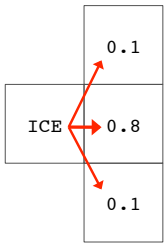
Artificial Intelligence (CS 452/552), Fall 2017
Assignment 05 (60 points)
Due before class, Wednesday, 29 November

You will write a program that reads in the given input file (`iceWorld.txt`); this file describes a grid-world environment consisting of:

- A *starting state*, marked by an `S`.
- A *goal state*, marked by a `G`.
- A set of states consisting of *open space*, marked by an `O`.
- A set of states consisting of *icy surface*, marked by an `I`.
- A set of *holes*, marked by an `H`.

Your program will then use two different reinforcement learning algorithms, SARSA and Q-learning, to learn the optimal policy, where we have the following dynamics:

- An agent can move in one of four directions—up, down, left, right—by a single grid-square.
- If the agent is in either the start state or in open space, it can move in any of the four directions deterministically, except that it cannot leave the grid. Any move that attempts to move off the grid will cause the agent to stay exactly where it is. The goal location functions as an absorbing state, so that once the agent has entered that state, any movement has no effect, and they remain at the goal.
- If the agent is on an icy surface, any attempt to move in a given direction will succeed with probability 0.8; in other cases, the agent will move diagonally to the left or right in the given direction, with probability 0.1 of each possibility. For example, if the agent is trying to move to the right in the icy square shown below, they will end up in one of the 3 locations shown, with the probabilities given:



- Any move into a hole will cause the agent to return to the location from which they just moved (after climbing out of the hole), and will incur a cost of -50; all other movements into open space or on icy surfaces incur a cost of -1. For any action that results in entering the absorbing goal state, the reward received is $+100$.

To do the learning, you should implement each algorithm to use the following parameters:

- An *episode* for learning is defined as $\min(M, N)$, where M is any number of moves that take the agent from the start state to the goal, and $N = (10 \times w \times h)$, where w and h are the width and height of the grid-world, respectively. That is, each episode ends as soon as the agent reaches the goal; if that doesn't happen after N time-steps (actions taken), then the episode terminates without reaching the goal. After each episode, the agent will return to the start state.

- The future discount factor is set to $\gamma = 0.9$; it never changes.
- The policy randomness parameter is initially set to $\epsilon = 0.9$; every 10 episodes it is updated. In particular, if E is the number of episodes already past, then for all values $E >= 10$, we set $\epsilon = 0.9/\lfloor E/10 \rfloor$. This means that after 1000 episodes, $\epsilon = 0.009$; at this point, we should set it to 0, and no longer act randomly.
- The step-size parameter for learning updates is set to $\alpha = 1.0$, and can be omitted from calculations; for this application, we do not need to reduce it, as we are not particularly interested in the values to which it converges, only the policy that is produced.

For each algorithm, you will do 2000 episodes of learning. Thus, there will be 1000 episodes during which the agent will act randomly, and 1000 for which it will always act greedily (although those greedy actions can change over time, since it will still be updating values and learning). You will produce and hand-in the following:

1. All source code for your program.
2. A text-file consisting of the reward gained for each of the 2000 episodes, for each algorithm.
3. A graph (in PDF form) that compares these reward values for each algorithm over time. That is, the graph will plot two data-series, one showing the value accumulated per episode by SARSA, and one showing the same results for Q-learning.
4. A text-file consisting of the best policies found after each 100 episodes (for a total of 20 policies), along with the episode number, for each algorithm. These policies will be in the form of a single action choice `U`, `D`, `R`, or `L` (up, down, right, left) for each of the non-hole states of the grid. Since we do not ever choose actions in the hole-states, these will simply be marked in the policy by an `H`. Thus, a final policy for one algorithm ***might*** look like the following:

```
Episode: 2000

RRRRRRRRRD
RRRRRRRRRD
RRRRRRRRRD
UUUHHHHRRD
UUUHHHHRRD
DDHHHHHHRD
RDDDDDDDDD
SRRRRRRRRR
RUUUUUUUUU
UHHHHHHHHU
```

Submit the work on D2L by the due-date and time (before class on the date given at the start of the assignment). Your submission should consist of an archive containing the source code and instructions for using it, along with the other materials outlined above.

Code should be written using sound software engineering principles and standard style, with comments preceding each method or function, and other generally sane practices throughout.