|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **AY -ODD 2024-25** | | | | | | | | | | | | | |
| **GUJARAT TECHNOLOGICAL UNIVERSITY** | | | | | | | | | | | | | |
| **SCHOOL OF ENGINEERING AND TECHNOLOGY** | | | | | | | | | | | | | |
| **PRACTICAL - 5** | | | | | | | | | | | | | |
| **Course Code & Name** | | | **ME01095021- Artificial Intelligence** | | | | | | | | | | |
| **Academic Term:** | | | **AY –ODD 2024-25** | | | | | **Semester** | | | | **I** | |
| **Student Enrollment No:** | | | **241370795004** | | | | | **Batch:** | | | |  | |
| **Student Name:** | | | **Dake Darsh Dhaneshkumar** | | | | | | | | | | |
|  | |  |  |  | |  | |  | | | |  |  |
| **AIM/Objective:** | | | | | | | | | | | | | |
| 1 | | To implement A\* algorithm for 8-Puzzle Problem with local & global heuristic functions. | | | | | | | | | | | |
|  | |  |  |  | |  | |  | |  | | |  |
| **Expected Outcome:** | | | | | | | | | | **CO/PO/PSO** | | | |
| 1 | | The program finds the solution path for the 8-puzzle problem using both **Misplaced Tiles** and **Manhattan Distance** heuristics. It displays all visited states and the solution path for each heuristic. | | | | | | | | CO5 | | | |
|  | | **Experiment Result and Analysis**  **Resources and Software used:**   1. Python 3.12.6 2. Jupyter Notebook   **Code:**  class Node:  def \_\_init\_\_(self, data, level, fval):  """ Initialize the node with the data, level of the node, and the calculated f-value."""  self.data = data  self.level = level  self.fval = fval  def generate\_child(self):  """ Generate child nodes by moving the blank space in all possible directions."""  x, y = self.find(self.data, '0')  moves = [[x, y - 1], [x, y + 1], [x - 1, y], [x + 1, y]] # Left, Right, Up, Down  children = []  for move in moves:  child = self.shuffle(self.data, x, y, move[0], move[1])  if child is not None:  child\_node = Node(child, self.level + 1, 0)  children.append(child\_node)  return children  def shuffle(self, puz, x1, y1, x2, y2):  """ Swap the blank space in the specified direction."""  if 0 <= x2 < len(self.data) and 0 <= y2 < len(self.data):  temp\_puz = self.copy(puz)  temp\_puz[x2][y2], temp\_puz[x1][y1] = temp\_puz[x1][y1], temp\_puz[x2][y2]  return temp\_puz  else:  return None  def copy(self, root):  """ Create a copy of the matrix."""  return [row[:] for row in root]  def find(self, puz, x):  """ Find the position of the blank space."""  for i in range(len(self.data)):  for j in range(len(self.data)):  if puz[i][j] == x:  return i, j  class Puzzle:  def \_\_init\_\_(self, size):  """ Initialize the puzzle size, open and closed lists."""  self.n = size  self.open = []  self.closed = []  self.visited\_states = []  def misplaced\_tiles(self, start, goal):  """ Heuristic: Count the number of misplaced tiles."""  return sum(start[i][j] != goal[i][j] and start[i][j] != '0' for i in range(self.n) for j in range(self.n))  def manhattan\_distance(self, start, goal):  """ Heuristic: Calculate the Manhattan distance for each tile."""  distance = 0  goal\_positions = {goal[i][j]: (i, j) for i in range(self.n) for j in range(self.n)}  for i in range(self.n):  for j in range(self.n):  if start[i][j] != '0' and start[i][j] in goal\_positions:  x, y = goal\_positions[start[i][j]]  distance += abs(x - i) + abs(y - j)  return distance  def f(self, start, goal, heuristic):  """ f(x) = h(x) + g(x) """  return heuristic(start.data, goal) + start.level  def process(self, heuristic, heuristic\_name):  """ Solve the puzzle using the specified heuristic."""  # Static input for initial and goal states  start = [  ['1', '2', '3'],  ['8', '0', '4'],  ['7', '6', '5']  ]  goal = [  ['1', '2', '3'],  ['8', '4', '5'],  ['7', '0', '6']  ]  start = Node(start, 0, 0)  start.fval = self.f(start, goal, heuristic)  self.open.append(start)  print(f"\nSolution Path using {heuristic\_name} Heuristic:")  while self.open:  cur = self.open[0]  self.visited\_states.append(cur.data)  print("\nCurrent State:")  for row in cur.data:  print(' '.join(row))  if heuristic(cur.data, goal) == 0:  print("\nGoal state reached!")  break  for child in cur.generate\_child():  child.fval = self.f(child, goal, heuristic)  self.open.append(child)  self.closed.append(cur)  del self.open[0]  self.open.sort(key=lambda x: x.fval)  print("\nAll Visited States:")  for state in self.visited\_states:  for row in state:  print(' '.join(row))  print("----------")  # Create the puzzle object  puz = Puzzle(3)  # Solve using Misplaced Tiles heuristic  puz.process(puz.misplaced\_tiles, "Misplaced Tiles")  # Solve using Manhattan Distance heuristic  puz.process(puz.manhattan\_distance, "Manhattan Distance")  **Inputs:**    **Output:** | | | | | | | |  | | |  |
|  | | |  |  | | --- | --- | | **Conclusion** | | | 1 | **Manhattan Distance** heuristic is generally more efficient as it considers the actual distance of tiles to their goal positions, leading to fewer steps compared to **Misplaced Tiles**, which only counts incorrectly placed tiles. | | | | | | | | | | | | |
|  | |  |  |  |  | |  | |  | |  | | |
| **Evaluation Rubrics** | | | | | **Marks** | | **Inadequate** | | **Good** | | **Excellent** | | |
| **0%** | | **50%** | | **100%** | | |
| 1 | The understanding of the Student regarding the objective of the given practical | | | | **2** | |  | |  | |  | | |
| 2 | Installation of Software or Hardware Setup level | | | | **2** | |  | |  | |  | | |
| 3 | Quality of the Analysis done | | | | **2** | |  | |  | |  | | |
| 4 | Quality of the report including concluding remarks and Findings | | | | **2** | |  | |  | |  | | |
| 5 | Question & Answer related to given practical & timely submission | | | | **2** | |  | |  | |  | | |
|  | | | | | **10** | |  | |  | |  | | |
| **Total Marks Obtained Out of 10** | | | | | | |  | | | | | | |
|  | |  |  |  | |  |  | | | | |  |  |
|  | |  |  |  | |  |  | | | | |  |  |
|  | |  |  |  | |  |  | | | | |  |  |
|  | | **Date of Completion:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | |  | | **Course**  **Coordinator Sign:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | | | | | | | |