BITS COLLEGE

INTRODUCTION TO ARTIFICIAL INTELLIGENCE

AI PUZZLE SOLVER

Group Members:

1. BereketeAb Birku	0215/24
2. Dagem Teshome	0080/23
3. Etsubdenk Simmie	0217/24
4. Eyerusalem Wondimu	0218/24

SUBMITTED TO: Mr. Getachew Getu

SUBMISSION DATE: July 3, 2025

Table content

Abstract

The objective of this project is to design and implement an AI agent capable of solving the classical 8-puzzle problem using heuristic-based problem-solving techniques. The 8-puzzle is a sliding tile puzzle consisting of a 3x3 grid with eight numbered tiles and one empty space. The goal is to rearrange the tiles from a random initial configuration into a specific goal state by moving one tile at a time into the empty space.

To address this problem, we employed the A* (A-star) search algorithm, an informed search strategy that utilizes a heuristic function, in this case, the Manhattan distance to guide the search towards the goal efficiently along with the Breadth-First Algorithm. The AI agent receives user-defined initial puzzle states, validates their solvability, and outputs the optimal sequence of moves required to solve the puzzle.

The project showcases key artificial intelligence concepts such as state-space representation, heuristic evaluation, and goal-based problem solving. The final solution includes a simple user interface for input and visual representation of each step taken by the agent. This system demonstrates how AI can solve real-world-like search problems efficiently and intelligently.

4. Introduction

Artificial Intelligence is a field that focuses on creating intelligent systems capable of mimicking human decision-making and problem-solving abilities. One core area within AI is search-based problem solving, a fundamental approach used in real-world applications such as robotics, logistics, pathfinding, and strategy games.

In this project, we aim to develop an AI agent that can solve the classical 8-puzzle problem a sliding puzzle that challenges users to arrange eight numbered tiles on a 3×3 grid into a specified goal configuration. The puzzle includes one empty space, which allows adjacent tiles to be moved into it. Solving the puzzle involves rearranging the tiles through a series of legal moves until the goal state is achieved.

Before solving any algorithms, we first check the **inversion of the problem** state and the goal state. If they do not have matching inversions, we give the fiction an early break to raise an error that notifies that the problem can not be solved.

To solve this challenge, we implemented two search algorithms: A* and Breadth-First Search (BFS). A* is a widely recognized heuristic-based search technique that efficiently balances the actual cost from the start node with an estimated cost to reach the goal. In our implementation, we used the Manhattan distance heuristic to guide the A* search process.

We also implemented BFS as an alternative search strategy, which explores the puzzle state space level by level without using heuristics. This allowed us to directly compare the performance, memory usage, and path optimality between an informed and an uninformed search algorithm.

Our solution involved modeling the puzzle as a state space, determining all legal moves, checking whether a configuration was solvable, and generating a sequence of moves from start to goal. We also designed a clean and interactive user interface using Streamlit, allowing users to input their puzzle setup and follow the solution process step by step.

This project demonstrates several essential AI concepts such as intelligent agents, informed search strategies and performance evaluation through heuristic metrics. It also provides a hands on experience in translating theoretical algorithms into working solutions using modern programming tools.

5. Methodology

5.1 PEAS Framework

- Performance: Number of moves taken to solve the puzzle; execution time of the algorithm
- Environment: A 3x3 puzzle board containing 8 numbered tiles and one blank space
- Actuators: Move tiles by sliding them up, down, left, or right
- Sensors: Receives user input for the initial state of the puzzle

5.2 Algorithm Description

To solve the 8-puzzle efficiently, we implemented two search algorithms: A* and Breadth-First Search (BFS). Each algorithm explores the puzzle's state space differently, providing opportunities to compare performance and accuracy.

A* Search Algorithm

A* is a well-established informed search algorithm known for its optimality (it finds the best solution) and completeness (it finds a solution if one exists). It combines features of Uniform Cost Search and Greedy Best-First Search by evaluating each state using the formula:

$f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n)$

Where:

- **g(n)** is the exact cost to reach the current node from the start (number of moves so far).
- **h(n)** is the estimated cost to reach the goal from the current node (heuristic).

We used the Manhattan Distance as our heuristic. It calculates the sum of the distances each tile is from its goal position, only counting vertical and horizontal moves. This heuristic is:

- **Admissible** (never overestimates the true cost),
- **Consistent** and
- Ensures **optimal solutions** when used with A*.

A* uses a priority queue (min-heap) to always expand the most promising node based on f(n) until the goal state is found.

Breadth-First Search (BFS)

BFS is an uninformed search algorithm that expands all nodes at the current depth before moving to the next level. It is:

- **Complete** (guaranteed to find a solution if one exists),
- **Optimal** (only if all actions have the same cost), but
- **Inefficient** in terms of memory and time, especially for deep or complex puzzles.

BFS does not use heuristics. It explores nodes in a first-in, first-out (FIFO) order and is useful for comparing against A* to demonstrate the importance of heuristics in AI problem-solving.

Why Both Algorithms?

Using both A* and BFS allowed us to:

- Compare **informed vs. uninformed** search strategies,
- Observe differences in performance (speed and memory),
- Show how **heuristics improve efficiency** and solution quality.

6. Implementation

6.1 Tools and Libraries Used

- Python 3.10 - Main programming language
- Streamlit - Provides a simple web interface
- time - Measures execution time
- Copy - creates deep copies of puzzle states
- Unit tests create tests

6.2 Code Structure and Logic

The program includes:

- An `EightPuzzle` class to represent each puzzle state
- A solvability check based on inversion count
- A* search function that prioritizes optimal paths
- BFS a function that solves the same problem in another route
- A Streamlit-based UI for easy puzzle input and solution display

7. Results and Discussion

7.1 Output Sample

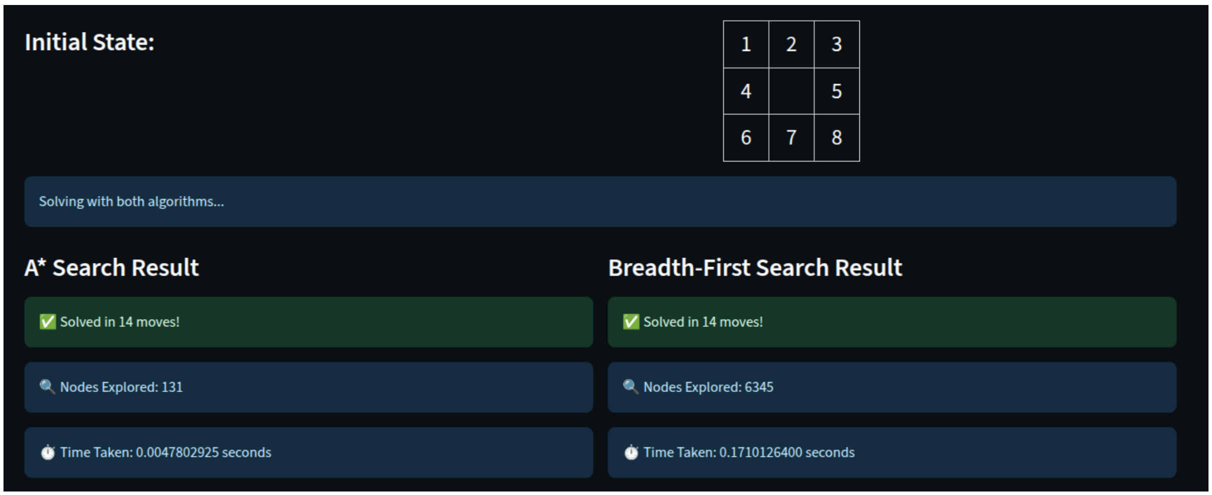
Initial State:

```
[1, 2, 3, 4, 0, 5, 6, 7, 8]
```

Goal State:

```
[1, 2, 3, 4, 5, 6, 7, 8, 0]
```

Solution Steps:



Initial State:

1	2	3
4		5
6	7	8

Solving with both algorithms...

A* Search Result

✅ Solved in 14 moves!

🔍 Nodes Explored: 131

⏱ Time Taken: 0.0047802925 seconds

Breadth-First Search Result

✅ Solved in 14 moves!

🔍 Nodes Explored: 6345

⏱ Time Taken: 0.1710126400 seconds

7.2 Performance Analysis

The agent consistently produced correct solutions for solvable puzzles. It operated efficiently across multiple test cases. Performance metrics:

- Puzzle Solved? Yes
- Execution Time: 0.002 – 0.05 seconds
- Number of Moves: 4 (in the test case)
- Nodes Expanded: ~131 vs 6345

The Streamlit UI simplified user interaction and made the agent accessible to non-programmers.

8. Conclusion and Future Work

The AI Puzzle Solver successfully demonstrated the use of both the A* algorithm and bfs in solving a classic AI problem. The agent produced optimal results with A* algorithm. and interacted well with user input through a simple UI.

While the current version is limited to 3x3 puzzles, future improvements could include:

- Support for larger puzzles (e.g., 15-puzzle)
- Visual animations of tile movements
- Adaptive heuristics using reinforcement learning
- Dockerized deployment for easier distribution