CS 461 – ARTIFICIAL INTELLIGENCE

HOMEWORK #3 (5%)

Assigned: Thu Nov 8, 2018

Due: Mon Nov 19, 2018, **2:00 pm**

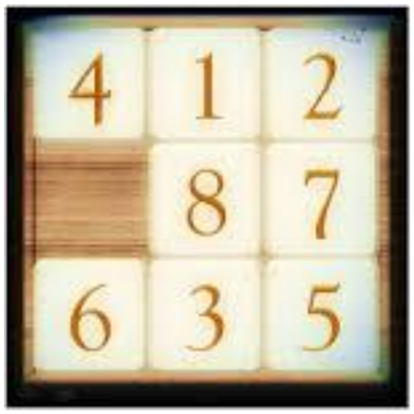
Do not forget to indicate the students submitting the homework (at most 5 names). Send your homework to our TA. (He'll most probably give you instructions.)

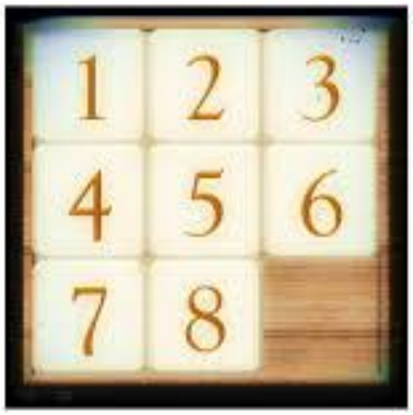
Feel free to use any programming language as long as any of you can do (if requested) a demo using your notebook computer.

The usual late submission policy applies.

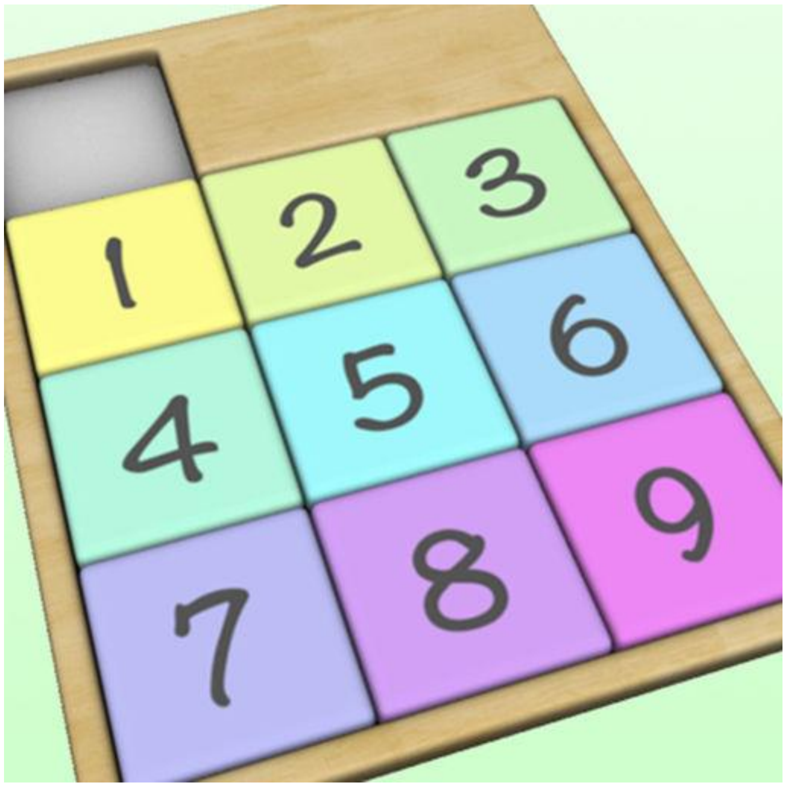
PROBLEM

The 8-puzzle is a game in which there are eight 1x1 tiles arranged on a 3x3 square so that there is one 1x1 uncovered (empty) area on the square. The tiles are labeled 1 through 8, and initially they are shuffled (cf. image below, left). The idea is to reach the goal state (cf. image below, right) from a given initial state by moving tiles one at a time.





However, and this is very important, you'll be solving not 8-puzzle but a variant of 8-puzzle in this homework. For lack of a better terminology, let me call this variant 9-puzzle. Here's a snapshot of 9-puzzle in its goal state:



You've already implemented BBS in Homework #2. **Now implement the A* search algorithm (no other algorithm is acceptable) and use it to solve a given instance of the 9-puzzle optimally, that is, with a minimum number of (sliding) moves.** (Winston covers A* in chapter 5. It is up to you to implement the dynamic programming add-on. I suggest that you do implement it so that you learn that idea well, but no points will be deducted for lack of it.)

As you know, A* requires admissible heuristics. You are free to use one of the following functions (but clearly state which one at the very beginning of your program):

- h1: Number of misplaced tiles (h1 is an admissible heuristic, since it is clear that every tile that is out of position must be moved at least once)
- h2: Sum of Euclidian distances of the tiles from their goal positions (h2 is an admissible heuristic, since in every move, one tile can only move closer to its goal by one step and

the Euclidian distance is never greater than the number of steps required to move a tile to its goal position)

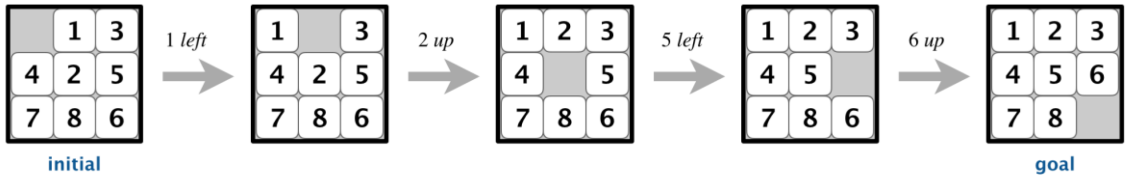
Here's what you must do:

- Write a 'puzzle generator' first. Starting from the goal state of 9-puzzle (cf. preceding image), the generator returns a reasonably garbled initial state (S) by randomly shuffling the puzzle. Notice that your generator thus guarantees that this initial state S will be solvable.
- Run your generator as many times as necessary to obtain 10 distinct initial states of 9-puzzle.
- Solve each of these 10 instances via A*. (Also solve each of them via BBS, as you have done in Homework #2. You'll see the reason presently.)

A submission consists of

- Listing of your program with a clear indication of what parts, if any, of it are borrowed from elsewhere. (It is best if you write your own original code because points will definitely be deducted for code heavily borrowed from another source.)
- For each of your 10 states, a graphical solution ("a sequence of drawings starting with the initial state and ending with the goal state") fully tracing the individual moves of your program to reach the goal. Under the graphical solution write the number of moves (call it N) that BBS made in order to reach a shortest-path solution. Obviously, (and if your BBS and A* implementations are working correctly), A* solutions should be making N moves too.

It is of utmost importance that a solution sequence is shown graphically, not symbolically. The drawings need not be sophisticated or in color, etc. Just to give you an example of what I've in mind, consider the following solution sequence for 8-puzzle (your drawings could be even simpler):



Thus, I expect 10 drawings like this from you but, needless to say, for 9-puzzle! (The arrows and their annotations are not compulsory but surely welcome.)

Finally, if you would like to explain your program (you should!), the best place to do so is inside your program (comments!). Do not attach nonexecutable appendices to your code listing.