**1. Define Your Research Question**

How can search algorithms like BFS, DFS, and A\* effectively solve the 8-puzzle problem, and how do their performances compare in terms of solution quality and computational efficiency?

**2. Introduction**

The 8-puzzle is a sliding puzzle consisting of a 3x3 grid with tiles numbered from 1 to 8 and a blank space. The objective is to rearrange the tiles to match a predefined goal configuration. This puzzle is a classic benchmark problem in artificial intelligence, particularly in studying state-space search and optimization.

Solving the 8-puzzle efficiently is crucial for understanding the practical applications of search algorithms in real-world scenarios, such as robotics navigation, logistics, and automated problem-solving. This project implements a GUI-based solver using BFS, DFS, and A\* to visualize and analyze their behavior in solving the puzzle.

**3. Related Literature**

1. **State-Space Search Techniques in AI:**  
   This work discusses the theoretical foundation of state-space search algorithms like BFS and A\*. However, it lacks practical visualizations and comparisons of algorithmic behavior.
2. **The Sliding Puzzle Problem and Heurist:**  
   Focuses on using heuristic functions like Manhattan distance for A\*. However, it does not explore interactive solutions or user-centric visualizations.

**How My Work Differs:**

* This project integrates BFS, DFS, and A\* into a user-friendly GUI for easy interaction.
* It visualizes the solution path, providing a deeper understanding of the algorithms' performance.
* The implementation highlights the challenges faced in each algorithm, including memory consumption (DFS) and heuristic tuning (A\*).

**4. Achievements, Challenges, and Workarounds**

**Achievements:**

* Successfully implemented BFS, DFS, and A\* to solve the 8-puzzle.
* Developed a GUI to visualize the solution step-by-step.
* Highlighted the efficiency and shortcomings of each algorithm.

**Challenges and Workarounds:**

* **Challenge:** DFS's memory usage for deeper states.  
  **Workaround:** Introduced cutoff limits to prevent stack overflow.
* **Challenge:** Optimizing A\*'s heuristic for complex configurations.  
  **Workaround:** Fine-tuned the Manhattan distance heuristic to improve performance without increasing computational cost.

**5. GitHub Link: https://github.com/Neeraj1404nani/8-Puzzel-solver-using-search-algorithm**