Machine Learning for Robotics SS 2017
Instructor: Georg Martius <georg.martius@tuebingen.mpg.de>
Due-date: 29.06.2017 (hand in at beginning of recitation session)
Exercise Sheet 8
June 23, 2017

1 Model free Reinforcement Learning

This time we will consider the case where the MDP of the system is unknown. You will implement Q-Learning and test it on a simple Gridworld domain and crawling robot.
The code for this exercise contains needs two more files than the previous exercise. You can get all of them from `qlearning.zip`:

Files:

agent.py The file in which you will write your agents (with small modification)

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. (with small modification)

crawler.py The crawler robot simulation code (new)

utils.py some utility code, see below.

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

Gridworld: Consult the previous exercise sheet for the general usage of the gridworld simulator. You also need the solutions from that exercise to compare to.

1.1 Qlearning

You will write a Q-learning agent, which does very little on construction, but then learns by trial and error interactions with the environment through its update(state, action, nextState, reward) method. A stub of a Q-learner is specified in QLearningAgent in agent.py, and you can select it with the option '-a q'. You should first run your Q-learner through several episodes under manual control without noise (e.g. '-k 5 -n 0 -m'), for example on the MazeGrid. Watch how

the agent learns about the state it was just in. Your actual agent should be an epsilon-greedy learner, meaning it chooses random actions epsilon of the time, and follows its current best q-values otherwise.
Written short questions:

(a) Train your Q-learner on the MazeGrid for 100 episodes.

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

How are the learned values different from those learned by value iteration (last exercise), and why? How can you make them closer to the optimal values?

(b) Train your Q-learner on the BridgeGrid with no noise (-n 0.0) for 100 episodes. How do the learned q-values compare to those of the value iteration agent? Why will your agent usually not learn the optimal policy?

(c) Train your Q-learner on the CliffGrid for 300 episodes. Compare the value it learns for the start state with the average returns from the training episodes (printed out automatically). Why are they so different?

1.2 Eligibility traces

Add an implementation of eligibility traces. Add a new class by copying QlearningAgent and also wire it up in gridworld.py to be selectable from commandline with '-a qe'. Hint: the opts.agent=='q' is used several times, so make sure you change the code appropriately. Also add a parameter '–gamma' to the commandline to set the eligibility parameter γ The agent superclass has now a function reset which is called at the start of each episode.

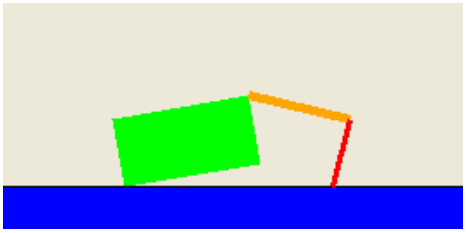
(a) check the values and Q-values after just 5 episodes with $\gamma = 0.5$ on the MazeGrid. `python gridworld.py -g MazeGrid -a qe --gamma=0.5 -k 5 -q` . Compare the result with version without the traces. What do you observe.

(b) Try the same with a larger value of γ. What do you observe and why?

(c) *Bonus:* On large grids, for instance, the normal ϵ-greedy exploration takes very long. Also in the BridgeGrid above we have seen that the agent does not explore very well. What would be done to improve the exploration? Try to implement it and test it on the Bridge and on a self made large grid.

1.3 Crawler Robot

In this part you will let a simple robot learn to locomote. The robot looks as follows:



Try the following command:

```
python crawler.py
```

This will invoke the crawling robot from class using your Q-learner. Play around with the various learning parameters to see how they affect the agent's policies and actions. Ensure that your Q-learner works in this non-episodic environment.