Project 6 IT3708:

Reinforcement Learning Using Q-Learning

a) Implementation

The learning agent gets to play the Flatland game from Project 3 k times. He gets rewards for his moves. That is a reward of 5 for eating food or coming back to the start position and -5 for eating poison. To prevent the agent from walking in circles, every step on an empty field costs -0.001.

With a learning rate of 0.6 and a discount rate of 0.4, the agent can solve all given flatland levels in k=500 iterations. These values where found by trying values walking downhill, but they don't change much on the performance. It is mainly set by the exploitation-vs-exploration-rate, which is described in b).

The Q-values are stored in a dictionairy, that has the tuple ((position, action), collected food) as keys and reward values as entrys. The position as well as the action are (x, y)-tuples of integers. The list of collected food fields is implemented as a string with their numbers.

$max_a Q(s,a)$ is found by getting the reward values vor Q(s,a) for the four possible actions in a given state from the dictionairy and comparing them to find the highest possible reward.

b) Action selection

With a probability of p, the agent selects the best action according to the values in the Q-dictionairy. If there are no entries for the given states, a random action is chosen. The probability p is set by the equation

$$p = \exp\left(\frac{-1}{T_{max}\left(1+\frac{t-i}{k}\right)}\right) \quad ,$$

where t is the number of steps taken in the current iteration and i is the number of iterations so far. This results in a growing exploitation during the k iterations. But also there is again more exploration in every step, while the agent walks through the maze. So he will stick to the best path he found for the first actions during one run, but keeps exploring the possibilities of collecting the last few food fields until the k iterations are over.

The value set for T_{max} is found to be best being 2048

c) Backup scheme

The used backup scheme is TD(x) but with a trace decay factor as in TD(λ) and without any eligibility trace. The eligibility trace would only facilitate loops of best moves in the flatland world, so the first implementation had a TD(λ) backup scheme. It was found that reducing the affected previous states to 20 gives an increase in speed, but no drawback in terms of good results. As a trace-decay factor, 0.2 was found to be good by trial and error.