ARTIFICIAL INTELLIGENCE

INTERMEDIATE REPORT

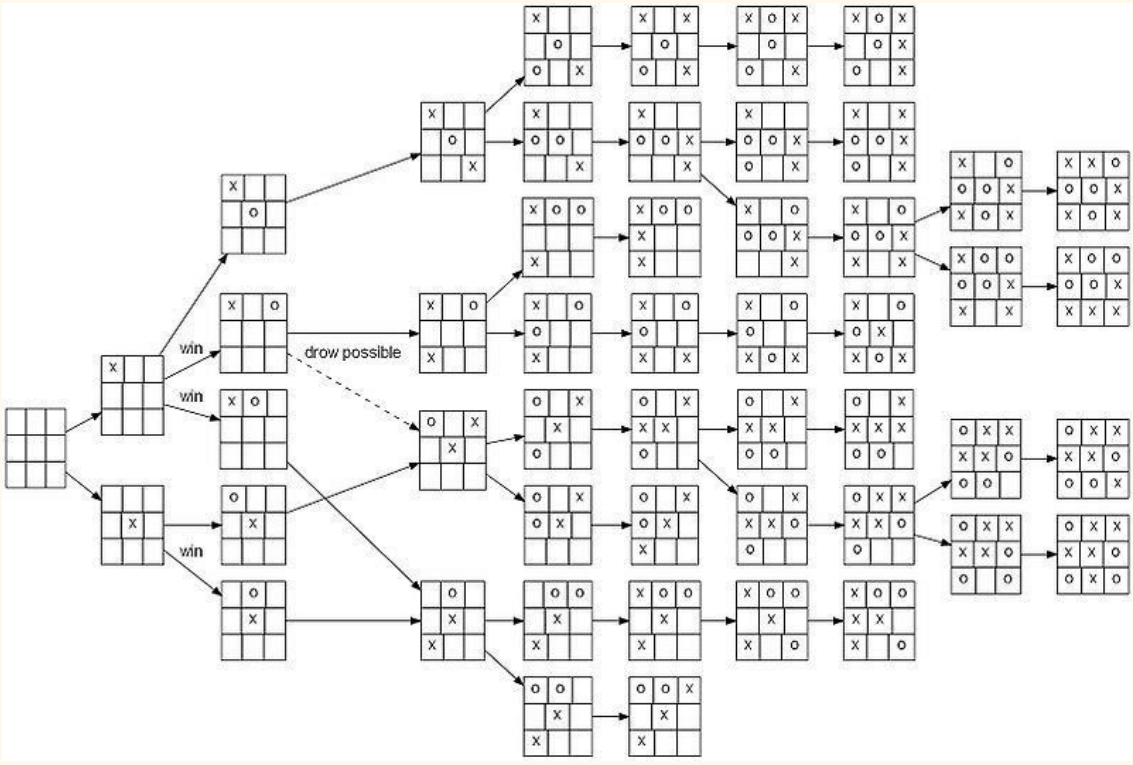
—

Team 18 - Kripa Anne (20171159) and Swaraj Renghe's (20171119) CASPER Bot

Tree Search Algorithm

Variants of MiniMax Algo with Alpha Beta Pruning; Monte Carlo Tree Search

1. About Mini Max



Minimax tree search algorithm searches through a directed graph where each node represents a state in the game. It presents an optimal method of computing best states for a two-player, zero sum game. Due to searching every node in the game tree, however, it needs to be sped up with different techniques to make it a feasible solution. It's time complexity is $O(b^m)$.

2. Possible Improvements to be made to the Base Minimax Algorithm

1. **Reducing Branching Factor** Since the complexity of the minimax algorithm is heavily dependent on the branching factor, introducing a limit on the branching factor will significantly speed up the algorithm. This can be done by sorting the next moves before increasing depth in the tree, so only the best nodes would be explored.

2. **Iterative Deepening until time limit is reached** Instead of calling the Minimax algorithm with a fixed depth, an alternative is to call it with a depth of 1, then with a depth of 2, and keep increasing the depth up to a fixed time period. This helps the Minimax algorithm limit the time taken for nodes with a large number of children, and helps it look deeper for nodes with lesser children.

3. **Alpha - Beta Pruning** This technique used in cognition with the Minimax algorithm provides a large time improvement over vanilla Minimax, without sacrificing optimality. It achieves this by stopping evaluation of a node when it detects that it is a definite worse path compared to a previously detected path. This provides large speed gains because it helps avoid computing whole subtrees.

3. Monte Carlo Tree Search Algorithm

A relatively new game AI tree search method, the Monte Carlo Tree Search presents a number of improvements over a Minimax implementation.

1. **No Evaluation Functions** It is a best first search algorithm, that runs simulations from the current node. Instead of simply running random simulations from the current board position, it uses the results of the simulations to generate a search tree, and then runs simulations from the leafs of the tree, propagating the results through the tree. In games like Tic-Tac-Toe, where writing a strong evaluation function is difficult, the Monte Carlo theorem thrives.

2. **Easy pondering** The algorithm can also be run while the opponent is thinking, further improving it's optimality. This allows it to play better while thinking lesser on it's own turn.

4. Improvement to the Monte Carlo Tree Search Algorithm

1. **Benefits Greatly from even a simple Heuristic Function** The advantage of the Monte Carlo algorithm is that it does not require a heuristic, but the presence of a heuristic function will allow the algorithm stop simulations early, before reaching a terminal state.

2. **Reinforced Learning** With an implementation for suitable Exploitation and Exploration in place, the algorithm will manage to use the results of it's previous simulations to help make future decisions, and will also explore nodes that might seem suboptimal in the hope of discovering a more optimal path down the tree.

3. **Upper Confidence Bound (UCT)** The Monte Carlo algorithm can be helped to select a node to visit next with the help of the following equation, where

$$\frac{w_i}{s_i} + c\sqrt{\frac{\ln s_p}{s_i}}$$

W_i = Number of wins after the i^{th} move

S_i = Number of simulations after the i^{th} move

C = Exploration Parameter

S_p = Total number of simulations for the parent node

A Baseline Heuristic

CASPER is primarily designed as an offensive player, so he will place more utility on obtaining a block than on defending. The approach followed is greedy in a sense, so as to prioritize winning boards over strategic positioning, due to the possibility of getting a bonus move which almost doubles chances of winning a block further into the game.

At the beginning, utilities are assigned to each cell on the BigBoards such that:

1. Centre - 3.5 points
2. Corner - 2 points
3. Edges - 1 point

Similarly, winning a block on any BigBoard (not taking into account opposing moves and positions) are:

1. Central block - 35 points
2. Corner block - 20 points
3. Edge block - 10 points

Starting Strategy (subject to change)

In the beginning, there are 81*2 possibilities for placing X. If we win the coin toss and start, we make the first move at centre of central block and keep sending the opponent to the central block till he captures it so as to get the centre in at least 2 other blocks. This would be our primary playing BigBoard. This is because a bonus move would be more beneficial towards the middle of the game than the beginning. If he mirrors the move on BigBoard2, we abandon the plan and continue with our arbitrary valuation.

If opponent starts with the same central tending move, we mirror his move on BigBoard2 and name that as our primary board. Otherwise, we follow our arbitrary search of states.

Mid-Game Plan

Our primary BigBoard would depend on a weighted function of number of X's on each board, number of patterns created by us and the opponent's future moves. During a normal move in the game, there will be at most 9*2 possibilities for movement. If we (X) have to play in a particular SmallBoard, the smallest, higher weight will be given to the primary BigBoard's SmallBoard. In a partially filled SmallBoard, a cell that wins the SmallBoard for us will be given a moderate weight and a cell that blocks the SmallBoard for the opponent will be given a lower moderate weight. In case of a block with no immediate possible move for either of us, we pick a cell in the path of the lowest Manhattan distance between 2 or more X's, or 2 or more O's and assign the cell a value of +3. A higher weight would be given to the function computing the relative value of the block we are sending the opponent to. If the block has 2+ sequential X's, we assign a -8 score to that SmallBoard cell. Else if the block has 2+ sequential O's, we assign a -7 score to the corresponding SmallBoard cell. The highest weight would be with respect to the function gauging if any player has reached any of the winning states on either BigBoard yet. At the end of each turn, we will recompute the Primary BigBoard based on new moves made by both players.

Winning States

A1	B1	C1
D1	E1	F1
G1	H1	J1

A2	B2	C2
D2	E2	F2
G2	H2	J2

All possible lines in the above 2 BigBoards are:

- Horizontal: (A, B, C), (D, E, F), (G, H, I)
- Vertical: (A, D, G), (B, E, H), (C, F, J)
- Diagonal: (A, E, J), (C, E, G)