**Project Title:**  
**AI-Based Solver for the N-Puzzle Problem Using Search Strategies**

**Submitted By:**  
Sameed Hasan (22k-4447)  
Muhammad Hamza Haroon (22k-4200)

**Course:**  
AI

**Instructor:**  
Muhammad Khalid Khan

**Submission Date:**  
[11-5-2025]

**1. Project Overview**

**● Project Topic:**  
This project focuses on solving the classic N-Puzzle game using a variety of AI-based search techniques. The solver supports board dimensions from 2x2 to 5x5 and allows users to define both the initial and goal states through a graphical user interface.

**● Objective:**  
The main objective is to develop a versatile AI agent capable of solving the N-Puzzle using various uninformed and informed search strategies, including A\*, BFS, DFS, and Greedy Best-First Search, and to compare their performance in terms of efficiency, optimality, and resource consumption.

**2. Game Description**

**● Original Game Background:**  
The N-Puzzle is a sliding puzzle consisting of an N×N grid with numbered tiles and one empty space. The goal is to rearrange the tiles to reach the desired configuration by sliding tiles into the empty space.

**● Innovations Introduced:**

* Custom puzzle size support (2x2 to 5x5).
* Graphical interface for setting start/goal states and visualizing solutions.
* Step-by-step solution playback with performance metrics.

**Impact on Gameplay Complexity and Strategy:**  
Introducing different board sizes and algorithm choices significantly affects the computational complexity, search space, and time required to find a solution, thereby offering insights into the practical performance of AI search strategies.

**3. AI Approach and Methodology**

**● AI Techniques to be Used:**

* A\* (with Manhattan distance heuristic)
* Uniform Cost Search
* Breadth-First Search (BFS)
* Depth-First Search (DFS)
* Greedy Best-First Search
* Iterative Deepening Search

**● Heuristic Design:**

* The Manhattan Distance heuristic is used to estimate the distance from the current state to the goal by summing the vertical and horizontal distances of each tile from its goal position.

**● Complexity Analysis:**

**1. Breadth-First Search (BFS)**

* **Time Complexity:** O(b^d)
* **Explanation:** Explores all nodes at depth d before moving deeper. Guarantees shortest path but uses a lot of memory.

**2. Depth-First Search (DFS)**

* **Time Complexity:** O(b^m)
* **Explanation:** Explores a single path to the maximum depth before backtracking. Not optimal and may get stuck if not bounded.

**3. Uniform Cost Search (UCS)**

* **Time Complexity:** O(b^(1 + C\*/ε))
  + where C\* = cost of the optimal solution, and ε = minimum step cost
* **Explanation:** Similar to BFS but prioritizes lowest cumulative cost g(n). Optimal and complete if all step costs are positive.

**4. Depth-Limited Search**

* **Time Complexity:** O(b^l)
  + where l = depth limit
* **Explanation:** DFS with a depth cutoff. Can miss solutions if l < d.

**5. Iterative Deepening DFS (IDDFS)**

* **Time Complexity:** O(b^d)
* **Explanation:** Repeated DFS from depth 0 to d. Combines the space-efficiency of DFS with the optimality of BFS.

**6. Greedy Best-First Search**

* **Time Complexity:** O(b^m) in the worst case
* **Explanation:** Selects nodes with the lowest heuristic value h(n) (here, Manhattan distance). Fast but not guaranteed optimal.

**7. A\* Search (with Manhattan Distance)**

* **Time Complexity:** O(b^d) (average), O(b^m) (worst case)
* **Explanation:** Expands nodes based on f(n) = g(n) + h(n). Very efficient with good heuristics and provides optimal solutions if h(n) is admissible.

**4. Game Rules and Mechanics**

**● Modified Rules:**

* User-defined board dimension (N×N)
* Initial and goal states are manually entered by the user.
* Moves allowed: Up, Down, Left, Right (only when adjacent to the blank space)

**● Winning Conditions:**

* Puzzle is solved when the current configuration matches the goal configuration.

**● Turn Sequence:**

* The AI executes moves in sequence without human input once the solving process begins.

**5. Implementation Plan**

**● Programming Language:**  
Python

**● Libraries and Tools:**

* **NumPy**: for matrix/state handling
* **Tkinter**: for GUI interaction
* **heapq, deque**: for efficient search structures
* **datetime**: for measuring execution time
* **Git**: for version control and collaboration

**● Milestones and Timeline:**

* **Week 1-2**: Puzzle structure and GUI design
* **Week 3-4**: Implementation of core search algorithms
* **Week 5-6**: Heuristic integration and optimization
* **Week 7**: Performance testing and result logging
* **Week 8**: Final report preparation and GUI polish

**6. References**

* Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th Edition).
* NumPy Documentation
* Python Official Documentation for Tkinter