VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.



LAB REPORT on

Artificial Intelligence (23CS5PCAIN)

Submitted by

SUJNYAN ULHAS KINI(1BM22CS340)

in partial fulfilment for the award of the degree of BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING



B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Sep-2024 to Jan-2025

B.M.S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering



CERTIFICATE

This is to certify that the Lab work entitled "Artificial Intelligence (23CS5PCAIN)" carried out by SUJNYAN ULHAS KINI (1BM22CS340), who is Bonafide student of B.M.S. College of Engineering. It is in partial fulfilment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Artificial Intelligence (23CS5PCAIN) work prescribed for the said degree.

Sneha P	Dr. Kavitha Sooda
Assistant Professor	Professor & HOD
Department of CSE, BMSCE	Department of CSE, BMSCE

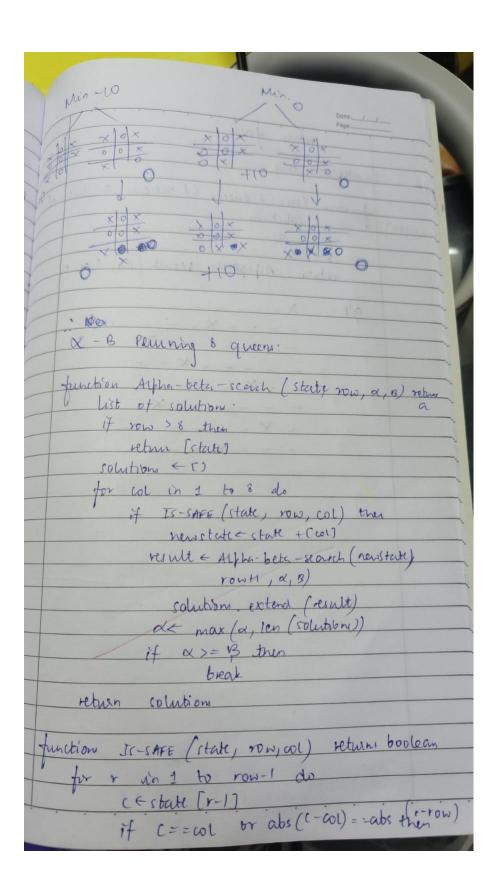
Index

Sl.	Date Experiment Title Page No		Page No.
No.		r	
1	24-9-2024	Implement Tic –Tac –Toe Game	1-7
2	1-10-2024	Implement vacuum cleaner agent	8-13
3	8-10-2024	Implement 8 puzzle problems using Depth First Search (DFS)	14-20
4	15-10-2024	Implement A* search algorithm Implement Iterative deepening search algorithm	21-31
5	22-10-2024	Simulated Annealing to Solve 8- Queens problem	32-35
6	29-10-2024	Implement A* search algorithm for N queens Implement Hill Climbing search algorithm to solve N-Queens problem	36-40
7	12-11-2024	Create a knowledge base using propositional logic and show that the given query entails the knowledge base or not.	41-43
8	19-11-2024	Implement unification in first order logic	44-46
9	3-12-2024	Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.	47-50
10	3-12-2024	Implement Min-Max Algorithm for Tic Tac Toe Implement Alpha-Beta Pruning for 8 queens	51-59

Github Link:

Implement Tic –Tac –Toe Game

Algorithm:
Dote Page
Tie Tac Toe (Min-max)
Define from unimax (board, dept, klayer). It the game has ended (wint bossificant) or board of
+10 if xwin
It it maximizing playes's turn (x): Initialize the best sure to small value for each empty Cell:
Make the more (n)
(all min may () receives bely
If it minimi sing players turn (0). Initialize best sure to a very large value for each empty cell. Make more (0) Call minimas () recursively!
Define function game ever (board) - Check if game has ended. (both lose draw) x o x (Mox) 0 0 x +10
10 × 100
XOX
X 0 X X 0 X +10.
(V).



return true Tetran true Touchton Solive 8 queers () retroine a list of Solive 9 queers () ret	Date Page	
return true Junction Solve-8-queens () returns a list of solutions X = 20 B = +00 return Alphanseta - search (C3) 1 mg of return Alphanseta - search (C3) 1 mg of	Jalu	/
Junction Solve-8-Queers () retraine a list of solvers solvers and solvers of solvers and solvers of solvers and solvers of solvers o		+
Junction Solve-8-Queers () retraine a list of solvers solvers and solvers of solvers and solvers of solvers and solvers of solvers o	return true	/
Return Authorite security (13, 1 2, 6) return Authorite security (13, 1 2, 6) o/r: X X	e-pugens () setroine a list	
return Aufmatteta - securch (13) 1 and 3 return Aufmatteta - securch (13) 1 and 3	Junchon solvers go	/
return Aufmatteta - securch (13) 1 acts of 1	X 6 - 00	/
		-
	return Alphangers scott	
And had all all and a line of the state of t		
The state of the second of the	- 14114 - 1- 14 1X 1 M	_
		_
A set parties and the formation of the first of the formation of the form		
And Columbian The second of t		
relation of to 8 de la fact for the fact of the fact o		
Ment columbian Here columbian Here columbian Here columbian Here columbian Here columbian	(1)	
Strand of the son cold that the son cold that the service of the son cold that the son the son that the son the son that the son the son that the son that the son the son that the son that		
Manual & Hotel State of the Holes South for South of the Control		_
colother extent formitel colother extent formitel if as is then bear	Total Hotels Hallace	_
constant sold sold sold sold sold sold sold sold	the face I want find a thing of the thing	-
colubbin enterd frequents of any for lon (solubbins) if as > 15 then beat		-
if as is then bear		1
tehen colubion		-
tehan coluntion		-
Hebrer Colubbon		_
		_
thather 1-see [that respond when below	Hebre Colubian	_
The Marine 1. sar (Marin of the order of the order		/
	- hacker I state I state reason betales added	
the same of the same of		
id do = (100-1) do 70		
=101=1) +i	100==3 41	
		,

```
import random
def win(board):
  for row in board:
    if row[0] == row[1] == row[2] != "":
       return True
  for col in range(3):
    if board[0][col] == board[1][col] == board[2][col] != "":
       return True
  if board[0][0] == board[1][1] == board[2][2] != "":
    return True
  if board[0][2] == board[1][1] == board[2][0] != "":
    return True
  return False
def printBoard(board):
  print("\n".join([" | ".join(row) for row in board]))
def draw(board):
  return all(cell != "" for row in board for cell in row)
def user_move(board):
  while True:
     try:
       move = int(input("Enter your move (1-9): ")) - 1
       row, col = divmod(move, 3)
       if board[row][col] == "":
          board[row][col] = "X"
          break
       else:
```

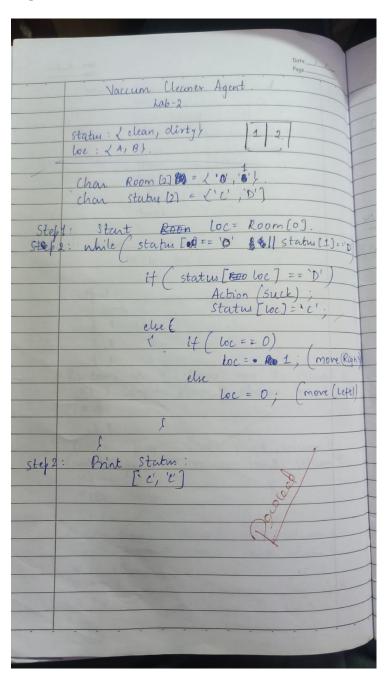
```
print("That space is already taken. Try again.")
    except (ValueError, IndexError):
       print("Invalid input. Please enter a number from 1 to 9.")
def computer_move(board):
  while True:
    move = random.randint(0, 8)
    row, col = divmod(move, 3)
    if board[row][col] == "":
       board[row][col] = "O"
       break
def _main():
  board = [["" for _ in range(3)] for _ in range(3)]
  while True:
    printBoard(board)
    user_move(board)
    if win(board):
       printBoard(board)
       print("You win!")
       break
    if draw(board):
       printBoard(board)
       print("It's a draw!")
       break
    computer_move(board)
    if win(board):
       printBoard(board)
       print("Computer wins!")
       break
    if draw(board):
```

```
printBoard(board)
print("It's a draw!")
break

if __name__ == "__main__":
   _main()
```

```
Enter your move (1-9): 2
X
0 1
Enter your move (1-9): 9
X
0 | X
Enter your move (1-9): 1
 X
 OIX
Enter your move (1-9): 5
 | X
 X
   OIX
You win!
```

Implement vacuum cleaner agent



```
def printArr(arr):
  for row in arr:
     print(row)
  print()
def clean(arr, x, y):
  if arr[x][y] == 1:
     arr[x][y] = 0
def check(arr):
  for row in arr:
     if 1 in row:
       return True
  return False
# Directions: right (0,1), down (1,0), left (0,-1), up (-1,0)
directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
direction_index = 0 # Start moving right
# Get room status
print("Enter the status of the rooms (0 for clean; 1 for dirty):")
arr1 = []
for i in range(2):
  row = []
  for j in range(2):
     a = int(input(f"Status of room ({i}, {j}): "))
     row.append(a)
  arr1.append(row)
x, y = 0, 0 #Start cleaning from the first room
while True:
  printArr(arr1)
  if not check(arr1):
```

```
break
clean(arr1, x, y)

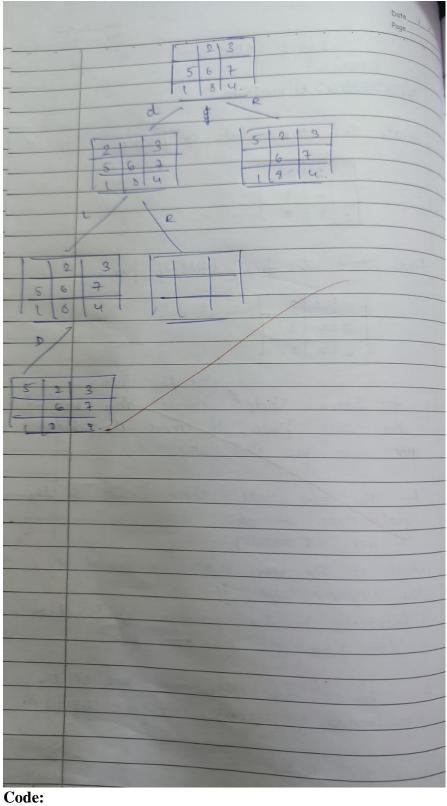
#Move to the next room in the current direction
dx, dy = directions[direction_index]
new_x, new_y = x + dx, y + dy

#Check bounds
if 0 <= new_x < 2 and 0 <= new_y < 2:
    x, y = new_x, new_y
else:
    #Change direction(turn right)
    direction_index = (direction_index + 1) % 4
    dx, dy = directions[direction_index]
    x, y = x + dx, y + dy #Move in the new direction
print("All rooms are cleaned!")
```

```
Enter the status of the rooms (0 for clean; 1 for dirty):
Status of room (0, 0): 1
Status of room (0, 1): 0
Status of room (1, 0): 1
Status of room (1, 1): 0
[1, 0]
[1, 0]
[0, 0]
[1, 0]
[0, 0]
[1, 0]
[0, 0]
[1, 0]
[0, 0]
[0, 0]
All rooms are cleaned!
```

Implement 8 puzzle problems using Depth First Search (DFS)

Algorithm:
Desi
potanathanteen of Algoerethm:
Greate the start state of puggle
2 3 5 6 4 1 0 4
1.04
Final Store the final state of puzzle
3 4 5
(first the street
stope the state. Stabl: for Every State, choose any lone of the many places to more the blank space to get New state.
Step? For the new state check if it is the goal state if not if yes got to step?
Check if the new state is present with
Backtrack to the previous step & choose of diff place for the place space.
If no, for the new choose any one of the legal moves & traverse. (Repeat Step 2).
Steps Print the state.
N



class PuzzleState:

def __init__(self, board, moves=0, previous=None):

```
self.board = board
  self.moves = moves
  self.previous = previous
  self.empty_pos = self.find_empty()
def find_empty(self):
  for i in range(3):
     for j in range(3):
       if self.board[i][j] == 0:
          return (i, j)
def manhattan_distance(self):
  dist = 0
  for i in range(3):
     for j in range(3):
       tile = self.board[i][j]
       if tile != 0:
          target_x = (tile - 1) // 3
          target_y = (tile - 1) \% 3
          dist += abs(i - target_x) + abs(j - target_y)
  return dist
def generate_moves(self):
  moves = []
  x, y = self.empty\_pos
  directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
  for dx, dy in directions:
     new_x, new_y = x + dx, y + dy
     if 0 \le \text{new}_x < 3 and 0 \le \text{new}_y < 3:
       new_board = [row[:] for row in self.board]
       new_board[x][y],
                               new_board[new_x][new_y] =
                                                                         new_board[new_x][new_y],
```

```
new_board[x][y]
          moves.append(PuzzleState(new_board, self.moves + 1, self))
     return moves
def dfs(start_board, max_depth):
  stack = [PuzzleState(start_board)]
  visited = set()
  goal\_state = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]
  while stack:
     current_state = stack.pop()
     if current_state.board == goal_state:
       return current state
     visited.add(tuple(map(tuple, current_state.board)))
    if current_state.moves < max_depth:
       for next_state in current_state.generate_moves():
          if tuple(map(tuple, next_state.board)) not in visited:
            if next_state.manhattan_distance() < 10:
               stack.append(next_state)
  return None
def print_solution(solution):
  path = []
  while solution:
     path.append(solution.board)
     solution = solution.previous
  for step in reversed(path):
     for row in step:
       print(row)
    print()
  print(f"Total moves taken to reach the final state: {len(path) - 1}")
```

```
initial_board = [[1, 2, 3], [4, 0, 5], [7, 8, 6]]
max_depth = 10
solution = dfs(initial_board, max_depth)
if solution:
   print("Solution found:")
   print_solution(solution)
else:
   print("No solution found.")
```

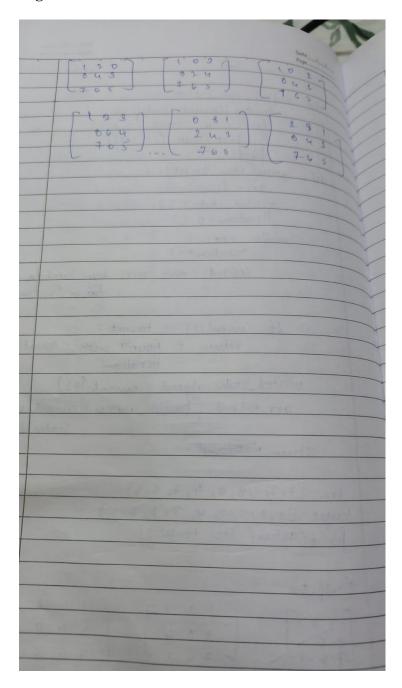
```
Solution found:
[1, 2, 3]
[4, 0, 5]
[7, 8, 6]

[1, 2, 3]
[4, 5, 0]
[7, 8, 6]

[1, 2, 3]
[4, 5, 6]
[7, 8, 0]

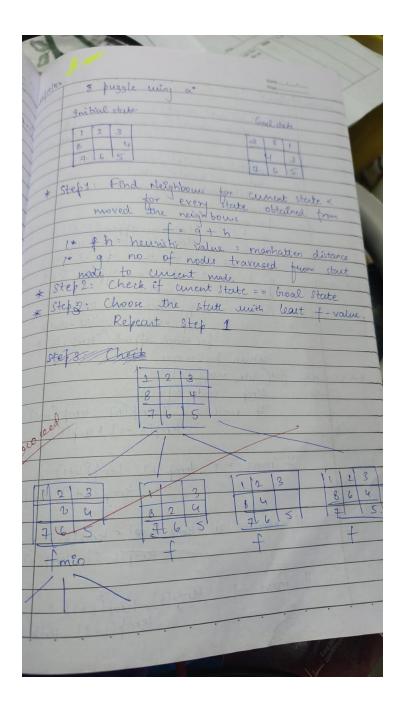
Total moves taken to reach the final state: 2
```

Implement A* search algorithm



Dote Page
return tem
Tala state (state):
det display-state (state). print ('current state')
por i in range (0, a, 3) post (stute (i: (+3))
brite (state (i: (+3))
from trunt)
det astructure (02)
visited states = C]
iterations = 0
while arr:
ament = nuis (arr, key= lambda a: for n (n, tanget))
Joseph (A) Congression
if curent co] == tanget
return f' found with (iteration)
tterations
visited_states_append (curent (0))
arrestend possible, moves (ament, virited-
States))
· latter and
12 Jun Notford
C1 2 2 2 2 4 7 6/57
src=[+1213,8,0,4,7,6(5)
C . 1. D1 (1. 31+10)
brist (aster (src, target))
1711740
Output:
F 3 3 4 0
084 165
265
····

310
Octe Proge
(5/10/24) Codes
def M_n (state, tauget): return sum (n 1 = y for m, y in zile
return sum (n = y for my in 21/2
Catalogue
def fin (state-with-(x), Visited States);
state IVI = state with - IVI
b=state.index(0) direction = C]
por movu = C)
of bess direction abbend ('d')
if b>=3: direction, append ('u')
et b 1/3 >0: directions. append (11)
gor more in directions append (in)
temp gen (state, move, b)
if temp not in winted - state:
for moves append (Hemp, 1x147)
return pos_moves
def gen(state, move 16):
temp = state colu()
st move=='l':temp[b], temp[b-1]=
templo -1) templo)
1
If more == 'r', temp (b], temp (b+1)
= temp (6+1), ten /(6)
if move == 'u' - temp [600] o temp[6-3]
= temp[b-3], temp[b]
1
if move==d' : temp[b], temp[b+s] =
temp[6+3], temp[b].
: 16+3] , templo).



```
def H_n(state, target):
    return sum(x != y for x, y in zip(state, target))
def F_n(state_with_lvl, target):
    state, lvl = state_with_lvl
    return H_n(state, target) + lvl
def possible_moves(state_with_lvl, visited_states):
```

```
state, lvl = state_with_lvl
  b = state.index(0)
  directions = []
  pos_moves = []
  if b <= 5: directions.append('d')
  if b \ge 3: directions.append('u')
  if b % 3 > 0: directions.append('l')
  if b % 3 < 2: directions.append('r')
  for move in directions:
     temp = gen(state, move, b)
    if temp not in visited_states:
       pos_moves.append([temp, lvl + 1])
  return pos_moves
def gen(state, move, b):
  temp = state.copy()
  if move == 'l': temp[b], temp[b - 1] = temp[b - 1], temp[b]
  if move == 'r': temp[b], temp[b + 1] = temp[b + 1], temp[b]
  if move == 'u': temp[b], temp[b - 3] = temp[b - 3], temp[b]
  if move == 'd': temp[b], temp[b + 3] = temp[b + 3], temp[b]
  return temp
def display_state(state):
  print("Current State:")
  for i in range(0, 9, 3):
     print(state[i:i+3])
  print()
def astar(src, target):
  arr = [[src, 0]]
  visited_states = []
  iterations = 0
  while arr:
     iterations += 1
     current = min(arr, key=lambda x: F_n(x, target))
```

```
arr.remove(current)
display_state(current[0])
if current[0] == target:
return f'Found with {iterations} iterations'
visited_states.append(current[0])
arr.extend(possible_moves(current, visited_states))
return 'Not found'
src = [1, 2, 3, 8, 0, 4, 7, 6, 5]
target = [2, 8, 1, 0, 4, 3, 7, 6, 5]
print(astar(src, target))
```

```
Current State:
[1, 3, 4]
[0, 8, 2]
[7, 6, 5]
Current State:
[8, 1, 0]
[2, 4, 3]
[7, 6, 5]
Current State:
[8, 0, 1]
[2, 4, 3]
[7, 6, 5]
Current State:
[0, 8, 1]
[2, 4, 3]
[7, 6, 5]
Current State:
[2, 8, 1]
[0, 4, 3]
[7, 6, 5]
Found with 40 iterations
```

Implement Iterative deepening search algorithm

Code: Iterative Despiners
det iterative deepening search (graph, start, goal) det dept currented search (mode, goal, dept): if hode = goal:
det dept curitide scarch (mode and).
H node
if node = = goal : seturn [node]
else: node
elif dept 0:
elif dept 0: for chold on graph, get (node, c): result = dept lunited scance (chold, goal, dept)
result = dalit , graph get (rode, ());
Librated search (child
if result is not Hone;
return [Mode) + Funt
debt = 0 12 days same same
dept = 0 phate Inte
The wast - mount - arable 19.
det get wer input - graph ():
num-edges = int (intent (" Enter no of edges"))
edecs")
mut (Enter each edge").
print ("Enter each edge"). for i in range (num-edges) node 1 node 2 = input () split()
roade 1 / node 2 = inhut () shirt()
V let
the mode is a graph
grapt node 1 - append (node 2)
graph(nodel) - (node 2)
graph (node a) append (mode)
graph i nou & stapes
6/10
else graph(nodei) = (nodi)
July Cooper
Ved

	Enter no of edges : 14	1
	Enter each edge	1 124
	чх	
	PR	
	P S XF	
	XH	
	R B	
	e c	
	S X	
	SZ	
	FU	
	FE	
	HL	
	Enter goal node: &F Path Juind: M-X-F	
	Enter starting hour	
	Enter goal node: 12 F	
	Park Award: M-X-F	
	Part Proses	
	V	
	The state of the s	
	THE RESERVE THE PARTY OF THE PA	(
42 7 7 1		
	to the same of the	
	A STATE OF THE PARTY OF THE PAR	
	The state of the s	
	To the same distance	
-		
ļ		
	The state of the s	
		The second second second second

```
def iterative_deepening_search(graph, start, goal):
  def depth_limited_search(node, goal, depth):
     if depth == 0:
       if node == goal:
          return [node]
       else:
          return None
     elif depth > 0:
       for child in graph.get(node, []):
          result = depth_limited_search(child, goal, depth - 1)
          if result is not None:
            return [node] + result
     return None
  depth = 0
  while True:
     result = depth_limited_search(start, goal, depth)
    if result is not None:
       return result
     depth += 1
def get_user_input_graph():
  graph = \{\}
  num_edges = int(input("Enter the number of edges: "))
  print("Enter each edge in the format 'node1 node2':")
  for _ in range(num_edges):
    node1, node2 = input().split()
    if node1 in graph:
       graph[node1].append(node2)
     else:
       graph[node1] = [node2]
    if node2 in graph:
```

```
graph[node2].append(node1)
else:
    graph[node2] = [node1]
return graph

def main():
    graph = get_user_input_graph()
    start_node = input("Enter the starting node: ")
    goal_node = input("Enter the goal node: ")
    path = iterative_deepening_search(graph, start_node, goal_node)
    if path:
        print(f"Path found: {' -> '.join(path)}")
    else:
        print("No path found")

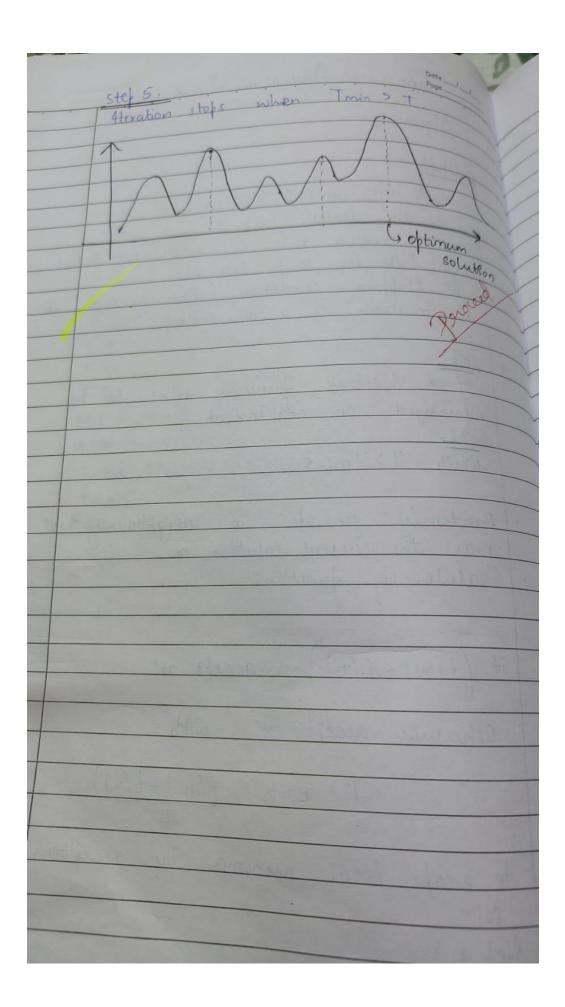
if __name__ == "__main__":
    main()
```

```
Enter the number of edges: 14

Enter each edge in the format 'node1 node2':
Y P
Y X
P R
P S
X F
X H
R B
R C
S X
S Z
F U
F E
H L
H W
Enter the starting node: Y
Enter the goal node: F
Path found: Y -> X -> F
```

Simulated Annealing to Solve 8-Queens problem

SIMULATED ANNEALING AGORITHMEN ALING	
oglo and	1
Algorithm:	>
Step 15	1
	1
Start with initial solution no	
feet a cooling rate of Court	
fet a cooling rate of (0 co 61) to control how fact the temp decreases	
St Stablishes Condition	
St stopping condition (aither a man as	
of iteration or a min temp Imin)	
d. Step 2:	-
Defined objective function $f(n)$ to be minimized or maximized	-1
The state of the s	
mininger or maximized	
Step 2:	
While T> Tring	
· Random y generate a neighbouring 101	n
Pandom by generate a relighbouring toler	
Qualita the tracking	
, Evalute obj. function.	
if (t(n') < f(n)) accept n'	
it It (n') < t(n) accept in	
Otherwise accept n' with	
Otherwise allege in with	1
	HARM
$P = enp \left(-f(n') - f(n)\right)$	
The state of the s	
	- about
to escape local minima by accepting	Worse
to make local minima	
TO Excuse	
Soln	
300	-
3teb 41	
C dona.	
Cooling Criteria:	THE REAL PROPERTY.
1-41	



```
import random
import math
def energy(x):
  return x ** 2 + 5 * math.sin(x) + math.exp(-x)
def adaptive_simulated_annealing(start, temp, cooling_rate, lower_limit, upper_limit):
  current = start
  current_energy = energy(current)
  while temp > 1:
    # Adaptive step size based on temperature (larger steps when hot)
    step_size = random.uniform(-1, 1) * temp
    new = current + step\_size
    # Ensure new solution is within bounds
    if new < lower_limit or new > upper_limit:
       continue
    new_energy = energy(new)
    # If the new spot is better, move there
    if new_energy < current_energy:
       current = new
       current_energy = new_energy
    else:
       # Acceptance probability (explore worse spots)
       probability = math.exp((current_energy - new_energy) / temp)
       if random.uniform(0, 1) < probability:
         current = new
```

```
current_energy = new_energy
    # Adaptive cooling based on progress
    if abs(new\_energy - current\_energy) < 0.01:
       temp *= 0.98 # Slow cooling near solution
    else:
       temp *= cooling_rate
  return current
# Run the simulation multiple times from different starting points
best_solution = None
for _ in range(10): # 10 runs
  result = adaptive_simulated_annealing(start=random.uniform(-10, 10), temp=100,
cooling_rate=0.99, lower_limit=-10, upper_limit=10)
  if best_solution is None or energy(result) < energy(best_solution):
    best_solution = result
print(f"Best solution found: {best_solution}")
```

Best solution found: -0.7323104061658242

Implement A* search algorithm for N queens

A	A start.
4	Code for Weld Costabiling Algo sutton
	import heap
1	det heuritic (board)
3/	det conflicte =0
	for i is range (len(board)):
1	for in range (it len (board))
V	for i is range (len(board)); for i is range (i+1, len(board)); if (board[i] = board[j] or abr (board[j] - board[j])=j-1.
N	conflicts +=1
1	return conflicts.
V	*
V	det a stant -8 - queen (1:
V	n > 8
Y/	
1	heapq. heappush (open set, (o, c1))
	while open set: f, board = heapq. heapfor (open-set)
	f, board = heapq. heappop (open-set)
/	
/	if len(board) == n & heuristic (board)=0.
/	Letun board
_	NAME OF STREET
_	I (local)
	row = len (board)
	for col un range(n):
	have board = board + Every
	if (heuristic (new_board) == 0:
	h-heunitic (new board) h-heunitic (new board) heafq. heappourh (open-set, 19th, new, boar
	h-heunitic hew does of the new.
	hearg hearbruh often sor boar
	return None Solution = a (tent-8-quem c) point ('solution', solution)
	Colution = a start 8 - quem 9
-	solution)
	mht (Solution)
	To differ the second

Output. Solution: [0,417, 5,2, 6,1,3] Parieto tooke with anting

2-D
Stept: Create an visibal state with a Represent this with queens of states
Stef 2 Let 'g' value to 0.
Add states to pa pased on persons over quest $f(n) = g(n)$ to the hours
Step4: Inent iniatial empty state to parionly
stefs: Identify the next empty now g-o (how) for each col 0 to n-1, place a queen & create a new state (Increment 9)
Gueen & vieate a new state (Ancrement 9) Step 6: Choose that state from periowing queue with lowest $f(n)$ value
Check if h=0 & g=N return Solution.
tlee Repeat Step 5.

import heapq

```
# Helper function to calculate the heuristic (number of conflicts)
def heuristic(board):
  conflicts = 0
  for i in range(len(board)):
     for j in range(i + 1, len(board)):
       if board[i] == board[j] or abs(board[i] - board[j]) == j - i:
          conflicts += 1
  return conflicts
# A* Search for 8-queens
def a_star_8_queens():
  n = 8
  open_set = []
  # Initial state: empty board
  heapq.heappush(open_set, (0, [])) # (f, board)
  while open_set:
    f, board = heapq.heappop(open_set)
     # Goal check
     if len(board) == n and heuristic(board) == 0:
       return board
     # Generate successors
     row = len(board)
     for col in range(n):
       new\_board = board + [col]
       if heuristic(new_board) == 0: # No conflicts so far
          g = row + 1
          h = heuristic(new_board)
```

```
return None # No solution found

# Run A* search

solution = a_star_8_queens()

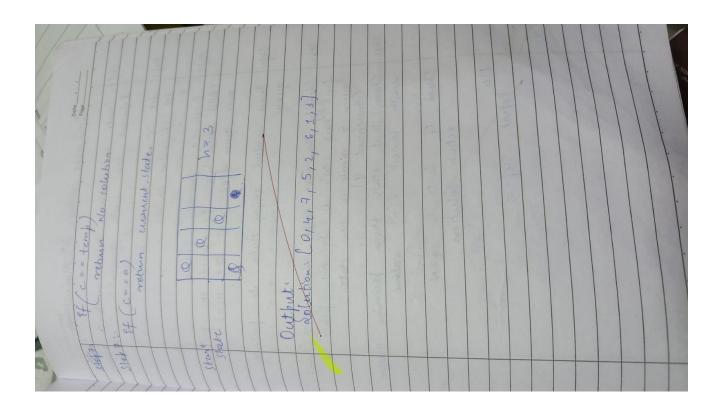
print("Solution board (column positions for each row):", solution)
```

heapq.heappush(open_set, (g + h, new_board))

```
Solution board (column positions for each row): [0, 4, 7, 5, 2, 6, 1, 3]
```

Implement Hill Climbing search algorithm to solve N-Queens problem

Algorithm: conflicts (board



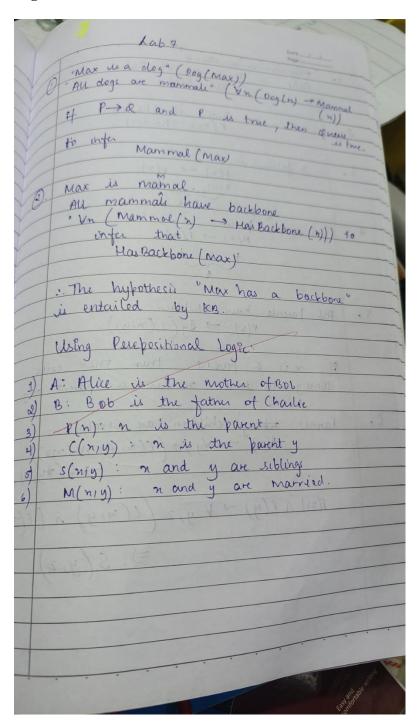
import random

```
while True:
    current_h = heuristic(board)
    if current_h == 0:
       return board # Solution found
    # Find the best neighbor by moving each queen to every other column in its row
    best_board = board[:]
    best_h = current_h
    for row in range(n):
       for col in range(n):
         if col == board[row]:
            continue
         new board = board[:]
         new_board[row] = col
         new_h = heuristic(new_board)
         # If the new board has fewer conflicts, update the best board
         if new_h < best_h:
            best_h = new_h
            best_board = new_board
    # If no improvement, we're stuck in a local minimum; restart
    if best_h >= current_h:
       board = [random.randint(0, n - 1) for _ in range(n)]
    else:
       board = best board
# Run hill climbing search
solution = hill_climbing_8_queens()
print("Solution board (column positions for each row):", solution)
```

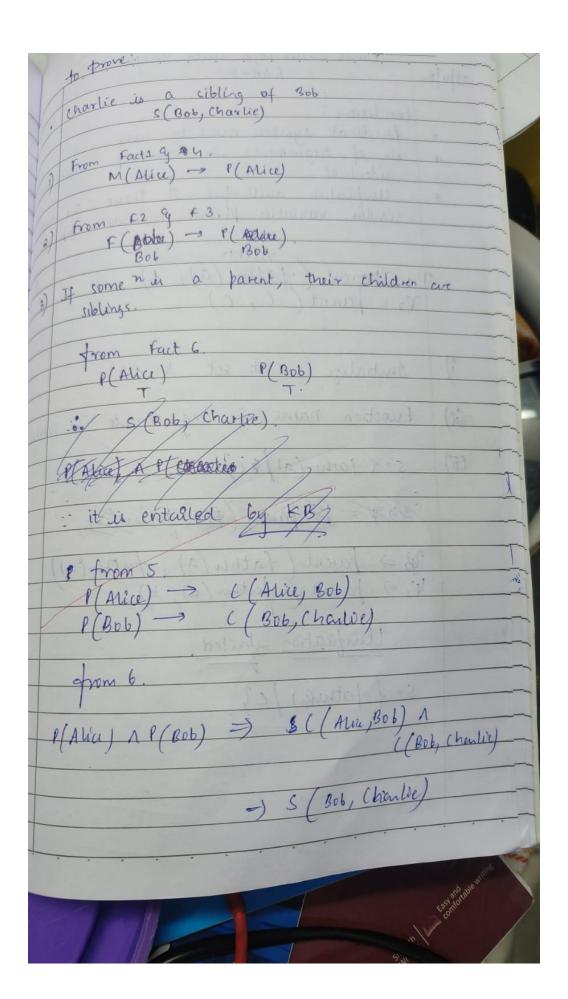
Solution board (column positions for each row): [0, 6, 3, 5, 7, 1, 4, 2]

Program 7

Create a knowledge base using propositional logic and show that the given query entails the knowledge base or not.



	1000
i	· thankie in a selding
J	Alice is the mother of 306:
	C(Alice, Bob)
2.	Bob in the state of the li
	Bob in the father of Charlie (1306, Charlie)
	A fatur is a parent
	$f(n) \rightarrow p(n)$
	true true
	true.
- 4	A mother parent
	$M(n) \rightarrow P(n)$
	transatabalani
	The bullettiech "Max has a backbeau
5.	All parents have Children
	P(x) -> fy (c(n,y))
	() ()(()())
	If n'is a parent then there exists
	some y such that n is a parent of
	ashand to what and a parent of y
C .	Parent's siblings children are sibling:
	P(n) -> +y, z (C(n/y) Ac(n/2))
	remide to a but a continue
	horizon to p bow is founded
	$P(n) \wedge P(y) \rightarrow \forall y, 2 \left(C(n, y) \wedge C(y) \right)$
	3) 312 (3) 1 (4)
	\Rightarrow . $S(y, 2)$
100	The second secon



import random

```
# Helper function to calculate the heuristic (number of conflicts)
def heuristic(board):
  conflicts = 0
  for i in range(len(board)):
     for j in range(i + 1, len(board)):
       if board[i] == board[j] or abs(board[i] - board[j]) == j - i:
          conflicts += 1
  return conflicts
# Hill climbing for 8-queens
def hill_climbing_8_queens():
  n = 8
  # Generate a random initial state
  board = [random.randint(0, n - 1) for _ in range(n)]
  while True:
     current_h = heuristic(board)
     if current h == 0:
       return board # Solution found
     # Find the best neighbor by moving each queen to every other column in its row
     best_board = board[:]
     best_h = current_h
     for row in range(n):
       for col in range(n):
          if col == board[row]:
            continue
          new_board = board[:]
```

```
new_board[row] = col
new_h = heuristic(new_board)

# If the new board has fewer conflicts, update the best board
if new_h < best_h:
    best_h = new_h
    best_board = new_board

# If no improvement, we're stuck in a local minimum; restart
if best_h >= current_h:
    board = [random.randint(0, n - 1) for _ in range(n)]
    else:
    board = best_board

# Run hill climbing search
solution = hill_climbing_8_queens()
print("Solution board (column positions for each row):", solution)
```

The hypothesis 'Charlie is a sibling of Bob' is FALSE.

Program 8

Implement unification in first order logic

19/11/24. First Order Logic with Unifications	/
t Conditions: * Peredicate symbols nums be same * No of arguments in both exp mist be identical * Unification will fail if there are bro similar variables present in same is * Father * 2 - Darrent (* father (a), b) * 2 = parent (C, C)	7 100
1) Instialize sub set S= XY	
ii) Function name & arguements in same	-
(ii) s- < father(a) / (c)	_
Start (father (a), father (a)) \(\psi\) \(\psi\) parent (father (a), father (a)) \(\psi\) \(\psi\) parent (father (a), b) \(\psi\) \(\p	

Date
We parent (X, V) Dote page Page Page
gnitialize subset S= 12
for variable & in P2, term father (a)
S= 2 father (a) /x }
13) W. = pavent (father (a), b) Y2 = pavent (father (a), y)
for variable V, term b. S= & father(a) / X, b/4/
5 = parent (father (a), b) We fasent (father (a), b) Unified Succonfully
Unified Succonfully
-MGU=S.
deed in the transport of the state of the
Code Output:
Unification failed:
Unified Succentrally: (b': 2, X:7(4); 7.9(1)
H. de
Edgette

```
def unify(x, y, subst=None):
  ,,,,,,
  Unification Algorithm: Unifies two terms, X and Y.
  if subst is None:
     subst = \{\}
  if x == y: # Step 1(a): If X and Y are identical
     return subst
  elif isinstance(x, str) and x.islower(): # Step 1(b): If X is a variable
     return unify_variable(x, y, subst)
  elif isinstance(y, str) and y.islower(): # Step 1(c): If Y is a variable
     return unify_variable(y, x, subst)
  elif isinstance(x, tuple) and isinstance(y, tuple): # Step 2: Check predicates and arguments
     if x[0] != y[0] or len(x) != len(y): # Predicate symbol or argument count mismatch
       return None
     for x_i, y_i in zip(x[1:], y[1:]): # Step 5: Recurse through arguments
       subst = unify(x_i, y_i, subst)
       if subst is None:
          return None
     return subst
  else:
     return None # Step 1(d): Failure case
def unify_variable(var, x, subst):
  ,,,,,,
  Unify variable with another term.
  ,,,,,,
  if var in subst:
```

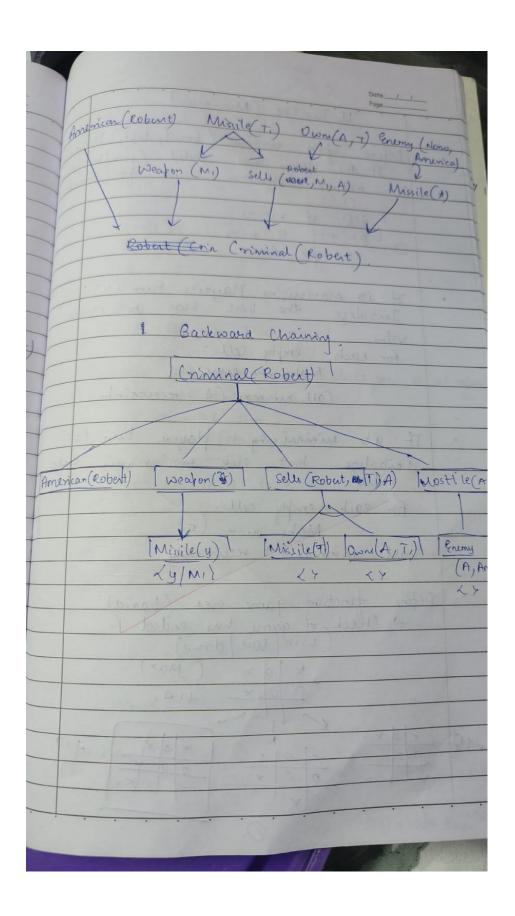
```
return unify(subst[var], x, subst)
  elif occurs_check(var, x, subst): # Check if var occurs in x
     return None
  else:
     subst[var] = x
     return subst
def occurs_check(var, x, subst):
   ,,,,,,
  Check if a variable occurs in a term.
   ,,,,,,,
  if var == x:
     return True
  elif isinstance(x, tuple):
     return any(occurs_check(var, xi, subst) for xi in x)
  elif isinstance(x, str) and x in subst:
     return occurs_check(var, subst[x], subst)
  return False
# Test cases for unification
x1 = ("P", "a", "x")
y1 = ("P", "a", "b")
x2 = ("Q", "x", ("R", "x"))
y2 = ("Q", "a", ("R", "a"))
print("Unifying", x1, "and", y1, "=>", unify(x1, y1))
print("Unifying", x2, "and", y2, "=>", unify(x2, y2))
```

```
Unifying ('P', 'a', 'x') and ('P', 'a', 'b') => {'x': 'b'}
Unifying ('Q', 'x', ('R', 'x')) and ('Q', 'a', ('R', 'a')) => {'x': 'a'}
```

Program 9

Create a knowledge base consisting of first order logic statements and prove the given query using forward reasoning.

```
2069:
   Forward Chaining
    To puoue: Robert is cuininal
             Carininal (Robert)
   It is a cuine for an american to weapon to a hostile nations.
   Take & of ane variables.
  American (1) 1 weapon (9) 1 reliefts
  Country & has some nuisiles
   In Owns (A, n) in missie (n)
  All the missiles were sold to country !
   by report Robert.
 Vn Missile(n) 1 Own (A, n) → Sells (Roberton
 Missiles are weapons
    Missile (m) -> Weapon(n)
Enemy of america is hostile
Vn Enemy (n, America) -> Mostile (n)
```



```
Code:
# Define the knowledge base (KB) as a set of facts
KB = set()
# Premises based on the provided FOL problem
KB.add('American(Robert)')
KB.add('Enemy(America, A)')
KB.add('Missile(T1)')
KB.add('Owns(A, T1)')
# Define inference rules
def modus_ponens(fact1, fact2, conclusion):
  """ Apply modus ponens inference rule: if fact1 and fact2 are true, then conclude conclusion """
  if fact1 in KB and fact2 in KB:
     KB.add(conclusion)
    print(f"Inferred: {conclusion}")
def forward_chaining():
  """ Perform forward chaining to infer new facts until no more inferences can be made """
  # 1. Apply: Missile(x) \rightarrow Weapon(x)
  if 'Missile(T1)' in KB:
    KB.add('Weapon(T1)')
    print(f"Inferred: Weapon(T1)")
  # 2. Apply: Sells(Robert, T1, A) from Owns(A, T1) and Weapon(T1)
  if 'Owns(A, T1)' in KB and 'Weapon(T1)' in KB:
    KB.add('Sells(Robert, T1, A)')
    print(f"Inferred: Sells(Robert, T1, A)")
  # 3. Apply: Hostile(A) from Enemy(A, America)
```

if 'Enemy(America, A)' in KB:

```
KB.add('Hostile(A)')

print(f"Inferred: Hostile(A)")

# 4. Now, check if the goal is reached (i.e., if 'Criminal(Robert)' can be inferred)

if 'American(Robert)' in KB and 'Weapon(T1)' in KB and 'Sells(Robert, T1, A)' in KB and 'Hostile(A)' in KB:

KB.add('Criminal(Robert)')

print("Inferred: Criminal(Robert)")

# Check if we've reached our goal

if 'Criminal(Robert)' in KB:

print("Robert is a criminal!")

else:

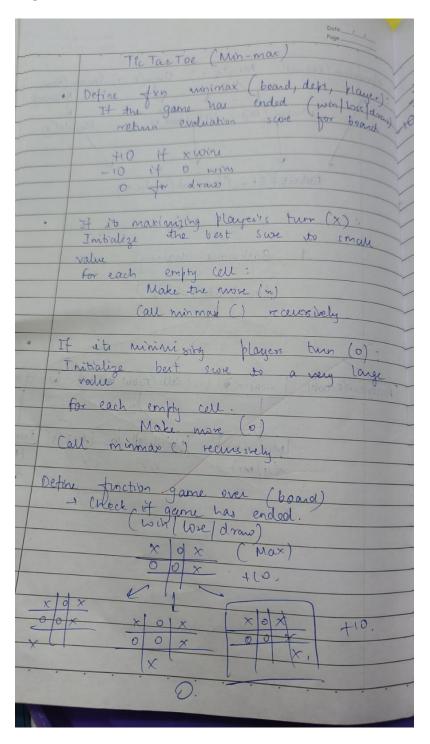
print("No more inferences can be made.")

# Run forward chaining to attempt to derive the conclusion forward_chaining()
```

```
Inferred: Weapon(T1)
Inferred: Sells(Robert, T1, A)
Inferred: Hostile(A)
Inferred: Criminal(Robert)
Robert is a criminal!
```

Program 10

Implement Min-Max Algorithm for Tic Tac Toe



X - B Pluning & queen: function Alpha-beta-scarch (starte now, a, B) repres list of solution. if now > 8 then return [state] solutions (1) for col in 1 to 8 do if IS-SAFE (state, row, col) then newstate + state + (col) result & Alpha-bets-search (newstarte) rowth, a, B) solutions extend (result) de max (a, len (solution)) if x>= B then break return colutions Is-sAFE (state, row, col) return boolean function r in 1 to row-1 do c ← state [r-1] if c== col br abs (c-col) = -abs ther

Dote Page
return Jalu
return true
Junchion Solve-8-Queens () setroins a list of solutions Solutions B < +00 return Arphaneta - securch (53, 1 × 6)
0/r: X
X
- X - + 2 - 1 C MANUEL
X = /41 - 12 - 414
(1) mortigar
at 8 of the state of the
And the state (state now call this
Class Hoto Stalmant
Misull & Mills feet want (new text)
(8, 10, Hove.
relative extend frenches
tout of a tree
midness of the second
Auction 11-14 (these reason) returns boules
the least of the state of
20 das (100-1) 1 (1-x) 21 mb >)
100 - (100 - 1) 2ds 2d = 100 = 2) 1;

Code:

import math

Constants for the players

AI = 'X'

```
HUMAN = 'O'
EMPTY = '\_'
# Function to print the board
def print_board(board):
  for row in board:
    print(" ".join(row))
  print()
# Function to check if a player has won
def check_winner(board, player):
  # Check rows, columns, and diagonals
  for row in board:
     if all(cell == player for cell in row):
       return True
  for col in range(3):
     if all(row[col] == player for row in board):
       return True
  if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in range(3)):
     return True
  return False
# Function to check if the game is a draw
def is_draw(board):
  return all(cell != EMPTY for row in board for cell in row)
# Minimax algorithm
def minimax(board, depth, is_maximizing):
  if check_winner(board, AI):
     return 10 - depth
  if check_winner(board, HUMAN):
     return depth - 10
```

```
if is_draw(board):
    return 0
  if is_maximizing:
    best_score = -math.inf
    for i in range(3):
       for j in range(3):
         if board[i][j] == EMPTY:
            board[i][j] = AI
            score = minimax(board, depth + 1, False)
            board[i][j] = EMPTY
            best_score = max(best_score, score)
    return best_score
  else:
    best_score = math.inf
    for i in range(3):
       for j in range(3):
         if board[i][j] == EMPTY:
            board[i][j] = HUMAN
            score = minimax(board, depth + 1, True)
            board[i][j] = EMPTY
            best_score = min(best_score, score)
    return best_score
# Function to find the best move for AI
def find_best_move(board):
  best score = -math.inf
  move = (-1, -1)
  for i in range(3):
    for j in range(3):
       if board[i][j] == EMPTY:
         board[i][j] = AI
```

```
score = minimax(board, 0, False)
         board[i][j] = EMPTY
          if score > best_score:
            best_score = score
            move = (i, j)
  return move
# Example usage
if __name__ == "__main__":
  # Initialize a sample board
  board = [
    ['X', 'O', 'X'],
    ['O', 'X', 'O'],
    ['_', '_', '_']
  ]
  print("Current Board:")
  print_board(board)
  best_move = find_best_move(board)
  print(f"The best move for AI is: {best_move}")
```

```
Current Board:
X O X
O X O
---
The best move for AI is: (2, 0)
```

Implement Alpha-Beta Pruning for 8 queens

Code:

```
class EightQueens:
    def __init__(self, size=8):
        self.size = size

def is_safe(self, board, row, col):
    """Check if placing a queen at board[row][col] is safe."""
    for i in range(col):
        if board[row][i] == 1: # Check this row on the left
            return False

for i, j in zip(range(row, -1, -1), range(col, -1, -1)): # Check upper diagonal
        if board[i][j] == 1:
            return False

for i, j in zip(range(row, self.size), range(col, -1, -1)): # Check lower diagonal
        if board[i][j] == 1:
        return False
```

return True

```
def alpha_beta_search(self, board, col, alpha, beta, maximizing_player):
  """Alpha-Beta Pruning Search."""
  if col >= self.size: # If all queens are placed
    return 0, [row[:] for row in board] # Return 0 as heuristic since it's a valid solution
  if maximizing_player:
     max_eval = float('-inf')
    best_board = None
    for row in range(self.size):
       if self.is_safe(board, row, col):
          board[row][col] = 1
          eval_score, potential_board = self.alpha_beta_search(board, col + 1, alpha, beta, False)
          board[row][col] = 0
          if eval_score > max_eval:
            max_eval = eval_score
            best_board = potential_board
          alpha = max(alpha, eval_score)
          if beta <= alpha: # Beta cutoff
            break
    return max_eval, best_board
  else:
    min_eval = float('inf')
    best_board = None
     for row in range(self.size):
       if self.is_safe(board, row, col):
          board[row][col] = 1
          eval_score, potential_board = self.alpha_beta_search(board, col + 1, alpha, beta, True)
          board[row][col] = 0
          if eval_score < min_eval:
            min_eval = eval_score
            best_board = potential_board
```

```
beta = min(beta, eval_score)
             if beta <= alpha: # Alpha cutoff
               break
       return min_eval, best_board
  def solve(self):
     """Solve the 8-Queens problem."""
     board = [[0] * self.size for _ in range(self.size)]
     _, solution = self.alpha_beta_search(board, 0, float('-inf'), float('inf'), True)
     return solution
   def print_board(self, board):
     """Print the chessboard."""
     for row in board:
       print(" ".join("Q" if col else "." for col in row))
     print()
if __name__ == "__main__":
  game = EightQueens()
  solution = game.solve()
  if solution:
     print("Solution found:")
     game.print_board(solution)
  else:
     print("No solution exists.")
```

#