8-Puzzle Solver: Report

Title: 8-Puzzle Solver
Name: VIVEK KUMAR
Roll No: 202401100300282
Date: 11/03/2025

INTRODUCTION

The 8-puzzle is a classic sliding puzzle that challenges you to arrange tiles in the correct order by moving them into an empty space. Imagine a 3×3 grid with numbered tiles (1-8) and one blank space (0). The goal is to shift the tiles around until they match a given target arrangement.

This project is an **AI-powered 8-puzzle solver** that uses the *A search algorithm** to find the most efficient solution. It takes a starting puzzle configuration from the user and calculates the shortest sequence of moves needed to reach the goal state. The program uses the **Manhattan distance heuristic**, a method that helps the AI determine how close each tile is to its correct position.

Whether you're a student learning about AI search algorithms or just someone who enjoys solving puzzles, this solver provides a great way to explore intelligent problem-solving techniques!

The 8-puzzle is a widely studied problem in Artificial Intelligence (AI) and search algorithms. It serves as an excellent example of problem-solving using heuristic search techniques.

Example Configuration

Initial State:

1 2 3

4 0 5

6 7 8

Goal State:

1 2 3

4 5 6

7 8 0

In this project, we implement an **AI-based 8-puzzle solver** using the *A search algorithm**. The program uses the **Manhattan Distance heuristic** to determine the optimal path to reach the goal configuration from the initial state.

METHODOLOGY

Approach Used

The solution is implemented using the *A (A-Star) search algorithm**, which is an informed search technique. The A* algorithm evaluates each possible move based on the function:

Where:

- **g(n)**: The cost to reach the current state (depth of the search tree).
- **h(n)**: The heuristic cost to reach the goal state (Manhattan Distance heuristic).
- **f(n)**: The total estimated cost (used to prioritize states in a priority queue).

Manhattan Distance Heuristic

This heuristic calculates the sum of the distances each tile is from its goal position:

Where:

- represents the current position of a tile.
- represents the goal position of the tile.

Steps to Solve the Puzzle

1. **Input:** User provides the start and goal configurations as a 3×3 grid.

2. **Processing:**
    - Generate possible moves from the current state.
    - Evaluate each move using the heuristic function.
    - Select the best possible move and continue until the goal state is reached.
3. **Output:** Display the sequence of moves to reach the goal and the total number of moves required.

CODE:

```
import heapq

import numpy as np


def get_user_input():

    print("Enter the puzzle configuration as a 3x3 grid (use 0 for empty space):")

    board = []

    for i in range(3):

        row = list(map(int, input().split()))

        board.append(row)

    return np.array(board)


class Puzzle:

    def __init__(self, board, parent=None, move="", depth=0, cost=0):

        self.board = board

        self.parent = parent

        self.move = move
```

```
        self.depth = depth
        self.cost = cost

    def __lt__(self, other):
        return (self.depth + self.cost) < (other.depth + other.cost)

    def __eq__(self, other):
        return np.array_equal(self.board, other.board)

def heuristic(board, goal):
    distance = 0
    for i in range(3):
        for j in range(3):
            if board[i][j] != 0:
                x, y = np.where(goal == board[i][j])
                distance += abs(i - x[0]) + abs(j - y[0])
    return distance

def get_neighbors(node, goal):
    moves = []
    x, y = np.where(node.board == 0)
    x, y = int(x[0]), int(y[0])

    directions = {"Up": (-1, 0), "Down": (1, 0), "Left": (0, -1), "Right": (0, 1)}

    for move, (dx, dy) in directions.items():
```

```
        new_x, new_y = x + dx, y + dy

        if 0 <= new_x < 3 and 0 <= new_y < 3:
            new_board = node.board.copy()
            new_board[x][y], new_board[new_x][new_y] =
new_board[new_x][new_y], new_board[x][y]
            moves.append(Puzzle(new_board, node, move, node.depth + 1,
heuristic(new_board, goal)))

    return moves

def solve_puzzle(start, goal):
    start_node = Puzzle(start, None, "", 0, heuristic(start, goal))
    open_list = []
    closed_set = set()

    heapq.heappush(open_list, start_node)

    while open_list:
        current_node = heapq.heappop(open_list)

        if np.array_equal(current_node.board, goal):
            path = []
            while current_node.parent:
                path.append(current_node.move)
                current_node = current_node.parent
            return path[::-1]
```

```
        closed_set.add(str(current_node.board))

        for neighbor in get_neighbors(current_node, goal):
            if str(neighbor.board) not in closed_set:
                heapq.heappush(open_list, neighbor)

    return None

if __name__ == "__main__":
    print("Enter the start state:")
    start_state = get_user_input()
    print("Enter the goal state:")
    goal_state = get_user_input()

    solution = solve_puzzle(start_state, goal_state)
    if solution:
        print("Solution found in", len(solution), "moves:", solution)
    else:
        print("No solution possible.")
```

OUTPUT/RESULT:

```
Enter the start state:
Enter the puzzle configuration as a 3x3 grid (use 0 for empty space):
1 2 3
4 0 5
6 7 8
Enter the goal state:
Enter the puzzle configuration as a 3x3 grid (use 0 for empty space):
1 2 3
4 5 6
7 8 0
Solution found in 14 moves: ['Right', 'Down', 'Left', 'Left', 'Up', 'Right', 'Down', 'Right', 'Up', 'Left', 'Left', 'Down', 'Right', 'Right']
```

REFERENCE:

1.**Algorithm & Theory:**

- Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall.
- Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

2. **Python Libraries Used:**

- NumPy: https://numpy.org/
- Heapq (Priority Queue Module): https://docs.python.org/3/library/heapq.html

4. **Online Resources & Tutorials:**

- GeeksforGeeks: A* Algorithm for 8-Puzzle Problem - https://www.geeksforgeeks.org/a-search-algorithm/
- Stack Overflow discussions for implementation ideas.