Solving 8-puzzle using A-star in Java

You are to implement the A* search algorithm to solve the 8-puzzle. The 8-puzzle consists of a 3x3 grid of squares, tiles numbered 1-8, and one blank space. For instance, an initial configuration of the puzzle might look like this:

2	3	6
	4	5
8	1	7

The goal of your program is to get it into the following arrangement:

1	2	3
8		4
7	6	5

Note that the blank space is in the middle. A move consists of moving one of the tiles next to the blank space into the spot where the blank space is. This leaves the spot previously occupied by the tile blank. We will call the possible moves L, R, U, and D based on whether the blank space moves to the left, right, up or down respectively.

Your task is to write a Java program which takes a file describing an initial puzzle configuration, and uses the A* algorithm to search for a series of moves which will solve the puzzle. (If you wish to use a different programming language, please consult the instructor).

Heuristic Function: Recall that A* evaluates nodes according to the path cost to the node plus the *estimated* path cost to the goal node. It expands the node with the smallest total first.

Your job is to implement and test any two different heuristic functions. One of these heuristics must be *admissible*, and at least one must be *non-admissible*. The heuristics you implement should be the "**Manhattan distance**" heuristic.

Program Specifics: You will write a file called Project.java (notice the upper case) which has the calling convention:

java Project <input-file>

where <input-file> is the name of a file that contains the initial state of the 8-puzzle in row-major-order (first row followed by second row followed by third) with 0 representing the location of the blank space. For example, the goal configuration depicted above would represented as:**?** 1 2 3 8 0 4 7 6 5

Your program should output:

1. Whether or not a solution is found
2. The number of nodes expanded during the search
3. The length of the solution (number of moves)
4. The solution itself (in the format: UDDL... meaning "Up, Down, Down, Left, ...")

It is okay to allow your program to take extra arguments beyond the input file name to identify which of your heuristic functions to use, but the "Manhattan distances" heuristic should be the default which is used when your program is run with only one command line argument, java Project <test-input>.

Be careful about using *random* configurations to test your program. Not all 8-puzzle configurations are solvable. Your program should return "No solution" if there is indeed no solution to a given 8-puzzle configuration, but it will take quite a long time to search all of state space (several hours, for example). For debugging purposes, feel free to use the following configurations which are all solvable:

2 3 6 0 4 5 8 1 7

8 3 2 7 0 4 6 1 5

7 8 3 5 0 6 1 4 2

0 4 8 7 5 3 1 2 6

7 8 0 2 4 1 5 6 3

8 4 2 6 1 3 7 5 0

7 8 4 3 5 6 0 1 2

7 8 4 3 5 6 1 0 2

7 6 8 3 2 0 1 4 5

2 0 6 7 3 8 1 4 5

Although not required, feel free to write code for visualization, showing, for example, a movie of how the initial configuration was transformed move-by-move to the goal configuration