Naser Alqimlas
CSCI 3202
Final Practicum
12/12/2018

Using Alpha Beta Pruning To Create an Unbeatable Tic Tac Toe Agent

Github link for my code: https://github.com/NaserAlqimlas/CSCI3202Practicum

Introduction and Background Research:

For my practicum I decided to create an unbeatable tic tac toe agent. I decided to conduct some research on the different ways to find a solution to this project. I found several solutions to the problem, One was via a table lookup, in which the agent would place a move in the center, then one of the corners then one of the sides in that order (if available of course). This was done by the Nantang Technological University in Singapore. It seemed like a naive and rudimentary solution so I kept searching.

The way most people solved it was by using a minimax agent. Aaron Wang, a poster on hackernoon.com stated that his minimax agent, which was an application of depth first search, had a running time of o(b^n). His implementation was done using a double for-loop which immediately put me off as a sub optimal solution.

Through searching for minimax solutions I stumbled upon a research paper which compared its results to that of an agent that randomly selects the next move. (Kobti, 2007) I decided would serve as a good comparison to my solution.

Q-learning was another common solution, in which the agent is trained through multiple iterations. This was done by Amresh Venugopal, who posted an article about it becominghuman.ai. I decided not to go with Q-learning since I was more interested in the topic of Alpha Beta Pruning. Since I had struggled with it earlier in the semester during the pacman assignment I decided this would be the perfect opportunity to strengthen my understanding of the topic. The way alpha beta pruning works is the same as minimax, but it performs a check to decide whether the next child node in the game tree will produce a desirable result, if the check fails, it prunes the nodes and proceeds.

Methods:

Github link for the code: https://github.com/NaserAlqimlas/CSCI3202Practicum

First, to have something to compare with, I implemented a naive solution in the niaveTicTacToe.py file that would select a position on the board at random. This was done to compare the Alpha Beta Pruning process with a random selection process. I did this by doing random.choice, which chooses a random move to make.

Second, I implemented a smarter solution using Alpha Beta Pruning. Since Alpha beta pruning works by going down every possible path of the game tree and pruning the redundant options, which in this case would be moves to make. It then maximizes the score for the computer player, and minimizes for the human player (named player in the code).

First, I declared a newStateGenerator Method, which would take in the current state and generate a new state given a move. This helps in the pruning process since we would look at these new states and decide whether to prune or not. I then have two functions, minimizer and maximizer which would maximize/minimize given the current game state, as well who is playing next (upNext in the code).

The Minimizer function would first check to see if the game is over, in which case it would check if the game is over, in which case it would return the score along with None as a tuple, since there would be no current best move. We then set our best score to infinity which is the worst case best score and declare a best move variable which would be initialized to None. Then we run through every possible move, and generate a new state and then take the minimum of the best score and the output of our maximizer function, which would be alpha in this case. This is stored in the newScore variable. We then compare our new score with the best score, and if our new score is better (less in this case) we would set it as our best score. If our best score is less than alpha then we prune. If that is not the case, then we set beta to the minimum of best score and beta. Finally when we loop through all possible moves, we would return a tuple of beta and the best move. The Maximizer works in a similar fashion except it would try to maximize the values and prunes on beta (rather than alpha).

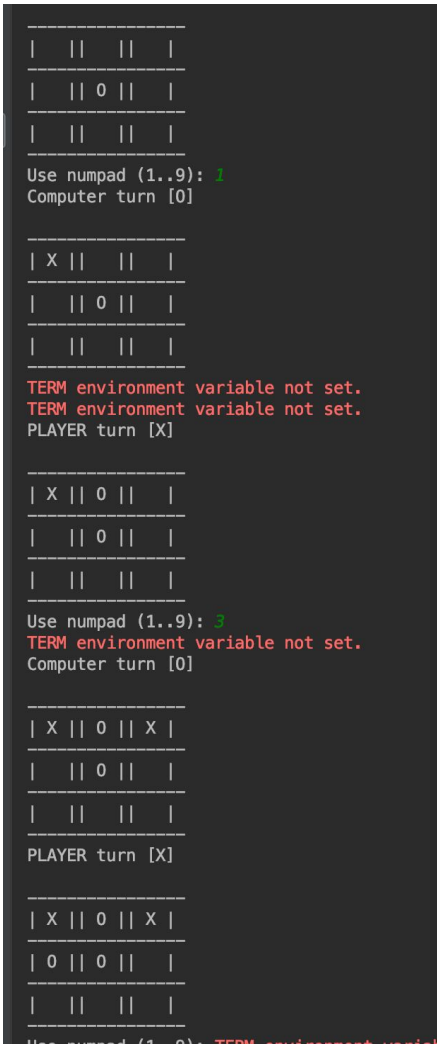
The AlphaBetaPruning function sets both alpha and beta to -infinity and infinity respectively. It checks if the current turn is the computers; if that is the case, then it would attempt to maximize the moves. Otherwise, it would minimize the moves. It would then return the best move's x coordinate, y coordinate, and the best score.

Due to recursion, the AI would flip flop between minimizing and maximizing, and play a "game" down each tree while attempting to maximize the score. In this implementation it is attempting to maximize the score of the computer.

Results:

There were several questions I aimed to answer by conducting this research. The first was to see if Alpha Beta Pruning would be successful at beating a human in a tic tac toe game. To do this I challenged the alpha beta pruning agent five times and the naive agent twice, sometimes I was playing, other times I let friends try to beat my agent for the sake of variety. In some cases I tried to see the agents behaviour, whereas in others I tried my best to win here were the results:

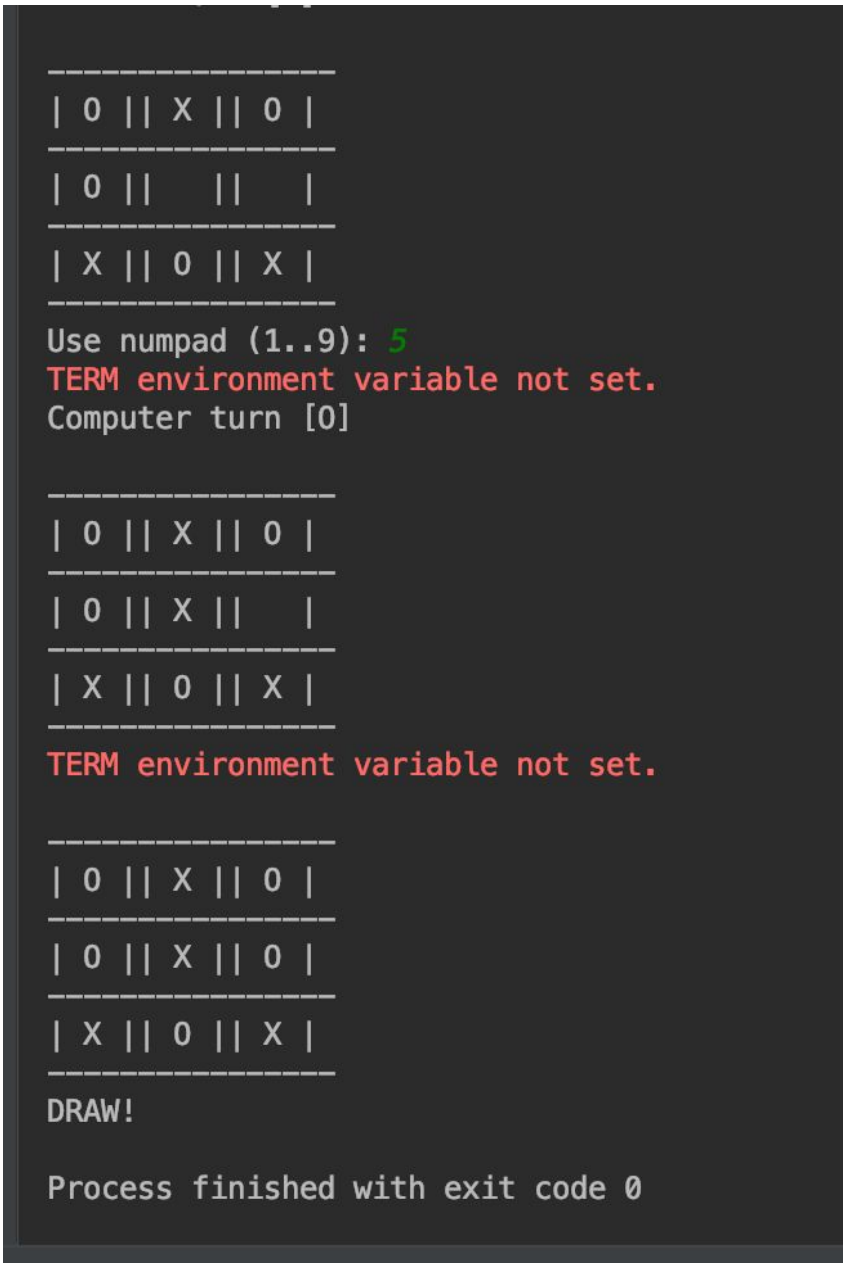
Game 1: Agent Wins



In this case I tried to see the agents behaviour by allowing it to win. As we can see the agent seemed to play safe in a sense by creating more possible win states rather than choosing to win.

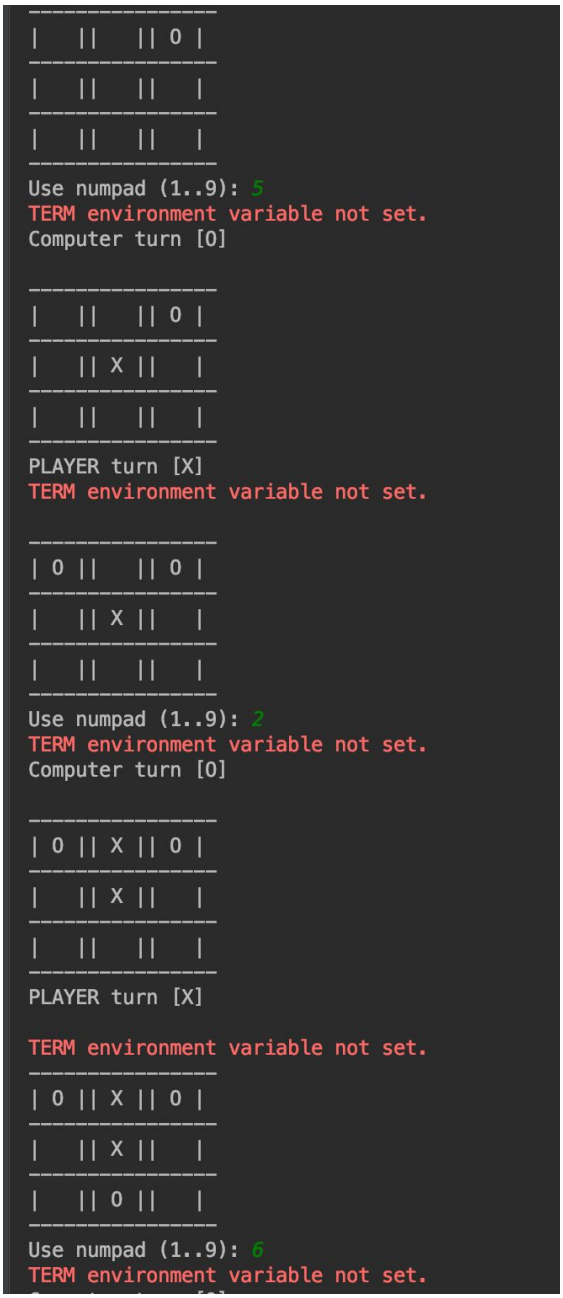
We can see this in the fifth game board, where it could have played in position (2,1) but instead, it played position (0,1). Ultimately it resulted in a win for the agent.

Game 2: Draw



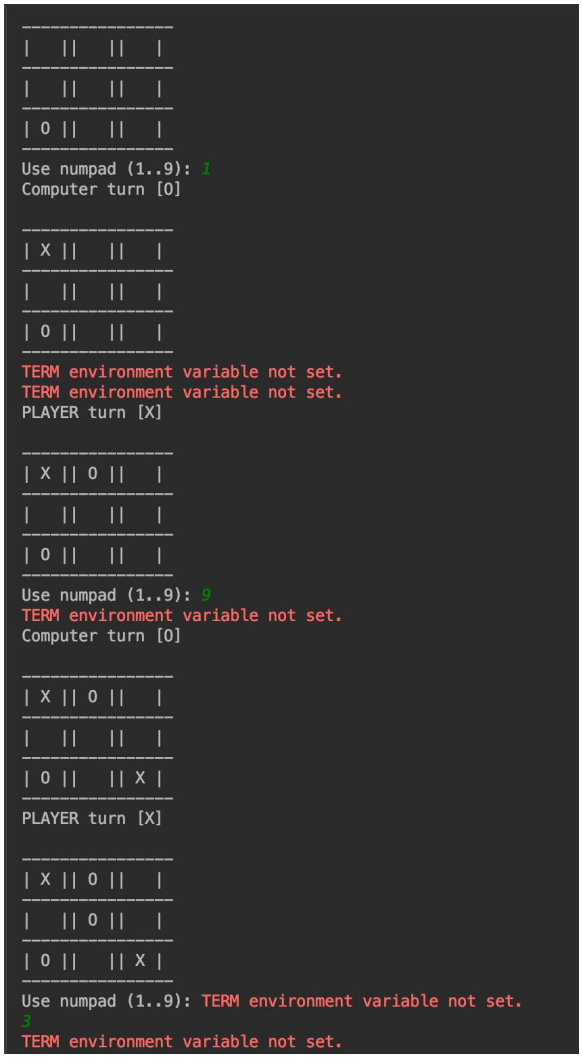
In this case I decided to try my best to win. As we can see it manage to successfully pressure me into playing defensively, ultimately resulting in a draw.

Game 3: Draw



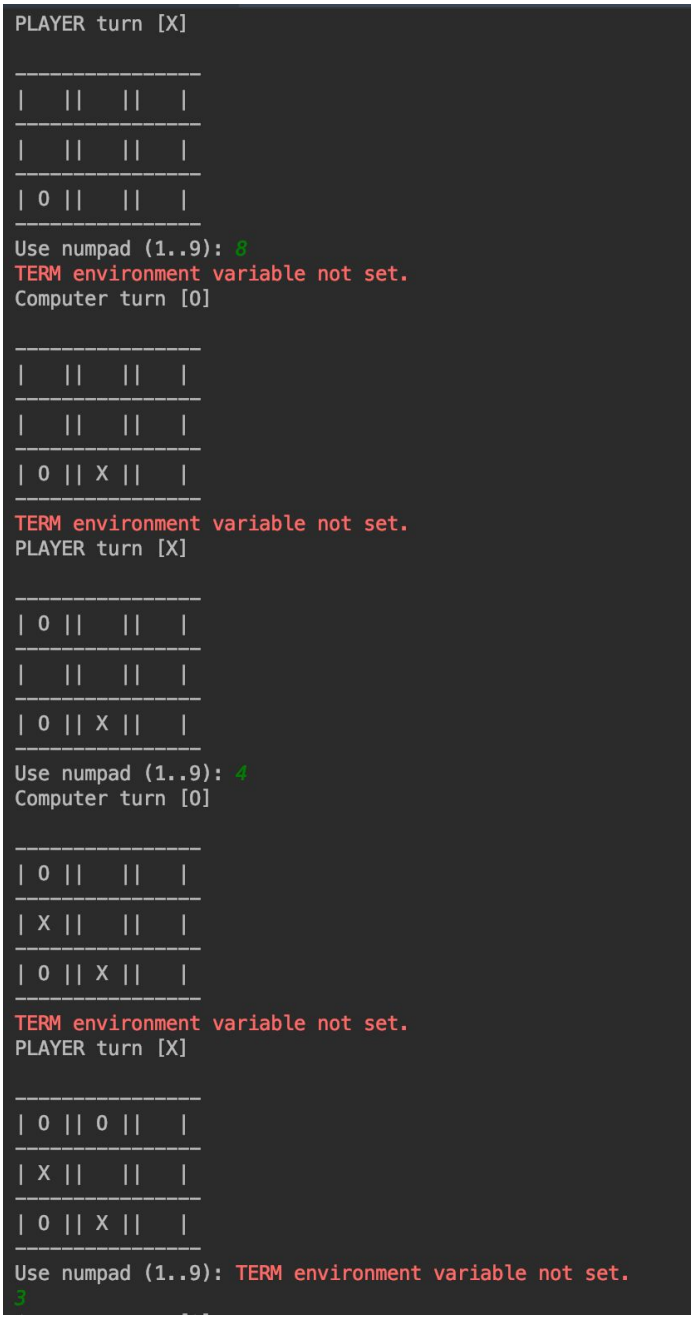
The result of this game was similar to the previous one. A friend of mine, Yousef Al-Musaibeeh, played this one attempting to best the agent. He was also pressured into playing defensively.

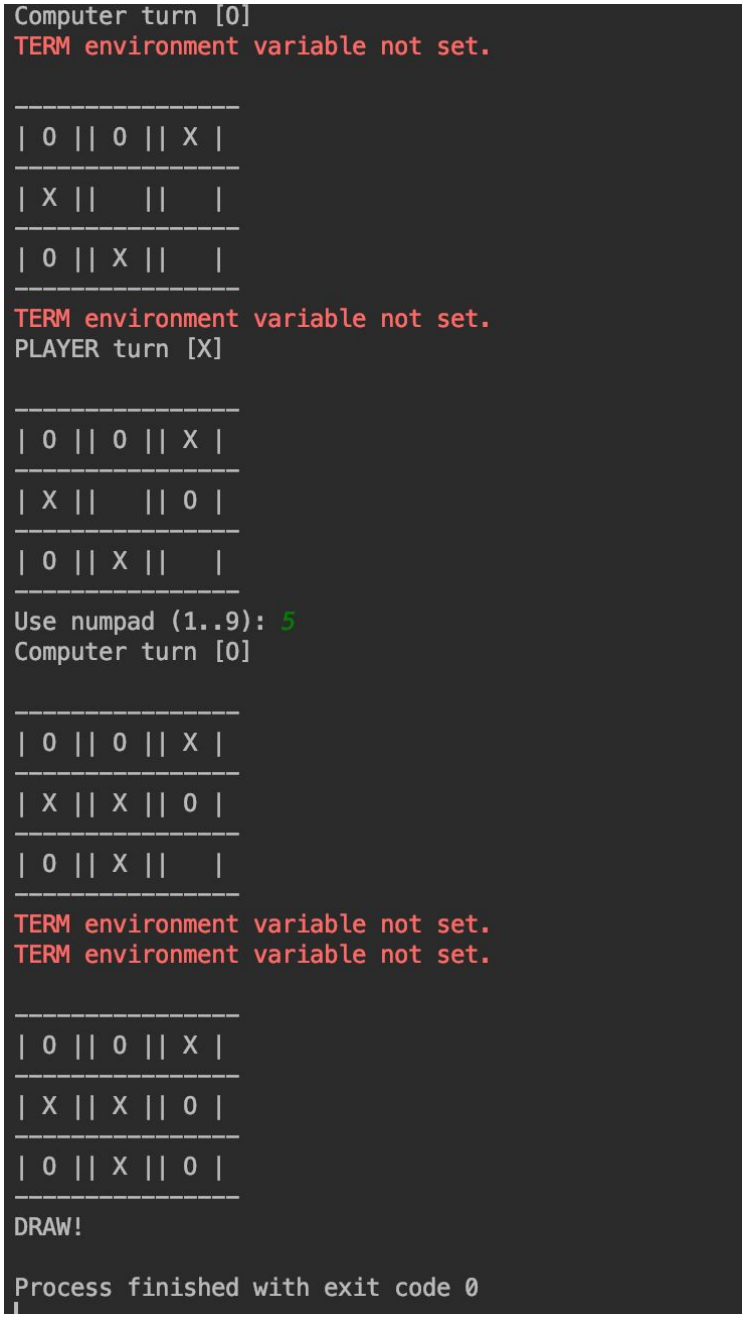
Game 4: Agent Wins



In this game, my friend decided to test out the behaviour of the agent, in which we begin to see in the 7th game board that the agent prioritized making him lose, over itself winning. Ultimately it managed to win after guaranteeing his loss.

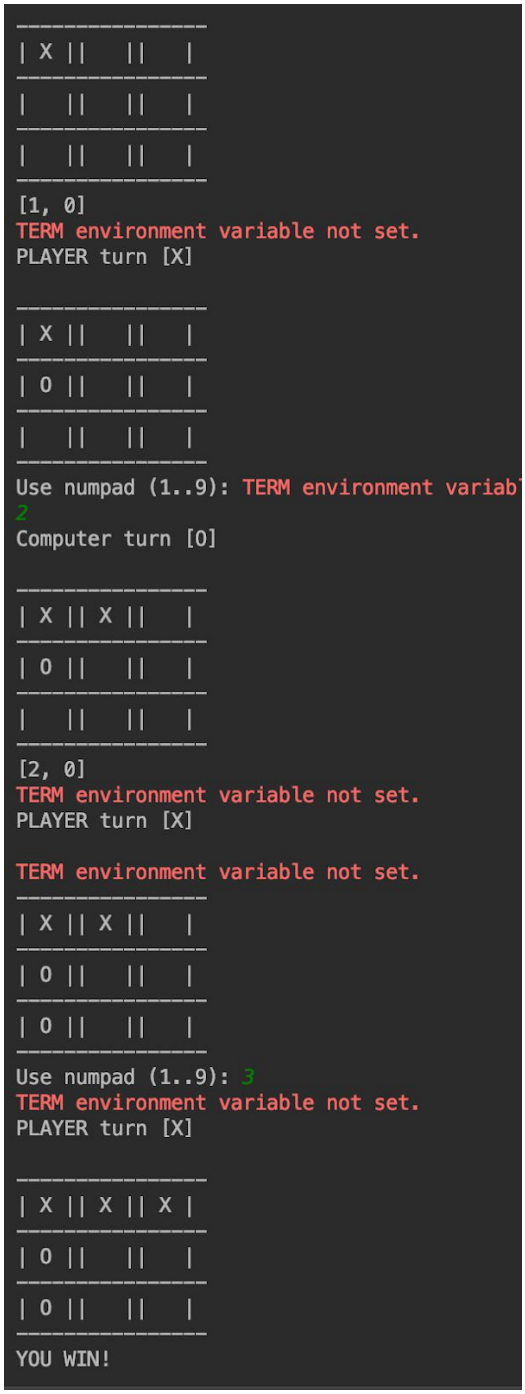
Game 5: Draw





In this game I decided once again to try again to beat the agent, but again it was futile since I was forced onto the defensive after the agents second turn.

Game 1 (Naive Agent):



We can see here that the random agent fares poorly against a real person since it is just a naive solution.

Game 2:

```
 _____________
|   ||   ||   |
 _____________
|   || O ||   |
 _____________
|   ||   ||   |
 _____________
Use numpad (1..9): TERM environment variable no
1
TERM environment variable not set.
Computer turn [O]

 _____________
| X ||   ||   |
 _____________
|   || O ||   |
 _____________
|   ||   ||   |
 _____________
[2, 1]
TERM environment variable not set.
PLAYER turn [X]

 _____________
| X ||   ||   |
 _____________
|   || O ||   |
 _____________
|   || O ||   |
 _____________
Use numpad (1..9): 2
TERM environment variable not set.
Computer turn [O]

 _____________
| X || X ||   |
 _____________
|   || O ||   |
 _____________
|   || O ||   |
 _____________
[2, 2]
PLAYER turn [X]
```

```
 _____________
| X || X ||   |
 _____________
TERM environment variable not set.
|   || O ||   |
 _____________
|   || O || O |
 _____________
Use numpad (1..9): 3
PLAYER turn [X]

 _____________
| X || X || X |
 _____________
|   || O ||   |
 _____________
|   || O || O |
 _____________
YOU WIN!
TERM environment variable not set.
```

Similar to the last game, the random choice here resulted in an easy win for myself, though it did get close to winning after placing two in a row as in the third board.

Conclusion:

To conclude, I was able to successfully create an alpha beta pruning tic tac toe AI agent. The only caveat was that my agent would not successfully prune when the player went first, instead it would fill out the board from left to right and once the row was filled it would go to the next. I wasn't able to find a solution for this, but I believe a few possible solutions include:

- A primitive, hacky solution would be to program the game such that the agent always goes first.
- A more eloquent solution would be to set the agent to a specific "player" (i.e: either X or O) and have the player always be the other "player" Then based on the rules of the game in which X always goes first, you would prune based on that.
    - In my solution I gave the player the freedom of choice since I did not expect this to be an issue, since most people disregard the rule.

Beside that fault, the pruning agent works properly if it goes first. It would at worst, tie with me resulting in a draw, in most cases, it would win, even when I tried my best to beat it. This agent felt like a much more "real" opponent compared to the naive implementation, which felt like playing versus a computer due to the random choice factor. Though it did constantly prioritize the player losing over itself winning, a fix to the way the agent generates new states in the newStateGenerator function may be a fix to this, but due to time constraints I was not able to implement this.

Citations

Wong, Aaron. "I Created an AI That Beats Me at Tic-Tac-Toe – Hacker Noon." *Hacker Noon*, Hacker Noon, 25 Feb. 2018, hackernoon.com/i-created-an-ai-that-beats-me-at-tic-tac-toe-3ea6ba22cd71.

Venugopal, Amresh. "Reinforcement Learning: Train a Bot to Play Tic-Tac-Toe." *Becoming Human: Artificial Intelligence Magazine*, Becoming Human: Artificial Intelligence Magazine, 9 Sept. 2018, becominghuman.ai/reinforcement-learning-step-by-step-17cde7dbc56c.

Kobti, Ziad & Sharma, Shiven. (2007). A Multi-Agent Architecture for Game Playing. 276 - 281. 10.1109/CIG.2007.368109.

Xiang, Dong. "Java Graphics Tutorial." *Relational Database Design*, May 2012, www.ntu.edu.sg/home/ehchua/programming/java/javagame_tictactoe_ai.html.

Dyer, Charles. "Adversarial Search AKA Games." *Csee.umbc.edu*, 2012, www.csee.umbc.edu/courses/671/fall12/notes/07/07.ppt.pdf.