A blue circle with white text

Description automatically generated

|  |  |
| --- | --- |
| Name | Mahrukh Muzzamil  Sania Nisar |
| Roll Number | 21L-6084  21L-6065 |
| Section | BSE-7A |
| Submitted to | Mr. Mubashir Baig |
| Date | September 20, 2024 |

**Applied Artificial Intelligence**

**Assignment 1**

National University of Computer and Emerging Science

**Documentation of the code**

**Introduction**

The code below uses the A\* algorithm to solve 2x2 (3 tiles), 3x3 (8 tiles), 4x4 (15 tiles), and 5x5 (24 tiles) sliding puzzles. The problems were reformulated movements of numbered tiles so that they would be in a given order. Legal moves take any adjacent tiles to an empty square.

**Classes and Methods**

1. **Board Class**

**Fields:**

1. **ngcheck:** The current state of the puzzle represented as a list
2. **hslot:** Size of the grid of the puzzle.
3. **moves**: Number of moves made thus far to reach the current state.
4. **previous:** The reference of the previous Board state.
5. **blank\_pos:** The position of blank tile (0).

**Methods:**

1. **\_\_init\_\_:** Builds up the board state and finds the position of blank tile.
2. **blankfinding:** Finds the position of the blank tile
3. **goalcheck:** Checks if the current state is goal state
4. **movegen:** Generate all possible moves by moving tiles to the position of blank
5. **\_\_lt\_\_:** Compares two Board object instances based on move count.
6. **manhattan\_dist function:**

* Returns the Manhattan distance heuristic as the sum of the absolute differences between the current positions of tiles and the corresponding target positions.

1. **astar\_search function**

Implementation of A\* Algorithm is as as follows:

1. Maintains an open list priority queue of boards to be explored and a closed set of boards already evaluated
2. It explores the board states; it generates neighbors and evaluates their costs.
3. Returns the path of the solution and the number of moves if one exists otherwise, it returns None
4. **pathtrace Function**

* Re-tracing back the path taken from the initial board state to the solution based on references of the previous step

1. **puzzleload Function**

* Reads the puzzle configuration from a text file. File consists of the size of the puzzle, maximum number of moves, and an initial state with a corresponding target state.

1. **solutiondisplay Function**

* Displays the sequence of moves that was taken to achieve the goal

**Heuristic Used**

* Manhattan Distance: It forms an admissible heuristic because it never overestimates the actual cost to reach the goal; it is also helpful in prioritizing states close to the goal configuration by the A\* algorithm.

**Results and Statistics**

1. The code measures the time taken from the time it begins solving the puzzle until it finds a solution or exhausts all its possibilities.
2. The performance will vary depending on the initial configuration and on the maximum allowed moves.

A screenshot of a computer program

Description automatically generatedThis will generate some example results by running the program with different input files to see how fast it can find solutions for puzzles of different sizes.

**Performance Measurement**

The time module is used to measure the execution time. This gives an easily accessible way of

benchmarking the performance of the algorithm against various configurations of the puzzle. Attaching screenshots:

A screenshot of a computer program

Description automatically generated

**File Format**

The input file format should have in it:

1. The first integer indicating the size of the puzzle.
2. The second integer is the maximum number of moves allowed
3. The next n^2 integers are the starting state.
4. The last n^2 integers are the target state.