N-Puzzle

A simple program with GUI to solve an N-Puzzle of size 3x3 using heuristic assisted A* search.

Background :

An N-Puzzle is a childrens game involving a sliding puzzle with n X n boxes (3 x 3, 4 x 4, 5 x 5, etc…) and n-1 tiles. Typically the tiles are numbered 1 through N-1, and there is one empty space, the tiles are arranged randomly at start and the goal is to slide the tiles into a desired configuration. Below is a photo of some physical 15-Puzzles owned by the author. Besides being a childrens game the N-Puzzle has become a classical problem for modeling algorithms involving heuristics. Commonly used heuristics for this problem including counting the number of “wrongly” placed tiles (num wrong tiles), or summing the Manhattan distances between each block and its position in the goal configuration (Manhattan distance). **Note :** Both of these algorithms are admissible which is to say they never overestimate the number of moves left, which ensures optimality for certain search algorithms such as A*. A* is an informed search algorithm that can be used in situations like this to garuntee the optimal solution.

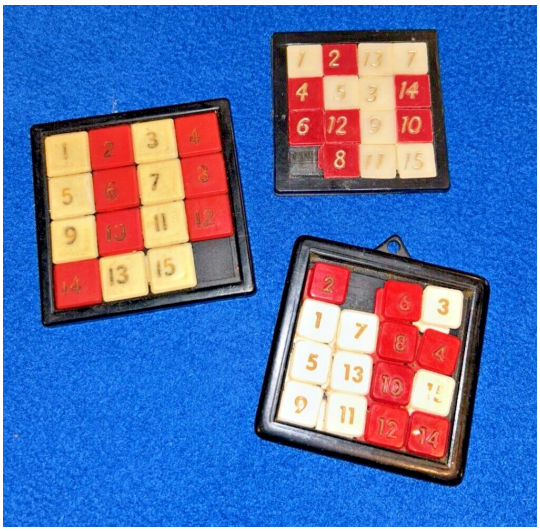

Figure I : Physical 15 puzzles owned by Jack.

For this project I created a Graphical User Interface (GUI) to simulate the physical board. The GUI will create a 9-Puzzle (3x3), however its worth nothing that the algorithms contained in “logic” can solve N-Puzzles of arbitrary size, the only drawback being space complexity. In my testing the program could easily solve N-Puzzles of size 3, 8 15, and 24. In order to effectively solve N-Puzzles of size larger then 5x5 (24 puzzle), the algorithm would need to be changed or updated to something like Simplified Memory Bounded A*. The desired goal state/solved state for the GUI is shown below.

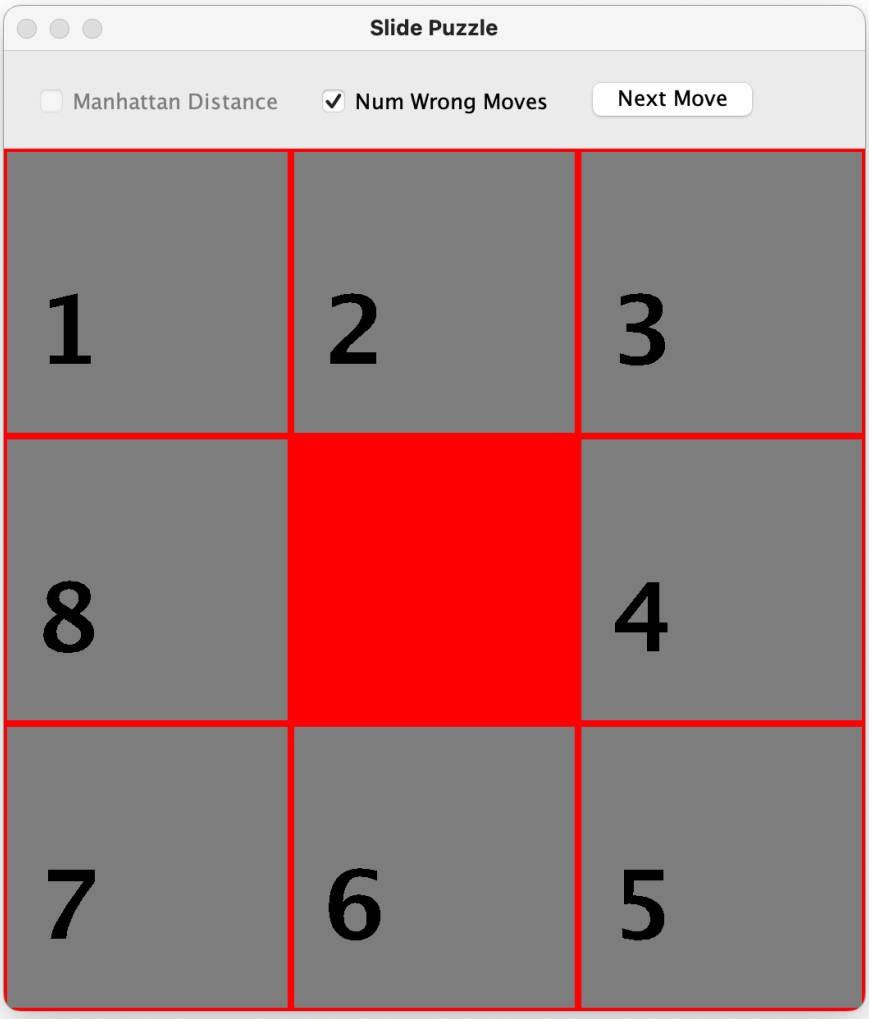

Figure II : The desired goal state for the 8 puzzle, with the 1 tile in the upper left corner, the tiles wrap around in numerical order with the empty tile in the center.

How to Use?

There are two folders for this project named UI, and Logic. You will need to obtain both of them on your local machine. Once you have both Logic and UI on your local machine (both in the same directory), navigate to that directory in your terminal. Now compile using the following command

javac ui/SlidePuzzleGUI.java

Now to run the program will require command line input repressenting the initial board state. For example the command

java ui.SlidePuzzleGUI 123457680

Corresponds to the following initial (unsolved state)

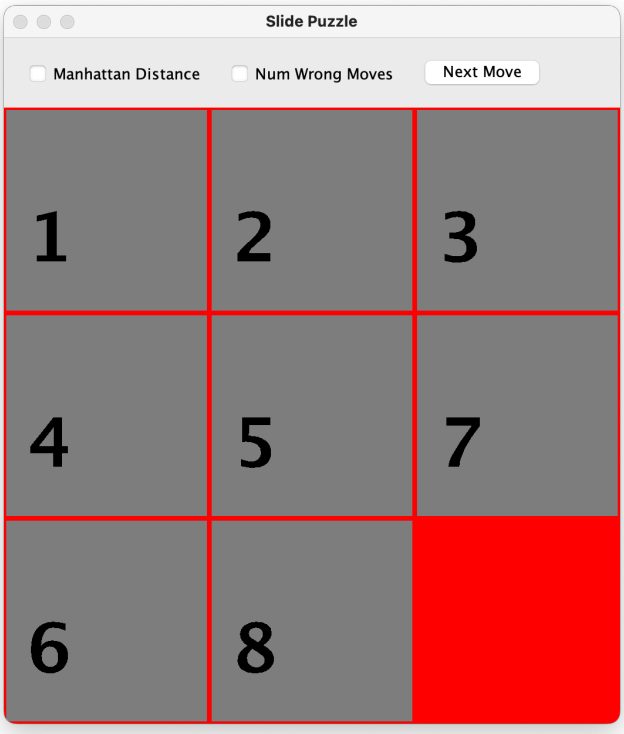

Figure III : The initial board state corresponding to the command line argument 123457680

Have fun experimenting with different other command line arguments/initial board states. At this point simply check the heuristic you wish to use, and continue clicking the next move button until the N-Puzzle is in the solved state shown in Figure II. Heuristic assisted A* will solve the puzzle configuration in figure III in just 12 moves.

```
Down, Right, Right, Up, Left, Left, Down, Right, Right, Up,
Left, Down
```

Future Changes :

I am close to automating the game play so that a user can just click a startGame button which will go through the steps automatically without the user having to click next move. I am also working on a pop up that would allow the user to choose the size of the N-Puzzle. Lastly I am currently working on designing and implementing an algorithm to efficiently solve N-Puzzles of arbitrarily large size, something analogous to Simplified Memory Bounded A*.