**AI Assignment 2B**

**Neeti Kurulkar**

**A\* Search Algorithm**

**Code:**

**class EightPuzzle:**

**def \_\_init\_\_(self, start\_state, goal\_state):**

**self.*start\_state* = tuple(map(tuple, start\_state)) # Convert to tuple**

**self.*goal\_state* = tuple(map(tuple, goal\_state))**

**def find\_blank(self, state):**

**""" Find the position of the blank (zero) tile """**

**for i in range(3):**

**for j in range(3):**

**if state[i][j] == 0:**

**return i, j**

**def get\_neighbors(self, state):**

**""" Generate possible next states by moving the blank space """**

**neighbors = []**

**moves = {'Up': (-1, 0), 'Down': (1, 0), 'Left': (0, -1), 'Right': (0, 1)}**

**x, y = self.*find\_blank*(state)**

**for move, (dx, dy) in moves.*items*():**

**new\_x, new\_y = x + dx, y + dy**

**if 0 <= new\_x < 3 and 0 <= new\_y < 3:**

**new\_state = [list(row) for row in state] # Convert tuple back to list**

**new\_state[x][y], new\_state[new\_x][new\_y] = new\_state[new\_x][new\_y], new\_state[x][y]**

**neighbors.*append*((tuple(map(tuple, new\_state)), move)) # Convert back to tuple**

**return neighbors**

**def h(self, state):**

**""" Manhattan Distance heuristic function """**

**goal\_positions = {1: (0, 0), 2: (0, 1), 3: (0, 2),**

**4: (1, 0), 5: (1, 1), 6: (1, 2),**

**7: (2, 0), 8: (2, 1), 0: (2, 2)} # Goal state positions**

**h\_value = 0**

**for i in range(3):**

**for j in range(3):**

**value = state[i][j]**

**if value != 0:**

**goal\_x, goal\_y = goal\_positions[value]**

**h\_value += abs(i - goal\_x) + abs(j - goal\_y)**

**return h\_value**

**def a\_star(self):**

**""" A\* search algorithm """**

**open\_set = {self.*start\_state*}**

**closed\_set = set()**

**g = {self.*start\_state*: 0}**

**parents = {self.*start\_state*: None}**

**moves = {self.*start\_state*: None}**

**while open\_set:**

**# Select node with the lowest f(n) = g(n) + h(n)**

**n = min(open\_set, key=lambda state: g[state] + self.*h*(state))**

**if n == self.*goal\_state*:**

**return self.*reconstruct\_path*(parents, moves, n)**

**open\_set.*remove*(n)**

**closed\_set.*add*(n)**

**for neighbor, move in self.*get\_neighbors*(n):**

**if neighbor in closed\_set:**

**continue**

**tentative\_g = g[n] + 1**

**if neighbor not in open\_set or tentative\_g < g[neighbor]:**

**g[neighbor] = tentative\_g**

**parents[neighbor] = n**

**moves[neighbor] = move**

**open\_set.*add*(neighbor)**

**return None # No solution found**

**def reconstruct\_path(self, parents, moves, node):**

**""" Reconstruct the path from the goal to the start state with states printed """**

**path = []**

**state\_sequence = []**

**while parents[node] is not None:**

**path.*append*(moves[node])**

**state\_sequence.*append*(node)**

**node = parents[node]**

**path.*reverse*()**

**state\_sequence.*reverse*()**

**print("Solution Steps:")**

**print\_state(self.*start\_state*)**

**print()**

**current\_state = self.*start\_state***

**for move, state in zip(path, state\_sequence):**

**print(f"Move: {move}")**

**print\_state(state)**

**print()**

**return path**

**def print\_state(state):**

**""" Utility function to print the puzzle state """**

**for row in state:**

**print(" ".*join*(str(cell) if cell != 0 else " " for cell in row))**

**start\_state = [[1, 2, 3],**

**[4, 0, 5],**

**[7, 8, 6]]**

**goal\_state = [[1, 2, 3],**

**[4, 5, 6],**

**[7, 8, 0]]**

**solver = EightPuzzle(start\_state, goal\_state)**

**solution = solver.*a\_star*()**

**if solution:**

**print("\nSolution Found!")**

**print("Steps to solve:", solution)**

**else:**

**print("No solution exists.")**

**Output:**

A black screen with blue and white text

AI-generated content may be incorrect.