AI-Based 64-Puzzle Solver with Graph Traversal Visualization

Submitted By: Muneeb Baig (22k-5095), Rafay Ahmed (22k-5030)

Course: Artificial Intelligence

Instructor: Ms Almas Ayesha Ansari

Submission Date: 26-Mar-2025

1. Project Overview

● **Project Topic:**

The project focuses on solving the classic 64-Puzzle game using Artificial Intelligence techniques. The 64-Puzzle consists of a 8x8 grid with numbered tiles (1-63) and one blank space. The goal is to arrange the tiles in order by sliding them into the blank space.

● **Objective:**

The objective is to design and develop an AI agent that can efficiently solve the 64-Puzzle using the A* Search Algorithm with Manhattan Distance heuristic. The project also aims to visualize the AI's traversal path and state transitions as a graph.

2. Game Description

● **Original Game Background:**

The 64-Puzzle is a sliding puzzle invented in the 1870s. It consists of fifteen numbered square tiles and one blank space on a 8x8 grid. Players slide the tiles horizontally or vertically into the empty space to achieve the ordered configuration.

● **Innovations Introduced:**

- Integration of A* Search Algorithm to solve the puzzle efficiently.
- Visualization of the AI's traversal using a graph (NetworkX and Matplotlib). - Development of an interactive Tkinter GUI to demonstrate the puzzle-solving process dynamically.

Impact:

These innovations enhance understanding of AI search algorithms by providing real-time visualization of node expansions, improving learning experience, and showcasing the puzzle-solving strategy.

3. AI Approach and Methodology

● **AI Techniques to be Used:**

- A* Search Algorithm: Utilized for finding the optimal path to the solution using cost and heuristic.

- Graph Search: The state space is represented and traversed as a directed graph. - Visualization: Graph traversal is visualized to demonstrate the AI's decision-making process.

● **Heuristic Design:**

- Manhattan Distance Heuristic: Calculates the sum of distances of each tile from its goal position, ignoring the blank tile. It ensures admissibility and optimality of the A* algorithm.

● **Complexity Analysis:**

- Time Complexity: Potentially exponential in the worst case due to the large branching factor. However, A* significantly reduces the number of nodes explored compared to uninformed search methods.
- Challenges: Efficiently handling visited states to avoid redundant computations and managing memory consumption during large state-space exploration.

4. Game Rules and Mechanics

● **Modified Rules:**

- The core 64-Puzzle rules are retained.
- The AI agent automatically performs moves based on the optimal path generated by the A* algorithm.
- The game state transitions are visualized as a directed graph showing explored paths.

● **Winning Conditions:**

The puzzle is considered solved when the tiles are arranged in order from 1 to 63 with the blank space (0) at the last position.

● **Turn Sequence:**

- There are no player turns. The AI iteratively selects the optimal move until the puzzle is solved.
- The GUI dynamically updates the board and visualizes each move made by the AI.

5. Implementation Plan

● **Programming Language:**

Python

● **Libraries and Tools:**

- Tkinter: For GUI development and interactive puzzle simulation.
- NetworkX: To create and manage the graph representing puzzle states.
- Matplotlib: To visualize the state traversal graph.
- Heapq: To efficiently manage the priority queue in A* implementation.
- Time, Random: For animation and random puzzle generation.