Project 5 IT3708:
Reinforcement Learning Using Q-Learning

Purpose: Use reinforcement learning in a changing environment to learn an optimal policy.

1 Assignment

For this assignment, you will revisit the Flatland environment from assignment 3. The goal this time isn't just to learn to eat food and avoid poison, but also to find the shortest route to eat all the food.

To do this, you will train the agent on a scenario using Reinforcement Learning, in which the agent receives feedback (*"reward"*) after each move. After enough runs, your agent should have learned a *policy* about what is good to do in each state in the scenario. So then the agent should play the scenario a final time, this time using what it believes to be the best moves, and record the number of steps taken to eat all the food.

2 Flatland: The Testing Environment

The environment is almost similar to the one you have already made. So you should be able to reuse most of the game logic and visualization code. The biggest difference is that the agent now also can move backwards. So the agent doesn't really have a direction any more, just a position from which it can move to the four neighbouring cells.

There is no longer a fixed timestep for a scenario. Instead, the goal now is to eat *all the food* without eating any poison and then *return to the start position*, all this in *as few steps* as possible. This is almost like an instance of Δ-TSP, using food instead of cities. As last time, there is wrap-around on the edges of the world.

2.1 File Format

You should be able to read Flatland scenarios from a file. Some example files are provided with this project description, and during the demo you will be given some new, unseen files to run your algorithm on. The first line in the file is 5 integers separated by spaces: W H X Y N. H is the height of the scenario, W the width. X and Y is the starting position, N the number of foods available. Then follow H lines with W integers each, describing the layout of the scenario. A value of -2 is the starting position, -1 is poison, 0 is a free cell, values above 0 are food. The foods have values from 1 to N.

Your system should be able to handle sizes of scenarios up to H=20, W=40, and with number of foods up to N=64.

3 Q-Learning

The Q-Learning algorithm will play the chosen Flatland scenario multiple times and record how well the different actions do in different states. For each action, the agent will receive an *immediate* reward: Some positive score for eating food, negative for poison and other metrics you may choose. It then uses this reward to update its beliefs about the world. An action here is moving left, right, up or down. A state is the position of the agent and which food that has been eaten so far.

Algorithm 1 Q-Learning Pseudocode

```
1: procedure Q-Learning
2:     Q[s, a] ← Initialize array that maps states and actions to a value
3:     for 1 to K do                                   ▷ K = number of iterations
4:         Game ← Restart scenario
5:         while Game is not finished do
6:             a ← Select action to do
7:             UpdateGame(Game, a)
8:             state ← State after doing a in the game
9:             r ← Reward after doing a in the game
10:            UpdateQArray(Q, state, a, r)
11:        end while
12:    end for
13: end procedure
```

3.1 Updating Q Values

The general formula for updating the value of doing action a in state s is:
$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
in which α is the learning rate and γ the discount rate. $max_a Q(s_{t+1}, a)$ is the best value for any action in the new state one arrived. For instance, being in state B and going left and arriving in state C, one would update $Q[B, left]$ using $max_a Q(C, a)$, which is the best value for any action one can do in state C, e.g. the best of $Q[C, left]$, $Q[C, up]$, $Q[C, right]$ and $Q[C, down]$.

3.2 Selecting Action

The action selection in the pseudo code should be made to let the agent learn the environment as efficiently as possible. Always choosing the currently believed best action for a state can make the agent miss good opportunities for better actions, but choosing a bad action can waste time.

To select the action, one can with probability p choose one at random, and with probability $1-p$ choose the best one. Having a fixed p can work well, but tweaking it to give good results can take a lot of time, and it depends heavily on the current scenario. A better way can be to have a p that starts high but becomes lower during the run, similarly to how it's done in Simulated Annealing. A third way is to select between the actions in the same fashion as you selected between individuals in your EA; using the current scores of the actions as the odds of choosing them.

3.3 Updating Multiple Q Values

Moving from state B to C, doing action a and receiving reward r, leads to an update of Q[B,a]. If the reward was big, the value should be high to indicate that doing a in B is probably a good idea in the future. In a new iteration in the Q-Learning procedure, the agent is in A and does action b to arrive in B. Since the agent has learned that being in state B has the possibility of moving to state C and receive a big reward, getting to state B is probably almost as good, so Q[A,b] is updated with a high value.

This can take many iterations: first learn that B is good as one can move to C and get a reward, and then later learn that A is good as one can move to B and get a reward in the future. If the agent the first time moved from A to B to C, why not update A straight away when receiving a reward in C, not just B?

To do this, the agent should remember previous states in the game and what action it did. When getting a reward in the future, the agent can update the value of doing these actions in the remembered states right away, instead of chancing on stumbling over these combinations again in the future. This is called a *backup scheme*. Two types can be used here: TD(x) and TD(λ). In TD(x), the agent remembers x previous states, and updates these when learning something new. In TD(λ), λ is a trace decay factor deciding by how much the old beliefs should be updated.

More details about these can be found in the lecture slides. Make sure you look at the equations specifically for Q-Learning. Some authors recommend resetting e(s,a) to 0 when the action taken was not the currently believed best action when using eligibility traces in Q-Learning.

4 Visualization

After your agent has trained on the scenario, you should let it run on a fresh version of the scenario. Now the agent should select the action it believes to be best for each timestep. When the agent has eaten all food and returned to the start position, the number of steps taken and the number of poison eaten should be shown.

The visualization should show arrows pointing where the agent would go if it were in that cell, given the current combination of food in the world. An example is shown in Figure 1. Of course the agent, food and poison should also be shown. The time between each timestep in the world should be adjustable, so we can see what happens.

For bigger scenarios, it will probably happen that you wont have an arrow for each cell, as the agent never visited that cell with that combination of food during learning. It may also happen that your agent becomes stuck in a loop of "best moves", this means that your results hadn't converged yet. This may have been bad luck, or you may need more iterations.

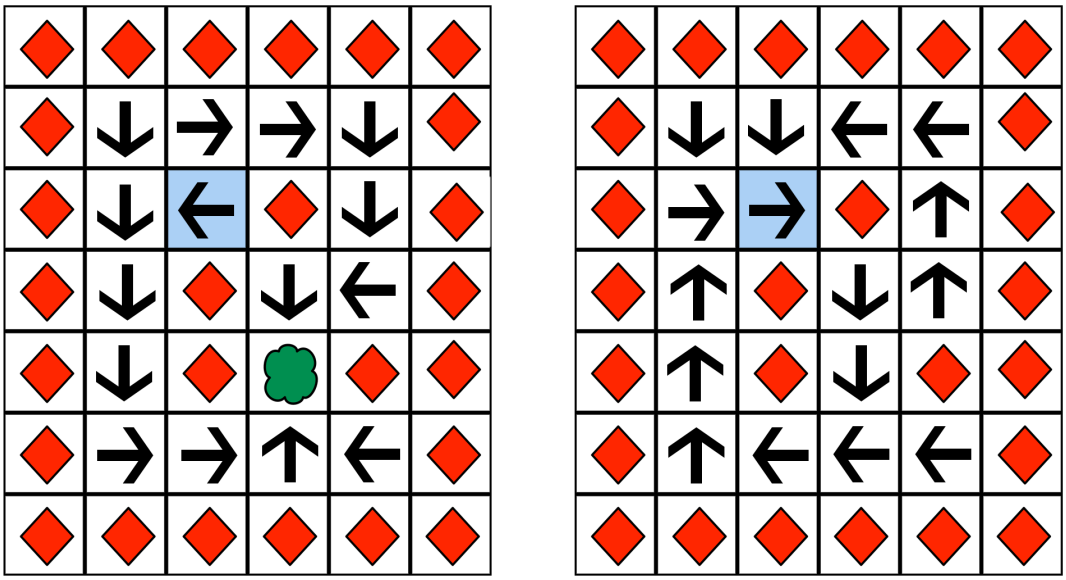

Figure 1: Red is poison, green is food. Left: Showing all the actions for all states having no eaten food, trying to get to the food. Right: Showing all the actions for all states with no food after eating it, trying to get to the start (blue shaded cell).

5 Report

You should write a report answering the points below. The report can give a maximum score of 6 points. Your report must not exceed 2 pages in total. Overlength reports will result in a point being deducted from your final score. Print on both sides of the sheets, preferably. Bring a hard copy of your report to the demo session.

a) Document your implementation (2p)

- Give an overview of your architecture, how you train your agent and how the Q-Learning is to connected to the Flatland.
- How you store Q[s,a] pairs and how you query for $max_a Q(s, a)$.
- Describe the values you use for the different parameters and why you have selected them.

b) Action selection (2p)

- Describe how you select which action the agent should take when learning, and why you selected this way. How is this in terms of exploration and exploitation?

c) Backup scheme (2p)

- Describe which backup scheme you use and why you selected it.

6 Demo

There will be a demo session where you will show us the running code and we will verify that it works. This demonstration can give a maximum of 14 points.

For this demo, you will use your Q-Learning algorithm to train an agent in the Flatland world, and show it taking the best actions it learned. You will be given some new scenario-files to run, and the goal is to find the shortest path *without* eating poison.

You should be able to choose which scenario-file to use and other parameters of your system without having to modify any code during the demo.

7 Delivery

You should deliver your report + a zip file of your code on It's Learning. The deadline is given on the assignment on It's Learning, and it is **before the demo sessions start in the morning**. The 20 points total for this project is 20 of the 100 points available for this class. For this project you should work alone.