INDEX

Name : Govind   Class :

Section :   Roll No. :   Subject : AI

Sl. No.	Date	Title	Page No.	Teacher's Sign. / Remarks
01	01/10/24	Tic-Tac-Toe game implementation	01	01.10
02	01/10/24	vaccum cleaner agent.	04	01.10
03	07/10/24	Solve 8 puzzle problem using BFS and BFS	05	08.10

Artificial Intelligence :
_____ × _____

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 function

  → 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.

-> insert Letter(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 score.

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:

Initial Board

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 adjacny dirty cells.

    → If the adjacny cell is dirty, move to that cell.

    → if no adjacny dirty cell exid.

3. Continue this process untoll all cells are clean.

5. Implementation steps.

→ loop untoll All cells are clean
   * check the current cell
   * If dirty : clean the cell.
   * If clean : check adjacny cell.
   * move to the first adjacny. dirty cell found.
   * If no adjacny dirty cells, use a movement stratagy.

Open state span Tru!

(1 1 A)   o   (A , 1 , B)
clean A   clean B

(0 , 1 , A)   (1 , 0 , B)
↓ move B   ↓ move A

(0 , 1 , B)   (1 , 0 , A)
↓ clean B   ↓ clean A

(0 , 0 , B)   (0 , 0 , A)
↑   ↑

Goal state   goal state.

output

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

Initialied location condition : {'A' : '0', B : '0'}

vaccum is placed in 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 complexitify.

01-10