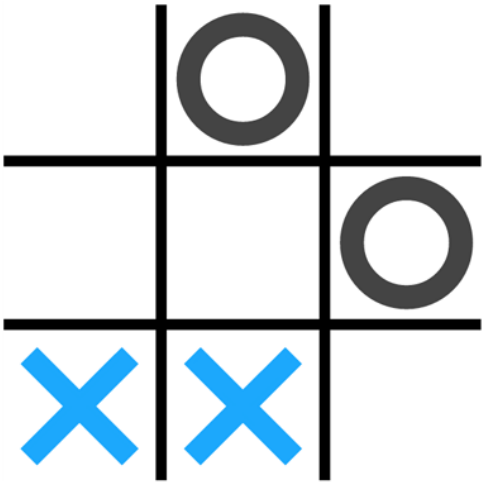
Reinforcement Learning



The game is first setup with a 4x4x4 array of empty values. From there, each player takes turns making a move. x_score and o_score hold the utility values x_board and o_board are the different grid positions. Each player first checks if they can make a move to win a game, stop the other player from winning, or catch a potential fork, and if so it makes that move. Otherwise it makes a move based on exploring vs. exploiting. After each move is made, checkWinner() determines if someone has won the game. If someone won, the utility values are updated accordingly.

Explore

The explore function is called when the game is in early stages where curiosity and learning is more rewarding than going to the square with the highest utility. The explore function looks for the square that has been explored or tried the least amount of times in previous attempts from the current game state and chooses that square for the next move in order to learn how the overall utility for the player will be impacted if that square were chosen. The equation that we utilized in order to determine the next square for the explore or exploit functionality was the equation:

$$a = argmax\ f(\sum_{s'} P(s' | s,\ a')U(s'),\ \ N(s', a'))$$

where N(s, a) is the number of times that have taken the action in a in state s and the the first portion of the equation (argmax … U(s')) chooses the action that maximizes the future utility. By using this equation, we were able to determine whether or not we would explore if we did not have a lot of information on other moves and we would do choose the square based on N(s,a). The benefits of exploring rather than exploiting using the given equation are that we are able to get a more accurate model of the environment and can uncover higher value states than we have already found. By

utilizing the equation, we were able to find game states for our tic tac toe game that previously were unknown which ultimately led to maximizing our utility.

Exploit

The exploit function goes through each of the possible moves that the player can take and makes the move that yields the highest utility for the player. This tends to occur in later stages of the game when winning yields the highest utility, over curiosity and learning for the future.

Updating Utility

The utility values are updated in the q_update_learn function. This occurs for each player after the game is won, lost, or tied. The function takes in the path used to win, the board, a learning rate, score, and a discount rate to update. The function starts with the winning move and recursively goes backwards in moves and updates the utility from there. Once it gets each move, it checks if that board exists it the played boards. If it is it updates that utility value for that state. Otherwise, it creates a utility value for that state. This continues for each of the states that occurred during the game. After this is completed, the states are reset and a new game takes place.