**Name:** Daniel Ramirez

**Course:** Intro. to A.I.

**Date:** 11/12/20

**Subject:** N-Puzzle Term Project Report

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Introduction:**

For this term course project, I choose to do the N-Puzzle problem which will use the A\* algorithm to find the solution of a given starting state. The main purpose for this project is to apply well known artificial intelligence algorithms such as A\* to a given problem to get to a solution efficiently. For this project, the system will use this algorithm to determine the minimum distance between the starting state and the goal states and use this calculated distances to slide the number panels around to reach the goal state. As stated at the beginning of this course, a system that can solve a problem such as a N-puzzle, can be concluded to have a form of system intelligence. The language I coded in for this project is in java. The reason is that java is great language to visualize these algorithm concepts via objects. Also, java has many different libraries that can make it easier for me to create and allocate objects more effectively, without running into errors.

**The Project Task:**

The main task for this project is to find the best way of moves from starting state to the goal state while using the A\* algorithm. The implementation of the A\* algorithm for N-Puzzle problem can be done in many different way but the general objective is to get define each parent and child states with the lowest costs from the starting state to the goal state. As describe in the project handout, there are a couple of requirements that are supposed to take place during the execution of the program. These are:

* At the beginning of the program the user inputs the N number of values for the puzzle.
* For each N, the output should include:
  + the number of nodes that have been traversed to the solution.
  + the state sequence from the starting state to the goal state.
  + the action sequence (each action transfer the current state to the next state).
  + the execution time.

In order to have the user not input the wrong number of N values I designed the beginning to only ask the user for the size or dimension of the board. So, if the user wanted to have a 3 x 3 board then the user will input 3 for the N value. For the time being, I only allow the user to input nothing greater than 5 because of the sheer magnitude of time that it will take the machine to process each state and the amount of memory that is needed to temporary store each state. The reason is that for simply a 4 x 4 board there will be 16 factorial different state which is 20.1 trillion states.

As for the rest of the objectives for this project, I display each state as it move to one state to another. After the everything is complete, I then display the minimum amount of moves it can take to solve the puzzle as well as the total amount of nodes states the machine took to get to the solution, and finally the total amount of time it took to solve the puzzle.

**The Project Algorithm:**

For this term project, the N-puzzle project will be using the A\* algorithm as the way to solve the puzzle. Here is a general summary of the A\* algorithm:

**A\* Algorithm:**

This algorithm is commonly used in pathing routes and traversing graphical panels. The reason it is a great algorithm to use for this puzzle is because it is an algorithm for finding the shortest path from one node to another efficiently. Other simple algorithms will try to fully process through every option to just find an ideal path. While the A\* algorithm finds best paths based on calculated values of costs from one to another point. This algorithm uses the following function to calculate each nodes value:

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

This is the cost function that finds the value for which the *f*(*n*)is total estimated cost of the path through node *n*, *g*(*n*) is the cost of the path from start node to node *n*, and *h*(*n*) is a heuristic function that estimates cost of the cheapest path from node *n* to the goal node. The calculation for the heuristic function *h(n)* can be done in many ways. The heuristic function I will be us using is the Manhattan distance formula, which will calculate the sum of the difference of each distance point within the current state and the goal states. This is also suitable for N-Puzzle because it will only give us the closest and correct path to the goal state.

The general layout of the A\* algorithm goes a follows:

* Step 1
  + Define a list OpenState.
  + Initially, OpenState consists solely of a single node, the start node S.
* Step 2
  + While the OpenState list is not empty, continue; else return failure and exit.
  + Remove node n with the smallest value of f(n) from OpenState and move it to the list ClosedState.
  + If node n is a goal state, return success and exit.
* Step 3
  + Expand all possible node n states that can be a successor.
* Step 4
  + If any successor to n state is the goal state, return success and the solution by tracing the path from goal state to S.
  + Otherwise, proceed to next step.
* Step 5
  + For each successor node:
    - Apply the evaluation function f to the node.
    - If the node n has not been in either OpenState and ClosedState list, add it to OpenState.
  + Go back to Step 2

**Project Implementation**

This snippet of code shows the Manhattan distance function that is written in Java. This function returns an integer which represents the Manhattan distance between current and the goal states. The function Next the function calculates distance between the current element i in both states. Each iteration of the for loop adds distance to the previous value which eventually returns Manhattan distance between the current and goal states.

