Artificial Intelligence
Extreme Tic Tac Bot : Strategy and Heuristics

TEAM

DeMoss

TEAM MEMBERS

AadilMehdi Sanchawala (20171043)
Rohan Chacko (20171061)

Abstract

In this report we analyse the Extreme Tic Tac Toe game, and our approach of building an AI bot for the same. This report will be focussed on Adversarial Search Techniques, specifically Minimax, its variants and $\alpha\beta$ Pruning. We will also discuss the heuristics and evaluation functions proposed for the search algorithm. Further on we propose additional strategies that would be used with Minimax algorithm with $\alpha\beta$ Pruning to increase our average time per move.

Search Algorithm

The search algorithm we plan to implement is Minimax Search algorithm. We are choosing this algorithm since Extreme Tic Tac Toe is a deterministic, turn-taking, two player perfect information game or a zero sum game. The environment is fully observable in which both the players with move alternatively and the utility values in the end will always be equal and opposite. The players in the game are adversaries of one another. Hence, minimax search algorithm is chosen for the AI bot.

Advantage over other common Search Algorithms

Expectimax

Expectimax is commonly used in environments where the outcome depends on a combination of the player's skill and chance elements. Since Xtreme Tic Tac Toe is a deterministic game meaning it does not involve chance elements, Minimax would work better.

Monte Carlo Tree Search

Monte Carlo Tree Search is based on exploring the search tree based on randomly choosing moves and associating optimal weights with every move while backtracking from goal states. Though this search technique does provide optimal weights to board states, it consumes a lot of time and memory in unrolling the tree especially when the board states exponentially grow. Given the time constraints, Monte Carlo Tree Search would not be an appropriate algorithm.

Beam Search

A heuristic algorithm that works on expanding the most promising node in a set of limited nodes. Although Beam Search improves on your memory requirements, you make a huge tradeoff for the optimality of the Search Algorithm. Hence, we chose against Beam Search.

Heuristic & Evaluation Function

Key insights from Gameplay

- A player could be sent to a SmallBoard many more times than the opponent. Thus it is possible that only one player makes a move on a SmallBoard always.
- Macro view of the board dominates the micro view of the board. Thus, even if I have a winning chance in the next move to capture a SmallBoard, it doesn't imply that my next move would be to capture that SmallBoard.

Heuristic influence factors

- Winning BigBoard: Forming a Bigboard winning position pattern
- Winning a SmallBoard
- 1-step force: Forcing opponent to gift me an open move
- 1-step block: Blocking adversaries winning Bigboard position pattern
- Blocking opponent winning position in a Smallboard
- Gifting an open move to opponent

The weights are currently decided in the range [-20, 20]. Weights would be finalized after satisfactory number of games are played and analysed.

Every cell in a SmallBoard would have weights based on the number of winning patterns the corresponding cell occurs in. The cells would be weighted in the following way:

SmallBoard_Weight Matrix:

+3	+2	+3
+2	+4	+2
+3	+2	+3

Additional weights are given in the following board states:

- Winning a SmallBoard such that a winning BigBoard pattern is formed is given a weight of 20.
- Winning a SmallBoard is given a weight of 18.
- Forcing opponent to gift an open move is given a weight of 15.
- Blocking adversary from winning a SmallBoard such that consequently a winning BigBoard pattern is blocked is given a weight of 12.
- Blocking adversaries winning SmallBoard pattern is given a weight of 10.
- Gifting an open move to opponent is given a weight of -15.

Every other move made is given an additional weight 0.

Simulated Annealing combined with Minimax

We define a temperature variable which gets colder as we go deeper in the game tree. Everytime our agent decides on a move, it flips a coin with a probability relating to the current temperature and according to the result it should either take a random move OR search for an optimal move using the Minimax algorithm with Alpha Beta pruning.

The premise of this strategy is that the colder the temperature the lesser is the probability of a random move.

The rationale behind that mechanism is, that at the beginning of the game the agent's choices are less critical than choices made deeper in the game, therefore, we allowed random moves with higher probability on earlier stage.

Our hypothesis is, that such a random jumps with decreasing probability will not impair the winning rate severely, and on the same time improves the average time per move dramatically