4100/5100 Assignment 5: MDPs and Q-Learning ... On Ice!
Due June 14 9PM

In this assignment, we'll revisit Markov Decision Processes while also trying out Q-Learning, the reinforcement learning approach that associates utilities with attempting actions in states.

The problem that we're attempting to solve is the following variation on the "Wumpus World" sometimes used as an example in lecture and the textbook.

1. There is a grid of spaces in a rectangle. Each space can contain a pit (negative reward), gold (positive reward), or nothing.
2. The rectangle is effectively surrounded by walls, so anything that would move you outside the arena, instead moves you to the edge of the arena.
3. The floor is icy. Any attempt to move in a cardinal direction results in moving a somewhat random number of spaces in that direction. The exact probabilities of moving each number of spaces are given in the problem description. (If you slide too far, see rule #2.)
4. Landing on a pit or gold effectively "ends the run," for both a Q learner and an agent later trying out the policy. It's game over, win or lose. (To simulate this during Q learning, set all Q values for the space to equal its reward, then start over from a random space.)

A sample input looks like this:

```
MDP
0.6 0.3 0.1
- P - G - P - - G -
P G - P - - - P - -
P P - P P - P - P -
P - - P P - - - - P
- - - - - - - - P G
```

The first line reads "MDP," which says we should solve this like an MDP. (The alternative is "Q".) The second line says that the probabilities of moving one, two, or three spaces are 0.6, 0.3, and 0.1. The rest is a map of the environment, where a dash is an empty space, P is a pit, and G is gold.

Your job is to finish the code in OnIce.java for MDPSolver.solve() and QLearner.solve(). These take a problem description like the one pictured above, and return a policy giving the recommended action to take in each empty square (U=up, R=right, D=down, L=left).

1. MDPSolver should use value iteration and the Bellman equation. ITERATIONS will refer to the number of complete passes you perform over all states. You can initialize the utilities to the rewards of each state. To keep things simple, for the MDP solver you

don't have to handle the reward spaces differently just because they'll end the trial; treat them like any other space, aside from their rewards.

2. QSolver will run ITERATIONS trials in which a learner starts in a random square and moves until it hits a pit or gold, in which case, the trial is over. (If it was randomly dropped into gold or a pit, the trial is immediately over.) The learner moves by deciding randomly whether to choose a random direction (if rng.nextDouble() < EXPLORE_PROB) or move according to the best Q-value of its current square (otherwise). Simulate the results of the move on slippery ice to determine where the learner ended up - then apply the Q-learning equation given in lecture and the textbook. (Not SARSA, the very similar other kind of Q-learning.)

The fact that a trial ends immediately on finding gold or a pit means that we want to handle those spaces in a special way. Normally Q values are updated on moving to the next state, but we won't see any next state in these cases. So, to handle this, when the agent discovers one of these rewards, set all the Q values for that space to the associated reward before quitting the trial. So, for example, if gold is worth 100 and it's discovered in square x, Q(x,UP) = 100, Q(x,RIGHT) = 100, Q(x, DOWN) = 100, and Q(x, LEFT) = 100. There's no need to apply the rest of the Q update equation when the trial is ending, because that's all about future rewards, and there's no future when the trial is ending. But now the spaces that can reach that space will evaluate themselves appropriately.

You should use the GOLD_REWARD, PIT_REWARD, LEARNING_RATE, and DISCOUNT_FACTOR constants at the top of the file.

Test and submit your code at:

https://www.hackerrank.com/cs41005100-assignment-5-mdps-and-q-learning-on-ice

Notice, however, that Q-learning involves some random steps, as well as occasionally arbitrary decisions. It's possible you won't match the tests exactly for Q learning, and that's fine, although the provided code does use a fixed seed for the random number generator to take away some of the variation. To help the chance that these tests match, you can do the following (which is entirely optional, and might be what you would have done anyway):
1) When generating random coordinates, always do it in the order (row,col).
2) Decide whether to explore before deciding the displacement from slipping.
3) When creating the final policy, evaluate clockwise from UP and change best action only on improvement of the utility.

When you've finished both solvers, and you're satisfied with the code you submitted to HackerRank, try running on the files bigMDP.txt and bigQ.txt with the command-line argument "eval", like so:

java OnIce eval < bigMDP.txt

java OnIce eval < bigQ.txt

This will run 10,000 trials where an agent dropped on a random empty space follows the policy until gold or a pit is encountered; and then it will calculate the average reward per move over all trials.

In a PDF submitted on Blackboard, **report these two numbers** and answer the following:

Which method did better, and why?