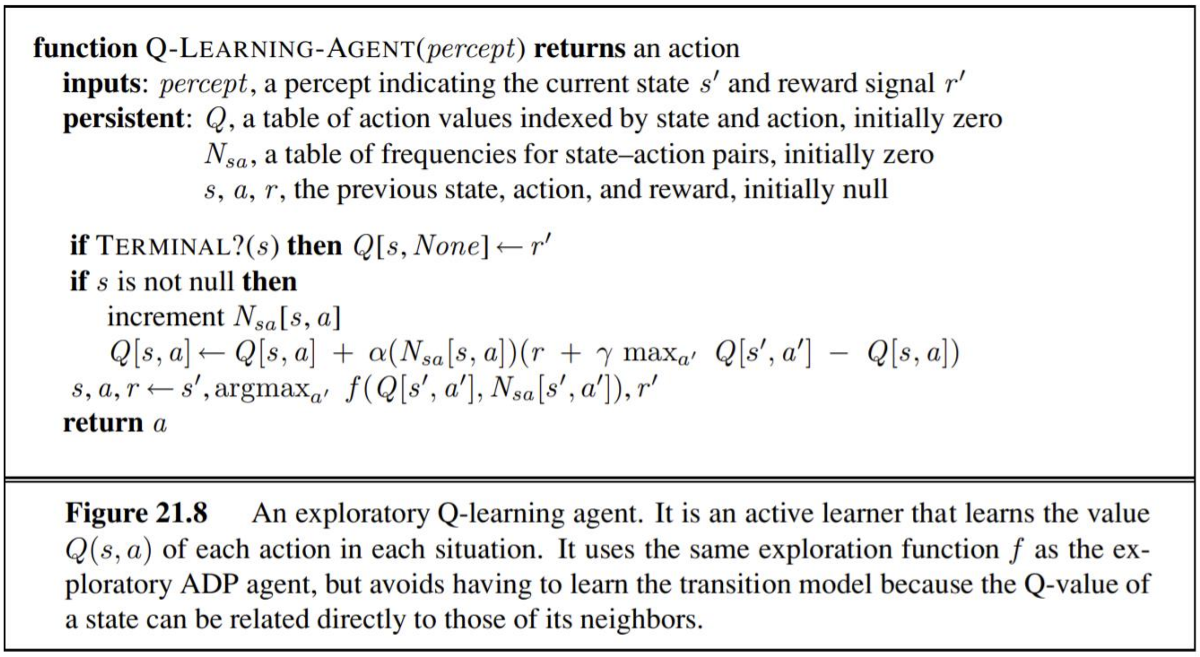
Q-Learning Agent

The software designed contains a Q-Learning agent which navigates in a maze in order to reach the goal position. The Q-Learning algorithm followed is as follows,



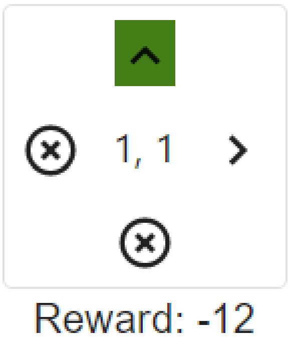
```
function Q-LEARNING-AGENT($percept$) returns an action
  inputs: $percept$, a percept indicating the current state $s'$ and reward signal $r'$
  persistent: $Q$, a table of action values indexed by state and action, initially zero
              $N_{sa}$, a table of frequencies for state–action pairs, initially zero
              $s, a, r$, the previous state, action, and reward, initially null

  if TERMINAL?($s$) then $Q[s, None] \leftarrow r'$
  if $s$ is not null then
      increment $N_{sa}[s, a]$
      $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$
  $s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$
  return $a$
```

Figure 21.8 An exploratory Q-learning agent. It is an active learner that learns the value $Q(s, a)$ of each action in each situation. It uses the same exploration function f as the exploratory ADP agent, but avoids having to learn the transition model because the Q-value of a state can be related directly to those of its neighbors.

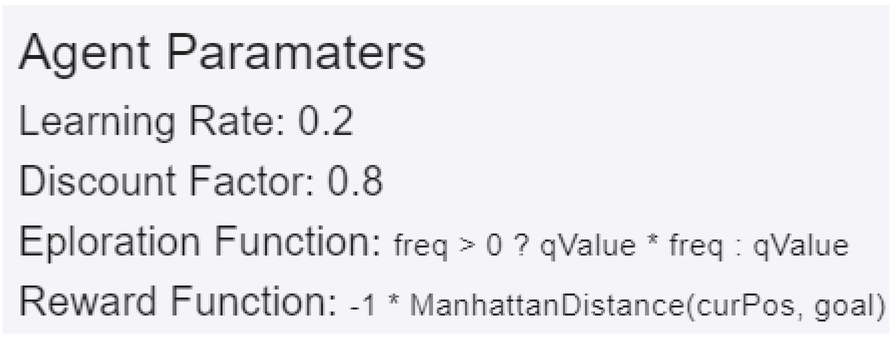
The implementation for this agent can be found in the file: RLAGENT/SRC/GAME_CORE/QLEARNINGAGENT.TS

At every position the agent has at most 4 actions available (UP, DOWN, LEFT, RIGHT). The application highlights the recommended action in green. The agent selects the recommended action according to the equation for 'a' in the above algorithm. The reward for each position is calculated as follows,

$$\text{reward(pos)} = -1 * \text{ManhattanDistance(pos, goal)}$$



By keeping the reward -ve, we can encourage the agent to find the shortest path to reach to the goal node instead of roaming around in the environment.



Agent Paramaters

Learning Rate: 0.2

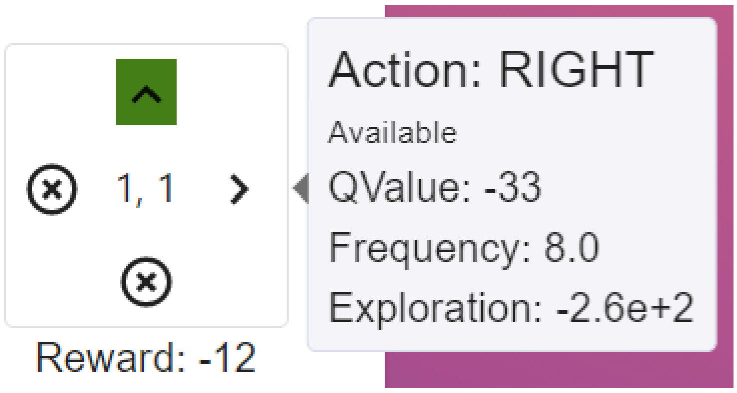
Discount Factor: 0.8

Eploration Function: freq > 0 ? qValue * freq : qValue

Reward Function: -1 * ManhattanDistance(curPos, goal)

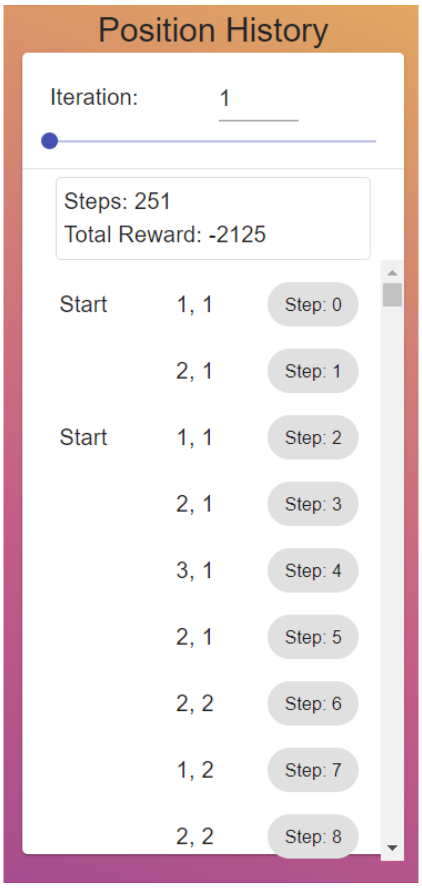
The agent also uses an exploration function to deviate from the usual route in order to find new potentially better routes. The exploration function is proportional to frequency of a state-action pair since the reward is -ve and the agent gets more penalized for taking the same route multiple times.

The knowledgebase consists of the Q-Values and Frequencies of every state-action pair explored by the agent. The application shows this on hovering over the possible actions for this state and also while hovering over the path histories. The exploration value here is the value returned by the exploration function.



Action: RIGHT
Available
QValue: -33
Frequency: 8.0
Exploration: -2.6e+2

The position history column shows the entire path taken by the agent for every iteration. It lists the position of the agent for each step and also shows the total reward collected by the agent in that iteration along with the number of steps taken to reach the goal position.

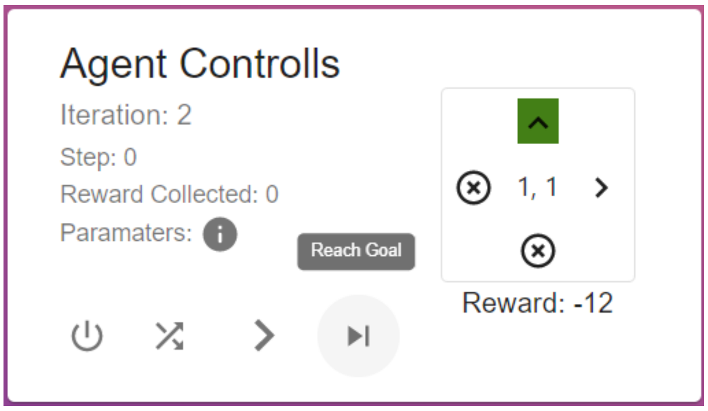


Position History

Iteration: 1

Steps: 251
Total Reward: -2125

Start	1, 1	Step: 0
	2, 1	Step: 1
Start	1, 1	Step: 2
	2, 1	Step: 3
	3, 1	Step: 4
	2, 1	Step: 5
	2, 2	Step: 6
	1, 2	Step: 7
	2, 2	Step: 8



Agent Controlls
Iteration: 2
Step: 0
Reward Collected: 0
Paramaters:

The application allows users to control the agent step-by-step or iteration-by-iteration. A button to randomly reposition the goal is also available and a button to reset everything is present.