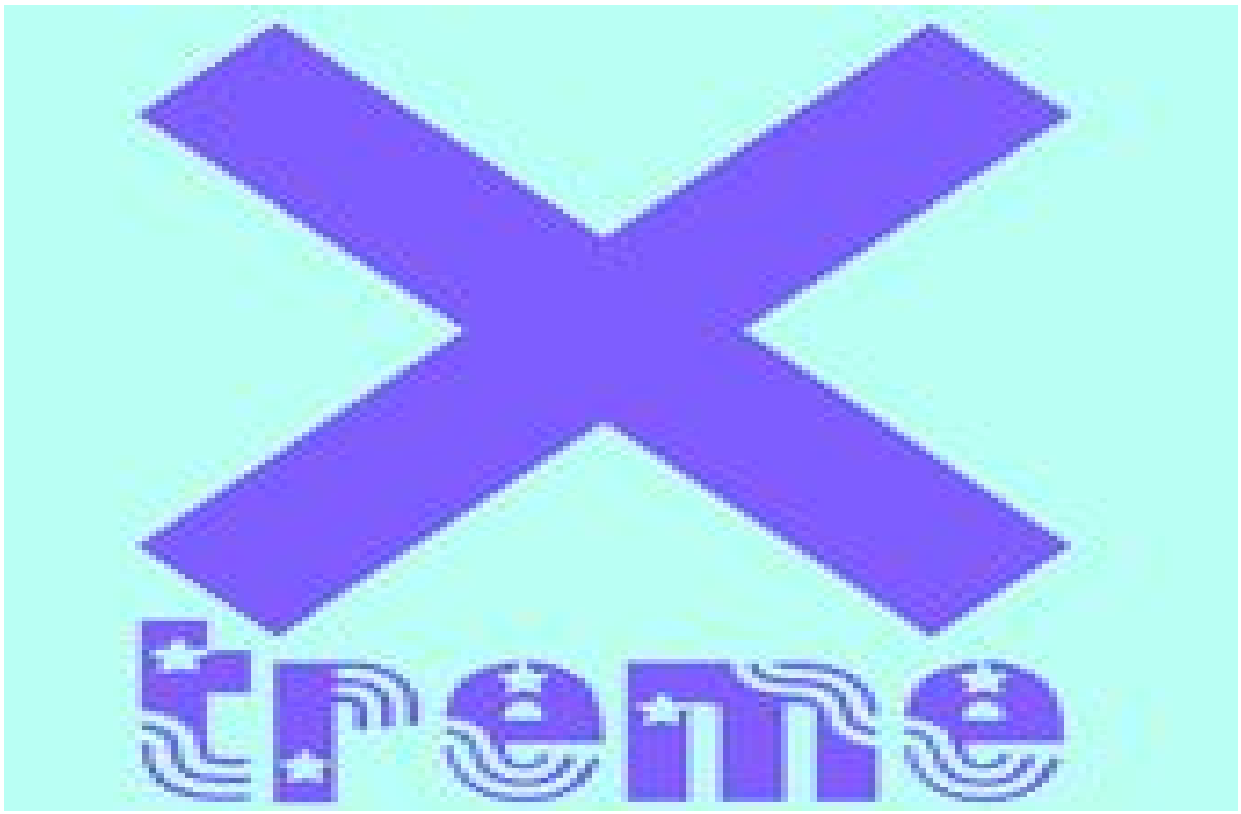
ARTIFICIAL INTELLIGENCE

NOTSOARTIFICIAL

Xtreme Tic Tac Toe Tournament

K.SV SAMHITA(2018121005)

FREYA MEHTA(20171184)



Introduction

- The Xtreme Tic Tac Toe is 3X3X3X3X2 game board.
- There are two Bigboarads of 9X9 size.
- Each Bigboard has 9 Small boards of 3X3 size.
- Each small board has 9 cells of size 1X1.

Heuristic:

- The main goal of our heuristic to maximise the value S, where S = MyScore - OpponentScore.
- We do this by calculating the weight for each board and combine them at stage.
- For this we have separate heuristics for calculating weights of cell , Smallboard, Bigboard.
- From this, We get the best possible move available.

Cell weights:

3	2	3
2	4	2
3	2	3

- Cell weight is calculated by observing the no.o f winning pattern possible including that particular cell.
- Ie:- Cell_weight(1,1) = 3 , As it can form three patterns (one in row, one in column and one diagonal)
- Cell_weight(i,j) = Cell(i,j)

SmallBoard heuristic:

- Every SmallBoard has 8 winning patterns.
- If a SmallBoard is draw or won by Opponent , the SmallBoard_weight = 0
- If a SmallBoard is won by us , the SmallBoard_weight = 50
- If it is neither of the above cases, We calculate the SmallBoard weight considering the Pattern contribution and cell contribution

- In this case, the SmallBoard has some partial patterns:
    - ➔ If the pattern is ⅓, Pattern_contribution = 1
    - ➔ If the pattern is ⅔, Pattern_contribution = 10
    - ➔ If any pattern that has an opponent or the pattern that is 0/3, Pattern_contribution = 0
- Cell contribution to the SmallBoard_weight, Cell_contribution = 0.1*cell_weight

SmallBoard_weight = Sum(Pattern_contriubution) + Sum(Cell_contribution)

BigBoard heuristic:

- Every BigBoard similar to the SmallBoard has 8 winning patterns.
- Here we consider the SmallBoard weights and the patterns formed the Smallboards which won.
- If Bigboard is already having a pattern with SmallBoards we won, that means we won the game. Ie:- BigBoard_weight = 50.
- If the Bigboard is draw or won by opponent, BigBoard_weight = 0
- If there are Partial Patterns, we calculate the pattern weight as follows:
    - ➔ If the pattern is ⅓, Pattern_contribution = 1
    - ➔ If the pattern is ⅔, Pattern_contribution = 10
    - ➔ If any pattern that has an opponent or the pattern that is 0/3, Pattern_contribution = 0
- Smallboard contribution to the weight, SmallBoard_contribution = 0.5*SmallBoard_weight.

BigBoard_weight = Sum(Pattern_contriubution) + Sum(SmallBoard_contribution)

Overall heuristic:

- ❑ Total_weight = BigBoard1_weight + BigBoard2_weight
- ❑ MyScore = Total_weight(calculated using our marker)
- ❑ OpponentScore = Total_weight(calculated using opponent's marker)
- ❑ **StateScore = MyScore - OpponentScore**

SEARCH STRATEGY :

- **MDTF algorithm** with iterative deepening search shall be used. It is more efficient than normal minimax search algorithm.
- Along with MDTF minimax algorithm we shall use alpha beta pruning to find the best move at each turn.
- **Iterative Deepening Search** (IDS) to progressively increase the search depth by 1 after every iteration.

OPTIMIZATIONS :

- **Transposition table** shall be used for making the pruning faster.
- Transposition table shall be implemented with hashing. In our case, we shall use **Zobrist Hashing** :

  Zobrist hashing is a hash function we shall use which is based on the xor operation. We store a SmallBoard position of a BigBoard on the basis of bits. We have 3 states for a particular cell :-

  1. X
  2. O
  3. Empty

  So, state of a cell can be represented by 2 bits and as there are 9 cells in a SmallBoard so total no. of bits used for representation are 2*9 = 18 for a SmallBoard. Thus a SmallBoard can be represented within 18 bits and storing the values corresponding to these in a dictionary speeds up the code .

- Zobrist Hashing shall speed up the computation since if the same state is seen again in future, its heuristic need not be calculated all over again.

- **Global Hash** - when you enter a board state, its value is added to the hash and when we return the value of the state shall be removed from the hash. We add and remove the value using XOR operation as shown in above point.
- **Our own Update function** - As the update function in simulator checks that the move is valid or not and we only call the update function on valid moves. so, we don't need to check its validity again since that is time consuming because of that we shall make our own update function. This shall saves computations in our program and hence our code should run much faster which shall eventually help the minimax function to go more deeper in the tree for finding best move.

BONUS MOVE :

It is mentioned in rules that if you win any SmallBoard in game then you get a single bonus move (i.e if you win a SmallBoard again in bonus move, you won't get a bonus move again). As in last stages of game most of the SmallBoard are just one or two move ahead from ending so, the significance of bonus move increases. Whenever during search, if the bot wins a SmallBoard, if it is at a max node, it consider its child node also as max nodes instead of min nodes. Similarly, if bot wins a SmallBoard at a min node, it consider its child nodes as a min nodes instead of max nodes.