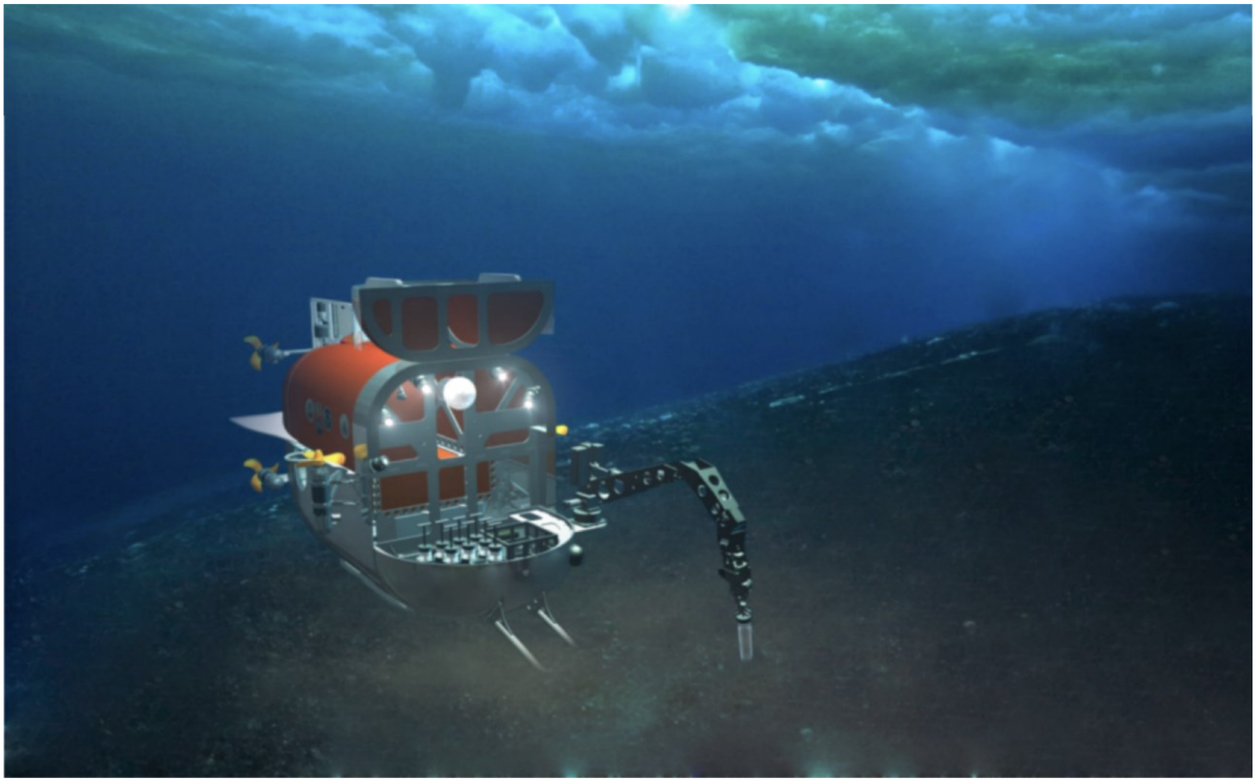
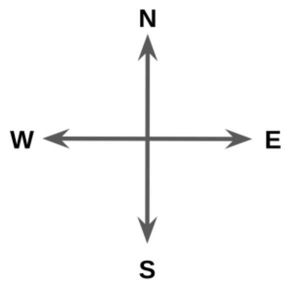
Creating exploration policies using iterative value iteration



https://www.whoi.edu/multimedia/nui-manipulator-integration/

Problem Description:

You are an expert on "underwater robotics" and you are conducting a research project funded by the government, to explore an underwater archaeological site discovered in the Atlantic Ocean. Based on the collected evidence, you believe it is likely the ruins of *Atlantis* -- a mythical island mentioned by *Plato* in his works nearly 2,400 years ago. To collect more evidence, you plan to develop an underwater robotic system that can navigate in the deep ocean.



The robot can move North, South, East, or West (see the figure to the right). However, there are uncertainty in the robot's navigation due to water movement in the ocean. As a result, it moves in the intended direction only 70% of the time, and in each of the other three directions 10% of the time. When it tries to move into off the site, it will end up staying in the current position (e.g., at [0,0], if the robot intends to move North, it has 80% chance of staying in [0,0], 10% chance of going to [1,0] and 10% chance of going to [0,1]).

The archaeological site can be represented by the grid as below:

0,0	1,0	2,0	3,0	4,0
0,1	1,1	2,1	3,1	4,1
0,2	1,2	2,2	3,2	4,2
0,3	1,3	2,3	3,3	4,3
0,4	1,4	2,4	3,4	4,4

In this grid, we use **(col, row) coordinates**. There will be obstacles in the site, such as ruined ancient temples and greek pillars, that the robot must avoid. If your robot crashes into an obstacle, you have to pay $100 to repair the robot. Fortunately, you know the locations of these obstacles, and these locations will not change over time. You will spend $1 for wear-and-tear each time your robot moves. The robot will start from a given location, and try to reach a location of interest. When it reaches the destination location, it will collect some important data that is worth $100. **Your goal is to compute a policy given the terrain of the site and the uncertainty of the navigation using value iteration.**

To be more concrete, the deep ocean navigation problem is part of a larger set of problems that involve decision making in the presence of uncertainty, in a fixed, known world. Given knowledge of this world (in the form of a 2-dimensional grid), a negative reward for movement cost (-1), a positive reward for reaching the destination (+100), and a negative reward for hitting obstacles (-100), you are asked to compute the optimal policy given that each movement has uncertainty using **value iteration**. **The policy is a mapping that tells you, in each grid location, where your robot should go, to achieve the highest accumulated reward over time with the greatest likelihood.**

Instructions:

You are to compute a policy for a given grid, which has a fixed set of unmoving obstacles and one destination location.

For each test case, your script will read input data from a file (named “input.txt” in the current directory) and write the result to a file (named “output.txt” in the current directory). Your script will not take any arguments from the command line.

Input: The input file is formatted as follows (all arguments are 32-bit integers):
<grid_size> // strictly positive
<num_obstacles> // non-negative
Next num_obstacles lines: <x>, <y> // non-negative, the locations of obstacles
<x>,<y> //destination point

Output: You compute a policy using value iteration, and write the policy into the output file in the following format:

- Obstacles are represented by the letter ‘o’
- Move East is represented by the right-caret character ‘>’
- Move West is represented by the left-caret character ‘<’
- Move North is represented by the hat symbol ‘^’
- Move South is represented by the letter ‘v’
- The destination is represented by a period symbol ‘.’

Example:

Input.txt:

```
4
2
0,0
3,0
```

output.txt

```
ovvo
vvvv
>>.<
>>^<
```

- In the Bellman equation for value iteration, there are two parameters “Gamma” (γ) and “Epsilon” (ε). Please set the value of gamma to be **0.9** and epsilon to be **0.01**.
- **Stopping Criterion:** We use a simple criterion here, i.e., we stop the algorithm when the change of the value of each position is less than “Epsilon” (ε).
- Moving off the grid is considered a valid action (for example, at state (0, 0) moving North is off the grid). In this case, consider this a transition from (0, 0) to (0, 0) with action North (i.e., it will end up staying in the current position).
- Treat obstacles as non-terminal, meaning that the robot can (theoretically) move into and over an obstacle.
- **Tie Breaking:** If values are the same for your available moves, choose to move in directions in this order of preference: North, South, East, West.