01/10/24

Artificial Intelligence

Ex 01 Tic-Tac-Toe game implementation.

Algorithm

01. Initialize the game Board
   → create a dictionary 'board' with key from 1 to 9, all initialized to empty space (' ').

02. Define Helper functions
   → print_board(board): print the current state of the board.
   → space_free(pos): Return True, if position 'pos' is free, otherwise return false.
   → check_win(): check all possible win conditions. (horizontal, vertical, diagonal) and return True.
   → check_draw(): check if all spaces are filled and no player has won, Return True. if it's draw.

-> insertLetter(letter, position):
place the letter (either 'x' or 'O') on the board at the specified position.
check for win or draw.

3. Define the Maximum Algorithm.
-> if the computer ('x') has won
return 1
-> if the player('O') won,
return of -1
-> if the board is full (draw),
return a score of 0.

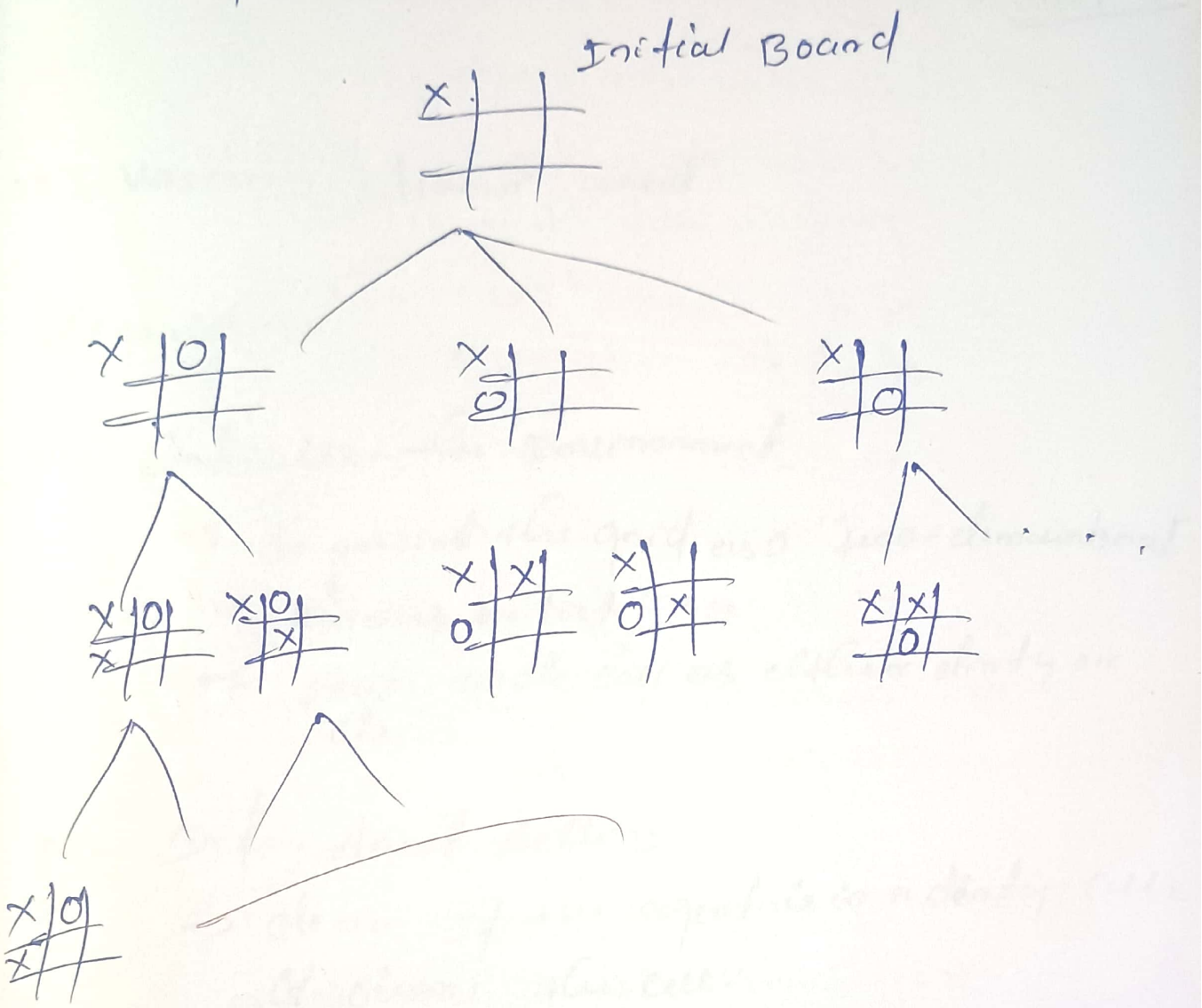
4. Game loop:
-> while the game is not won:
- call comp_move() to calculate the computer move.
- check if the computer has won, if so, exit the loop.
- call "player_move()" to allow the player to make a move.

5. End of Game.
-> If a player wins, display the winning message.
-> If it's draw, display a draw message.

TC = O(n)

Time complexity.

State space Tree:



01/10/24

02) Vaccum cleaner agent.

Algorithm:

01. Initialize the Environment
   -> Represent the grid as a two-dimensional array or list
   -> Make each cell as either dirty or clean.

02. Define Agent Actions
   -> clean: If the agent is in a dirty cell, it cleans the cell.
   -> move up: move the agent one cell up
   -> move down: move the agent one cell down
   -> move left: move the agent one cell left
   -> Move left: move the agent one cell right.

03. Define Sensing:
   -> The agent can sense whether it's current cell is dirty or clean.
   -> The agent can determine it's position relative to the grid boundaries.

4. Agent's Decision-making process.

→ while there are dirty cell in the grid

1. If the current cell is dirty, perform the clean action
2. If the current cell is clean, check for adjacent dirty cells.
   - → If the adjacent cell is dirty, move to that cell.
   - → if no adjacent dirty cell exist
3. Continue this process until all cells are clean.

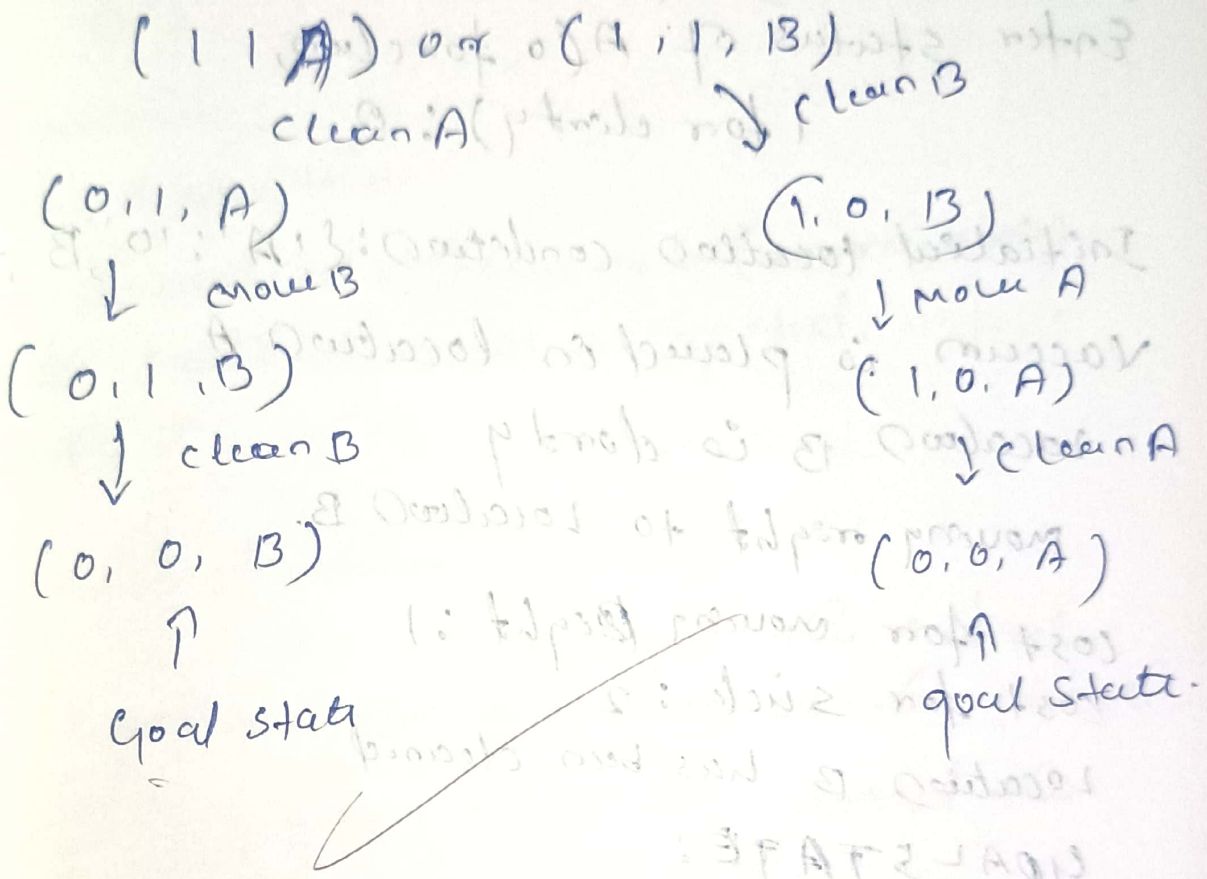
5. Implementation steps.

→ loop until All cells are clean

* check the current cell
* If dirty: clean the cell.
* If clean: check adjacent cell.
* move to the first adjacent dirty cell found.
* If no adjacent dirty cells, use a movement strategy.

output

Goal state Space Tree:



output

Enter location of vaccum (A or B) : A
Enter status of A (0 for clean,
1 for dirty) : 0

Initial location condition : {'A' : '0', 'B' : '0'}
vaccum is placed on location A
location B is dirty
moving right to location B.
cost for moving right : 1
cost for suck : 2
Location B has been cleaned
GOAL STATE :
{'A' : '0', 'B' : '0'}
performance : 2

| TC = O(1) |

Time complexity.

01.10