My fantastic report for Assignment 1

Kalle Josefsson (ka5532jo-s) student of EDAP01

September 17, 2024

1 Description of the solution (1-2 pages)

- Describe your approach

  My initial approach was just to get the programming running which turned out to be the most difficult task during the assignment, this is probably since it is my first time programming in python and I tried copying the board and other tricks but it did not work. Thankfully one of the TA's helped me and showed that I could copy an entire environment and play my simulated games on the copies instead. Now to the interesting part of the task. I started out by implementing my minimax algorithm by creating two functions, one min-player and one max-player. The purpose of these were to call them recursively and let them play out every possible game given a board when first called in my function studentmove, which an environment with a board was sent in as parameter. They played out every possible combination of the game and doing so creating a tree. At first I had winning giving a score of 1, loosing -1 and draw or unfinished game a score of 0. This however worked really really slow so I added a depth to the minimax function which made the minimax stop calling each other after a depth of five in the decision tree and if this was not a winning state I would just return zero, in other words the program did not understand which were good moves vs bad moves if it did not win within five moves, this had to be fixed some way and I did that by creating an evaluation function.

  An evaluation function is a function which evaluates the game state given that the game is not finished, which is just what we needed when we reached a max-depth without being in a winning state. My evaluation function is rather simple and straightforward and also the first one that came to mind when trying to find a good one. It is to take the dot product of the board with a board valuing positions based on how many times you can win in a so the corner position from which you can only win in 3 ways, 1 horizontal, 1 diagonal and the ones in the middle of the board having 13 combinations of wins giving them a higher value. Hence our eval function would take the number on every position on our board 1,-1 or 0 for, my piece, opponents piece and a position that is empty. And multiply them with the value of the corresponding position in our eval-matrix and adding them all together to get a picture of the game state. This turned out to be very useful since my program stopped putting pieces in the left most column trying to win there all the time, since that was seen as an equally

valid move as the middle pieces. One thing to note, is that winning and loosing was still scored as 1 and -1, which is not good for us since we can gain 13 points from just taking a middle position with our disc instead of winning, to solve this issue and to make my program block the opponents winning plays I just altered the reward of winning and loosing to +10000 and -10000 this ensured that if my program saw an opportunity to win it stopped trying to catch important positions and instead went for the win, and in the same manner avoided to loose at all cost. This worked exceptionally but there was still an issue, the computation time was to long so we had to optimize our algorithm in order to make it faster when deciding the next move.

In order to reduce computation time one could just decrease the depth of which the program is searching however this would also make it worse so it was not an option for me instead I used this really smart thing called Alpha-Beta pruning which is a method to reduce the amount of lines down the decision tree which have to be looked out,in other words we prune some trees which we do not look at. We can use this method since our min-player function will always choose the best move for them, i.e minimizing the score and our max-player will choose it's best move by maximizing the score hence I my program maximize the score when knowing that the opponent will try to minimize it therefor we can prune decisions that we make and therefor not calculate the score in each node to give an example which makes this a bit more clear, lets say i am considering between two different moves and looking a at a depth of two, first I look at the first move and then let my opponent make their different available moves and calculate the score for each of these resulting states, since my opponent tries to do what is best for them it will always choose the score which has the lowest score. Now i go into the other move I could make and look at the result of the first move that my opponent makes, if this has a lower score than the lowest from my first move I prune the other available moves and don't even continue looking at the other decisions since this will result in worse score for me. Even if lets say the third available moves gives me a great score my opponent will never make that move so there is no point in looking further. Now let's say I have a third option and the all of the scores in this option is higher than the lowest score from my first option, then I update my parameter to the lowest score from the third option and continue this way. In theory this can prune a lot of decisions and at best decreasing the time-complexity for search by O(x*exp(n/2)) and therefor letting us go to a depth of n*2. This worked amazing for me the program became much much faster, maybe I was lucky and the value of the first node after a step was the lowest/highest and therfor could skip the other available moves. The final result when I tested it vs the server was 20 straight wins which I am rather pleased with after working hard for a week.

- Mention the relevant lines of your code that show your minimax implementation

  My minimax implementation can be seen from line 122 to line 168 int eh code and being called in my student move function at line 137.

- Mention the relevant lines of your code that show the alpha-beta pruning

  My alpha beta:implementation can be seen in my min-player and max-player functions in the same lines of code.

- Explain your evaluation function

  Evaluation function is explained above

2 How to launch and use the solution (as much as needed)

Describe how to launch and use your solution. Prerequisites, dependencies and so on.

Not that much done, except I had to import pip install, gym, pygame and also copy but other than that there is nothing special and you will have to run it in a python environment. Just write python skeleton.py –local or –online based on who you want to verse. And doing the same when running the BOT made program.

3 Peer-review (1-2 pages)

Name your partner and the points you discussed The student which I peer reviewed was Vidar Tobrand.

3.1 Peer's Solution

Describe the solution of your peer. The solutions were similar and we could tell that both me and him used Simon's lecture notes as inspiration for the min, max player methods, our adversarial search as well as the alpha beta pruning. However the code differed for the implementation but basically did the same thing. He had the same eval function as me, which we thought would be a reoccurring theme when discussing our solutions. He also uses alpha beta pruning as I did which seemed like the easier and best optimizing strategy.

3.2 Technical Differences of the Solutions

Which things were done differently?

There are some key differences, I noticed that Vidar did not end the search if the first move led to a state which is done. I had this in my code, since if my first move leads to a state which is done, either it's a draw and the board is full or the move is a winning move, hence before going into max and min player recursion we can just return the action which lead to done = true after a step and avoid unnecessary computations, I was happy with this little computation because not only does this reduce computation, it also makes sure that our program always does the winning move when it is presented (it would anyways though). Vidar also stores the state, reward, done after every step which I don't know if it affects the speed, but I just update the environment when taking the step and actually storing them maybe does not even make a difference. He changes player when it is not needed, when max depth is reached or a step results in

winning state he changes player which does not effect so could remove to make code shorter. Overall his code is very short and does the job well so I would say well done to him.

3.3 Opinion and Performance

- Which differences are most important? Our programs played a bit differently since he chose to value the absolute value of a win much larger than the absolute value of a loss. Which I had set to the same but different signs. This makes his player win a bit faster maybe but also riskier, and at a higher risk at loosing, but we discussed this and did not come to a conclusion to which was best, probably does not matter since he also won 20 in a row versus the server. I think the performance of each is pretty similar but mine is maybe a bit faster since the optimization I mentioned above. It would be interesting to see which one performed better versus a better player than the server to see how valuing loss and win would affect against player at same level as our programs. All and all a very interesting discussion and I think both of us got a deeper understanding after having struggled alone which , also reassuring that we had the same thought process.
- How does one assess the performance?
- Which solution would perform better?

4 Interaction summary with an AI chatbot (1-2 pages)

4.1 Interaction prompts

The prompts that made the chatbot to produce code that could run the 4inarow game against the server. In ordet to get the chatbot to write the code and strategy for creating a program which plays connect four I just wrote the simple message: Can you write the code and idea behind creating an AI which plays connect four, you do not need to include the creation of the board etc. Then to make it understand how the environment worked I presented it with the code we were given, both skeleton and the connect four environment. However when I did this I got tons of errors trying to run the code so I sent in the error messages a lot of times progressively making the code better and then it finally worked. The code will be published on canvas in a file called chatgpt_skeleton.

4.2 Difference to the own solution

Elaborate on the differences of the obtained code to your solution. Funny enough ChatGPT answered with first a text which stated that I needed to implement a minimax algorithm, an evaluation function as well as alpha-beta pruning which it stated was optional (at first) and therefor it did not include it in the code it presented but then i told it to add the code for alpha beta pruning as well which it did. Now to the code, the code differs a lot from mine, the structure is different since chatgpt did nestled functions which I did not even know was

doable. Instead of having two methods for min and max player it has one minimax method which works essentially the same, it choose to use depth four instead of five which of course makes the program worse since the search does not foresee the result of the fifth move. The alpha beta pruning works the same and is implemented in the same way as mine. The biggest difference however is the eval function. Chatgpt used an eval function which scored based on how many in a row you have, giving two in a row a score of 10, three in a row a score of 100 and four in a row (a win) a score of 1000. Which is a really bad strategy, since always going for three in a row when playing vs a good player they will just block it. It also does not have a high negative score for loosing meaning that it would rather go for winning than preventing loosing which is also a bad strategy.

4.3 Performance

Is there any substantial difference of performance between your and chatbot's solutions?

When it comes to the performance of the AI generated version I would say that it wins about 70 percent of the time vs the server when my implementation won 22 times in a row. SO you could say the chat generated version is a lot worse. This is probably due to both the depth being smaller, the eval function being a lot worse strategically and also of course not preventing losses. I gave loosing the same score (but negative) as winning, meaning that I prevent the losses instead of trying to go for a win at all cost, meaning maybe longer games but also a lot higher win percentage (100 percent).