**A\* Implementation of the 8 puzzle problem**  
Charles Arvey (9008151720)  
Michael Palmer (900757121)

**Implementation Architecture:**

Our implementation uses two primary classes: Puzzle and PuzzleState. A Puzzle instance consists of one start state and goal state, both of which are instances of PuzzleState. More PuzzleState instances will be generated by Puzzle as the A\* algorithm generates new state permutations.

When the implementation is run, a Puzzle instance is created. This instance will initialize the puzzle's start and goal states based on the input file. A\* search is used to find the best path from the start state to the goal state. On initialization, the Puzzle instance also performs a check to determine if the puzzle is solvable in a finite amount of time.

Dictionaries are used in our PuzzleState class to track states and map the values of the puzzle board to their respective positions.

For each possible action (move), a PuzzleState instance is created to represent the current state of the puzzle. In order to actually move states, a new PuzzleState instance is generated with the switched (new) positions in the underlying dictionary. Valid movement positions are mapped out based on the value of the key in a PuzzleState’s dictionary. For instance, if a puzzle’s empty space is at position 0 (top left), it’s valid movements positions would be 1 (top middle) and 3 (left middle).

**Heuristics:**  
We used a combination of the Manhattan Distance and Linear Conflict for our heuristic.

Manhattan Distance calculates the distance between two nodes using only vertical and horizontal moves, as opposed to euclidian distance.

We used an aggregate manhattan distance to calculate the cumulative costs for an entire puzzle state. This gives us an estimate of whether or not we are moving closer to the goal state or not.

In addition to the Manhattan Distance, we also calculate the linear conflict and add that value to the total heuristic. A tile t is considered in linear conflict with another if:

* Tiles ta and tb are in the same line
* The goal positions of ta and tb are both in that line
* Tile ta is to the right of tb
* The goal position of ta is to the left of the goal position of tb

**Requirements:**  
Python 2.7  
  
**Input:**  
The input file will contain the initial and goal board configuration.  
A sample can look like this:  
  
 7 2 4  
 5 0 6  
 8 3 1  
  
 0 1 2  
 3 4 5  
 6 7 8  
  
The first block of numbers is the start state and the second block of numbers is the goal state.  
  
**Output:**  
If the goal configuration is reachable in a finite number of steps, all board configurations from initial to goal and the minimum number of steps required to reach the goal are printed out. If the goal configuration is not reachable in a finite number of steps, "This Puzzle is not solvable" is displayed.  
  
**Compilation and Usage:**  
python a\_star.py --file [path to input file]

EX:

python a\_star.py --file sample-problems/test3\_0

**Work Distribution:**

Michael worked on the linear conflict and priority queue implementations and implementing code optimizations.

Charles worked on the manhattan distance, inversion logic, file parsing, and movement logic.

All other aspects of the code were worked on collectively by both Charles and Michael.