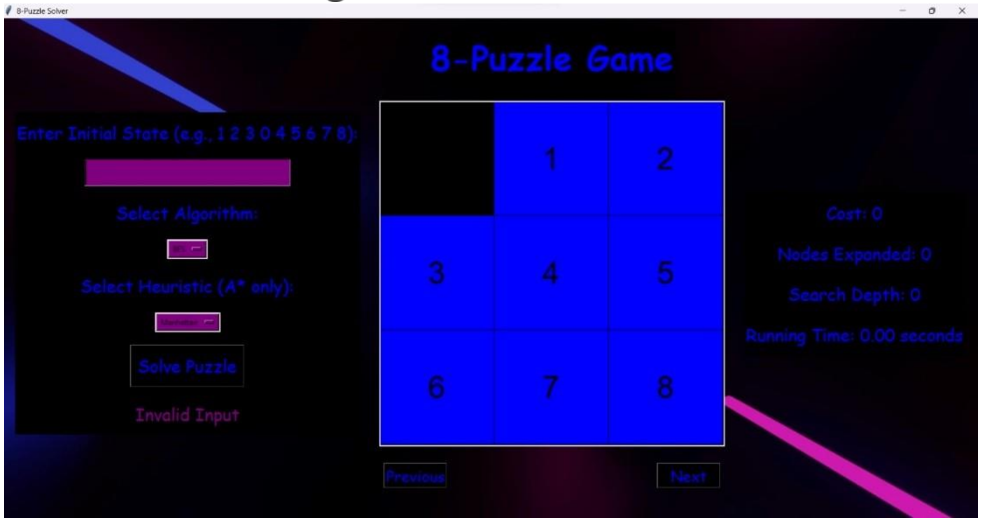
Artificial Intelligence

Assignment 1:

Using Informed and Uninformed
Search Algorithms to Solve 8-Puzzle



8-Puzzle Game

Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8):

Select Algorithm: BFS

Select Heuristic (A* only): Manhattan

Invalid Input

Cost: 0

Nodes Expanded: 0

Search Depth: 0

Running Time: 0.00 seconds

Under the supervision of:

- **Dr. Marwan Torki**

Names:	IDs:
Saad El Dine Ahmed Saad	7370
Morougue Mahmoud Ghazal	7524

8-Puzzle Solver

The **8-Puzzle** is a classic sliding puzzle game that consists of a 3x3 grid with eight numbered tiles and one empty space.

An instance of the **8-puzzle game** consists of a board holding 8 **distinct** movable tiles, plus an **empty space**. For any such board, the empty space may be legally swapped with any tile **horizontally** or **vertically adjacent** to it.

The objective is to arrange the tiles in the correct order by sliding them into the empty space.

In this project, we implemented three search algorithms: Breadth-First Search (**BFS**), Depth-First Search (**DFS**), and the **A*** algorithm with two heuristic functions (Manhattan and Euclidean distance) and to provide our game with an efficient algorithm we used 1D array instead of 2D array to avoid large time complexity.

Algorithms

➔ A precise and finite set of well-defined instructions or steps used to solve a problem, perform a task, or achieve a specific goal.

➢ Breadth-First Search (BFS):

Assumption: The BFS algorithm assumes that all moves have equal costs, and it explores the puzzle space level by level.

Data Structure: BFS uses a queue data structure to explore states in a first-in-first-out (FIFO) manner.

Operation: The algorithm starts from the initial state and expands all possible moves from each state while maintaining a queue of states to explore. It continues this process until it reaches the goal state.

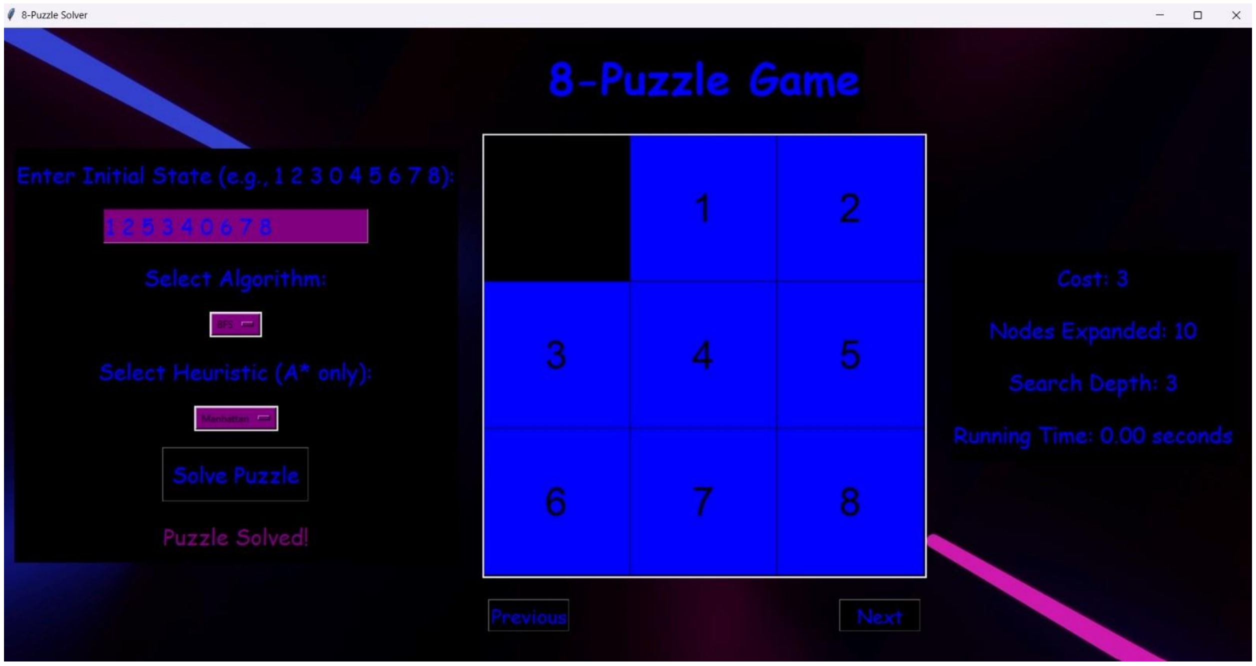
Path to Goal: The path to the goal state found by BFS is a sequence of board states that leads from the initial state to the goal state.

Cost of Path: The cost of the path is the number of moves needed to reach the goal state.

Nodes Expanded: BFS counts the number of nodes (states) it explores during the search.

Search Depth: The search depth is the max depth reached while expanding the nodes searching for the goal state.

Running Time: The running time is the time taken to find the solution.



Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8): 1 2 5 3 4 0 6 7 8

Select Algorithm: BFS

Select Heuristic (A* only): Manhattan

Puzzle Solved!

Cost: 3

Nodes Expanded: 10

Search Depth: 3

Running Time: 0.00 seconds

➢ Depth-First Search (DFS):

Assumption: DFS explores one branch of the search tree as deeply as possible before backtracking.
Data Structure: DFS uses a stack data structure to explore states in a last-in-first-out (LIFO) manner.

Operation: The algorithm starts from the initial state and explores a single branch of state as deeply as possible. If it reaches a dead-end, it backtracks and explores other branches until it finds the goal state.

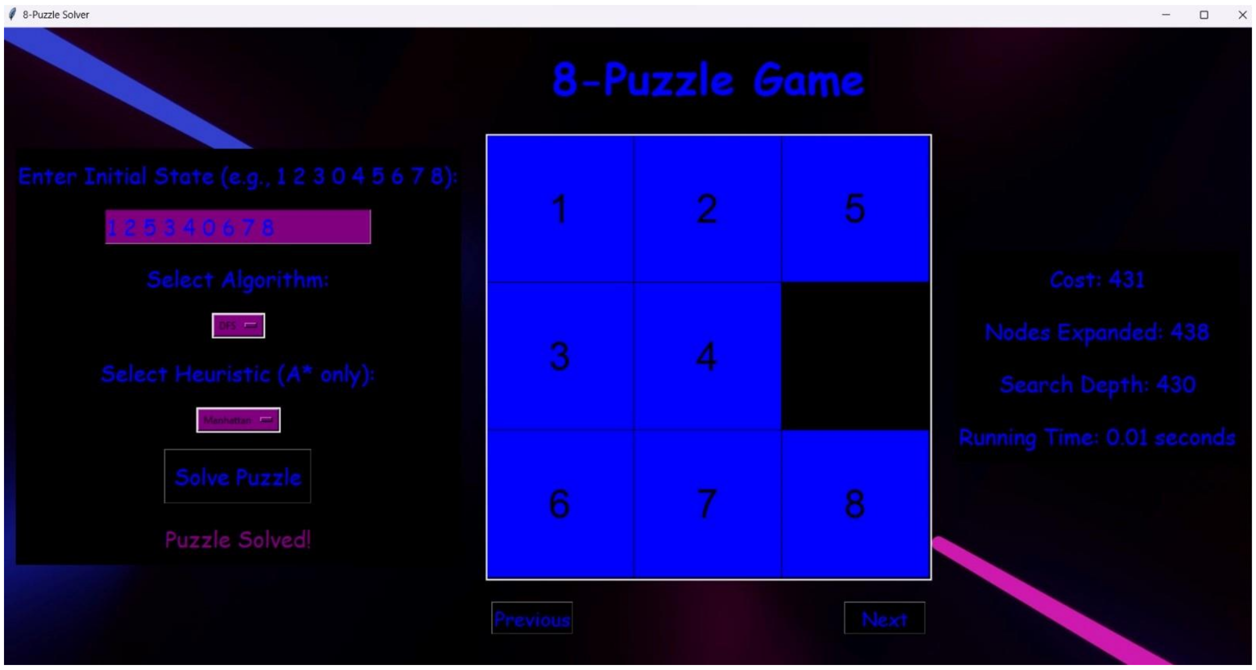
Path to Goal: The path to the goal state found by DFS is a sequence of board states that leads from the initial state to the goal state.

Cost of Path: The cost of the path is the number of moves needed to reach the goal state.

Nodes Expanded*:* DFS counts the number of nodes (states) it explores during the search.

Search Depth: The search depth is the depth of the goal state in the search tree.

Running Time: The running time is the time taken to find the solution.



Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8): 1 2 5 3 4 0 6 7 8

Select Algorithm: DFS

Select Heuristic (A* only): Manhattan

Puzzle Solved!

Cost: 431

Nodes Expanded: 438

Search Depth: 430

Running Time: 0.01 seconds

➢ A Star (A*):

Assumption: The A* algorithm is an informed search algorithm that uses heuristics to estimate the cost of the cheapest path to the goal.

Data Structure: A* uses a priority queue data structure to explore states in order of estimated cost.

Operation: The algorithm starts from the initial state and explores states based on their estimated total cost, which is the sum of the actual cost (path length) and a heuristic cost. It chooses states with the lowest total cost for exploration.

Path to Goal: The path to the goal state found by A* is a sequence of board states that leads from the initial state to the goal state.

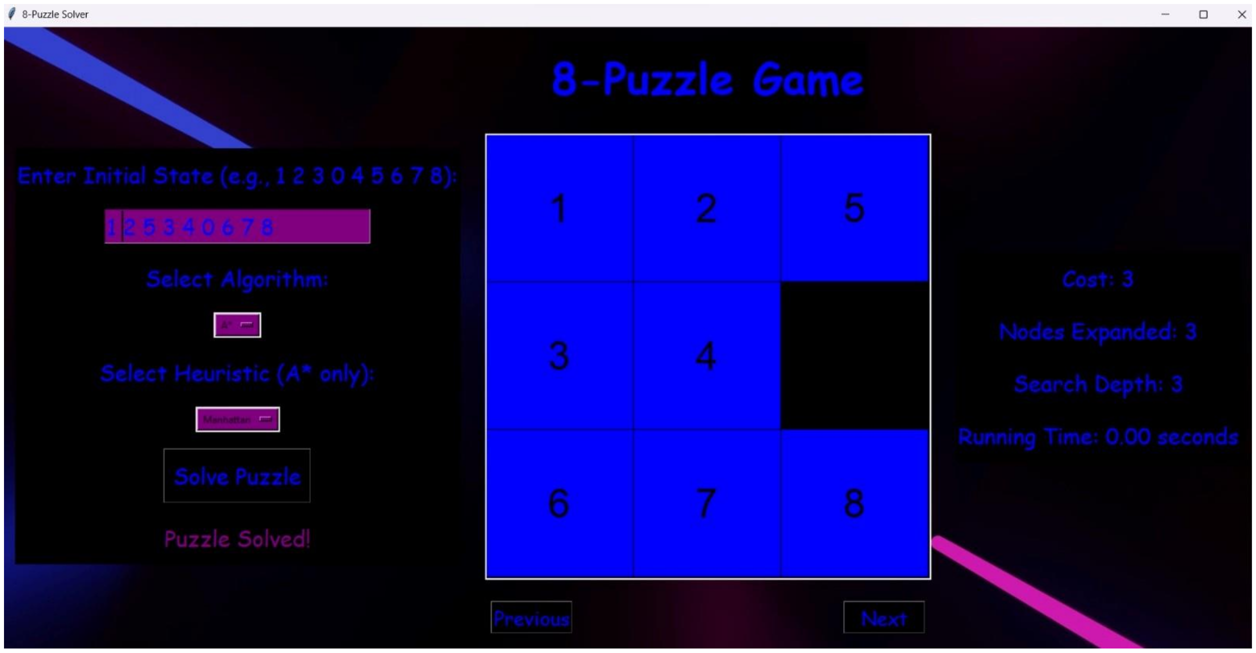
Cost of Path: The cost of the path is the number of moves needed to reach the goal state.

Nodes Expanded: A* counts the number of nodes (states) it explores during the search.

Search Depth: The search depth is the depth of the goal state in the search tree.

Running Time: The running time is the time taken to find the solution.

A* using Manhattan Distance:



8-Puzzle Solver

8-Puzzle Game

Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8):

1 2 5 3 4 0 6 7 8

Select Algorithm: A*

Select Heuristic (A* only): Manhattan

Solve Puzzle

Puzzle Solved!

1	2	5
3	4	
6	7	8

Cost: 3

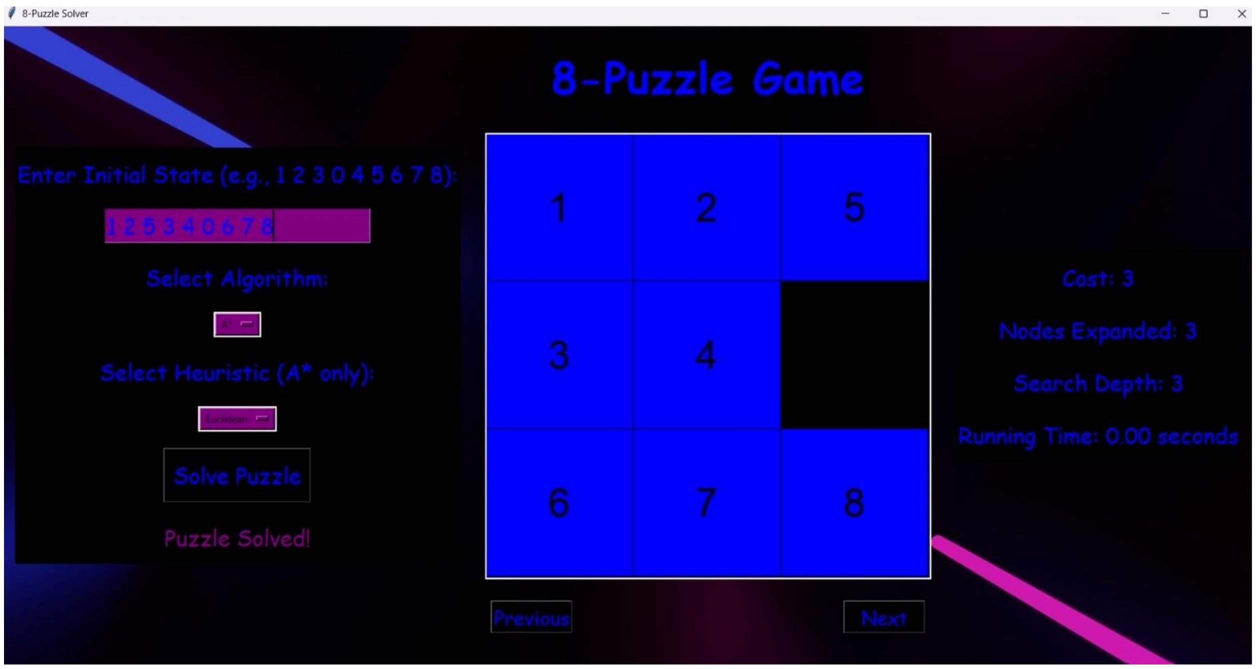
Nodes Expanded: 3

Search Depth: 3

Running Time: 0.00 seconds

Previous Next

A* using Euclidean Distance:



8-Puzzle Solver

8-Puzzle Game

Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8):

1 2 5 3 4 0 6 7 8

Select Algorithm: A*

Select Heuristic (A* only): Euclidean

Solve Puzzle

Puzzle Solved!

1	2	5
3	4	
6	7	8

Cost: 3

Nodes Expanded: 3

Search Depth: 3

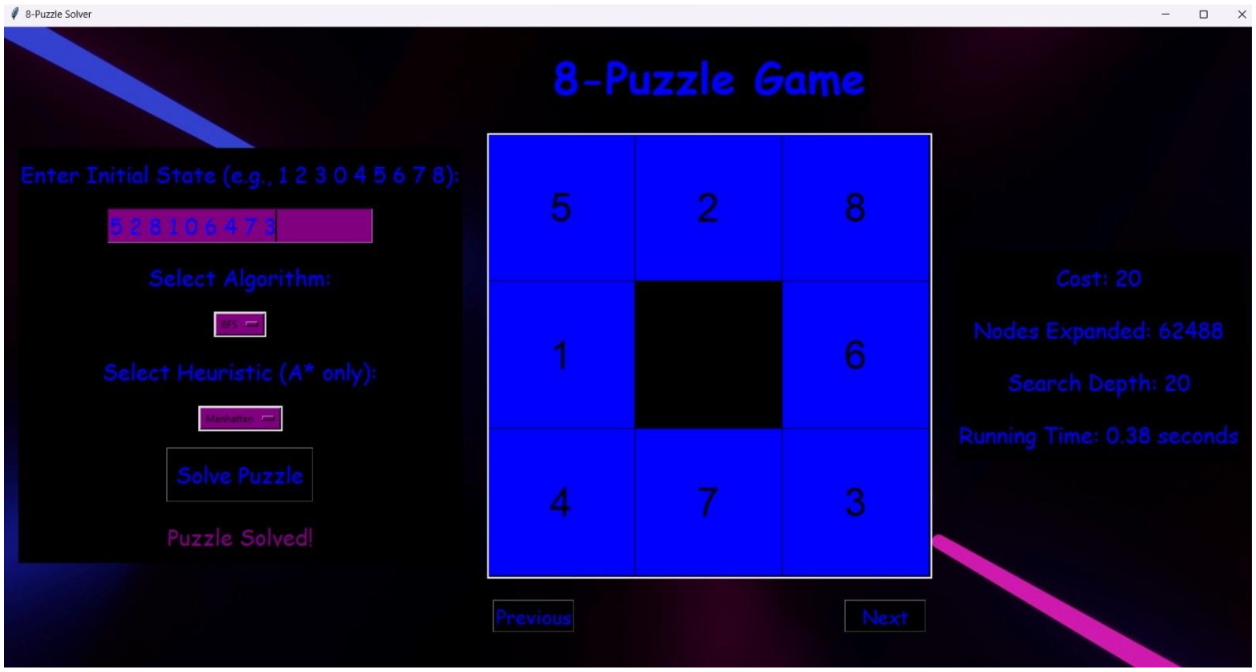
Running Time: 0.00 seconds

Previous Next

Sample Runs

We conducted sample runs of our 8-Puzzle solver using the provided graphical user interface (GUI).

1. BFS Algorithm:



Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8): 5 2 8 1 0 6 4 7 3

Select Algorithm: BFS

Select Heuristic (A* only): Manhattan

Puzzle Solved!

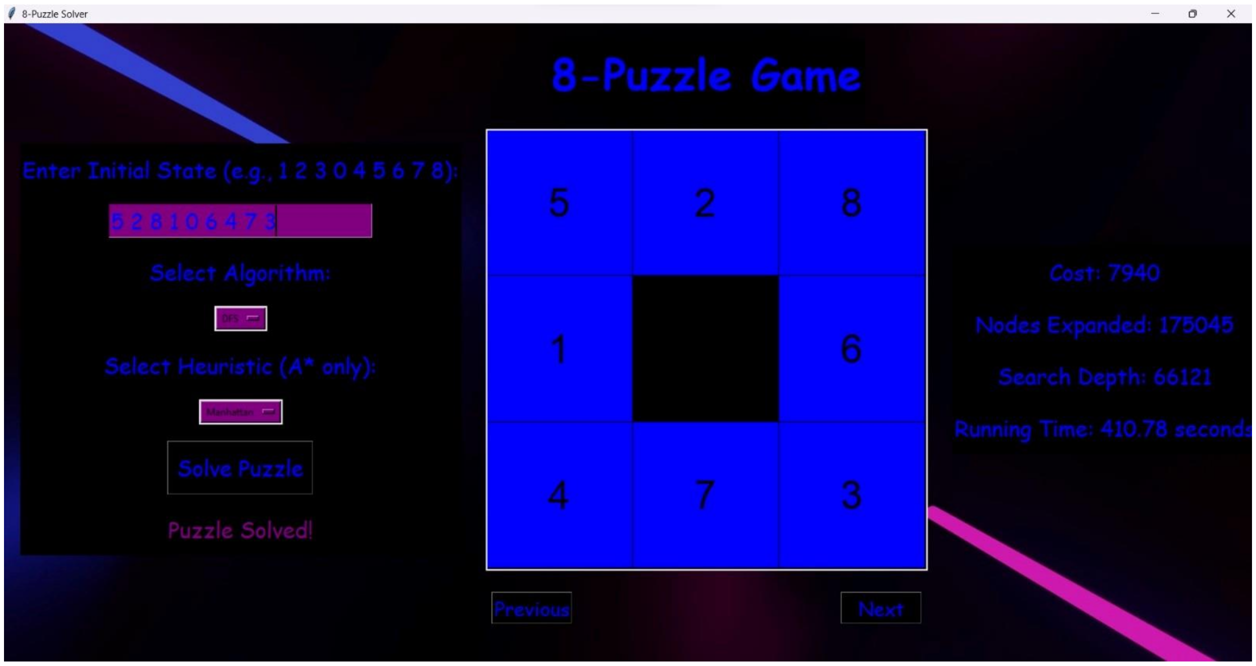
Cost: 20

Nodes Expanded: 62488

Search Depth: 20

Running Time: 0.38 seconds

2. DFS Algorithm:



Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8): 5 2 8 1 0 6 4 7 3

Select Algorithm: DFS

Select Heuristic (A* only): Manhattan

Puzzle Solved!

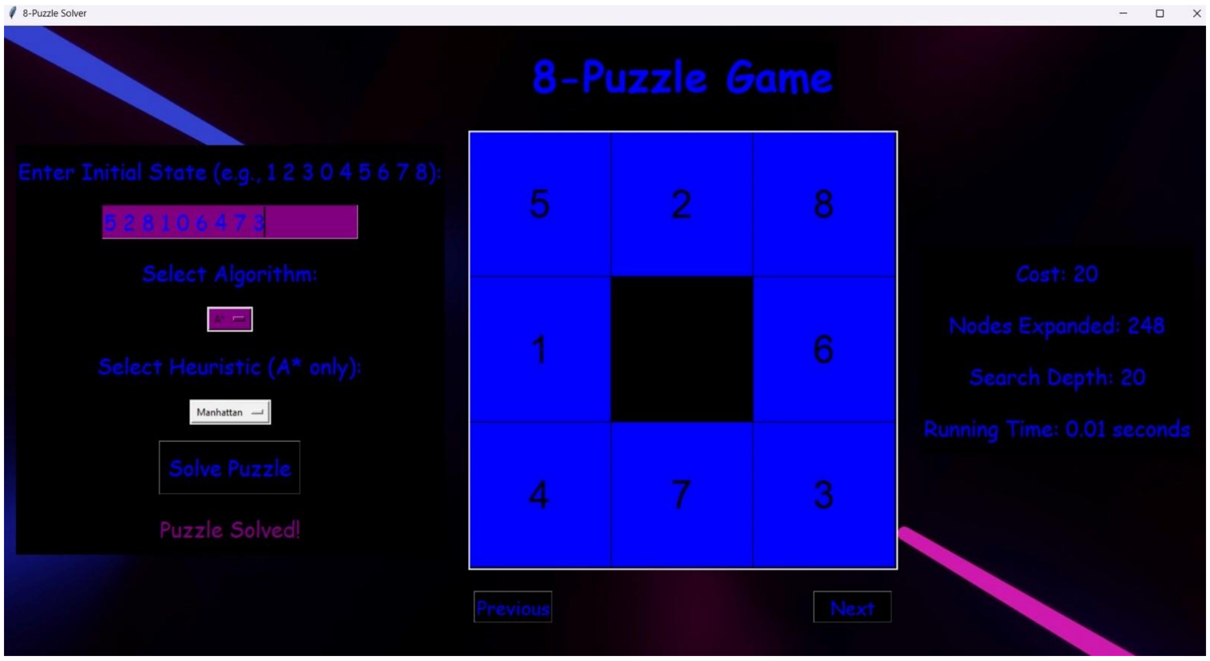
Cost: 7940

Nodes Expanded: 175045

Search Depth: 66121

Running Time: 410.78 seconds

3. A* Algorithm (Manhattan / Euclidean Heuristic):



8-Puzzle Game

Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8): 5 2 8 1 0 6 4 7 3

Select Algorithm: A*

Select Heuristic (A* only): Manhattan

Puzzle Solved!

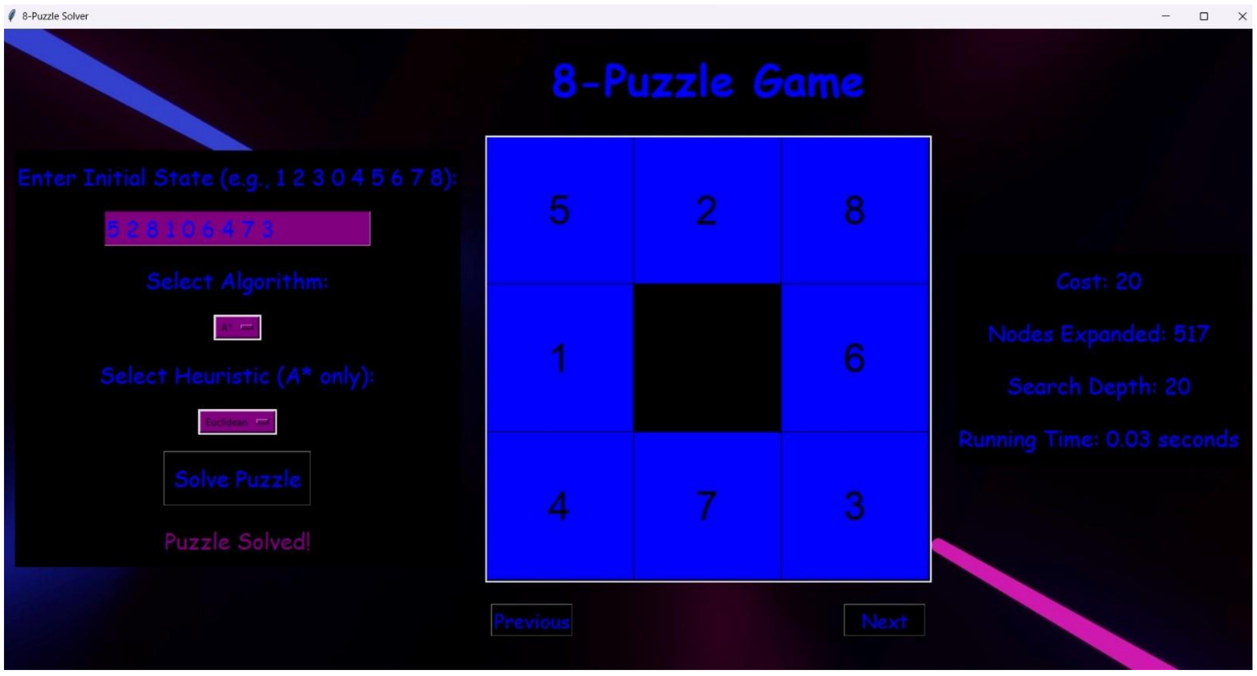
5	2	8
1		6
4	7	3

Cost: 20

Nodes Expanded: 248

Search Depth: 20

Running Time: 0.01 seconds



8-Puzzle Game

Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8): 5 2 8 1 0 6 4 7 3

Select Algorithm: A*

Select Heuristic (A* only): Euclidean

Puzzle Solved!

5	2	8
1		6
4	7	3

Cost: 20

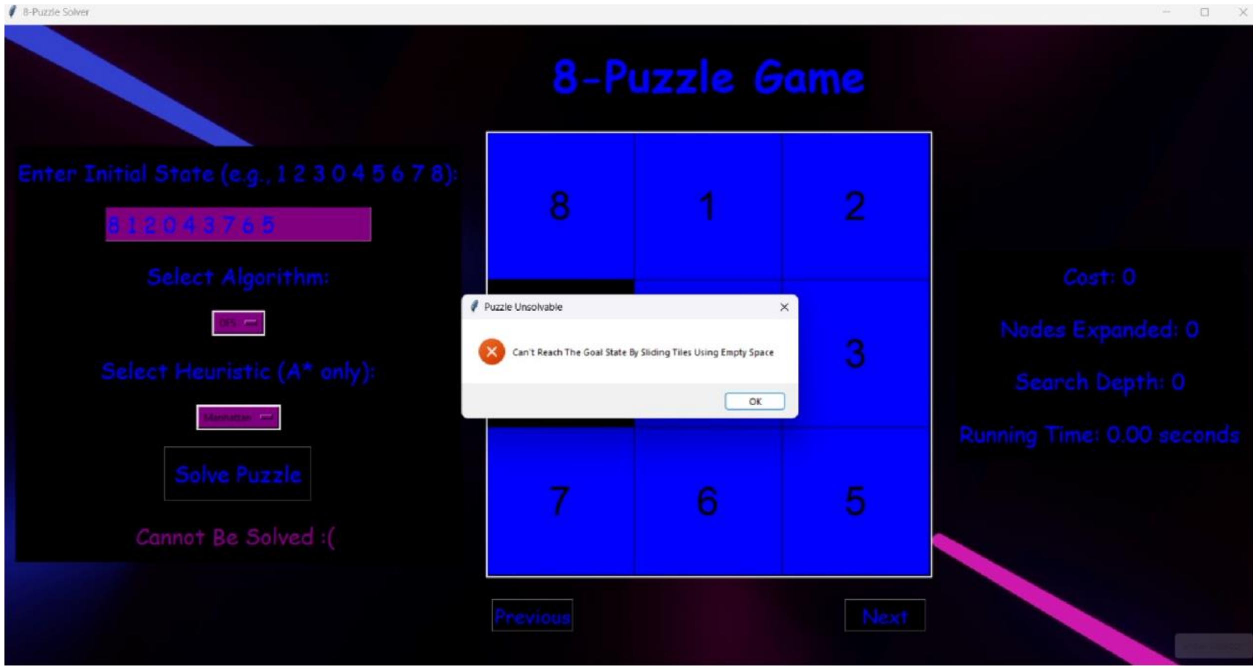
Nodes Expanded: 517

Search Depth: 20

Running Time: 0.03 seconds

Here are samples of some handled cases:

➔ User inputs an unsolvable puzzle:


8-Puzzle Solver

8-Puzzle Game

Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8): 8 1 2 0 4 3 7 6 5

Select Algorithm: DFS

Select Heuristic (A* only): Manhattan

Solve Puzzle

Cannot Be Solved :(

Puzzle Unsolvable

Can't Reach The Goal State By Sliding Tiles Using Empty Space

OK

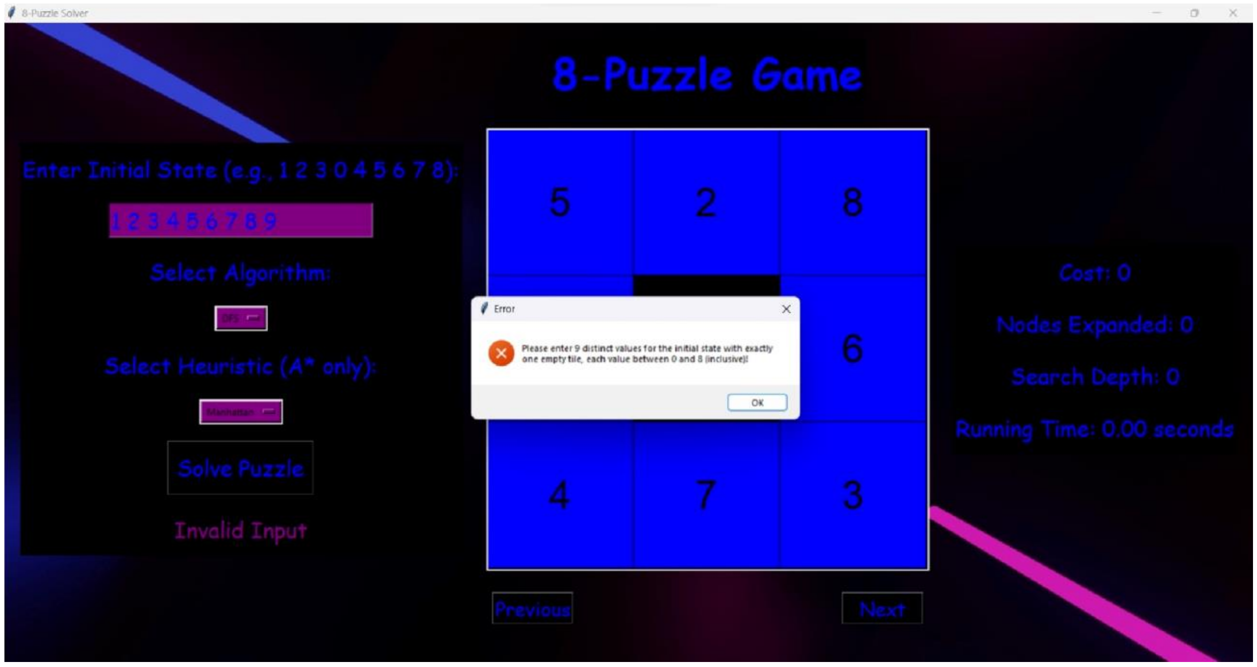
Cost: 0

Nodes Expanded: 0

Search Depth: 0

Running Time: 0.00 seconds

➔ User inputs digits out of the game range (0-8):


8-Puzzle Solver

8-Puzzle Game

Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8): 1 2 3 4 5 6 7 8 9

Select Algorithm: DFS

Select Heuristic (A* only): Manhattan

Solve Puzzle

Invalid Input

Error

Please enter 9 distinct values for the initial state with exactly one empty tile, each value between 0 and 8 (inclusive)!

OK

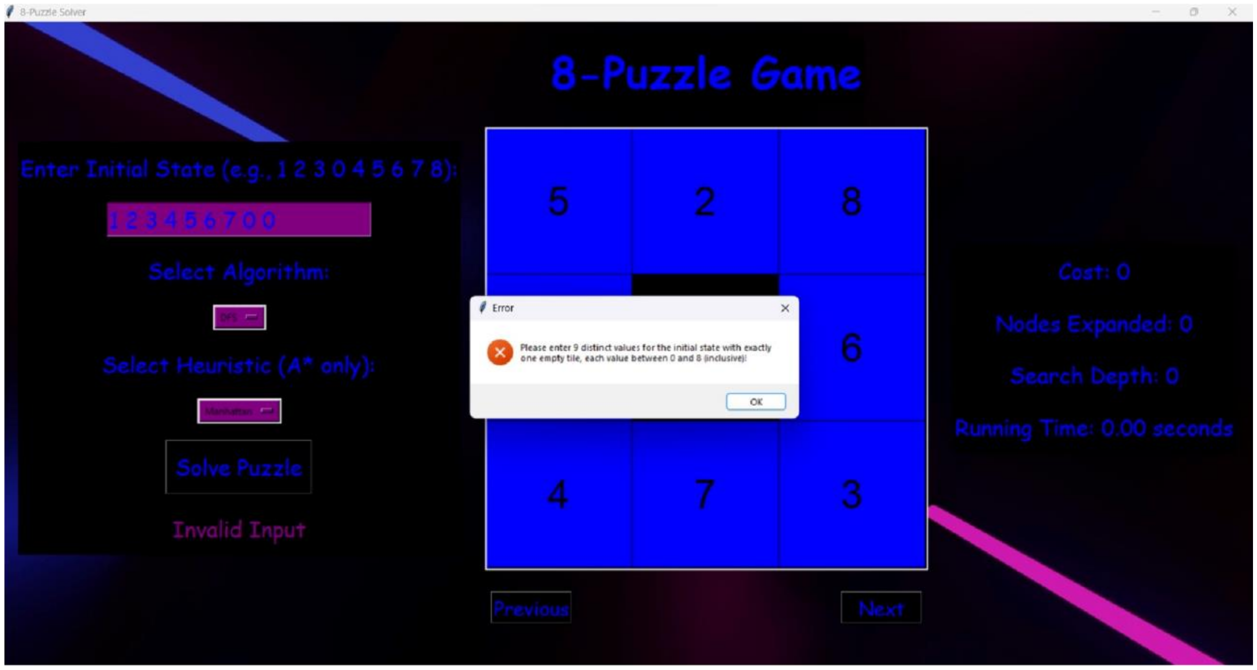
Cost: 0

Nodes Expanded: 0

Search Depth: 0

Running Time: 0.00 seconds

➔ User inputs identical numbers instead of distinct ones:



Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8): 1 2 3 4 5 6 7 0 0

Select Algorithm: DFS

Select Heuristic (A* only): Manhattan

Invalid Input

Cost: 0

Nodes Expanded: 0

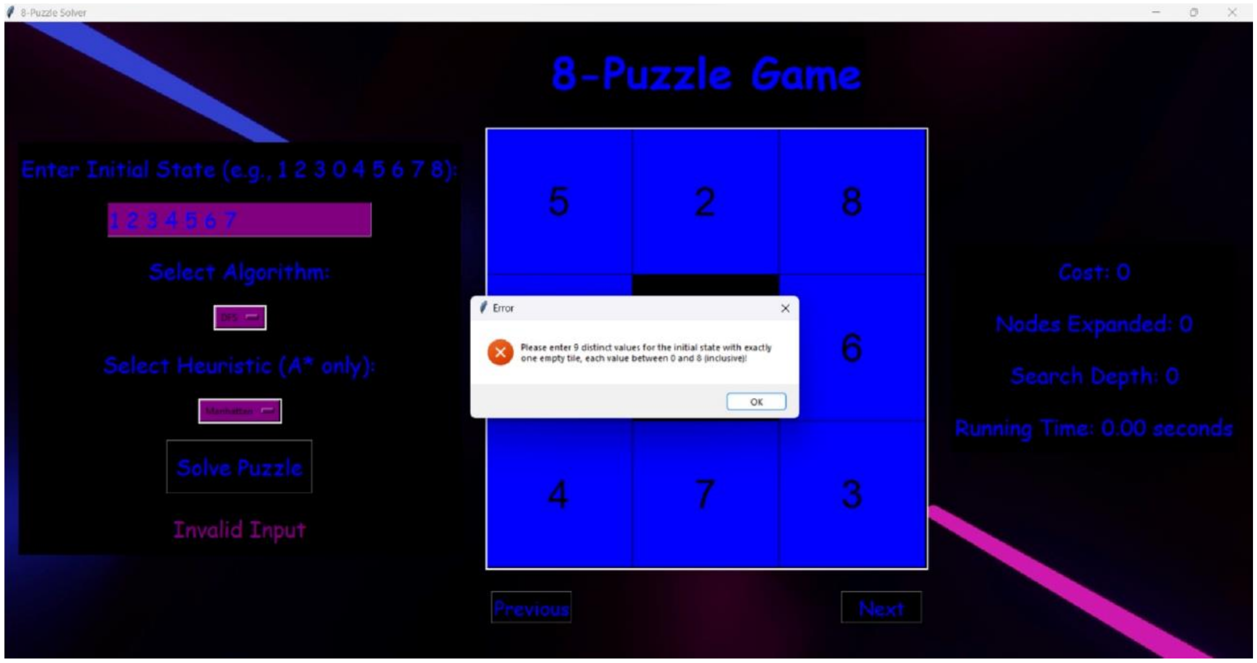
Search Depth: 0

Running Time: 0.00 seconds

Error

Please enter 9 distinct values for the initial state with exactly one empty tile, each value between 0 and 8 (inclusive)!

➔ User inputs less than 9 digits:



Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8): 1 2 3 4 5 6 7

Select Algorithm: DFS

Select Heuristic (A* only): Manhattan

Invalid Input

Cost: 0

Nodes Expanded: 0

Search Depth: 0

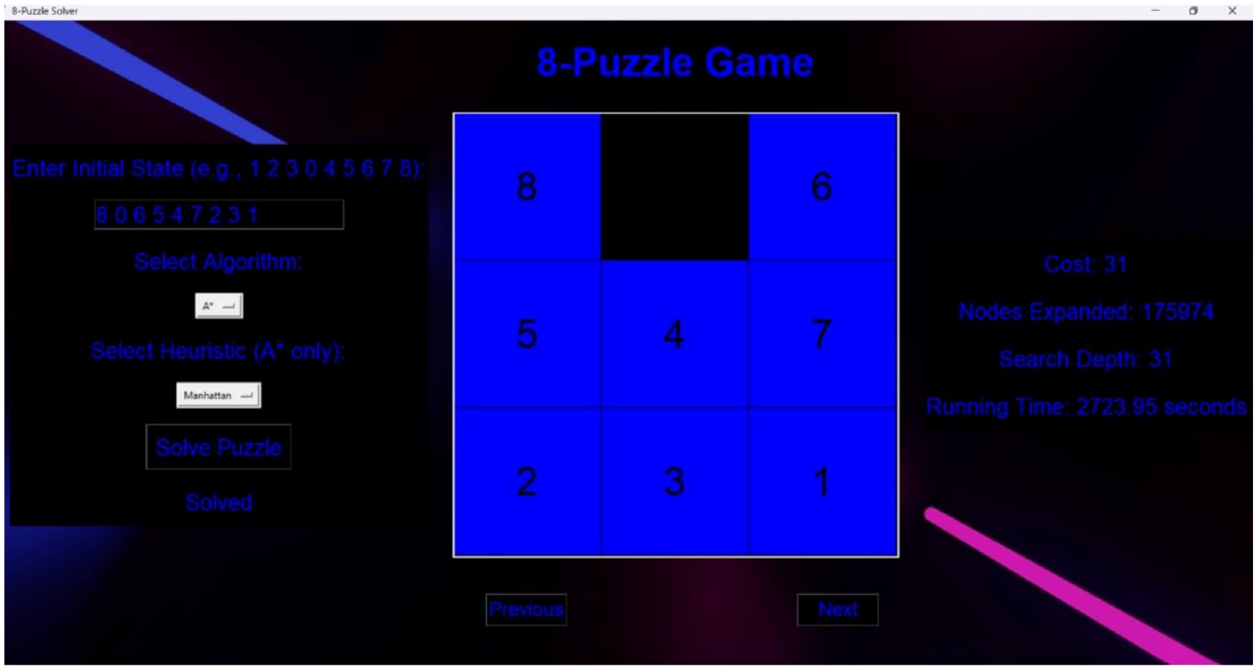
Running Time: 0.00 seconds

Error

Please enter 9 distinct values for the initial state with exactly one empty tile, each value between 0 and 8 (inclusive)!

➔ **Algorithms used to take too much time before using 1D array instead of 2D array to avoid large time complexity in addition adding a set to check the explored nodes in DFS instead of looping inside the explored queue which slows down the search:**

Before:



Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8): 8 0 6 5 4 7 2 3 1

Select Algorithm: A*

Select Heuristic (A* only): Manhattan

Solved

8		6
5	4	7
2	3	1

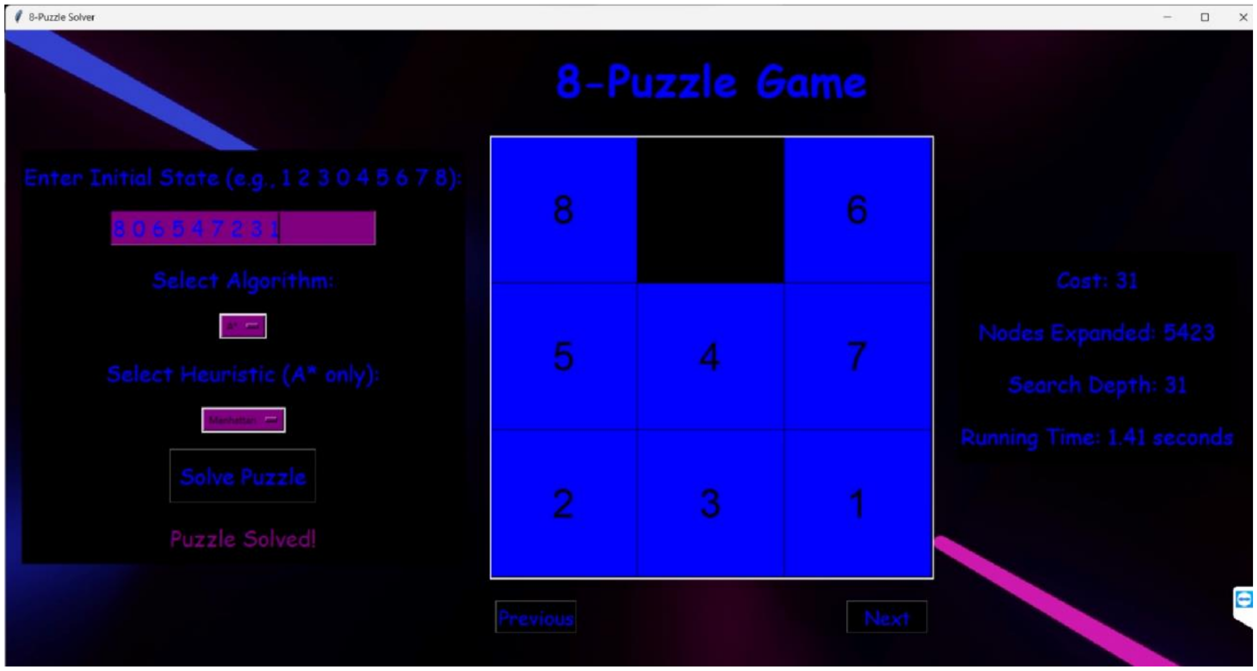
Cost: 31

Nodes Expanded: 175974

Search Depth: 31

Running Time: 2723.95 seconds

After: **[Notice the difference in Running Time]**



Enter Initial State (e.g., 1 2 3 0 4 5 6 7 8): 8 0 6 5 4 7 2 3 1

Select Algorithm: A*

Select Heuristic (A* only): Manhattan

Puzzle Solved!

8		6
5	4	7
2	3	1

Cost: 31

Nodes Expanded: 5423

Search Depth: 31

Running Time: 1.41 seconds

Assumptions Made by the User while using the app or While Writing the Code:

In any software development project, the assumptions made during the coding process are fundamental in **guiding** the behavior and functionality of the software. Assumptions serve as implicit or explicit **expectations** regarding the environment, input data, user behavior, and other critical factors that **influence** the code's **performance** and **reliability**.

Python Environment:

The user assumes that they are working in a Python environment or jupyter notebook with the necessary libraries, such as matplotlib, numpy, and tkinter, installed.
This code will not work in an environment where these libraries are not available.

Initial State Input Format:

The user assumes that the input for the initial state of the puzzle will be provided as a string with nine space-separated values (e.g., "1 2 3 0 4 5 6 7 8"). The code does not handle input validation for other formats, otherwise, an error message will pop up.

Solvable Puzzle:

The code doesn’t assume that the initial puzzle state is always solvable. It always checks whether the provided initial state can be solved, and if it's not solvable, an error message will pop up.

Algorithm and Heuristic Selection:

The user can select the algorithm (DFS, BFS, A*) and heuristic (Manhattan or Euclidean) for the A* algorithm.

The user assumes that they have selected the required algorithm and heuristic for the given puzzle, which might not always be the case.

Time Limit:

The user assumes that the puzzle will be solved within a specific time limit (e.g., 5 minutes). If the puzzle is not solved within this time frame, the code returns a timeout error.

User Interface Understanding:

The user assumes that users interacting with the GUI understand the purpose and functionality of the provided buttons, labels, and dropdown menus.

Visual Output:

The user assumes that users can visually interpret the puzzle board and results using the provided graphical user interface. The code relies on a graphical representation to show the puzzle board and its solving steps.

Path Drawing:

The code assumes that it is acceptable to draw the entire path of puzzle states for visualizing the solution. This can result in a lengthy path that might not be practical to display or navigate.

User-Friendly Input:

The user assumes that users will provide valid and meaningful input, adhering to the format and constraints mentioned in the code.

Assumption of Distinct Values:

The code assumes that all values in the initial puzzle state are distinct integers from 0 to 8, with exactly one 0 indicating the empty space. Any deviation from this assumption may lead to unexpected behavior.

Manhattan and Euclidean Heuristics:

The code assumes that the Manhattan and Euclidean heuristics are valid and effective for estimating the cost to reach the goal state. The user may not consider other heuristics that might be more suitable for certain puzzle configurations.

Summary:

In this project, we implemented **three search algorithms** to solve the **8-Puzzle game**.

We **provided a graphical user interface** for users to **input initial states** and **select algorithms**.

The algorithms (BFS, DFS, and A*) were able to find solutions for **solvable puzzles** and **report metrics** such as path length, nodes expanded, search depth, and running time.

The **choice of algorithm** and **heuristic** can significantly affect the **efficiency** of solving the puzzle, as demonstrated by the sample runs.

This project demonstrates the application of different search algorithms and heuristics in solving a classic puzzle.

Note:
All tree levels start from 0, also search depth