PAGE NO :
DATE :

```
if move < 0 or move > 8:
    raise ValueError
row, col = divmod(move, 3)
if board[row][col] == " ":
    return row, col
else:
    print("Invalid input, please enter a
          number 1 and 9.")

def computer_move(board):
for move in get_available_moves(board):
    board[move[0]][move[1]] = "O"
    if check_winner(board) == "O":
        return move

return random.choice(get_available_moves
                     (board))

def main():
    board = [[" " for in range(3)] for i in

while True:
    print board(board)
    row, col = human move(board)
    board[row][col] = "X"

if check_winner(board) == 'X':
    print board(board)
    print("congratulation! you win!",
    break
```

PAGE NO :

DATE :

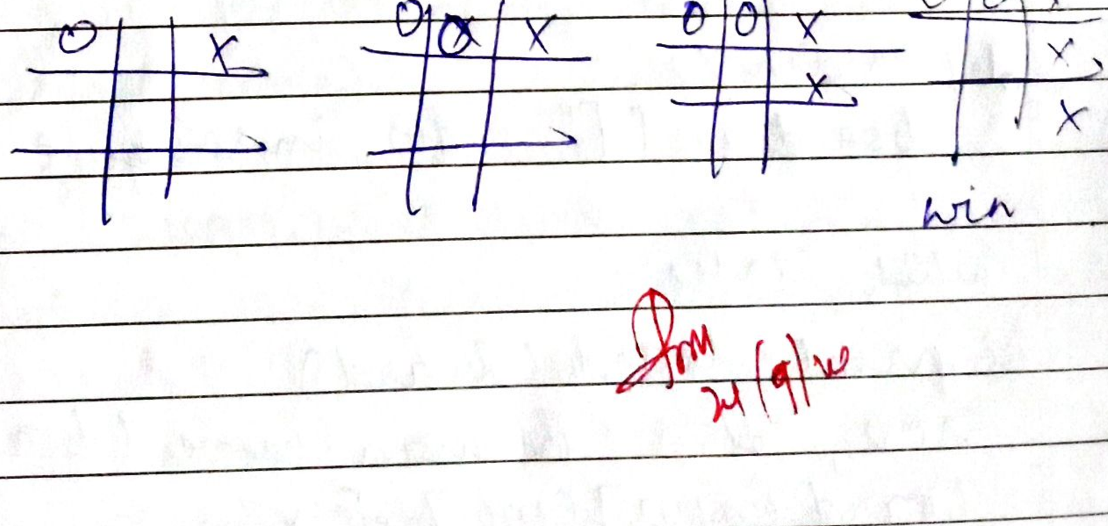
```
if is_board_full(board):
    print_board(board)
    print("It's a tie!")
    break

print("computer's turn: ")
row, col = computer_move(board)
board[row][col] = "O"

if check_winner(board) == 'O':
    print_board(board)

if _name_ == "_main_":
    main
```

output



3/10/24

PAGE NO :
DATE :

LAB - 2

Vacuum cleaner Algorithm.

Step 1 : Start.

Step 2 : select two rooms A and B.

step 3 : Take input from the user about clean/dirt status of the room and location.

step 4 : if A is clean move to Room B
if A is dirty suck the dirt

if B is clean move to Room A
if B is dirty suck the dirt.

do it until both Room A and Room B becomes clean,

step 5 : End

Initial position A.

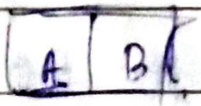
Percept sequence

Room (Location)	status	action.
A	dirty	clean (suck
A	clean	move to B
B	dirty	suck
B	clean	move to A

A[dirty] B[dirty] clean A

PAGE NO :
DATE :

Percept sequence	action
A [dirty] B [dirty]	clean A, move to B
A [clean] B [dirty]	clean B, move to A
A [dirty] B [clean]	clean A, move to B
A [clean] B [clean]	Do successful cleaning.



PAGE NO :
DATE :

8/10/24.

algorithm

step 1: start
step 2: assign initial and Goal state.

initial state = {
[1 2 3]
[4 0 5]
[7 6 8] }

goal_state =
[1 2 3]
[4 5 6]
[7 8 0]

step 3: calculate manhattan distance

```
for i in rang(3):
    for j in range(3):
        title = self.state[i][j]
        if title != 0:
            goal_x, goal_y = divmod(tile-1, 3)
            total_distance += abs(goal_x - i)
```

return total distance.

step 4: Get possible move by finding the position of empty space and swapping it with neighbors

PAGE NO :
DATE :

Step 5: find_empty_space function:

finds and returns the co-ordinates of empty space in the current state.

dfs_with_manhattan (Initial_state, goal_state)

stack = (Node (initial_state))
visited = set()

while stack

step 6: construct solution Function:
Backtracks from the goal node to the start node to reconstruct the sequence of moves leading to the solution.

Proceed

15/10

PAGE NO :
DATE :

Iterative deepening Algorithm

Implementation

Algorithm :- // returns true if the destination is reachable within the max depth.

step 1: start
step 2: input the maximum depth for a graph
step 3: get the goal destination
step 4: bool IDDFS(src, target, limit)
for i from 0 to max_depth

DFS

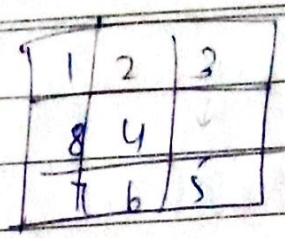
```
{
    if DLS( src, target, limit) == true
        return true ;
    else
        return false
}

bool DLS ( src, target, i )
    if ( src == target )
        return true

    else
        return false ;

    foreach adjacent i of src
        if DLS ( i, target, l-1 )
            return true

return false.
```



8 1 2
4 3
7 6 5

8 1 2
4 3
7 6 5

8 2
4 1 3
7 6 5

8 1 2
4 3
7 6 5

4 8 2
1 3
7 6 5

8 puzzle a* algorithm

step 1 : start
step 2 : create a puzzle with 9 tiles
3×3 board
step 3 : Input initial and goal states
step 4 : Heuristic function
Total cost function ''
~~f(g) = h(g) +~~
$f(n) = g(n) + h(n)$

step 5 : Keep track of visited states
step 6 : choose the state with
least total cost
step 7 : Repeat until we reach final
state

step 8 : stop.

10/10/24

22/10/24

PAGE NO :
DATE :

Simulated annealing algorithm.

step 1: start select the maximum no of iterations for each temp

step 2: set T = T_initial
T_optimal = T_current

while T > T_min
for i from 1 to max_iteration

new solution x_new is found by slightly modifying x_current

ΔE = f(x_new) - f(x_current)

if (ΔE < 0)
accept x_new solution
x_current = x_new.

else
accept x_new with probability
P = exp(-ΔE / T)
if accepted
x_current = x_new

If f(x_current) < f(x_best):
update x_best = x_current

Reduce the temperature : T = α * T

return x_best as best solution.

PAGE NO :
DATE :

29/10

8-Queens A* implementation.

Algorithm.

step 1: start with an empty board.

step 2: successor Function : From the current state generate the next state by placing a queen in the next row, only valid column positions.

step 3: Heuristic h(n)
use a heuristic that estimates the no of attacking pair.

step 4: cost Function 'g(n)
since adding a queen to the board is a single step, we can use g(n) = depth of the state

step 5: Evaluation function $f(n)$
$f(n) = g(n) + h(n)$

step 6: Goal state $h(n) = 0$
the Goal state is reached.

PAGE NO :
DATE :

N Queens Problem using hill climbing algorithm.

step 1: Initialization
Generate a random initial state, which is a list of integer representing the positions of the queens on the chessboard

step 2: calculate attacks.
Define function that counts how many pair of queens are attacking each other.

step 3: hill climbing.
Generate all possible neighbor state by moving each queen to every column in its row, excluding its current position
calculate no of attack of each neighbor state
select the neighbor with fewest attacks.

step 4: If solution is found with 0 attacks then break out of the loop early.

21/10/24

$P \wedge Q$.

12/11/24 Propositional - Logic

Knowledge base
Alice is mother of Bob (P)
Bob is father of charlie (Q)
A father is a parent (R)
A mother is a parent (S)
All parents have children (T)
If someone is parent, their children are siblings (U)
Alice is married to David (

Hypothesis.

charlie is a sibling of Bob

Alice is mother of Bob
Bob is father of charlie
Alice is a parent of Bob
Bob is a parent of charlie

so charlie is a sibling of Bob

Entailment process
Alice is mother of Bob
Bob is father of charlie
so Alice and Bob are parents
All parents have children
and if someone is parent, their

PAGE NO :
DATE :

children are siblings

~~Hence~~

conclusion: Hence charlie is a sibling of Bob

$P \rightarrow S$

$Q \rightarrow R$

$R \rightarrow U$

$S \rightarrow U$

Hence charlie is a sibling of Bob

11/4/24

PAGE NO :
DATE :

Algorithm : unify (E_1, E_2) — x, y

step 1 : if ~~E_1, E_2~~

if x , or y is a variable or constant,

a) if x and y are identical, then return NIL

b) Else if x is a variable

a. then if x occurs in y then return failure

b. else return $\{(y/x)\}$

c. else if y is a variable

a. if y occurs in x then return failure

b. else return $\{(x/y)\}$

c. else return Failure

step 2 : if the initial predicate symbol in x & y are not same, then return failure

step 3 : if x & y have a different no of arguments, then return Failure

step 4 : set substitution set (SUBST) to NIL

Ans 19/1/24

PAGE NO :
DATE :

2/12/24

Forward Reasoning Algorithm.

function FOL-FC-ASK (KB, α) returns a substitution or false

input : KB, the knowledge base, a set of first order definite clauses α, the query, an atomic sentence.

local variables: new, the new sentences inferred on each iteration

repeat until new is empty

new $\leftarrow$ { }

for each rule in KB do

$(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow$ standardize-variables (rule)

for each θ such that SUBST (θ, $p_1 \wedge \dots \wedge p_n$) = SUBST ($\theta$, $p_1' \wedge \dots p_n'$)

for some $p_1', \dots, p_n'$ in KB

$q' \leftarrow$ SUBST (θ, q)

if q' does not unify with some sentence already in KB or new then

add q' to new

$\phi \leftarrow$ unify (q', α)

if ϕ is not fail then return ϕ

add new to KB

return false.

2) N-queens Implementation using Alpha-Beta Algorithm

3) Min max Algorithm

=) Back Tracking Algorithm (Depth first

=) Best move strategy. (best move)

=) Max will try to maximize the utility
Min will try to minimize the utility of opponent (worst move)

2) Algorithm N queens using Alpha-beta pruning

1. Initialize the Board
   create a board size N and set $\alpha = -\infty$ and $\beta = +\infty$.
2. Recursive Function
   Recursive function is used to place queens row by row
3. Best case
   If all queens are placed (row == N), a valid solution is found.
4. Pruning check.
   for each potential ~~check~~ column in the current row, check if placing a queen is valid
   update α and β values dynamically
5. Terminate Unnecessary Branches:
   if $\beta \leq \alpha$, prune the current branch

PAGE NO :
DATE :

3) MinMax Algorithm Tic-Tac-Toe Implementation.

pseudo code:-

```
function evaluate (board)
  if row/column/diagonal is all 'x': return 10
  if row/column/diagonal is all 'O': return -10
  return 0 # No winner.

function minimax (board, is_max):
  score = evaluate (board)
  if score != 0 or no moves left: return score

  if is_max:
    best = -∞
    for each empty cell:
      make move 'x'
      best = max(best, minimax(board, False))
      undo move
    return best
  else:
    best = +∞
    for each empty cell:
      make move 'O'
      best = min (best, minimax (board, True))
      undo move
  return best_move.
```

PAGE NO :
DATE :

```
function play_tic_tac_toe():
    initialise 3x3 board
    while True:
        if no moves left: print("Draw!"); break
        get human move, update board
        if evaluate(board) == -10: print("You win!"); break
        ai_move = find_best_move(board)
        update board with ai move
        if evaluate(board) == 10:
            print("AI wins!"); break
```

Ans
17/1/24