Programming Project 1
Solving the 8-puzzle using A* algorithm

Team Member: Shubhangi Alande (801104235)

Problem Formulation:

8-puzzle problem is played on a 3-by-3 grid with 8 tiles labeled 1 through 8 and a blank tile. Goal is to rearrange the tiles so that they are in given order. You are permitted to slide grid blocks horizontally or vertically into the blank square. The following shows a sequence of legal moves from an initial board position (left) to the goal position (right).

```
  1 3      1   3      1 2 3     1 2 3      1 2 3
4 2 5 => 4 2 5 => 4   5 => 4 5   => 4 5 6
7 8 6      7 8 6      7 8 6     7 8 6      7 8
```

The blank tile helps us to achieve the goal. The problem formulation for the puzzle is as given below:

1. **States**: The state description consists of the location of the 8 tiles on the grid as well as the blank tile.
2. **Initial State**: Any state provided by the user can be the initial state of the puzzle. But the algorithm does not guarantee to generate the goal state from all the possible initial states.
3. **Actions**: Based on the location of the blank tile, the possible movements for any given state are a subset of sliding the tiles: Right, Left, Up and Down. Thus, the actions are {RIGHT, LEFT , UP, DOWN}.
4. **Transition Model:** Applying any action possible to any tile in any state leads to a new state.
5. **Goal Test:** The Goal Test checks if the current state matches the goal state configured for that problem. Algorithm stops when Goal test returns true.
6. **Path Cost:** The step cost is 1, so the path cost is the number of steps to get to the goal configuration.

A* Algorithm to solve 8-puzzle Problem:

In A* for 8-puzzle problem, we first move the blank tile in all the possible directions in the initial state and calculate **f-score** for each newly generated state. The next state chosen from the **Fringe** is based on its **f score**, the state with the least f score is picked up and explored.

f-score = h-score + g-score

A* uses a combination of heuristic value (h-score: how far the goal node is) as well as the g-score (i.e. the number of nodes traversed from the start node to current node) to calculate f-score.
After expanding the current state, it is pushed into the **Explored** list. This process continues until the goal state occurs as the current state. Heuristics can be calculated in below two ways:

Programming Project 1
Solving the 8-puzzle using A* algorithm

Team Member: Shubhangi Alande (801104235)

1.Misplaced Tiles: This heuristic returns the number of tiles that are not in their required goal position. Let's take below puzzle instance,

INITIAL STATE

5		**8**
4	**2**	**1**
7	**3**	**6**

GOAL STATE

1	**2**	**3**
4	**5**	**6**
7	**8**	

We can see that tiles with number 5, 8, 2, 1, 3, 6 are placed in wrong place if we compare the initial state with the goal state. Hence, the heuristic value h(n) = 1 + 1 + 1 + 1 + 1 + 1 = 6 for the initial state. By comparing current state and goal state h(n) is calculated for every successor and the state with lowest heuristic value is selected for expansion. This procedure is repeated till we get goal state.

2. Manhattan Distance: Manhattan Distance of a tile is the distance or the number of slides/tiles away it is from its goal state. Thus, for a certain state the Manhattan distance will be the sum of the Manhattan distances of all the tiles except the blank tile. For example,

INITIAL STATE

5		**8**
4	**2**	**1**
7	**3**	**6**

GOAL STATE

1	**2**	**3**
4	**5**	**6**
7	**8**	

For the state above,
h(N) = sum of the (Manhattan) distance of every numbered tile to its goal position
= 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1
= 13

For some special cases, however, A* algorithm cannot find a solution to 8-puzzle problem. The algorithm then needs to terminate if it is unable to reach the goal state at all after a specified number of iterations.

Programming Project 1
Solving the 8-puzzle using A* algorithm

Team Member: Shubhangi Alande (801104235)

Program Structure:

I have implemented the A* search Algorithm to solve the 8-puzzle problem using **C#** Programming Language.

- The program accepts the initial state and the goal state from the user and those are given as inputs to A* algorithm in the form of a 2-dimensional array.
- The algorithm proceeds as follows:
  1. Checking goal test. If not equal to goal state then follow,
  2. Calculating index for blank tile,
  3. Generating successors for the current state,
  4. Calculating heuristics for each state,
  5. Adding the state to the priority queue,
  6. Selecting the sate with lowest heuristics for expansion,
  7. Checking if the selected state is already exists in the list of explored states. If not present then,
  8. Adding selected state to the explore list and expanding it,
  9. Repeat above steps till current state is not equal to goal state.

- The output is the solution path from the initial state to the goal state, the number of nodes generated, and the number of nodes expanded.

Global Variables:

1. _initState = 2-dimensional array to store initial state accepted from the user.
2. _goalState = 2-dimensional array to store goal state accepted from the user.

Functions/Procedures:

1. GoalTest (): checks if the current state is equal to the goal state.
   Input Parameters: Current State, Goal State
   Returns: True if Current State = Goal State, false otherwise.

2. FindIndexOfZero (): Calculates the blank tile position in the current state.
   Input Parameters: Current State
   Returns: Row Index and Column Index of 0's location in the matrix (considering 0 as Blank Tile)

3. GenerateChildrenStates(): Generates the successors of the current state by swapping the blank tile(number 0) with all of its possible adjacent tiles. The movement can be right, left, up, down.

Input Parameters: Current state, Goal State, Row Index and Column Index of 0 in that state.
Returns: Generated successors.

4. MoveToTheRight (): Swaps the tile with number 0 to its right adjacent tile.
   Input Parameters: Current state, Goal State, Row Index and Column Index of 0 in that state.
   Returns: New State

5. MoveToTheLeft (): Swaps the tile with number 0 to its Left adjacent tile.
   Input Parameters: Current state, Goal State, Row Index and Column Index of 0 in that state.
   Returns: New State

6. MoveToUp (): Swaps the tile with number 0 to its above adjacent tile.
   Input Parameters: Current state, Goal State, Row Index and Column Index of 0 in that state.
   Returns: New State

7. MoveToDown (): Swaps the tile with number 0 to its below adjacent tile.
   Input Parameters: Current state, Goal State, Row Index and Column Index of 0 in that state.
   Returns: New State

8. ManhatanDistance(): Calculates the Manhattan distance between the current state tiles and the goal state tiles.
   Input Parameters: Current State and Goal State.
   Returns: The Manhattan distance for current state.

9. CalculateMisplacedTiles():Calculates number of misplaced tiles by comparing current state and goal state.
   Input Parameters: Current State and Goal State.
   Returns: Number of misplaced tiles.

10. ExploredTest():Checks if state with lowest heuristic is already expanded or not.
    Input Parameters: State with lowest heuristic from Fringe, List of Explored States.
    Returns: True if state is explored otherwise False.

Programming Project 1
Solving the 8-puzzle using A* algorithm

Team Member: Shubhangi Alande (801104235)

Sample Input and Number of nodes generated and expanded:

INITIAL STATE	**GOAL STATE**	**Misplaced Tile # of nodes generated**	**Manhattan Distance # of nodes generated**	**Misplaced Tile # of nodes expanded**	**Manhattan Distance # of nodes expanded**
1 2 3 4 8 0 7 6 5	1 2 3 4 5 6 7 8 0	22	8	27	6
2 8 1 3 4 6 7 5 0	3 2 1 8 0 4 7 5 6	29	17	11	7
1 2 3 5 6 0 7 8 4	1 2 3 5 8 6 0 7 4	10	10	4	4
0 1 3 4 2 5 7 8 6	1 2 3 4 5 6 7 8 0	12	12	5	5
5 0 8 4 2 1 7 3 6	1 2 3 4 5 6 7 8 0	786	144	296	44

Conclusion: Thus, observing the above table we can conclude that theoretically Manhattan distance is a better heuristic than Misplaced Tiles.