Machine Learning for Robotics SS 2017
Instructor: Georg Martius `<georg.martius@tuebingen.mpg.de>`
Due-date: 22.06.2017 (hand in at beginning of recitation session)
Exercise Sheet 7
June 12, 2017

1 Value Iteration

We will get our hand on value iteration for known MDPs. You will test your systems on a simple Gridworld domain.
The code for this exercise contains the following files, available as zip archive: `gridworld.zip`:

Files:

agent.py The file in which you will write your agents.

mdp.py Abstract class for general MDPs.

environment.py Abstract class for general reinforcement learning environments (compare to mdp.py)

gridworld.py The Gridworld code and test harness.

utils.py some utility code, see below.

The remaining files graphicsGridworldDisplay.py, graphicsUtils.py, and textGridworldDisplay.py can be ignored entirely.
You will need to fill in portions of `agent.py` and bits in `gridworld.py`.

1.1 Gridworld: Getting started

To get started, run the Gridworld harness in interactive mode:

```
python gridworld.py -m
```

You will see the an two-exit Gridworld. Your agent's position is given by the blue dot, and you can move with the arrow keys. Notice that the agent's value estimates are shown, and are all zero. Manual control may be a little frustrating if the noise level is not turned down (-n), since you will sometimes move in an unexpected direction. Such is the life of a Gridworld agent! You can control many aspects of the simulation. A full list is available by running:

```
    python gridworld.py -h
```

You check out the other grids, change the noise or discount, change the number of episodes to run and so on. If you drop the manual flag (-m) you will get the RandomAgent by default. Try:

```
python gridworld.py -g MazeGrid
```

You should see the random agent bounce around the grid until it happens upon the exit. Not the finest hour for an AI agent; we will build better ones soon.
Next, either use the text interface (-t) or look at the console output that accompanies the graphical output. Do a manual run through any grid you like, and notice that unless you specify quiet (-q) output, you will be told about each transition the agent experiences. As in previous code, coordinates are in (row, col) format and any arrays are indexed by [row][col], with 'north' being the direction of decreasing row, etc. By default, most transitions will receive a reward of zero, though you can change this with the living reward option (-r).
Note particularly, that the MDP is such that you first must enter a pre-terminal state and then take the special 'exit' action before the episode actually ends (in the true terminal state (-1, -1)). Your total return may have been less than you expected, due to the discount rate (-d).
You should definitely look at agent.py, mdp.py, and environment.py closely, and investigate parts of gridworld.py as needed. The support code should be ignored.
You can use the util.Counter class in util.py. It acts like a dictionary, but has a getCount() method which returns zero for items not in the Counter (rather than raising an exception like a dictionary), though it is not really required.

1.2 Implement and test value iteration

Write a value iteration agent in `ValueIterationAgent`, which has been partial specified for you in `agent.py`. You can select this agent with '-a value'. Your value iteration agent is an offline planner, not a reinforcement agent, and so the relevant training option is the number of iterations of value iteration it should run (-i). It should take an MDP on construction and run value iteration for the specified iterations on that MDP before the constructor returns. Recall that value iteration computes estimates of the optimal values. From these value estimates, you should synthesize responses to getPolicy(state) and getQValue(state, action). (If your implementation is such that the policy or q-values are precomputed, you may simply return them.) You may assume that 100 iterations is enough for convergence in the questions below. (Note: to actually run your agent with the learned policy, use the -k option; press enter to cycle through viewing the learned values, q-values and policy execution.)
Hint: the function `MDP.getReward` takes 3 arguments: state, action and nextstate. However, `nextstate` is ignored in the current implementation, such that you can pass `None`.

(a) How many rounds of value iteration are needed before the start state of `MazeGrid` becomes non-zero? Why?

(b) Consider the policy learned on `BridgeGrid` with the default discount of 0.9 and the default noise of 0.2. Which one of these two parameters must we change before the agent dares to cross the bridge, and to what value?

(c) On the `DiscountGrid`, give parameter values which produce the following optimal policy types or state that they are impossible:

   (a) Prefer the close exit (+1), risking the cliff (-10)
   (b) Prefer the close exit (+1), but avoiding the cliff (-10)
   (c) Prefer the distant exit (+10), risking the cliff (-10)
   (d) Prefer the distant exit (+10), avoiding the cliff (-10)
   (e) Avoid both exits (also avoiding the cliff)

(d) On the `MazeGrid`, with the default parameters, compare the value estimate of the start state after 100 iterations to the empirical returns from running 10 episodes (-k 10 -q). How close is the value estimate to the empirical average discounted rewards? How about if we run 10000 episodes (-k 10000 -q)? Does the value estimate predict the long-term average discounted rewards correctly?

Note that your value iteration agent does not actually learn from experience. Rather, it ponders its knowledge of the MDP to arrive at an optimal policy before ever interacting with a real environment. This distinction may be subtle in a simulated environment like a Gridword, but it's very important in the real world, where the real MDP is not available.

1.3 Create your own gridworld

Go and create your own gridworld in `gridworld.py`. Check the value function etc. for different noise, discount and livingReward.