt) (get-content .\proof\pruning.txt)

InputObject                  SideIndicator
-----------                  -------------
Finished in  4.633665 seconds =>
Finished in  6.575271 seconds <=
```

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

We based our pruning from the Minimax algorithm already given to us, we didn't use any additional data structure. The functional decomposition is the same, *max_value*(), which works the same as minimax but pruning, *min_value*(), and next_move which is a slightly modified *max_value*() so that it returns the node and not the integer.

iii. Implementation.

As mentioned previously the strategy is implemented using three functions:

- _max_value(): first checks if the state received is a terminal node (in which case it returns as minimax_value the result of evaluating it with the corresponding heuristic). If it is not a terminal node it iterates through its successors, performing pruning when the minimax_value is higher than beta. The implementation is the same as the pseudocode given to us.

- _min_value(): does the same chack as _max_value() to see if the state is a terminal node, and if it is not it also iterates through the successors of the state, but instead of looking for the maximum value, it looks for the smallest return of _max_value() form all the successors. It also does pruning but instead of checking beta it checks alpha.
- next_state(): The function of minimax only calls min_value for all the nodes, but we implemented a very similar code as in *max_value()*, instead of returning the value, we return the node.

iv. Other relevant information.

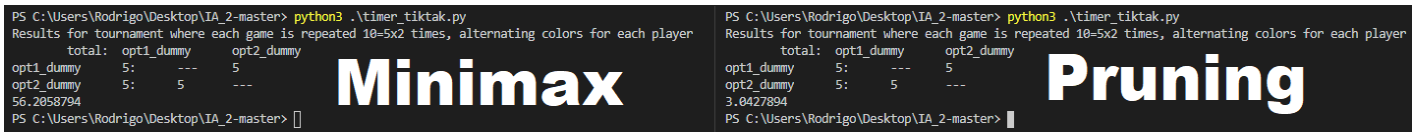
The pseudocode is not exactly the same as the photo, as it returns the max_value function directly and our implementation has a different function called next_move, but it works the same.

b. Efficiency of alpha-beta pruning.

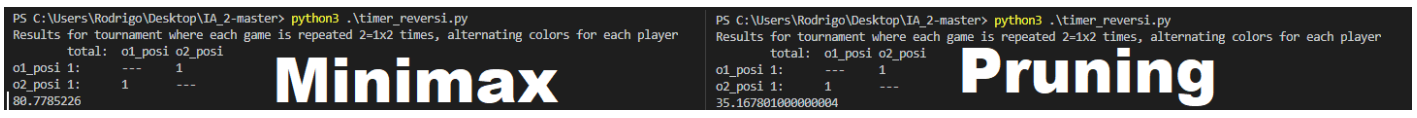
Complete description of the evaluation protocol.

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

Tictactoe tournament:



Reversi tournament:



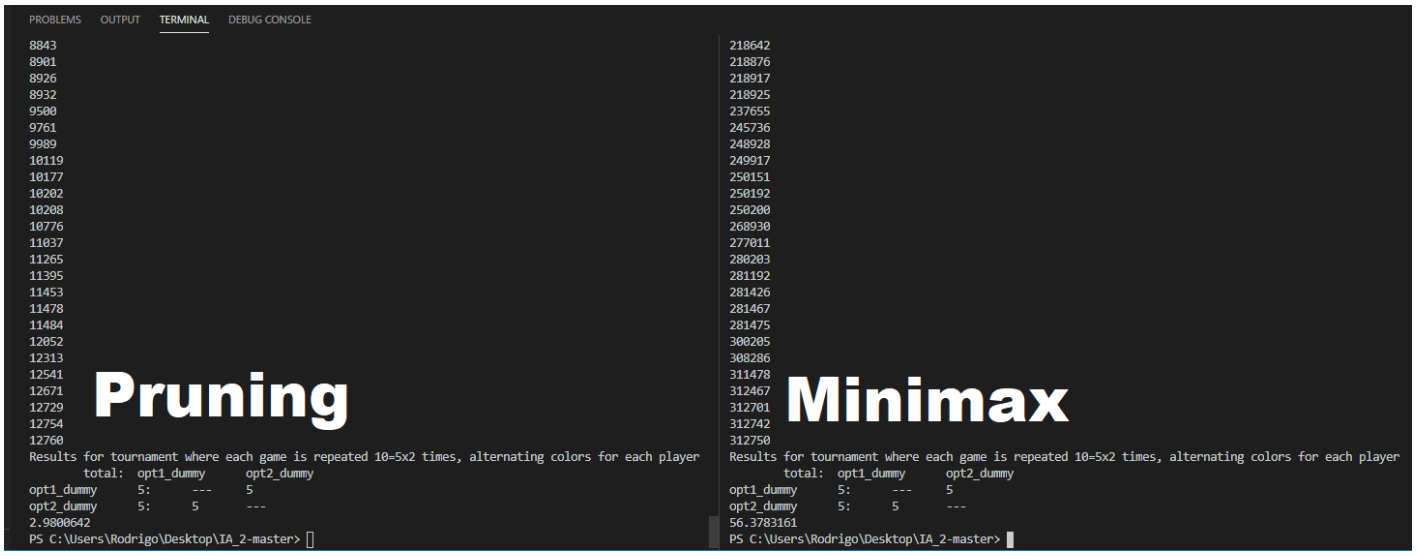
We first ran *demo_tournament.py* (we modified it to add the library *timeit* and stored it *timer_tiktak.py*).

We also run a reversi tournament with one of our heuristics, this is because we didn't have any heuristic that worked with TicTacToe, as we designed all for reversi, timer_reversi.py is again very similar to demo_tournament.py but instead of being TicTacToe the game is a 8x8 Othello, and we used one of our heuristics, the depth is 2 because setting it to 3 or 4 was extremely taxing on the cpu.

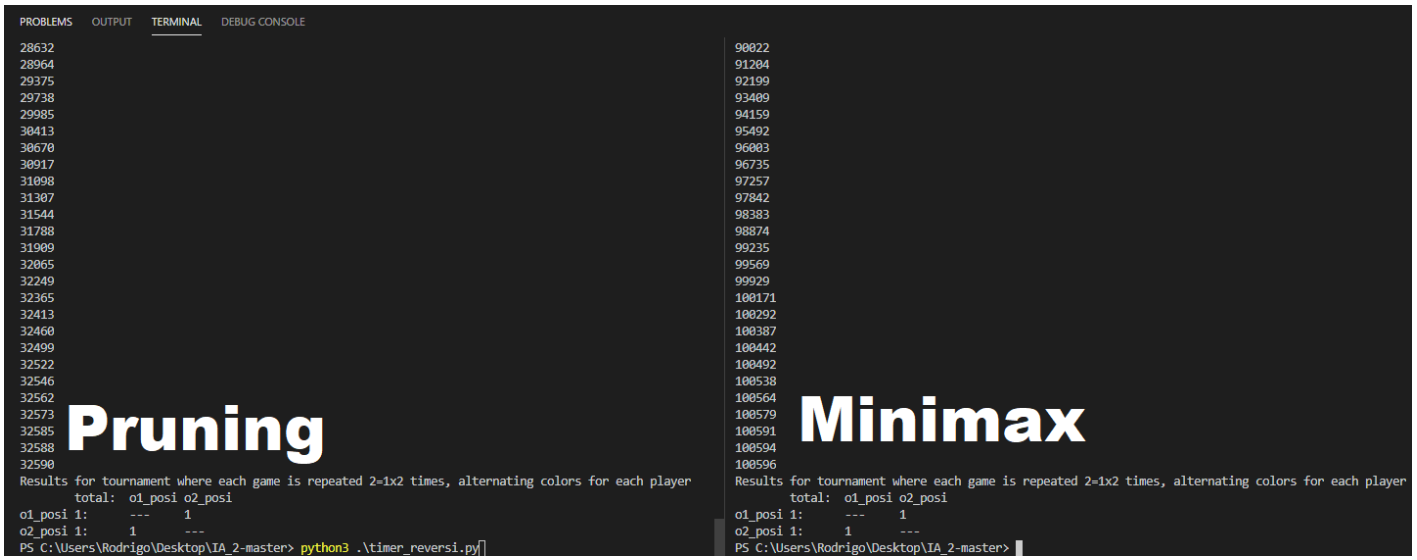
Game/Strategy	**Minimax**	**Pruning**
TicTacToe	56.2058794	3.0427894
Othello	80.7785226	36.4051095

ii. Computer independent measures of improvement.

Tictactoe:



Reversi:



We did the same as for the times, but this time we added a variable in the strategies called expanded, when we return the max node in *next_move* we print the value, this way we can see how it increments. The important values are the ones at the end.

Game/Strategy	**Minimax**	**Pruning**
TicTacToe	312750	12760
Othello	100596	32590

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

When timing TicTacToe it can clearly be seen that Pruning is 18.5 times faster than the stock Minimax algorithm, while in the Reversi tournament it achieves 2.22 times improvement, both are quite high, but the stark differences are due to the different heuristics and games, one can discard more childs.

When we obtained the metrics that do not depend on the computer we can see that the improvement is very similar to the one we timed, 24.5 and 3, this is because we tried to run the timing tests with the computer as stable as possible.
It's clear that Alpha Beta Pruning is a great way to optimice Minimax, it is very easy to implement and the impact can only be positive.

2. Documentation of the design of the heuristic.

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

We watched some Othello videos that explained the basics of the game, we also tried playing and finally we read some papers to try and we searched for previous work on the internet to see where to start coding.

References:

Cherry, Kevin Anthony, (2011). "An intelligent Othello player combining machine learning and game specific heuristics". *LSU Master'sTheses.767*.
https://digitalcommons.lsu.edu/gradschool_theses/767

Parth Parekh, Urjit Singh Bhatia, Kushal Sukthankar. (2013). "*Othello/Reversi using Game Theory techniques"*.
http://play-othello.appspot.com/

Amy S. Biermann (1994) *"Parallel Implementation and Optimization of the Minimax Algorithm with Alpha-Beta Cutoffs in the context of the game Othello".*
http://pressibus.org/ataxx/autre/minimax/paper.html

b. Description of the design process:

i. How was the design process planned and realized?

We first wrote the pseudocode of several basic heuristics, corners, coins, mobility and positional. After that we coded them we started combining them so we had more possibilities to play with. We also implemented heuristics that didn't depend on the basic ones such as stability.

ii. Did you have a systematic procedure to evaluate the heuristics designed?

Yes, we obtained the best heuristics from a starting board (corners_diff) and then we used it as a baseline. We then proceeded to add to the tournament other heuristics such as the stability, or others that depends on the state of the game, and discard them as they fell below the baseline or they didn't enter in the top 3 of the tournament.

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".

Yes the heuristics were one was dependant on the movements done and left, (Parth Parekh, Urjit Singh Bhatia, Kushal Sukthankar., 2013), we also thought about implementing the stability (Amy S. Biermann, 1994), but we finally decided not to

finish it and deliver it as early results didn't give great results and we read (Parth Parekh, Urjit Singh Bhatia, Kushal Sukthankar., 2013) that stability didn't win to other more aggressive heuristics such as corners. We implemented an evaluation function that consisted of combining all the basic heuristics, this achieved great results.
Our third heuristic is based on weights (Parth Parekh, Urjit Singh Bhatia, Kushal Sukthankar., 2013) we obtained from a paper.

c. Description of the final heuristic submitted.

1. Positional: Has a representation of a board with weights, we simply add the weights, if they're from the max_player, or subtract them if they're from the adversary, from the final heuristic. It's basic, but it works quite well. These weights represent the stability they have, its not a direct stable measurement as it doesn't change with the state but we can safely say that the borders and corners are quite important and from there we can assign more weights.
2. PPCr: Uses 3 of the basic heuristics based on how advanced the game is, in the early game we use corner_heuristic as we want to be very aggressive in the early game and capture first those, then, in middle game, positional, as once we have the corners we want stability first, and in the end_game there is not as high of a risk of the enemy flipping our coins so we will use coins.
3. Evaluation: It's a simple formula based on weights for each basic heuristic, where we assigned the most importance to corners (as is the one that best works), secondly mobility (in case corners do not matter any more) and finally coins. The weights are assigned based on testing and performance.