Assignment 1: Experimenting with the 8-puzzle

In this assignment you get a chance to play with some heuristic search algorithms.

In the textbook code from Github file `search.py`, take a look at the class called `EightPuzzle`. Take some time read and understand it, including the `Problem` class that it inherits from.

Put the coding part of you answers to the following questions in a Python 3 file named `a1.py` that starts like this:

```
# a1.py

from search import *

# ...
```

Do not use any modules or code except from the standard Python 3 library, or from the textbook code from Github.

Don't forget: If you get any kind of help from elsewhere, you must cite that fact in your assignment (a comment in the source code is usually fine). Accidentally forgetting to do this is no excuse!

Question 1: Helper Functions

Write a function called `make_rand_8puzzle()` that returns a new instance of an `EightPuzzle` problem with a random initial state that is solvable. Note that `EightPuzzle` has a method called `check_solvability` that you should use to help ensure your initial state is solvable.

Write a function called `display(state)` that takes an 8-puzzle state (i.e. a tuple that is a permutation of (0, 1, 2, ..., 8)) as input and prints a neat and readable representation of it. 0 is the blank, and should be printed as a `*` character.

For example, if `state` is `(0, 3, 2, 1, 8, 7, 4, 6, 5)`, then `display(state)` should print:

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

Question 2: Comparing Algorithms

Create 10 (more would be better!) random 8-puzzle instances (using your code from above), and solve each of them using the algorithms below. Each algorithm should be run on the exact same set of problems to make the comparison fair.

For each solved problem, record:

- the total running time in seconds
- the length (i.e. number of tiles moved) of the solution
- that total number of nodes that were *removed* from `frontier`

You will probably need to make some modifications to the A* code to get all this data.

Also, be aware that the time it takes to solve random 8-puzzle instances can vary from less than a second to hundreds of seconds — so solving all these problems might take some time!

The algorithms you should test are:

- A*-search using the misplaced tile heuristic (this is the default heuristic in the `EightPuzzle` class)
- A*-search using the Manhattan distance heuristic Please implement your own (correctly working!) version of the Manhattan heuristic.
    - Be careful: there is an **incorrect** Manhattan distance function in `tests/test_search.py`. So **don't** use that!
- A*-search using the max of the misplaced tile heuristic and the Manhattan distance heuristic

Summarize all your data in a single table in a spreadsheet as described below.

Based on your data, which algorithm is the best? Explain how you came to your conclusion.

Question 3: The House-Puzzle

(Duck-puzzle) Implement a new `Problem` class called `DuckPuzzle` that is the same as the 8-puzzle, except the board has this shape (that looks a bit like a duck facing to the left):

```
+--+--+
|  |  |
+--+--+--+--+
|  |  |  |  |
+--+--+--+--+
   |  |  |  |
   +--+--+--+

 1 2
 3 4 5 6   goal state
   7 8 *
```

Tiles slide into the blank (the `*`) as in the regular 8-puzzle, but now the board has a different shape which changes the possible moves.

As in the previous question, test this problem using the same approach, and the same algorithms, as in the previous problem.

Be careful generating random instances: the `check_solvability` function from the `EightPuzzle` probably doesn't work with this board!

Based on your results, how does the Duck-puzzle compare to the 8-puzzle: is it easier, harder, or about the same difficulty?

Presenting Your Results

For questions that ask for more than code, please put all your tables, data, and discussion into a standard Excel worksheet file named `a1.xlsx`. You can use Excel or Google Sheets to create it.

Make the spreadsheet beautiful, informative, and easy to read. Be sure to include helpful descriptive statistics like the min, max, average, and median values.

You are encouraged to include helpful or informative graphs of your data.

Spelling, grammar, and neatness count!

What to Submit

Please put all your code into a single Python 3 file named `a1.py`, and all your data, tables, charts, discussion, etc. in an Excel file named `a1.xlsx`. Compress these into a single archive named `a1.zip`, and submit it on Canvas before the due date listed there.

If you want to make changes to code in `search.py`, or in files other than `a1.py`, please copy the code you want to change into `a1.py`, and change it there. Don't make any changes to other files in the textbook code.

Hints

- When you are testing your code, you might want to use a few hard-coded problems that don't take too long to solve. Otherwise, you may spend a lot of time waiting for tests to finish!
- One easy way to time code is to use Python's standard `time.time()` function, e.g.:

  ```
  import time

  def f():
      start_time = time.time()

      # ... do something ...

      elapsed_time = time.time() - start_time

      print(f'elapsed time (in seconds): {elapsed_time}s')
  ```

  Python has other ways to do timing, e.g. check out the `timeit` module.

- If you download the textbook code as a Git repository, then you can create a branch called `a1` for this assignment, e.g. something like:

  ```
  git branch a1 git checkout a1
  ```

  This way you can make any changes you like while maintaining a copy of the original code.

  You are **not** required to use Git, but since the code is stored as a Git repository, it's probably the best way to get it.