Project 3

Directions

- Submit one file to Carmen: `qlearningAgents.py`
- Expected time commitment: 3-6 hrs
- Due: Friday, Oct. 19, 5:00PM
- Download this ZIP file with the Pacman source code: reinforcement.zip. (This code, and the idea for the assignment, comes from UC Berkeley.)

Task 1 — Passive RL (4 pts)

When solving an MDP (using either value iteration or policy iteration), the agent has perfect information about the problem that it is trying to solve. In order to come up with the best policy, the agent pondered the problem until it had the optimal policy, before even setting foot in the world. In this project, you will design an agent that instead of coming up with an optimal policy beforehand (which is usually not possible due to imperfect information), will explore the world to eventually arrive at a policy.

For this task, you will be writing a method to update the Q-value (i.e. the expected value of an action from a given state) after each action. Open the file `qlearningAgents.py` and take a look at the code for `QLearningAgent`. For this task, you will be completing the `update()` method.

The variable updated by this method is the `qValues` variable, which is a dictionary, with keys that are tuples: (state, action). The update equation is as follows:

$$Q_{i+1}(s,a) = (1-\alpha) * Q_i(s,\ a)\ + \alpha * [R(s) + \gamma * max_{a \in A(s')} Q_i(s',\ a)]$$

The max here is computed by the `computeValueFromQValues()` function, so your `update()` method will only consist of about 2 lines of code. When you've finished, watch your q-learner get better under manual control, using the following:

```
python gridworld.py -a q -k 5 -m
```

Check your work with the following command:

```
python autograder.py -q q4
```

Task 2 — Active RL (3 pts)

Now we need a way to explore the state space automatically. If you only choose random actions, you may not get much better at the task. If you only choose the best actions (according to your knowledge so far), there may be states that you don't explore. Agents must use a balance of **exploration** (random actions) and **exploitation** (best actions) to find a good policy. Typically, this is done by choosing a random action with a certain probability, ε, and the best action with probability (1 - ε).

The code for this task should be entered into the `getAction()` method. First, get the legal actions from this state. Then, with probability given by `self.epsilon`, make a random choice, otherwise use `computeActionFromQValues()` to find the current best action. Return the selected action.

Watch your q-learner figure out how to navigate the grid using the following:

```
python gridworld.py -a q -k 100
```

Test with:

```
python autograder.py -q q5
```

Task 3 — Approximate RL (3 pts)

What if the state space is large enough that you don't have time to explore all of the states? Instead of only associating a Q-value with each state + action pair, you could generate a set of **features** that would inform how similar a state + action pair is to other state + action pairs that you've seen before, in order to generate an approximate Q-value (even for states that you may not have seen before).

To do task 3, complete the `update()` method in the `ApproximateQAgent`. The features are generated for you, all you have to do is decide how to update a set of weights associated with each feature. Use the following equations:

$$w_i = w_i + \alpha * diff * f_i(s, a)$$
$$diff = [R(s) + \gamma * max_{a' \in A(s')} Q(s', a')] - Q(s, a)$$

Again, the max is computed for you in the `computeValueFromQValues()` method. You can see it in operation by using:

```
python pacman.py -p ApproximateQAgent -a
     extractor=SimpleExtractor -x 50 -n 60 -l mediumClassic
```

Check your work in the usual way: `python autograder.py -q q8`