The file works by first reading the file and creating a board.
I'm using a simple 2d array as it's what we used in the first assignment, a priority queue to sort the states of the board by their cost and a HashSet so I don't have duplicates of visited board states.

After creating the board using a heuristic the A* algorithm will check the tiles surrounding the empty tile and make moves accordingly depending on the board state.

It will also add these moves to a simple linked list which will later become the solution file.

Most of the program was quite simple swapping tiles, creating a board, a state class and a list of moves. Creating the A* algorithm took some time but after it started working most of what to optimize was which heuristic to use.
This was easily the hardest part, the first heuristic I tried was counting how many tiles were out of place. The problem with this heuristic was it took up too much memory and would give me an error for small boards.

I then tried to use the Manhattan heuristic which would solve boards up to 4x4s anything after that size would return a heap space error. I could have played around more with which data structures I used but I didn't want to spend all my final prep time on this so I thought this would suffice. Something I played around with but couldn't figure out was adjusting the size of my hashtable as I thought that might free up some heap space potentially however I couldn't figure it out in time and left it as the default.

I didn't have a lot of time to work on this project but if I had the time I wanted I had an idea of a pattern database that the algorithm would check against.

For a fixed-sized board, this would be very simple but for a dynamic-sized board, it got a little more complicated.

Say the board was of n x n size the algorithm would go in ascending order starting from 1 and place tiles in order. If the tile was in the corner (or how it would be represented in code is c(n-1) or cn where c is the row number) it would then put the two tiles into preparation spaces then execute a known set of moves which would put the tiles in place.

This would work until the last two rows then the algorithm would check for the (n-1)n +i th and the nn+i th element then repeat for all i=0-(n-2)

Then the last 3 tiles will cycle until all three are in place.

In my head this should produce a solution that isn't the best but not really bad.
Very simple solutions might suffer because of the "hard" coded aspect of this pattern database placing each tile in order, but the pattern database has an "upper limit" on the solution size based on the size of the board.

The pattern database especially seemed pretty cool to me because I know how to solve a Rubik's cube and it's identical to how I think when solving one.