

## LAB1:

### 1} IMPLEMENTATION OF TIC TAC TOE:

```
import math

def print_board(board):
    print("\n")
    for row in board:
        print(" | ".join(row))
        print("-" * 9)
    print("\n")

def check_winner(board, player):
    # Check rows
    for row in board:
        if all(cell == player for cell in row):
            return True

    # Check columns
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True

    # Check diagonals
    if all(board[i][i] == player for i in range(3)):
        return True
    if all(board[i][2 - i] == player for i in range(3)):
        return True

    return False

def is_full(board):
    return all(cell != " " for row in board for cell in row)

def player_move(board):
    while True:
        try:
            move = int(input("Enter your move (1-9): ")) - 1
            if move < 0 or move > 8:
                raise ValueError
            row, col = divmod(move, 3)
            if board[row][col] == " ":
                board[row][col] = "X"
                break
        else:
```

```

        print("Cell already taken. Try again.")
    except (ValueError, IndexError):
        print("Invalid input. Please choose a number from 1 to 9.")

def minimax(board, depth, is_maximizing):
    if check_winner(board, "O"):
        return 1
    if check_winner(board, "X"):
        return -1
    if is_full(board):
        return 0

    if is_maximizing:
        best_score = -math.inf
        for r in range(3):
            for c in range(3):
                if board[r][c] == " ":
                    board[r][c] = "O"
                    score = minimax(board, depth + 1, False)
                    board[r][c] = " "
                    best_score = max(score, best_score)
            return best_score
    else:
        best_score = math.inf
        for r in range(3):
            for c in range(3):
                if board[r][c] == " ":
                    board[r][c] = "X"
                    score = minimax(board, depth + 1, True)
                    board[r][c] = " "
                    best_score = min(score, best_score)
            return best_score

def computer_move(board):
    best_score = -math.inf
    best_move = None
    for r in range(3):
        for c in range(3):
            if board[r][c] == " ":
                board[r][c] = "O"
                score = minimax(board, 0, False)
                board[r][c] = " "
                if score > best_score:
                    best_score = score
                    best_move = (r, c)

```

```

    if best_move:
        board[best_move[0]][best_move[1]] = "O"

def main():
    board = [[" " for _ in range(3)] for _ in range(3)]
    print("Welcome to Tic-Tac-Toe!\nYou are X. Computer is O.")
    print_board(board)

    while True:
        player_move(board)
        print_board(board)
        if check_winner(board, "X"):
            print("You win!")
            break
        if is_full(board):
            print("It's a draw!")
            break

        print("Computer's move:")
        computer_move(board)
        print_board(board)
        if check_winner(board, "O"):
            print("Computer wins!")
            break
        if is_full(board):
            print("It's a draw!")
            break

if __name__ == "__main__":
    main()

```

OUTPUT:

Welcome to Tic-Tac-Toe!  
You are X. Computer is O.



```
| | |  
-----  
| | |  
-----  
| | |  
-----
```

Enter your move (1-9): 1

```
X | | |  
-----  
| | |  
-----  
| | |  
-----
```

Computer's move:

```
X | | |  
-----  
| O |  
-----  
| | |  
-----
```

Enter your move (1-9): 2

```
X | X |  
-----  
| O |  
-----  
| | |  
-----
```

Computer's move:

Computer's move:

```
x | x | o
-----
  | o |
-----
  |   |
-----
```

Enter your move (1-9): 4

IMPLEMENTATION OF VACCUM CLEANER:

```
import random
import time

class Environment:
    def __init__(self):
        # Each room is either "Dirty" or "Clean"
        self.rooms = {
            "A": random.choice(["Clean", "Dirty"]),
            "B": random.choice(["Clean", "Dirty"])
        }
        self.agent_location = random.choice(["A", "B"])

    def show_environment(self):
        print(f"Location: {self.agent_location}")
        print(f"Room A: {self.rooms['A']}, Room B: {self.rooms['B']}\n")

class VacuumAgent:
    def __init__(self, environment):
        self.environment = environment
        self.score = 0

    def clean(self):
        room = self.environment.agent_location
```

```

        if self.environment.rooms[room] == "Dirty":
            print(f"Cleaning room {room}...")
            self.environment.rooms[room] = "Clean"
            self.score += 1
        else:
            print(f"Room {room} is already clean.")

    def move(self):
        # Switch location
        self.environment.agent_location = "B" if
self.environment.agent_location == "A" else "A"
        print(f"Moving to room {self.environment.agent_location}...")

    def run(self, steps=4):
        for step in range(steps):
            print(f"Step {step + 1}:")
            self.environment.show_environment()
            self.clean()
            self.move()
            time.sleep(1)
        print("\nFinal State:")
        self.environment.show_environment()
        print(f"Total cleaned: {self.score} room(s)")

# Run the simulation
env = Environment()
agent = VacuumAgent(env)
agent.run()

```

OUTPUT:



Step 1:

Location: A

Room A: Dirty, Room B: Dirty

Cleaning room A...

Moving to room B...

Step 2:

Location: B

Room A: Clean, Room B: Dirty

Cleaning room B...

Moving to room A...

Step 3:

Location: A

Room A: Clean, Room B: Clean

Room A is already clean.

Moving to room B...

Step 4:

Location: B

Room A: Clean, Room B: Clean

Room B is already clean.

Moving to room A...

Final State:

Location: A

Room A: Clean, Room B: Clean

Total cleaned: 2 room(s)