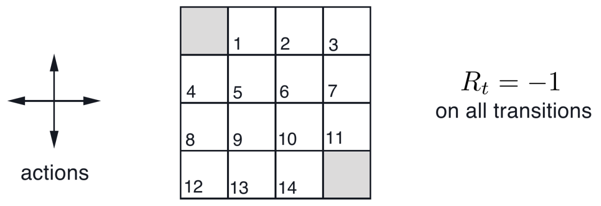
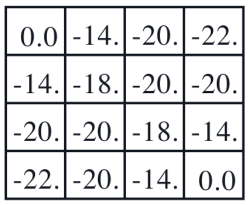
Reinforcement Learning WS 2024
TAs: Pierre Schumacher, Rene Geist
Due-date: **29.10.2024, 14:00** (upload to ILIAS as one file)
Filename: homework2-NAME1-NAME2-...zip
Homework Sheet 2
October 22, 2024

## 1 State-Action Value Function and Policy Iteration

(a) Consider the 4×4 gridworld shown below.



The nonterminal states are $\mathcal{S} = \{1, 2, ..., 14\}$. There are four actions possible in each state, $\mathcal{A} = \{\text{up}, \text{down}, \text{right}, \text{left}\}$, which deterministically cause the corresponding state transitions, except those actions that would take the agent off the grid leave the state unchanged. Thus, for instance, $p(6 \mid 5, \text{right}) = 1$, $p(7 \mid 7, \text{right}) = 1$, and $p(10 \mid 5, \text{right}) = 0$. This is an undiscounted, episodic task. The reward is $-1$ on all transitions until the terminal state is reached. The terminal state is shaded in the figure (although it is shown in two places, it is formally one state). The reward function is thus $r(s, a, s') = -1$ for all states $s, s'$ and actions $a$. Suppose the agent follows the equiprobable random policy $\pi$ (all actions equally likely). The value function is:



| | | | |
|---|---|---|---|
| 0.0 | -14. | -20. | -22. |
| -14. | -18. | -20. | -20. |
| -20. | -20. | -18. | -14. |
| -22. | -20. | -14. | 0.0 |

What is $q_\pi(11, \text{down})$? What is $q_\pi(7, \text{down})$? What is $q_\pi(9, \text{left})$?

from Sutton and Barto's Textbook

(b) Give an equation for the optimal value function $v_*$ in terms of the optimal state-action value function $q_*$.

(c) Give an equation for $q_*$ in terms of $v_*$, the transition probability $\mathcal{P}$, and reward function $\mathcal{R}$.

(d) Give an equation for the optimal policy $\pi_*$ in terms of $q_*$.

(e) Derive the Bellmann Expectation Equation for $q(s, a)$ using only $q$ not $v$. See Slide 32 of the lecture nodes. Similarly to the self-contained formula for $v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')\right)$

## 2 Value Iteration

We will get our hands on value iteration for known MDPs. You will use the simple Gridworld domain from last exercise. However, use the `2_3-gridworld.zip`, such that we all start from the same code. As before we have these files:

**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.

**gridworldClass.py** Definition of Gridworld class.

**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`.

### 2.1 Gridworld: Getting started

Please check the last exercise for the basic introduction.

### 2.2 Implement and test value iteration

Write a value iteration agent in `ValueIterationAgent`, which has been partially 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 for 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 extracted policy, use the -k option; press enter to cycle through viewing the computed 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`.
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 dictionary (rather than raising an exception like a dictionary), though it is not really required. (The name "Counter" is also a bit misleading, but nevermind.)

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

(b) Consider the policy computed 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 of value iteration to the empirical returns $\mathbb{E}[G_t]$ as you computed in the last exercise (-k 10 -q, and -k 10000 -q). Report the numbers. Why do you get different results?

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 is very important in the real world, where the real MDP is not available.

## 2.3 Create your own gridworld

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