“Create an algorithm and write pseudocode to play the game “Noughts and Crosses” (Also known as “Tic-Tac-Toe”)”

Define the problem:

“Noughts and crosses” is a game played by two people on a 3x3 grid. Each player takes turns placing one of their respective symbols in one of the positions on the grid (noughts or crosses). The aim of the game is to create a line of 3 of your own symbols in either the horizontal, vertical or diagonal directions whilst aiming to prevent the other player from completing this first. You can tie in noughts and crosses if there is no remaining positions to place symbols in and a line of 3 has not been achieved by either player.

Describe my algorithm:

My algorithm would consist of one main loop that repeated logic for each turn. First, the algorithm would check to see if placing their symbol in a square would cause them to win. If so, then a symbol would be placed there. If not however, the algorithm would perform the check to see any positions the enemy could place their symbol to cause them to win on the next turn. If they can, then the algorithm will place a symbol in that position to prevent this. If there are no winning, or possible losing squares, then the algorithm will place the symbol randomly in one of the remaining empty squares.

Structured English:

If turn:

Loop through each empty square:

If placing a symbol in that square would cause the algorithm to win, then place a symbol in that square

Else If the opposing player placed their symbol In that square and it would lead them to win then place your own symbol there

Else ( there are no winning or losing moves ):

Choose a random empty position and place your symbol there

Evaluate effectiveness of my algorithm:

There is an opportunity here to use recursion to allow the algorithm to perfectly play the game. By “simulating” each combination of moves for each current available move, the algorithm would be able to avoid situations where the enemy has two options for winning. In the current case, the algorithm would simply look for the first one and then prevent it. Furthermore, optimisations can be done through use of a Minmax algorithm with alpha-beta pruning. This would define each position and move as being good or bad for the algorithm and eliminates obviously bad options to make the search faster. The situation in which the algorithm would fail is as follows.

A game with noughts and crosses

Description automatically generatedHere, crosses has just moved by placing a cross in the lower-left square following the nought being placed in the top square. This could happen to my algorithm as it would see no issue with the enemy placing the cross in the bottom right even though it is obvious that it causes there to be two options for crosses to now win.