CMSC 671 (Introduction to AI) – Fall 18

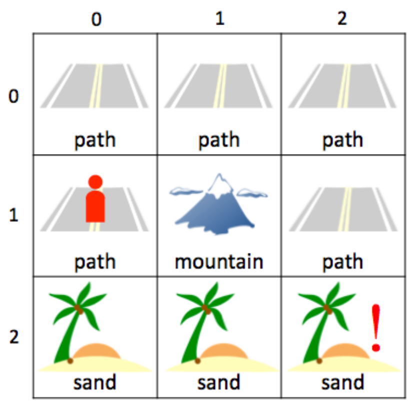
Homework 2: Search (75 points)
Turnin: Blackboard.
Please submit Part I as a **single PDF file** named *yourlastname*_hw2.pdf.
Please submit Part II as a **.py** file, named *yourlastname*_hw2.py.

All files must start with your last name and have your full name in the file, at/near the top.

PART I. NAVIGATING IN TERRAIN (SEARCH SPACES AND STATES) (35 PTS)

The General Idea

Consider navigating a 3 x 3 space, shown right. There are three kinds of terrain, each of which takes some amount of effort to traverse: entering a "path"[1] cell costs 10 calories, entering a "sand" cell costs 30 calories, and entering a "mountain" cell costs 100 calories. Your agent starts at coordinates **(1,0)**, as shown, and is trying to reach square (2,2) (marked !). The agent cannot move diagonally.



Assignment: Answer the following questions about the representation of this puzzle.

1. How would you represent this as a search problem?
   Your answer should be complete and relatively formal. (See the water jug and Sudoku examples from lecture.) *10 pts*
   (a) Describe the **state space**.

   (b) Provide a table of **actions/operators**, including **constraints**. Spell these out—don't provide "classes" of actions or constraints.

   (c) What is your **goal test**?

2. How many **unique, legal, reachable** states are there in this search space? *3 pts*

3. Draw the first three levels of the search tree. *5 pts*

4. If you were using heuristic search, what and why? *5 pts*

   (a) What **admissible** heuristic would you choose for evaluating states?

   (b) Explain how you know it is admissible.

5. **Choose algorithms** for this navigation problem in general (that is, with any possible arrangement of sand, paths, and mountains). In full sentences, justify your answers in terms of space and time complexity, completeness, and optimality. *6 pts*
   (a) **Uninformed** search:

   (b) **Informed** search:

   (c) **Local** search:

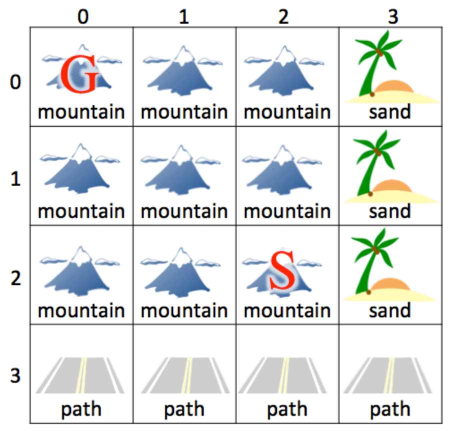
6. If the navigation square could be any size (1044 x 1044, for example), what algorithm (from any of the above) would you choose, and why? *6 pts*

[1] pixabay.com/en/road-crossing-crosswalk-street-304283, pixabay.com/en/sand-beach-island-palm-sun-tree-304525, pixabay.com/en/mountain-peak-snow-summit-304054

PART II. PATH-FINDING (40 PTS.)

The General Idea

Write a program to read in and solve puzzles of the type described above. The puzzle may be of any size, will always be square, and will have a random selection of path, sand, and mountain cells. The goal of the puzzle is to move your agent from a starting cell to a goal cell. S and G may be any cell.



The agent's task is to find either the lowest-cost path from S to G, **or** any path with a cost that is less than or equal to 300 calories. In the puzzle given in Part I, any non-looping path would be a correct solution; in the example given to the right, there is only one correct solution.

Some puzzles of this form may have multiple optimal solutions.

Details

You may assume:

- The size of the square is nonzero.
- The agent may only move one square at a time (not diagonally).
- The array is always 0-indexed.
- **Tuples are always in (x,y) order. (horizontal, vertical/row, column)**

You may *not* assume:

- The agent is unable to backtrack. (The search may and probably will, though!)
- Start or goal states will always be along an edge.
- The start state will be different from the goal state.

Assignment: Write a function called **solve** which takes a matrix (the problem) and two tuples (start and goal), and tries to find a path from the start state to the goal state that is either ≤ 300 calories, or—if it is more expensive than 300 calories—is optimal (lowest possible cost). Our two examples would be passed in as follows:

```
>>> solve((0,1), (2,2), [[p,p,p], [p,m,p], [s,s,s]])
>>> solve((0,0), (2,2), [[m,m,m,s], [m,m,m,s], [m,m,m,s], [p,p,p,p]])
```

Your solver should take these values, and **return** (not print) a string containing a list of moves. Moves should be represented by the capital letters N,S,E,W for north, south, east, and west moves. The optimal solution to the 3 x 3 board would be returned as "NEESS". Please don't print anything inside the function; everything must be in the return value.

Implement your solution to this problem as: A* search. Use the heuristic you gave in Part I. You **may** use any of the optimizations we covered in class (e.g., keeping track of previously expanded nodes).

Hints:

I recommend writing two kinds of tests. (1) Unit tests, which have known answers and can be rerun as you're writing to make sure your code is performing as expected. (2), A function that can generate random problems of various sizes and testing your code against those, ranging from 2x2 to at least 100x100, which will let you make sure your code doesn't crash on various edge cases and takes a reasonable amount of time.