EECS 492 Homework 2

Due Friday, 14 October

Fall 2022

Introduction

Deadline

This homework is due by 11:59 PM EST on Friday, 14 October. Late homework will be penalized by 10% per day (where each day starts at 12:00 AM on the due day). There are only **2** late days for this homework.

Gradescope

Please familiarize yourself with Gradescope if you have never used it before. In particular, when you submit an assignment, Gradescope asks you to select pages of your solution that correspond to each problem. This is to make it easy for graders to find where in your work they should grade.

For the written questions, put your answers in a PDF and submit to the corresponding assignment on Gradescope. You can submit this PDF as many times as you would like before the deadline. You can use the provided LaTeX template or Word template and fill in your answer. You may also submit your written response homework in a separate document. Please make sure to include your responses to the 'Understanding the Code' section of the coding question. For the coding questions, submit your code file (Agent.py) to the Gradescope autograder. A template starter code is provided for you. Do not modify any of the imports in these files. You will be graded on both visible and hidden test cases. Thus, you should make sure your solution is always correct (possibly by testing it on other cases that you write) and not just whether it succeeds on the visible test cases. This is an important skill for a practicing computer scientist! You can submit to the autograder as many times as you would like before the deadline. While you will not be graded on style, we encourage you to uphold best-practice style guidelines. For more information about the coding refer to the coding portion of the homework.

Setup

You are recommended to use a Python virtual environment for the coding portion.

You may follow this tutorial from EECS 485 to create and setup a virtual environment.

We have provided you with a requirements.txt file. After creating the environment as mentioned in the tutorial, activate the environment by executing source env/bin/activate in terminal. Install the necessary library by using pip install -r requirements.txt.

Please ensure you are using Python version 3.7 or greater. The chess library we will be using in this assignment is only supported for Python versions 3.7 and above. Feel free to visit us in office hours if you need help on the above or help on setting up tools for debugging. **We recommend you get your setup working well in advance before you actually start working on the coding questions.**

Working

Please show your working in order to receive partial credits. Answers should be clearly legible, brief and to the point.

Justification of Answers

When asked to justify your answers, a simple one-or-two line explanation will suffice.

Written Section [40 points]

1 Constraint Satisfaction [15 points]

1. A committee has decided that the new Leinweber Computer Science and Information building needs to be slightly larger than what current plans have marked out. They now need to level out a patch of land that is connected to the current construction site. To do this, they need to do the following things -

   - Task 1 : Remove the existing fencing and equipment in the area
   - Task 2 : Set up a new fence around the updated perimeter
   - Task 3 : Clear out the vegetation
   - Task 4 : Dig out the sand
   - Take 5 : Blast the rock
   - Task 6 : Level the patch to make it flush with the existing construction site.

   Tasks 1, 2 and 3 need to be completed first in parallel, following which 4, 5 and 6 must be completed one after the other. Assume that Tasks 1, 2 and 3 all finish at the same time. Management has decided to assign 4 construction specialists to these tasks to get them done as quickly as possible. Since the area is small, they want to get it all done in a single day. The specialists can do the following tasks

   1. A can do 1, 2 and 4
   2. B can do 1, 3, 4 and 5
   3. C can do 1. 2, 3 and 6
   4. D can do 3 and 6

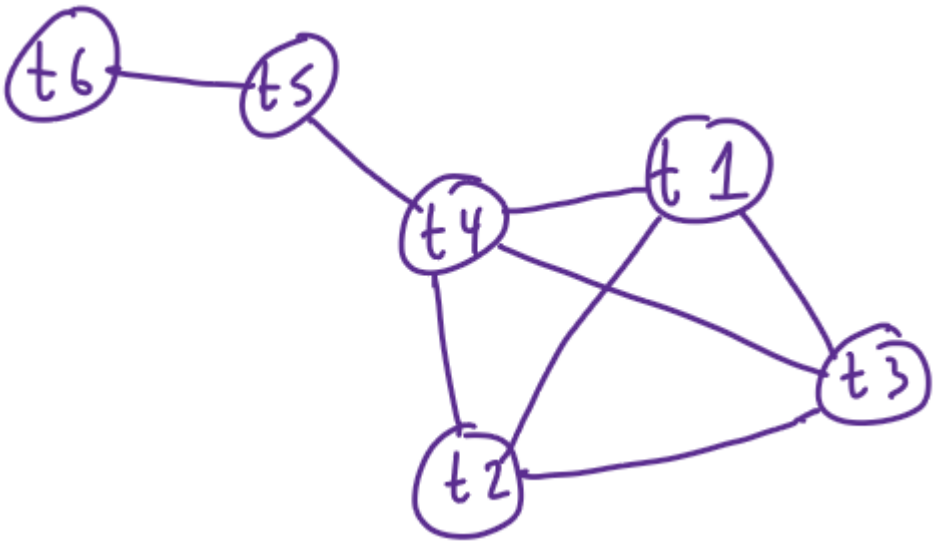
   Due to health and safety regulations, none of the specialists can work back-to-back. The specialists cannot do two tasks at the same time.

   (a) Formulate this problem as a CSP problem in which there is one variable per construction activity, stating the domains, and constraints. [2 points]

   Solution: Variables – Task 1-6 X: {t1, t2, t3, t4, t5, t6} Domain: each construction specialist d: {A (for 1, 2, 4), B (for 1, 3, 4, 5), C (for 1, 2, 3, 6), D(for 3, 6}
   Constraints: node constraints: C = {t1 ≠ D} {t2 ≠ B || D} {t3 ≠ A} {t4 ≠ C || D}{t5 ≠ A || C || D} {t6 ≠ A || B}
   Each specialist can't do more than one task from t1-3: C = {t1 ≠ t2}{t2 ≠ t3}{t1 ≠ t3}
   Specialists can't work back-to-back jobs: C = {t4 ≠ t1 || t2 || t3 || t5}, {t5 ≠ t6}
   shorthand: {t5 ≠ t6} → {(A, B), (A, C), (A, D), (B, A), (B, C), (B, D), (C, A), (C, B), (C, D), (D, A), (D, B), (D, C)}

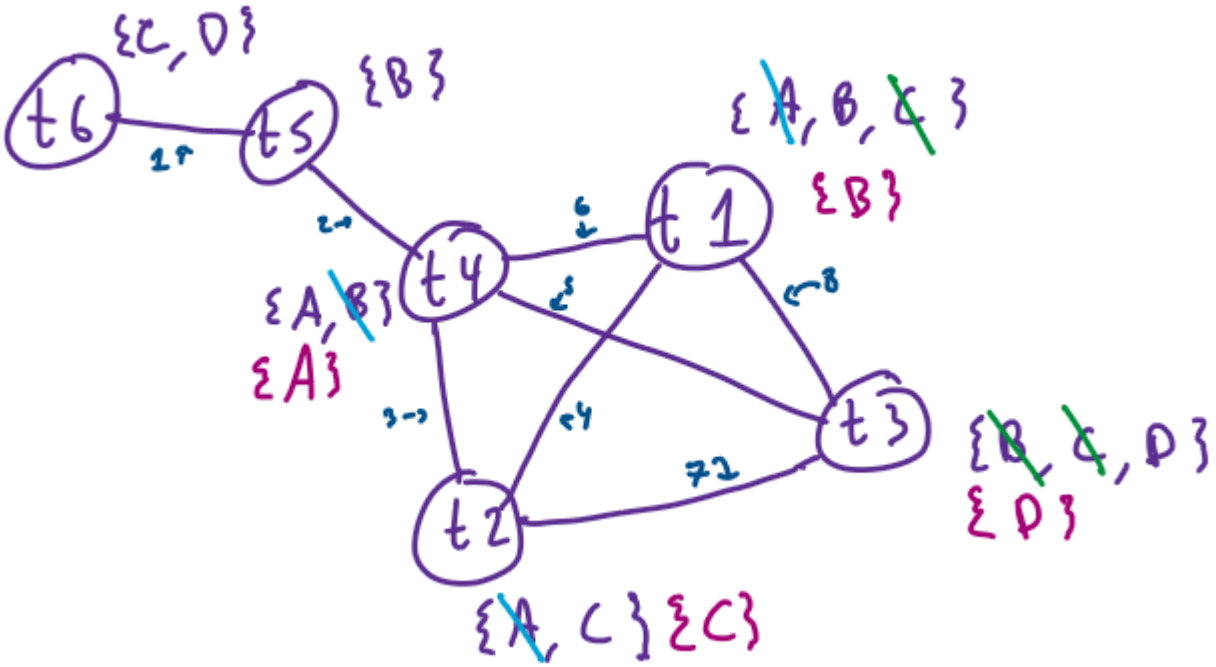
   t1: {A, B, C} t2: {A, C} t3: {B, C, D} t4:{A, B} t5: {B} t6:{C, D}

   (b) Draw the constraint graph [3 points]



Solution:

(c) Show the domains of each variable after running AC-3 on this initial constraint graph. [3 points]



Solution:

1. Start w/ t5 (MRV)
2. Check path 1 & 2 to queue
3. Remove B from t4 domains
4. Add path 3, 5, 6 to queue
5. Remove A from t2, A from t1
6. Add path 4, 6, 7, 8 to queue
7. Remove C from t1, B from t3, C from t3
8. Add path 6, 8, 5 to q
9. Complete, true

(d) What are the solutions to this CSP? [2 points]

Solution: There are two solutions: t1=B, t2=C, t3=D, t4=A, t5=B, t6=C and t1=B, t2=C, t3=D, t4=A, t5=B, t6=C

2. Constraint Satisfaction algorithms are very good at helping find if no solution to a problem exists. One way this is used in Computer Science is in an area called formal verification. Formal verification aims to provably show that an algorithm does what it intends to. It is possible to formulate formal verification as a constraint satisfaction program by showing that we cannot find an assignment of arguments and variables that make our code behave in a way we don't want.

   Consider this Python function:

```
def func (x , y , z ):
    a = x*x
    b = a + 2*x*y
    c = b – 2*x*z
    d = c + y*y
    e = d – 2*y*z
    f = e + z*z
    return f
```

   We want to ensure that this function always returns a value greater than or equal to zero for any value of x, y and z. We also know that each python assignment operation (i.e, '='), is effectively constraining the left hand side of the assignment to the right hand side.

   How would you formulate the CSP to test that func(x, y, z) will always return a value greater than zero? [5 points]

Solution: Variables X = {x, y, z} Domains: Di = Set of all Reals Constraints C = {x <= -y + z}, {y <= -x + z}, {x+y >=z}

f = 2x +2y -2z, 0 <= f, 0 <= 2x + 2y -2z

2 Probability [15 Points]

Please show your working for all these questions.

1. Consider a scenario where you've developed an AI model that is able to detect the presence of a disease *D* from a blood sample. From a controlled experiment, you were able to determine that the true positive rate of your model is 90%, i.e.

$$P(\text{ Model says the patient has } D \mid \text{ patient has } D) = 0.9$$

   In a similar fashion, you determine that your algorithm has a true negative rate of 80%

$$P(\text{ Model says the patient does not have } D \mid \text{ patient does not have } D) = 0.8$$

   Current studies show that the chance that a person gets *D* is 5%. Answer the following questions.

   (a) Given a random patient without any diagnostic history (that is, we do not know if they have *D* or not), what is the likelihood that your AI model says that the patient has *D*? [5 points]

Solution: P(Mod = Yes| pat = Yes) = 0.9

P(Mod = No | pat = no) = 0.8

P(mod = Yes) = P(pat = y && mod = yes)/ P(pat = yes | model = yes)

= P(mod ==

=[0.9/0.05]/P(

(b) Your test is relatively inexpensive, costing a patient just $50. Additionally, you observe that patients save a lot of money if they get diagnosed correctly, but have higher healthcare costs if they get diagnosed late. You perform a study to compute the average savings in healthcare costs experienced by the users of your model and get the following results.

	Patient has *D*	Patient does not have *D*
Your model says Positive	10,000	-1000
Your test says Negative	-30,000	-50

Table 1: Average savings provided by your model (in USD)

Using this study, what is the expected value of savings in healthcare costs that a patient will incur? Does your model actually save people money? [5 points]

Solution: 10000*.045 + -50 * 0.76 +

2. To increase participation in the EECS 492 discussion classes, the instructors have decided to create five discussions - A, B, C, D and E and hold a pop quiz in four of these discussion sections that they've randomly selected. Students need to decide which section they will go to on the previous day, and they will be told if they have a quiz only halfway through the discussion. Students must register for one of the classes.

   You don't think you're ready for it, so you want to do your best to avoid going to a section that has a quiz. At the same time, you need to attend one of the discussions. You randomly decide and pick Section C. After you pick, a GSI decides to help you out. He tells you that he knows which discussion sections are going to have the quiz. He also tells you the Section A is going to have a quiz, and gives you the option to swap the section you picked (C) with the others (B, D or E).

   Using probability, determine what choice (keeping your discussion or swapping for another one) gives you the highest odds to avoid the quiz.[5 points]

Solution: Swapping for another discussion gives you the highest odds of avoiding the quiz. Initially, the probability that each individual discussion is the one section without the quiz is the same – 20% each. Therefore, there is an 80% chance that A, B, D, or E has no quiz. The chance that A doesn't have a quiz is zero after the GSI tells you that discussion A has a quiz. Not so intuitively, the probability that a discussion section that isn't C (B, D, E) is still 80%, as that portion of the pie, has not been shrunk, A has just been taken out of it. Now B, D, and E individually have an approximately 26.667% chance of not having a quiz, a higher percentage than C. Therefore, by switching to one of the three other sections is your best bet, as the probability that you won't have to take a quiz increases by about 6.667%.

3 Adversarial Search [10 Points]

1. Use the $\alpha\beta$ pruned Minimax algorithm on the Game Tree described in Figure 1 to find the value of the root and the optimal series of moves. Mention which nodes get pruned. [10 points]

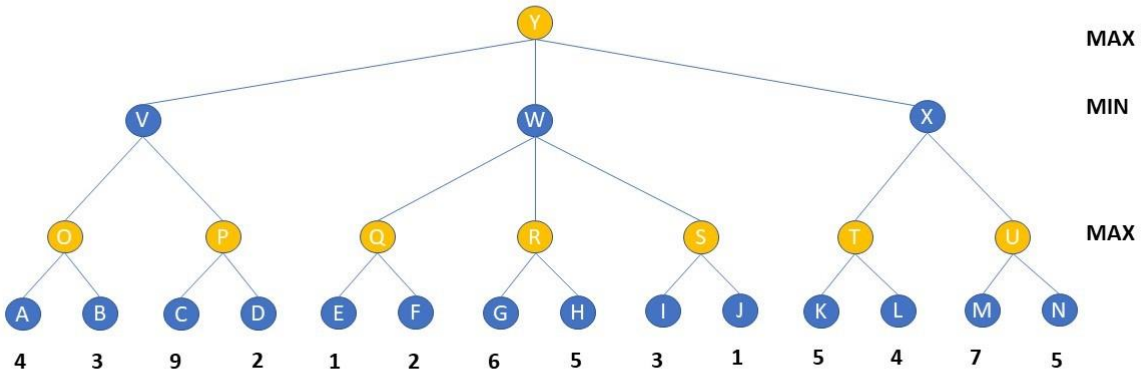
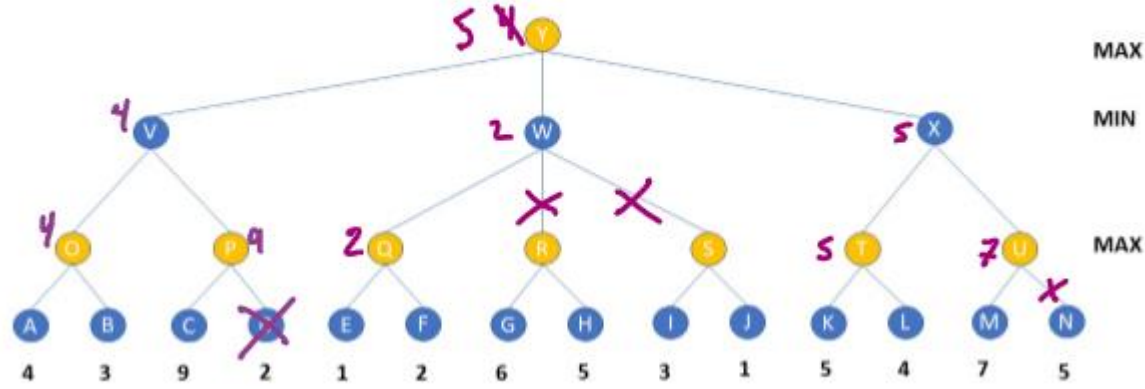


Figure 1: Game Tree for Question 4.1

Solution: Root value: 5 Moves: Y, X, T, K Pruned: D, G, H, I, J, N



Programming Section [60 Points]

Introduction

For the rest of the assignments this semester, we will work on the same problem - Chess! Our goal is going to be to iteratively work towards building a fairly competent chess AI. You do not need to learn how to play chess in order

to do these assignments, but we do recommend you familiarize yourself with the rules to make it easier for you to debug, understand and improve your work[1]. You'll probably pick them up as you work on these problems.

In this assignment, we'll be using Minimax with $\alpha\beta$ pruning and a simple heuristic to plan our moves to create Version 1 of our Chess AI[2].

There are two qualitative questions that are also present in this section! Please attempt them after you complete the programming task and include your answers along with the written section! Observability, FEN Notation and Ply

You might recall that the discussion around the observability of chess. We noted that when we talk about chess's observability, we simplified because of two special moves - en passant and castling, both of which require you to maintain some form of state history. In practice, we can also implement chess in a fully observable manner using FEN notation [3], which is the standard way to represent any chess board state as a single string.

You may also see the word 'ply' in a few places. A ply in chess is a half-move - meaning that it represents just one white move, or just one black move. In our terminology, the ply would be the depth that we search till (e.g., ply=3 would mean White's move, followed by Black's move, followed by White's move).

Objective [50 points]

Your goal is to complete the h-minimax function in the file Agent.py. Using this function, the Agent will take any FEN string and ply, and predict the expected set of moves that will maximise its chances of winning.

Overview

The assignment folder contains the following important .py files

- Agent.py
- Engine.py
- Evaluator.py
- Node.py
- Main.py
- Test.py
- BoardViewer.ipynb
- AgentVersusAgent.ipynb

Agent.py

The relevant members and methods are

1. expandNode(self, node, depth) : this creates the nodes of a ChessNode node. Since the minimax algorithm only cares about the values of the leaf nodes, this method generates heuristics only for those, using the depth argument.
2. h-minimax(self, depth, node, maximizingPlayer, alpha, beta) : this is a recursive function whose implementation you must complete. The output of this function must be the score and the list of moves. You must complete this function using $\alpha\beta$ pruning, as regular minimax will simply take too long to run.

[1] This website has a ton of great resources if you're interested.

[2] It might interest you to know that it was a UM alumnus that first came up with this idea! See this paper if you're curious. [3]If you're interested in understanding it - see this link

Node.py

This class describes the ChessNode object. It has the following members

- state : a variable of the chess.board class that is used to capture the current state of the board
- move : a string that represents the move that resulted in this node's state. This string is in the UCI format.[3]
- score : the heuristic score of a node
- isTerminal : a boolean variable that indicates if the node is a leaf. We need to include this in order to correctly handle a checkmate. A checkmate is the end of the game, and as such, if the node's state is a checkmate, then it cannot have any children. The minimax algorithms we've walked through assume that all the children are on the same level, so it is important to check for both the depth as well as if the node is terminal or not before proceeding on with the rest of the algorithm.

The methods in ChessNode are used to generate children with or without the heuristic. The expandNode method in Agent already provides a wrapper for this.

Main.py

This is the main file you will use to run and test your solution. This file reads an input test file and writes out an output result file. You do not need to worry about parsing the input or formatting the output as both are already taken care of for you.

The test input format is a tab separated text file. The first value is the ply of the agent, and the second is the FEN representation of the board state. We've included a expected results.txt file to give you an idea of what the expected result is, along with how long the algorithm should take to generate it. Compare your output with the result file to determine if your solution is acceptable.

Note: The execution time of the function can vary to a certain degree, but ideally it should be within 1020 seconds of the expected value. An order of magnitude greater is not acceptable. as the test cases on the autograder are timed.

Test.py

You can use this file to test individual test cases. Like Main.py, it loads an input file and compares the results to an expected output. Note that this file does **not** check if your algorithm runs within the expected time constraints. To use it, simply run py Test.py TESTNUMBER where TESTNUMBER ranges from 1 to 4.

Engine.py

This class acts as a wrapper for the python-chess Python library. It provides functionality to generate legal moves and possible board states. It also adds support for visualizations.

Evaluator.py

The evaluation method is already implemented for you, and you do not need to create any kind of heuristic. All of the evaluation is done by the Evaluator object, and the evaluation is done automatically when you expand a node.

The heuristic itself is based on the Simplified Evaluation Function, which is a fairly straightforward function that encapsulates the following information

1. If we can get a promotion - A promotion is an event where a Pawn reaches the end of the board. When this happens, we can swap the pawn out for a Queen, Bishop, Knight or Rook, resulting in a big advantage for the player who got the promotion.
2. If we can capture a piece - We want to reward capturing pieces, but its better to capture a strong piece with a weak one, instead of vice-versa. To do this, we assign numeric values to each of the pieces. For instance, a pawn has a value of 100 while the Queen has a value of 900.

[3] See here for more details

3. How well positioned our pieces are on the board - This is done by assigning a value to each piece called a piece value, and using a lookup table called a Piece-Square-Table (PST). The PSTs are available in a pickle file and get loaded by the evaluator object. The PSTs are just 8x8 matrices for each piece - Pawn, Rook, Knight, Bishop, Queen and King, that have scores that roughly indicate how good a particular square

   is for a given piece. These PSTs are based on empirical data that some expert players have come up with. In addition, towards the end of the game, a common tactic is to have the King move away from the edges and corners of the board to have more space to avoid checkmates. To allow for this, the King has two PSTs, one for the early and midgame, and an endgame specific table.

In addition, the heuristic captures the current value of the board by summing up the total value of your pieces, and subtracting the total value of your opponent's pieces.

Jupyter Notebooks

Chess boards can be hard to interpret on a terminal, so we've included two Jupyter notebooks to help you visualize what the board looks like, and to help you play around with the algorithm you write. To run the notebooks, after you set up the virtual environment, simply run jupyter notebook
This will open a browser window where you can open and view the .ipynb files. These files allow python code to be rendered into HTML, making it very easy to look at chessboards.

AgentVersusAgent.ipynb

This notebook is completely optional to use. You can use it to play one agent against another and see which agent wins. It may also help you in debugging, and if you're very curious, you can even modify it to play against the AI yourself.
Don't be alarmed if the game is played *really* slowly. A regular $\alpha\beta$-pruned minimax agent playing with a ply of 5 can take over a minute for some board states with a lot of valid moves. Subsequent assignments will have you come up with agents that are a lot more fun to watch!

BoardViewer.ipynb

This is a Jupyter notebook that makes it easier to actually see the chessboard and what each move looks like. Using this is completely optional, and it is solely to help you debug, The notebook contains cells that provide an example of how you can use it to display a sequence of moves.

Testing

To test your code, you can use the following options

1. Running Main.py will create a consolidated results file that you can compare with expected_results.txt in the Tests folder to check how long each test case took to run.

2. Running Test.py will let you validate a specific test case's move sequence. To use it, simply run
   Test.py
   followed by the test number (1 to 4). For example
   py Test.py 1 will run test case 1.[4]

3. You can also use the jupyter notebooks to help you debug and analyse the board states.

Understanding the Code [5 points]

Please include a written response to these questions when you submit the written section of this assignment.

1. We mentioned earlier that regular minimax will simply take too long to run. For the four test cases we've provided, run your algorithm with and without $\alpha\beta$ pruning and tabulate the time it takes to generate the move sequences. You can use Main.py to generate a results.txt file and copy your results from there. [2

[4] Depending on your operating system and Python version, you may need to use the command python instead of py

points]

Solution: 1: 313.05288 secs 2: 111.8610077 secs 3: over 30 min 4: over 30 min

2. Take a look at function used to generate children with their heuristic value.

```
# Use this function to generate child nodes with the heuristic value def generateWithHeuristic ( self ,
engine , evaluator ):
    childStates = engine . generateBoardStateTuples ( s e l f . state ) children = [ ]
    for child in childStates : terminal = False
        childBoardState = child [0]
        if childBoardState . is _checkmate ():
            terminal = True
            if childBoardState . turn == evaluator . player :
                score = −s e l f .CHECKMATESCORE else :
                score = s e l f .CHECKMATESCORE elif
        childBoardState . is game over ():
            terminal = True
        score = 0 else :
                score = evaluator . getMoveScore( s e l f . state , child [1] , evaluator . player )
            children . append(ChessNode( child [0] , child [ 1 ] . uci () , score , terminal ))
    children.sort(reverse=(self.state.turn == evaluator.player), key = lambda x : x.score)
    return children
```

The line in bold sorts child nodes based on their score in descending order if they're created by the Agent's move, and in ascending order if they're created by the opponent's move. Why do this? [4 points]

Solution: The nodes created by the agent's moves most recently are the most likely to be favorable to the agent and make it closer to winning the game. Therefore, sorting it in descending order prioritizes the most advantageous moves first. The nodes created by the opponent's moves are the most likely to result in the opponent's win and our agent's loss. The opponent knows that it's working directly against us, so it would factor into its move decision that our agent would have the low value nodes deep in the queue and not evaluate them in the decision process (if we didn't sort them in ascending order), making it the opponent's best move, and best chance to win.

3. Suppose your agent is now playing against a rival agent with the exact same architecture and heuristic. You play with ply=5, and make a prediction of a series of moves. The rival agent makes the move you predicted it would take. You now predict another set of moves with ply=5. Is your next move the same one that you predicted before your rival made their move? Why or why not? [4 points]

Hint: Consider two extreme cases - one where the game has just started, and another where the game is about to end. You can also use AgentVersusAgent.ipynb to help you out!

Solution: Not necessarily. When there are a plethora of moves available, even if the opponent played the predicted move, the tree has still changed and as such your next move isn't going to be the same as previously predicted.