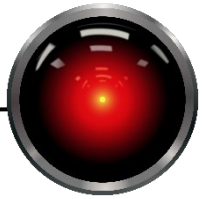


Assignment 01

1. (20 points) Agent environments. An agent has been designed to help air traffic controllers direct traffic. Answer the following question about task environment state:
    a. Briefly describe the state of the problem.
    b. Is state static, dynamic, or semidynamic? Justify your answer.

2. (20 points) A Pacman ghost agent uses the following strategy. If the Pacman has eaten a power pill in the last 30 seconds, the agent will move in a direction opposite of the Pacman's current position, subject to constraints of available moves. Otherwise, ghost moves in a valid direction that will move the ghost closer to the Pacman. Based on these simple rules, would you consider the agent type to be simple reflex, model-based reflex, utility-based, or learning? Briefly justify your answer.

3. (20 points) A plan for designing a smart city intersection calls for designing an agent that is responsive to pedestrians wishing to cross the street. Rather than pushing a button to activate a mid-block cross walk without traffic lights, a state-based agent activates the cross walk when pedestrians are waiting near the cross walk. Assume two actions: Activate crosswalk, deactivate crosswalk

    a. Assume a clock tick percept is noted every 1 s along with any other measured percepts. What other sensors and percepts should be used to provide service?
    b. Provide a state transition table that ensures that pedestrians have anywhere from a minimum of 30 s to a maximum of 90 s to cross the street. Also ensure that traffic is allowed to flow for a minimum of 120 s when traffic is present and the crosswalk signal is not engaged.

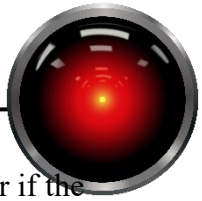
Getting your feet wet with Python (60 points)

Complete 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. You can force a specific configuration using the force_state argument and a list or tuple, e.g. (8,None,5,4,7,2,3,1) for the puzzle on the next page. This can be helpful for testing by letting you create easily solvable puzzles, e.g. (1,2,None,4,5,3,7,8,6) is solvable in two moves. If force_state is not specified, create a random board.

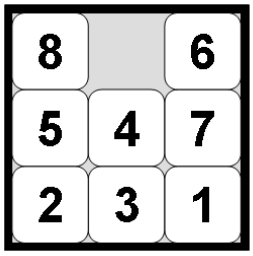
  Your puzzle must be solvable. Method TileBoard.solvable takes a list or tuple of tiles (like force_state) and checks whether or not the puzzle can be solved. For random puzzles, reshuffle until you get a solveable puzzle. When the puzzle is

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



forced to a specific configuration using force_state, throw a ValueError if the puzzle is not solvable.

- 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 provided by the parent class 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)
- 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 a [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: action [0,-1] would move the space zero rows and one column to the left, exchanging the blank and 8.

  Hint: To maintain multiple versions of the board in a later assignment involving search, 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:

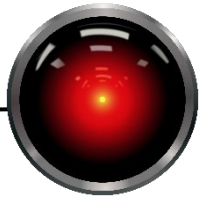
```
newboard = copy.deepcopy(self)  # Create a deep copy of the object
# make modifications to newboard
```

- 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).

The portions of the program that you need to modify are either marked as "todo:" or raise a NotImplementedError. A driver function (driver.py) is provided . It should print a board and allow you to play an 8 puzzle.

Some general hints:

- 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 skeleton code and modules driver and basic_search.board are provided for you. They are available on Blackboard. You should not modify any files other than tielboard.py.

What to Turn In

You must turn in:

- Paper copy of your work including the tileboard.py module, moves from a sample game (you can use force_state to make an easy game) 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. Links to the affidavits can be found on the assignments page of the course web site.
- An electronic submission of file tileboard.py 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.