CSE 368 - AI: Homework 8
Reinforcement Learning

In this homework you will use the MDP solution from last week and compare it to solutions from reinforcement learning. In the handout file there is an `AgentInteraction` class that abstracts physical interactions with the virtual maze. Write an reinforcement learning agent `RLAgent` that can take a sequence of interactions and compute a policy. This agent does not know about the transition matrices or the structure of the maze, but will keep track of rewards and the policy in a Q-function.

1) **(20) ϵ-Greedy Policy.** The trick with reinforcement learning is that the agent is going to use partially learned policies as it is gathering more data about the system. In order to trade off how much exploration an agent does vs how much an agent follows the policy which it thinks is beset, it uses a parameter ϵ. Following the current policy is called "greedy" since the agent optimized the known reward. An ϵ-greedy policy follows the greedy policy $(1-\epsilon)$ of the time and otherwise does a random exploration move with probability ϵ. Implement a function `epsGreedy` that returns the currently optimal action $(1-\epsilon)$ of the time and a random one otherwise. Write a helper function called `bestAction` that returns the best action from a given state (according to the current value of Q).

2) **(30) Q-learning Step.** Write a function that executes one step of Q-learning. The agent should apply an ϵ-greedy action from the current state, and use the returned reward and next state in updating the Q function. Use the learning rate α given in the `RLAgent` template file and the γ that is consistent with the associated MDP. This way you can compare the Q based policy to the optimal policy computed by the MDP. This function should use the `takeAction` function in the agent interaction interface.

3) **(20) Training Episode.** Write a function that randomly picks an initial state and then takes a sequence (use a default value of 20) of consecutive q-learning steps. Each one of these is called a *episode* and represents a path that an agent takes through the maze while executing and learning about the optimal policy. Write a function call `trainingEpisode` that uses the `reset` function in the agent interaction interface in order to reset the system to a random state.

4) **(30) Train the RL-Agent.** Write a function train that runs a number of training episodes and stores the quality of the Q function for each episode in the member qualityQ but appending the current value returned from the provided function `evalQ`, which returns the fraction of non-wall locations where the best action according to Q and the best action computed from the MDP agree. You can monitor the convergence using the supplied `plotQQual()` function in the `drawprob.py` file.

(Extra Credit) **(20) (Written) Find Better Learning Parameters**. Submit a plot for different values of α and ϵ, and identify parameter combinations that work better, i.e. converge faster than the default values. It is relatively easy to get %90 of the Q values to converge to the optimal solution, but the remaining few take many more training iterations. You don't need to show all the plots reaching 1. The plots have a 'dip' in the quality of the first few hundred iterations. Why? How could you change the initialization procedure to fix this problem?