Lab 2:
Reinforcement learning

N.B.

There are two versions of GridWorld on studentportalen. For this assignment please use `GridWorld2.jar`. It is an updated version from what used during the intro-lab. The old version is also kept at studentportalen if the new version would contain a serious bug.

We also wish to encourage you to read the instructions carefully, and if anything is unclear or if you have questions, please do not hesitate to contact us either at the lab, coming by our offices or send an e-mail:

- peter.backeman@it.uu.se
- aleksandar.zeljic@it.uu.se
- gustaf.borgstrom@it.uu.se

Introduction

Goals

After this assignment you will understand the idea behind temporal-difference learning: to update an output value towards the next output value (plus the immediate reward), rather than towards the final outcome of a training episode. You will use the Q-learning and SARSA algorithms to let an agent find an "optimal" policy for maximizing some metric of reward in a simple deterministic process.

Preparations

Read through these instructions *carefully* in order to understand what is expected of you. Read your lecture notes, handouts from lectures 7 and 8, and parts of chapter 6 in the text book.

Report

Hand in a report according to the instructions found on Studentportalen. It should contain answers to all questions, and all requested plots. Observe that some questions are marked with ★. These questions are of particular importance and should be answered thoroughly.

Files

Download the file `GridWorld2.jar` from the assignment page on Studentportalen. It is the program that you will use for this lab. It should be executable on any machine with a Java VM installed. You may have to give permission to run the file as a program (right click ⇒ properties ⇒ permissions ⇒ "Allow executing file as program").

Gridworld

The tool that you will use in this assignment is a program called Gridworld (see Figure 1). In Gridworld, an agent (the robot) is moving around in a two-dimensional grid. Each square in the grid represents a state that the agent can be in. In each state the agent can choose between four different actions (moving *up*, *down*, *left* or *right*), each of which moves the robot to a new state in the corresponding direction. Some states are occupied by walls, if the agent tries to move to such a state it will remain in the state that it was already in.

An action can take the agent to the goal (drawn as a trophy), which occupies one of the states. When the agent takes an action that leads it to the goal it is given an immediate reward of 1, and the episode ends (the agent moves no further). There are also danger zones, if a danger zone are entered, the episode ends and the agent receives a negative reward of -1. For all other actions (leading to some state which does not contain the goal) the agent does not receive any reward (or, equivalently, a reward of 0).

The states are numbered from 0 and up, going from left to right and from top to bottom. Thus, in the *Classic* map (shown in Figure 1) the state in the lower left corner has number 30, the state in the lower right corner has number 34, and the obstacles occupy states number 11, 13, 16, 17 and 18. The goal is in state number 12.

The arrows in the figure indicate current Q-values for the different actions in the states. The arrow lengths are proportional to the Q-values. The longest arrow in each square is drawn in red, to make it easier to see which path the agent currently believes is the best one. The figure shows the situation after a few episodes, before the values have converged. Note that you can also see the exact Q-values for a state in the bottom bar in the window. By default, you

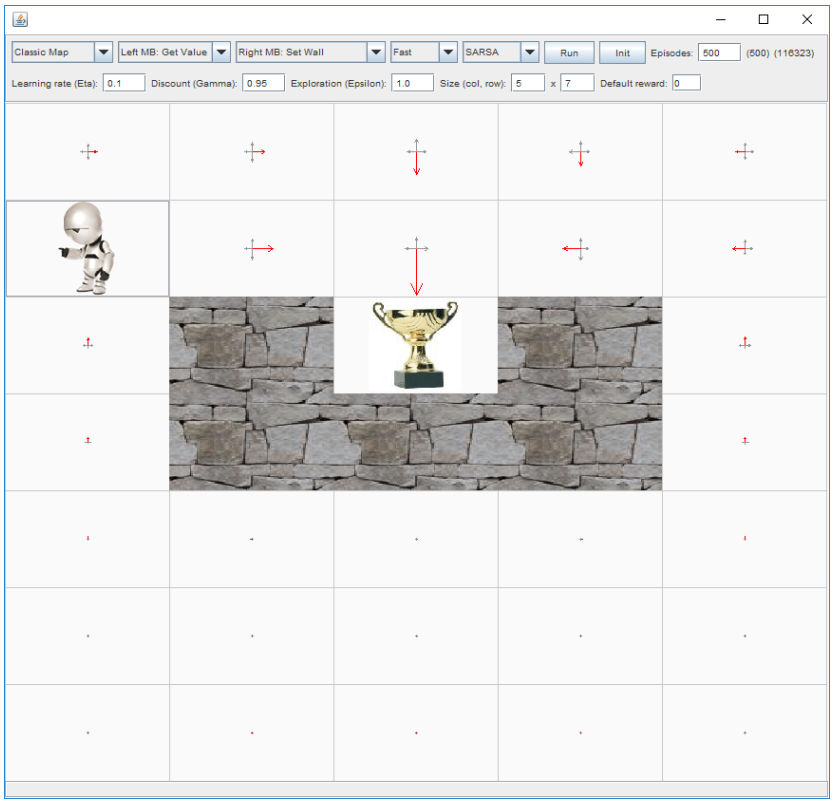

Figure 1: Gridworld with the Classic map.

choose a state to be displayed there by left-clicking the corresponding square.

Getting started

To start the tool, run the file `GridWorld.jar` (e.g., by double-clicking it or using `java -jar` from a command line). The combo-boxes at the top of the window define (in order from left to right):

1. The map to use.
2. The left-click action (i.e., what happens when you left-click in a square).
3. The right-click action (i.e., what happens when you right-click in a square).
4. The update speed.
5. The update rule (or learning algorithm).

Pressing **Init** will reset the world, and initialize the start state. The values in the parameter text-boxes will be used when you press **Run**, you cannot change

them when a simulation is running. To change them if a simulation is running, press the **Halt** button and then change the parameters. If you change the size, you must press **Init** for the change to take effect. The other parameters do not require you to press **Init** to take effect.

Task 1: Q-learning

The first task is to make the agent learn state-action values (Q-values) with Q-learning. In this task, action selection will be 100% random, i.e., the agent selects amongst the four possible actions with equal probability. This is achieved by using ϵ-greedy action selection with $\epsilon = 1.0$ (this value of ϵ makes the agent select all actions randomly). Even though the agent is actually updating its policy, it does not use the policy when making decisions.

1.1 Set-up

Use the following settings:

- The *Classic* map.
- *Fast* update speed. (You can change this later to very fast when convenient.)
- *Q-Learning* as the update rule.
- 500 episodes (the number of sessions to perform before stopping, 0 means never stop).
- The learning rate/step size $\eta = 0.1$.
- The discount factor $\gamma = 0.95$.
- The exploration parameter $\epsilon = 1.0$ ($\epsilon = 1$ means 100% random action selection, $\epsilon = 0$ means 100% greedy).
- A map size of 5x7 squares.

Now click **Init** and then **Run** and observe the arrows as they grow. Repeat the initialization and run procedure a few times. You may have noticed the grey square changing locations after each episode; it represents the start state and is randomly selected at the start of each episode.

Having observed the arrows, answer the following questions:

Question 1: *After the arrows have converged, some states will have longer arrows than others (i.e., larger Q-values). Why is this so?*

Question 2: *In the first few episodes, only the states closest to the goal will have their Q-values increased (even though the agent may start far from the goal). Explain why.*

Question 3: *Certain arrows (not necessarily in the same state) should converge towards the same lengths. Why is this the case? Give an example of two state-action pairs that get equal Q-values.*

Question 4 ★: *If you train the agent long enough, the red arrows will mark the shortest path to the goal. Why does it find the shortest path? (You may have noticed that the problem formulation—how the agent is rewarded—did not define shorter paths as "better".)*

Set the exploration rate ϵ to 0.1. Before, when $\epsilon = 1$, even though the Q-values were updated, the information was never utilized since the action selection was completely random. Now, in 90% of the cases a greedy choice based on the learnt Q-values directs the agent. So the agent will now change its behaviour over time. Set the speed to *Fast* and observe 500 episodes, repeat a couple of times, and make sure to press **Init** in between repeats.

Question 5: *Some arrows that previously converged to equal length (with $\epsilon = 1.0$) will not any more (unless you train for a very long time). Why?*

Reset the game state and set the discount rate (γ) to 1.0.

Question 6: *Try running for enough iterations for the arrows to converge. Is this solution different from the previous question? Why?*

In the following experiments you might see green and red arrows. A green arrow indicates a Q-value of above 1, while a yellow arrow indicates a value below -1.

Question 7 ★: *Try setting a default reward of 0.1 and run for enough iterations. Explain what happens and why. Do the same for a default reward of -0.1*

Question 8 ★: *Write down the update rule for Q-learning! Explain the purpose of the different constans and the intuition behind them.*

Question 9: *What is the true value (the value that Q-learning should converge to, Q*) of the action* down *in state 2 (top row, centre square)? Include the complete calculation, not just the answer.*

Task 2: SARSA

Start another instance of the simulator, and use the following parameters:

2.1 Set-up

- The *Classic* map.
- *Fast* update speed. (You can change this later to very fast when convenient.)
- *Q-Learning* as the update rule for one instance and *SARSA* for the other one.
- 500 episodes (the number of sessions to perform before stopping, 0 means never stop).
- The learning rate/step size $\eta = 0.1$.
- The discount factor $\gamma = 0.95$.
- The exploration parameter $\epsilon = 0.1$
- A map size of 5x7 squares.

You should now have 2 simulators running, one using Q-learning and the other using SARSA. Make sure that the parameters are the same. Observe 500 episodes from each simulator and notice how the arrows "behave".

Question 10: *With Q-learning the lengths of the arrows increased steadily, but using SARSA they sometimes decrease. Why is that?*

Whenever the exploration rate is larger than 0, SARSA does not converge since the arrows can decrease. However, when the exploration rates approaches 0, the arrows of SARSA does converge to some value. Try and make the arrows converge by running the game for 1000 epsiodes, divide the exploration rate by two and then repeat. Do this until the exploration rate is sufficiently small.

Question 11: *Do the arrows in the SARSA window converge to the same lengths as the corresponding arrows in the Q-learning window? Motivate your answer.*

Close the simulator using Q-learning. In the simulator running SARSA, change the exploration rate (ϵ) to 1.0, the number of episodes to 500, the speed to *Very Fast* and press **Init**.

Question 12: *The arrows look different from the result of running with Q-learning (as done in the start of the lab). Explain why.*

Select the following settings:

- Grid size 5x3.
- The *Cliff* map.
- *Very Fast* update speed.
- *Q-Learning* as the update rule.
- 10000 episodes.
- The step size $\eta = 0.1$.
- The discount factor $\gamma = 0.95$.
- The exploration parameter $\epsilon = 0.25$.

Consider the setup and think about what the result will look like.

Question 13: *Run the experiments a few times and study the resulting Q-values. Are they what you expected? Explain.*

Now change the update rule to SARSA and once again consider the setup.

Question 14 ★: *Run the experiments a few time and study the resulting Q-values. Are they what you expected? Explain. How to they differ from the Q-values from the previous question and why.*

Task 3: Two rooms

Select the following settings:

- Grid size 5x5.
- The *Two Rooms* map.
- *Very Fast* update speed.
- *Q-Learning* as the update rule.
- 50 episodes.
- The step size $\eta = 0.1$.

- The discount factor $\gamma = 0.95$.
- The exploration parameter $\epsilon = 0.1$.

Run 50 episodes, repeat a couple of times and make sure to press **Init** in between. Observe the number that is growing in the top right corner, the action count. It keeps track of how many actions have been selected since the last time you pressed **Init**. (The first number is the number of episodes passed, the second number is the total action count.) Try to get a sense of the average number of actions for 50 episodes.

Now press **Init** and select *Step* as the speed. You can now steer the agent by using the arrow keys. The agent will still update its Q-values as if it had selected the actions itself. For 10 episodes, manually guide the agent to the goal, making sure to take the shortest path. When the episode counter reaches 10 (to the left of the action counter) change the **Episodes** field to 40 and select *Very Fast* as the speed. Press **Run** (*Do not press* **Init***!*). Repeat the guiding experiment a few times.

Question 15: *What can you say about the average action count in the experiment? Compare guiding the robot to not guiding the robot.*

Question 16 ★: *In general, can you think of any disadvantages of guiding the agent in this way?*

Question 17: *Suggest an application where this method of initially leading the agent may be necessary (if not in theory, then at least in practice).*

To increase your understanding of reinforcement learning, we suggest you continue experimenting with the tool. For example, you could try different or larger maps. What happens if walls suddenly appear or disappear? How can you influence learning by manually selecting the starting positions while running? Go wild!