Exercise #1-Part 2:
Solving N-Puzzle Problem Using A*

Attached are the following files:

1. frontier.py – Implementing frontier with a priority queue. Will be used to prioritize states based on a value function, state.hdistance(s)+state.path_len(s). You will need to write state.hdistance(s) ☺
2. state.py – Implements an N-Puzzle, except for the heuristics. Your assignment!
3. search.py – Implements the search pseudocode from class.

You need to go through the attached files, understand them, and perform the following tasks:

1. Implement the hdistance1 function to accurately return the number of tiles out of place and change the code in frontier to return state.hdistance1(s)+state.path_len(s). Run the code 100 times and save the output of the number of states checked – (i) total items pushed, (ii) total items popped and (iii) path cost of the solution – as averaged from 100 runs in a 4X4 puzzle (previously not always possible).
2. Implement the hdistance2 function to accurately return the Manhattan distance of the tiles from their target position and change the code in frontier to return state.hdistance2(s)+state.path_len(s). Run the code 100 times and save the output of the number of states checked – (i) total items pushed, (ii) total items popped and (iii) path cost of the solution – as averaged from 100 runs in a 4X4 puzzle.
3. Re-run Steps 1 and 2, as a *weighted A* search*. That is, with priority function 2*state.hdistance1(s)+state.path_len(s) and 2*state.hdistance2(s)+state.path_len(s), respectively. How does this affect the runtime (number of states checked) and optimality (actual cost of found solution) of the code?

* Reminder from the lecture:

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Figure 3.29 Comparison of the search costs and effective branching factors for the ITERATIVE-DEEPENING-SEARCH and A* algorithms with h_1, h_2. Data are averaged over 100 instances of the 8-puzzle for each of various solution lengths d.