Noughts and crosses Paper

My goal:

My goal is to make a program that can play noughts and crosses (tic tac toe) optimally .

* I cannot use any form of preprocessing.
* It should run relatively fast.
* It must be dynamic (it cannot be a series of if statements, this means it could be expanded to connect four more easily).
* No external libraries should be used.
* It must be my own idea (I cannot watch a video that details how to make a perfect tic tac toe script and take inspiration)

Algorithm:

The way I went about making my program was a simple idea, but everything is easier said than done.  
Let us take a different game first. In this game two players alternate picking branches until we hit the base node of the tree in which that value is selected:

P1 goes first and wants the highest score possible.

A picture containing diagram

Description automatically generatedP2 goes second and wants the lowest score possible.

In this game one strategy/algorithm we can use to play optimally is called minimax (min max). This is when a player picks the instance in which he maximizes his results even if the opponent was to play the most detrimental move towards him. Maximizing the yield under the assumption that the opponent plays perfectly (the best move from the opponent’s perspective)

In this scenario that would entail player one picking node c. This is because it yields the best results when playing against a perfect opponent. The reason for this pick, would be because you would go to the base node and assign the node above the highest or lowest value depending on the turn. If the turn is the opponents at that level of the tree, then assign the lowest value to the parent node. However, if it is your turn at that level of the tree, assign the highest value.

Along with other limitations which I am sure I haven’t thought of one of the most major ones is of identical parental node values. For instance, if we were to change node b1’s value to 3. In this scenario node a and node b would be given the same value when it may be more beneficial to pick node b since the mistake the opponent could make may be more beneficial towards us. Moreover, another limitation is that this would not work for complex games where there are too many base nodes or in imperfect information games.

My method:

Diagram, schematic

Description automatically generatedIn my scenario I will have some function create a tree of all noughts and crosses positions and assign the ending nodes different values:

1 for X winning.

-1 for O winning

0 for no one winning.

Then these values will be escalated like in the above game and a script will pick whichever it finds appropriate depending on if the computer is X or O. If it is X, it will pick the highest value while if its O it will pick the lowest values.

Limitations of my application:

One reason this works is because there aren’t a lot of base nodes. However, in a game like go or chess with a lot of possible combination (moves) it would be impossible computationally to create this tree. Furthermore, in games such as poker which is an imperfect information game a different aproach must be taken to consider the unknown

Rotation:

To avoid having a huge tree I considered rotation. What does that mean? When playing noughts and crosses on the first move you have 9 different possible position (squares) you can play. When writing these out it is evident that they are extremely similar, and some are the same board position just rotated. This can be utilized since now instead of having 9 starting nodes and paths to follow there are only 3. This greatly reduces the size of the tree. Here is an example:

A picture containing background pattern

Description automatically generated A picture containing background pattern

Description automatically generated

Further increase in efficiency

Another thing I did to increase efficiency, is I made the creation of the hash map and the escalation of the base node values happen at the same time. This avoids me having to go through the tree again just to escalate the values assigned to the nodes.

Ve.1 and Ve.2

I had originally made this 2-3 weeks ago. However, when I ran it took too long. I know believe, it was reasonably fast (ish) and would have worked, but I forgot a simple if statement which drastically increased time. So, I then decided to redo everything using hash maps and making everything more efficient since I thought this high run time was because the gods of programming were punishing me for using linked lists.

The tree:

The thing I frequently talk about above as “the tree” resembles a tree but it’s not really.

It takes a format as follows. One big hash map where each key is a board/position and each value of this, is a list with the first item being the node attributed to the position and the second another hash map of the possible iterations of the position:

Defining the functions:

rotate: Takes board which is a list and a different parameter that tells it how many time it should rotate.

remove\_rotation\_copies: Takes a list of boards removes copies from a list of multiple boards.

iterate: Takes a board and someone who is playing and produces a list of all the next boards.

game\_table: Takes a list that is a board and produces it in an aesthetically pleasing way for the user.

gui: Takes a board and who is playing, asks the user where they want to play and output a final board.

check\_won: Takes a board produces: 0 if draw 1, if X wins -1, if O wins -1, and None if it is not done.

convert: Takes a list, produces a string and is given a string turn produces a list (used for the hash map.

step: Takes a node and turn produces a list with a hash map with every move through time.

response: Takes a guiding hash map takes who it plays against and the board and finds best Node.

find\_offset: Takes hash map and a board and figures out how you must rotate it to fit the hash map.

The Code:

<https://github.com/philip344/Knots-and-crosses-game.git>

What did I learn:

This project has been unusually tough but so are most projects when you do them and has tested my motivation. I have furthered my knowledge in object-oriented programming, and it is my first encounter with trees (without using an external library). Lastly, it’s another good experience to deepen my understanding of optimization which is extremely crucial. The reason why I do this even though there is always someone better or something more efficient is because its mine. I have been able to complete this, from start to finish without peoples help, on my own, my own solution. Sure, it may not be the best or the fastest or the smallest, but it works and has a part of my thinking in it.

Bibliography:

[W3school.com](file:///C:\Users\pilih\OneDrive\Desktop\random%20stuff\W3school.com) (Things such as not remembering inbuilt function names)

[Google](file:///C:\Users\pilih\OneDrive\Desktop\random%20stuff\Google) (The rotation example picture)

[JetBrains](file:///C:\Users\pilih\OneDrive\Desktop\random%20stuff\JetBrains) (The IDE I used)

[Python](file:///C:\Users\pilih\OneDrive\Desktop\random%20stuff\Python) (The language I used)