05/10/24

PAGE NO : 1
DATE :

Artificial Intelligence

Implement Tic-Tak-Toe

Algorithm:

1. Initialize the Board:
   Create a 3x3 grid (or a dictionary) to represent the board, initializing all positions are empty.

2. Define Functions:
   * Print Board: Display the current state of board.
   * Check Space Free: Determine if a given position on the board is free (i.e, empty)
   * Check Win: Check if either player has won the game by examining all wining combinations (rows, columns, diagonals).
   * Check Draw: Check if the board is full and there is no winner, indicating a draw.
   * Insert letter: Place the player's or bot's letter (X or O) in the specified position if its free and check for win or draw.
   * Minimax Algorithm:
     - Implement the minimax Algorithm to evaluate the best possible move for the bot.
     - The bot ~~maximizes~~ its score while minimizing the player's score through recursive evaluation of future game states.

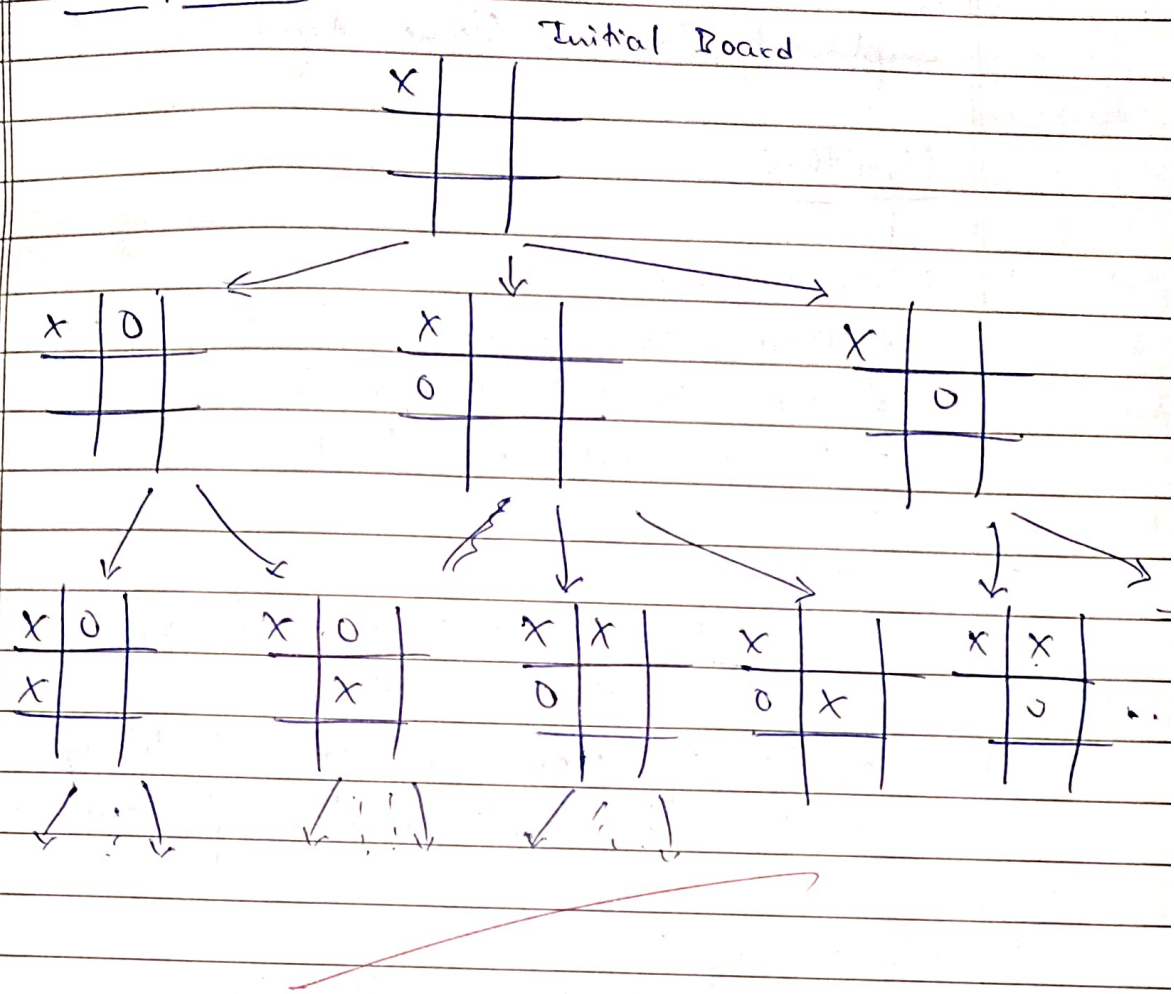
3. Game loop:
While the game is not over (no winner or draw):
- Prompt the player to make a move by entering a position (1 to 9)
- Call Insert letter for the player's move.
- If the game is still ongoing, call the Computer move function.
  * Use the minimax function to determine the best move for the bot.
  * call Insert letter for bot's move.
  * Check for win or draw after every move.

4. End conditions:
- If the player wins, display a message indicating the player won.
- If the bot wins, display a message that bot won
- If the game is a draw, display a message indicating the draw.

Time Complexity:

$$TC_{(tic\text{-}tac\text{-}toe)} : O(n)$$

State Space Tree:



Implement Vaccuum Cleaner Agent:

Algorithm:

1. Initialization:
   - Define the goal-state to indicate the cleanliness of each location (A and B)
   - Set an initial cost to zero

2. User Input:
   - Get the current location of the vaccuum cleaner (either A or B)
   - Get the cleanliness state of the current location
   - Get the cleanliness state of other location.

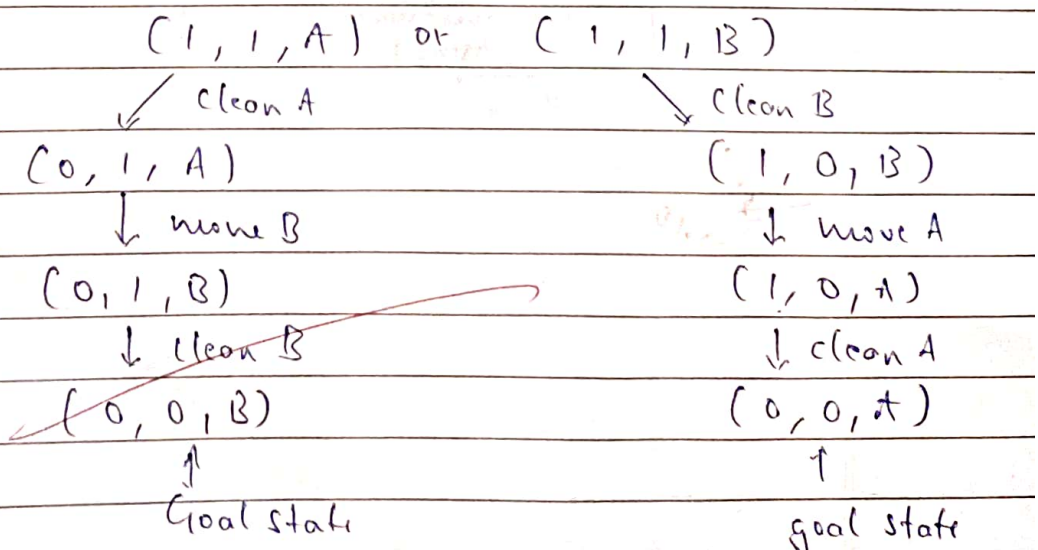
3. Cleaning logic:
   - check the location of the vaccuum.
     - if the vaccuum is on location A:
       - if location A is dirty (status_input is '1'):
         - clean location A, update goal-state, and increase cost.
       - if location B is dirty (status_input_complement is '1'):
         - move to location B, increase cost.
         - clean location B, update goal-state, and increase cost.
     - if location A is clean (status_input is '0'):
       - if location B is dirty, move to location B, increase cost, and clean it.
       - if location B is clean, do nothing.

PAGE NO :
DATE :

- if ~~location~~ vacuum is in location B:
  - if loc B is dirty (status_input is '1'):
    - clean loc B, update goal-state, and increase cost.
    - if loc A is dirty (status_input - complement is '1'):
      - move to loc A, increase cost
      - clean loc A, update goal-state and increase cost.
  - if loc B is clean (status_input is '0'):
    - if loc A is dirty, move to loc A, increase cost and clean it.
    - if loc A is clean, do nothing.

4. Output:
   - Print the final goal-state to show cleanliness of both locations.
   - Print ~~the~~ total performance measurement (total cost incurred).

State Space Tree:



Output:

Enter location of Vacuum : A
Enter status of A : 1
Enter status of other room : 1
Initial location Condition : {'A' : 0', 'B' : 0'}
Vacuum is placed in location A
Location A is Dirty.
Cost for cleaning A : 1
Location A has been Cleaned
location B is Dirty
Moving right to location B
Cost for moving right : 1
Cost for Suck : 3
Location B has been Cleaned
GOAL state :
{'A' : 0', 'B' : 0'}
Performance Measurement : 3

Time Complexity:

TC (Vacuum Cleaner Agent) = O(1)

01.10