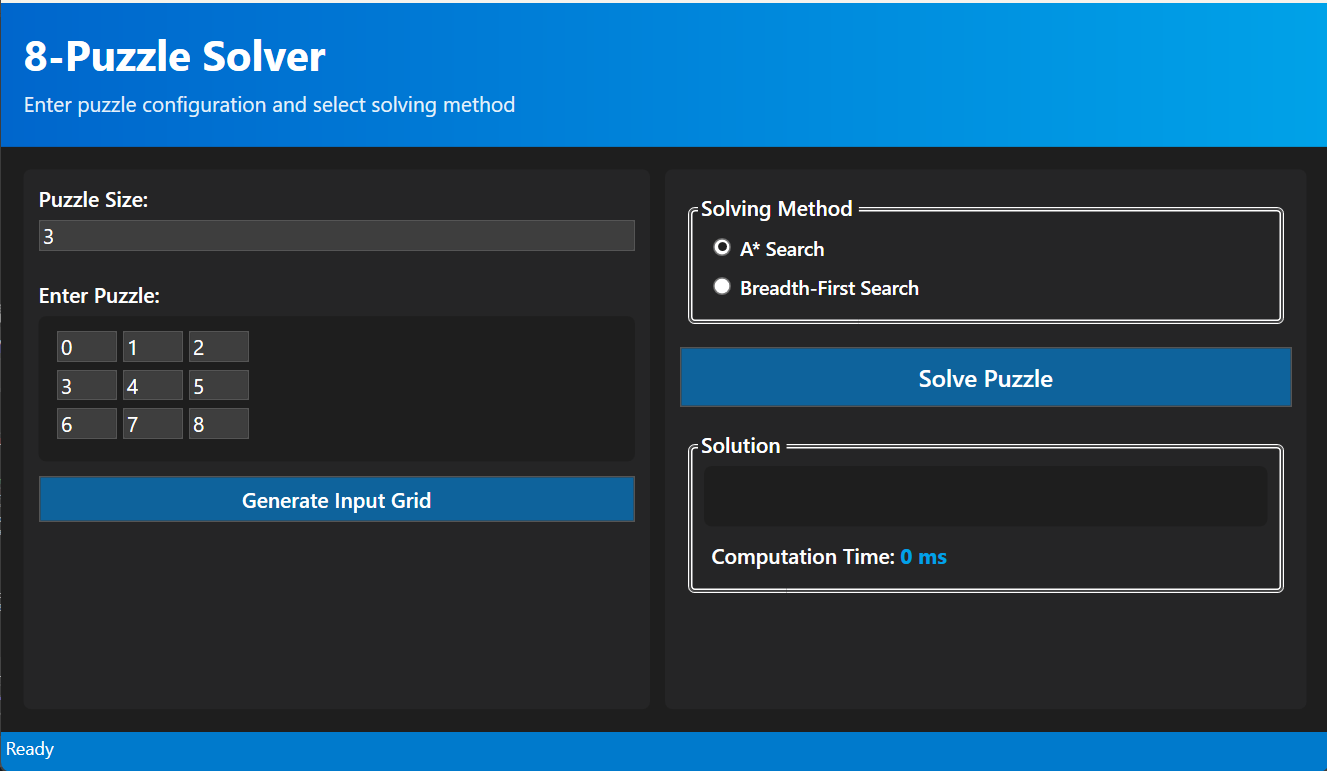


Puzzle Solver: Proposal Document



**1. Problem Statement**

The 8-Puzzle problem is a classic sliding puzzle consisting of a 3x3 grid with eight numbered tiles and one empty space. The objective is to rearrange the tiles from a given initial configuration to a goal state, typically with tiles ordered sequentially and the blank space at the end. This problem serves as a benchmark for evaluating search algorithms in artificial intelligence due to its manageable complexity and the necessity for efficient solution strategies.

**2. Objectives**

* **Develop a User-Friendly Interface**: Create a graphical user interface (GUI) using WPF (Windows Presentation Foundation) to allow users to input custom puzzle configurations and select solving algorithms.
* **Implement Solving Algorithms**: Integrate two primary search algorithms—A\* Search and Breadth-First Search (BFS)—to solve the puzzle efficiently.
* **Ensure Scalability**: Design the system to handle puzzles of varying sizes, from 2x2 up to 10x10 grids.
* **Provide Detailed Output**: Display the sequence of moves leading to the solution, along with performance metrics such as the number of moves and computation time.

**3. Chosen Techniques**

***3.1. A Search Algorithm*\***

A\* Search is an informed search algorithm that uses heuristics to guide the search process towards the goal state efficiently. In this project, the Manhattan distance heuristic is employed, which calculates the sum of the distances of each tile from its goal position.

*Implementation Details*:

* **Cost Calculation**: The getCost method in the State class computes the total cost as the sum of the path cost (level) and the heuristic cost (Manhattan distance).
* **Priority Queue**: A priority queue is used to select the next state to explore based on the lowest total cost.

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

BFS is an uninformed search algorithm that explores all neighboring nodes at the current depth before moving to nodes at the next depth level. It guarantees the shortest path in terms of the number of moves but can be memory-intensive.

*Implementation Details*:

* **Queue Structure**: A standard queue is used to manage the order of state exploration.
* **State Expansion**: All possible moves from the current state are enqueued for exploration.

**4. Expected Outcomes**

* **Functional Application**: A desktop application that allows users to input puzzle configurations, select a solving algorithm, and view the solution steps.
* **Algorithm Comparison**: Insights into the performance differences between A\* Search and BFS in terms of computation time and number of moves.
* **Scalability Assessment**: Evaluation of the application's ability to handle larger puzzle sizes and the corresponding impact on performance.