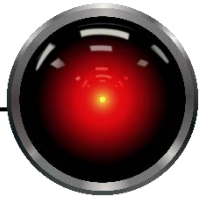


Assignment 01

1. (20 points) Agents
    a. A user wants to construct an agent to monitor production of soup. In particular, they are interested in making sure that the bacertia *Esecherichia coli* O157:H7 does not . Check the Foodsafety.gov web site for how to prevent *E. coli* and propose a percept for this agent.
    b. Propose a percept-action table for this agent.
2. (20 points) Work problem 3.6b from your book. Be sure to specify, the initial state, a goal test, successor and cost functions.

Getting your feet wet with Python (80 points)

Create class TileBoard in file boardtypes.py for representing n-puzzles. It should be derived from class Board (provided) and should support the following interface[1]:

- TileBoard(n, force_state=None) – Creates an n-puzzle of size n. Example TileBoard(8) creates the 8-puzzle shown below. (Note that we are using your book's notation which refers to the number of tiles as opposed to the number of squares as we did in class.)

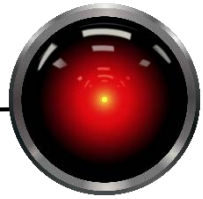
  Your puzzle must be solvable. You can check if a puzzle is solvable by checking the inversion order. Inversion order is defined by as the sum of the permutation inversions for each tile position. Using the example in the 8-puzzle below, the tiles from left to right and top to bottom are [8,None,5,4,7,2,3,1]. The permutation inversion of the 3rd position (tile 5) is the number of tiles that are <5 following 5: [4, 2, 3, 1] or 4 tiles. The inversion number is the sum of all permutation inversions for each tile, e.g. inversions of first, second, third, … positions. If the board has an even number of rows, the row number of the blank must be added to this. Puzzles with even inversion numbers are solvable, odd ones are not (proof of this is unrelated to the problem at hand and beyond the scope of our class). Reshuffle the tiles if your puzzle is not solvable.

  If the optional force_state is specified, it should contain a list (e.g. [8,None,5,4,7,2,3,1]) that will be used to populate the board instead of producing a random board. This will help you test your solved() and == methods without having to solve the puzzle.

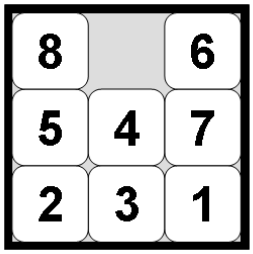
- Operator == to check if two tile boards are in the same state. Use method __eq__(other_obj) to overload the equality operator.

- state_tuple() - Flatten the list of list representation of the board and cast it to a tuple. e.g. [[1,2,3],[4,None,5],[6,7,8]] becomes (1,2,3,4,None,5,6,7,8)

[1] We describe the calling interface, the method signatures of objects require the argument self to be first.



- get_actions() – Return list of possible moves. It is easier to think of the blank space as being moved rather than specifying which numbered tiles can be moved. We will return this as a list of lists where each sublist is an [row_delta,col_delta] list specifying the offset of the space from its current position relative to the current row and column. Values of the deltas can be -1, 0, or 1 indicating that the space should be moved: left, no move, or right and up, no move, or down



  Concrete examples for the puzzle at right where space can be moved to the left, right, or down: [[0,-1], [0,1], [1,0]].

- move(offset) – Given a valid action of the rom [row_delta, col_delta], return a *new* TileBoard that represents the state after the move.
  Example: [0,-1] would move the space one column to the left, exchanging the blank and 8.

  Hint: To maintain multiple versions of the board in a later assignment, it is important that you not just modify the lists. As Python objects contain pointers to the list, changing the lists in the current object would also change the lists in any other object that was created from this one. There are many ways that you could prevent this, but one of the easiest ways to clone an object is to use the copy module's deepcopy method:

```
newboard = copy.deepcopy(self)  # Create a deep copy of the object
# make modifications to newboard
```

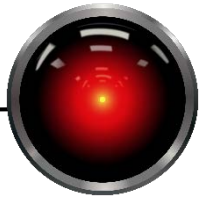
- solved() – Return True if the puzzle is in a goal state (the blank must be in the center of an odd sized puzzle. You may define it as you wish for even sized puzzles).

Create function driver in driver.py that is invoked when driver.py is called from the command line: python driver.py. It should print a board and allow you to play an 8 puzzle. If you want, you can have it ask you for the size of the board instead of hardcoding it.

Some general hints:

- Take advantage of inheritance. Parent constructors are not called automatically, but can be accessed with *super*.
  Example: super(TileBoard, self).__init__(*appropriate args*)
- The random module has a method shuffle that can be used to permute a list. This is useful for randomizing the initial state.
- If you need math functions (e.g. sqrt), they are in Python's math module

A zip archive with a module Python module called basicsearch_lib can be found on the Blackboard assignment page. It contains an expanded version of the Board class that we discussed in class. Expand the library such that directory basicsearch_lib is in the same



directory as your boardtypes.py file.

Sample import assuming the directory is in the same one you placed your solution in:
```
from basicsearch_lib.board import Board
```

What to Turn In

You must turn in:

- Paper copy of your work including the program, output and the questions. Pair programmers should turn in a single package containing separately answered questions and a single copy of the program.
- All students must fill out and sign either the single or pair affidavit and attach it to your work. A grade of zero will be assigned if the affidavit is not turned in.
- An electronic submission of your program shall be made in addition to the paper copy. A program comparison algorithm will be used to detect cases of program plagiarism.

Remember that all assignments are due at the beginning of class (i.e. if you skip the first ten minutes of class to work on your assignment it will be considered late), and the policy on late assignments is described in the syllabus.