ENDGAME(Team-8)
Nonidh Singh(20171203)
Sarthak Singhal(20171091)

AI(Tic Tac Toe Bot)

Deadline(25/2/19)

OVERVIEW

So basically in this assignment we have to implement an AI bot which would compete in a tic tac toe competition with other bots.

PROBLEM STATEMENT

- The Extreme Tic Tac Toe is an extension of 3x3 Ultimate Tic Tac Toe which in turn is an extension of standard 3x3 Tic Tac Toe. Extreme Tic Tac Toe comprises of 2 boards of 3x3 board in which each cell further is a 3x3 board.
- The game is between two teams.
- Coin is flipped to choose who will move first.
- Player 1 marks 'x' and Player 2 marks 'o' on the board.
- The player who makes a legitimate pattern wins the whole board.
- Refer to the PDF for more details.

SPECIFICATIONS

HEURISTIC

DETAILS

- So the heuristic we are planning to use is as follows:
- In each 3X3 normal board we consider all 8 winning combinations.
- They are r1, r2, r3, c1, c2, c3, d1 and d2.
- With each winning combination we can have 4 possible outcomes:
  - We are winning(eg X _ _)
  - Opponent is winning(eg O _ _)
  - Open for both(eg _ _ _)
  - Closed for both(eg X O _)
- We give certain values to each state:

    - 1 for winning
    - -1 for losing
    - ½ for both
    - 0 for none
- So for every small board we sum all the combinations and get utility for a small board.
- We do this for every 3X3 board(18 such boards) and get the utility for a particular state.
- In small boards we do the same but give different weights to different cells and then:
    - We may add it to utility.
    - Or give every 3X3 cell a weight equal to utility in small board.
- For deciding in which big board to play we are planning:
    - If opponent moves first, then move in the same board in which the opponent moves.
    - If we move first, then move in different board.
- For critical conditions:
    - If we have 2 same symbols in any winning combination, give it some weight say 10/-10.
    - If we have 3 same symbols in any winning combination, give it some weight say 100/-100.
- For extreme conditions(winning a combination in small board):
    - Flag the cells of small board and try to move in them only and finish the game early.
    - If opponent has any winning combination, try to win the cell or prevent him from winning in that cell.

WHY USED?

- We decided upon this heuristic by breaking the game into small 3X3 boards and then thinking on how to win it.
- So winning in a small board is quite intuitive:
    - We try to capture center(most paths can be made).
    - Then corner cells.
    - If opponent has made 2 moves in a winning combination, we prevent him from winning.
    - If 3 moves made in any combination, then the game is over which implies high utility.
- Later we'll give certain weights to each cell which would give preference to one cell over the other.
- So this heuristic covers the initial stage of the game when there is not much happened by occupying important positions in each cell.

- In later stages of the game, it will help us to move in best possible way by looking forward with the help of search algorithms.

SEARCH ALGORITHM

ALGORITHM USED AND IMPLEMENTATION DETAILS:

- We are using **alpha beta pruning** as an extension of the **minimax** to go as deeper as we can in the allotted time slot.
- Alpha beta pruning helps to remove redundant subtrees from the search path.
- We are improving upon the complexity of alpha beta using techniques of **MTD(f)** which is short window version of the alpha beta pruning.
- MTD(f) derives its efficiency by only performing zero-window alpha-beta searches, with a "good" bound (variable beta).
- To find the minimax value, MTD(f) calls AlphaBeta a number of times, converging towards it and eventually finding the exact value.
- A **transposition table** stores and retrieves the previously searched portions of the tree in memory to reduce the overhead of re-exploring parts of the search tree.
- Also to improve upon the search space we will use alpha beta pruning with an increasing depth like in case of the **idfs (iterative depth first search)** so that we can step by step reach a better depth efficiently.
- For the transposition table we are storing the states in the hash table using the **zobrist hashing**.
- The transposition table will be used for quick lookups in optimal states and starting and ending of the game.
- It will also remember the good moves so that when the bot encounters such path again it can return the value based on history rather than searching for the best node again in the path.
- We also plan to use some **learning techniques** inspired from the monte carlo tree search algorithm so that the bot can improve its performance over time.

ADVANTAGES:

- The advantages of the minimax based algorithms like alpha beta and its other optimal versions is that with a strong heuristic function it can cutoff search efficiently and result in most suitable action.
- Also a good heuristic can be used to counter wild swings that may occur due to opposition moves.
- It is better in comparison with some learning algorithms as it does not need extensive training of the model.

DISADVANTAGES

- ➔ The disadvantage of alpha beta is that for good evaluation of the game state it requires a very strong and optimal heuristic that requires very good domain knowledge.
- ➔ On the other hand the learning counterparts don't have such requirements and they can perform efficiently with better training.
- ➔ The other backlog is that it does improve upon its performance as the other learning algorithms do.