TDDE15 - Lab 3

Completion

2022-09-27

- Erik Jareman erija971
- Isak Berntsson isabe723
- Ludvig Eriksson Knast ludkn080
- Linus Bergström linbe580

Q-Learning implementation

Implementation of the GreedyPolicy and the EpsilonGreedyPolicy functions.

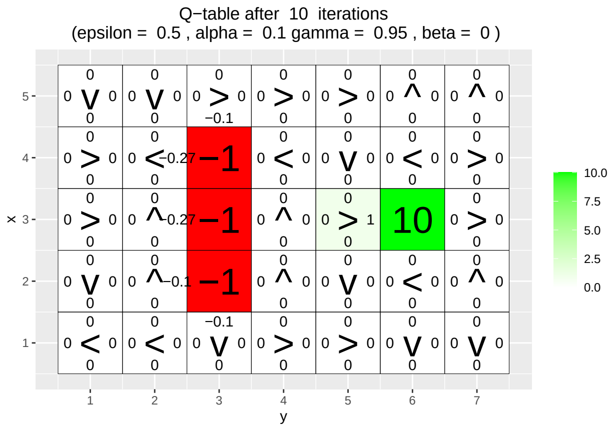
The GreedyPolicy has been fixed since the first hand in.

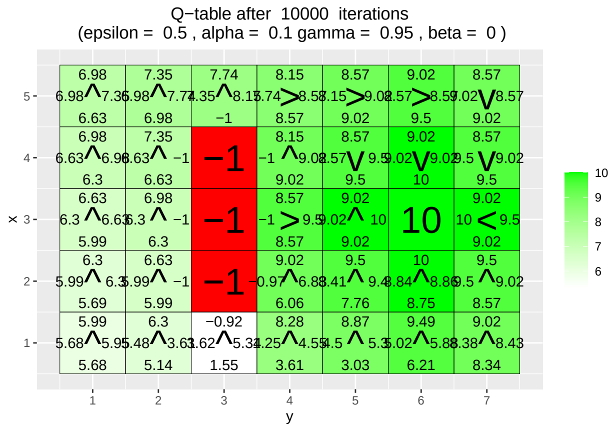
```r
GreedyPolicy <- function(x, y){

  # Get a greedy action for state (x,y) from q_table.
  #
  # Args:
  #   x, y: state coordinates.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.

  # Your code here

  #
   return(sample(which(q_table[x,y,]==max(q_table[x,y,])),1))

}
EpsilonGreedyPolicy <- function(x, y, epsilon){

  # Get an epsilon-greedy action for state (x,y) from q_table.
  #
  # Args:
  #   x, y: state coordinates.
  #   epsilon: probability of acting randomly.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.
  # Your code here

  if (runif(1) > epsilon)
  {
    return (GreedyPolicy(x, y))
  }

  return (sample(c(1, 2, 3, 4), 1))

}
```

Implementation of the q_learning function

```
q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95,
                       beta = 0){

  # Perform one episode of Q-learning. The agent should move around in the
  # environment using the given transition model and update the Q-table.
  # The episode ends when the agent reaches a terminal state.
  #
  # Args:
  #   start_state: array with two entries, describing the starting position of the agent.
  #   epsilon (optional): probability of acting greedily.
  #   alpha (optional): learning rate.
  #   gamma (optional): discount factor.
  #   beta (optional): slipping factor.
  #   reward_map (global variable): a HxW array containing the reward given at each state.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #
  # Returns:
  #   reward: reward received in the episode.
  #   correction: sum of the temporal difference correction terms over the episode.
  #   q_table (global variable): Recall that R passes arguments by value. So, q_table being
  #   a global variable can be modified with the superassigment operator <<-.

  # Your code here.

  episode_correction = 0

  repeat{
    # Follow policy, execute action, get reward.
    policy = EpsilonGreedyPolicy(x=start_state[1], y=start_state[2], epsilon=epsilon)
    end_state = transition_model(x=start_state[1], y=start_state[2], action=policy, beta=beta)

    # Get reward
    reward = reward_map[end_state[1], end_state[2]]

    # Calculate temporal difference, store in sum, and update Q-table
    temporal_diff =
      reward +
      gamma * (max(q_table[end_state[1], end_state[2], ])) -
      q_table[start_state[1], start_state[2], policy]

    episode_correction = episode_correction + temporal_diff

    q_table[start_state[1], start_state[2], policy] <<-
      q_table[start_state[1], start_state[2], policy] + alpha*temporal_diff
    # Move agent
    start_state = end_state

    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))
  }

}
```

Environment A



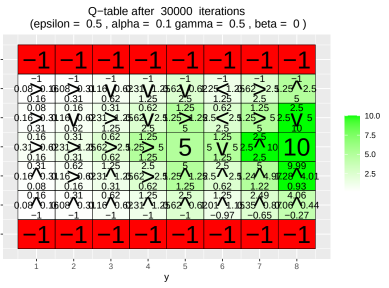


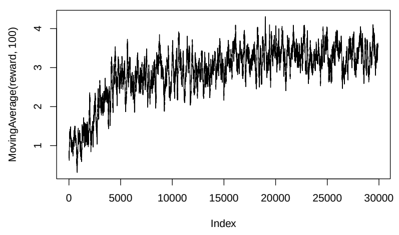
What has the agent learned after the first 10 episodes?
The agent is starting to learn that some of the actions taking the agent to the "-1" states will result in a negative reward. No positive rewards for actions has been set since none of the first 10 episodes has reached the "goal state" with reward 10.

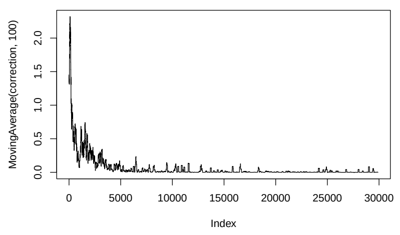
Is the final greedy policy optimal for all states?
The final greedy policy is not optimal since there are states where the agent does not take the shortest possible path to the +10 reward state.

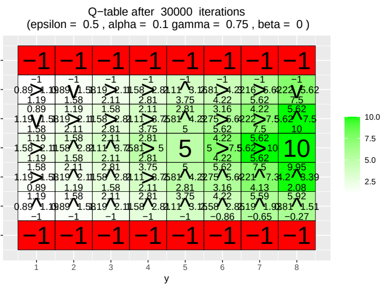
Do the learned values in the Q-table reflect the fact that there are multiple paths (above and below) to get to the possible reward?
The learned Q-values seem to prioritize one of the two possible paths, and thus, do not reflect on this fact. One way to make the agent find both paths could be to increase the probability for exploration.

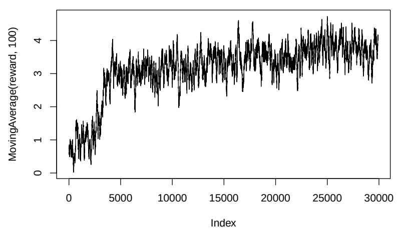
Environment B

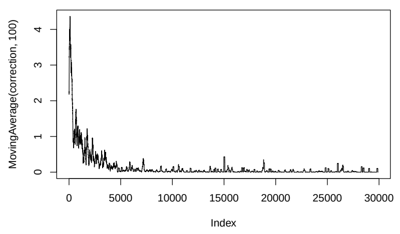


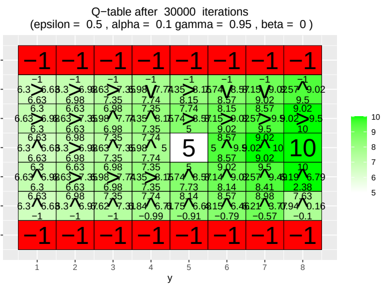


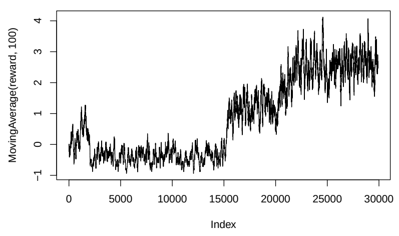


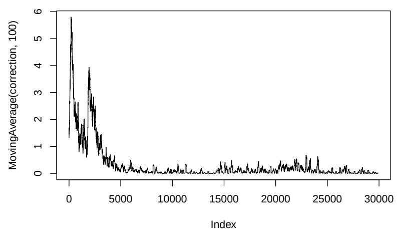


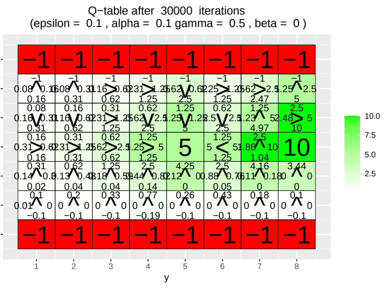


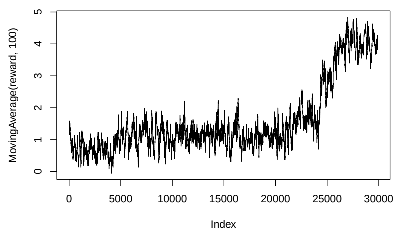


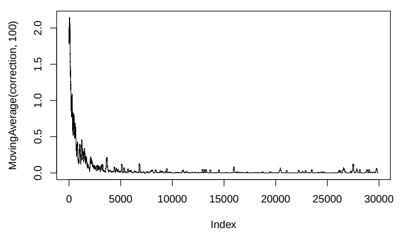


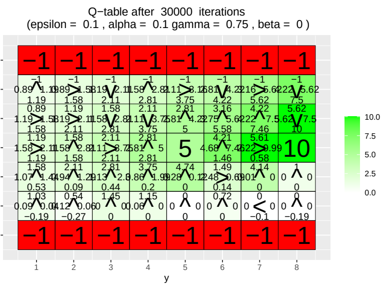


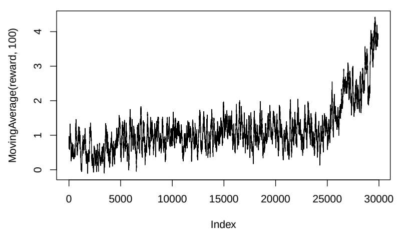


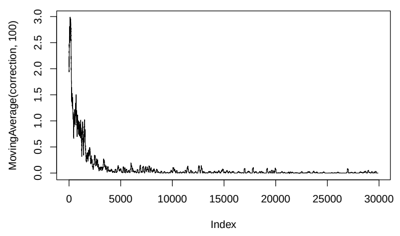


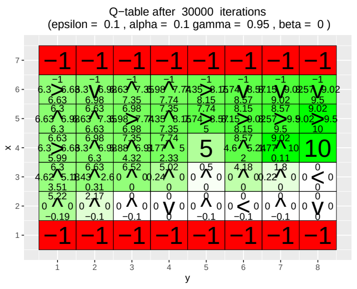


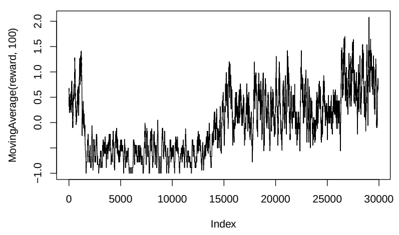


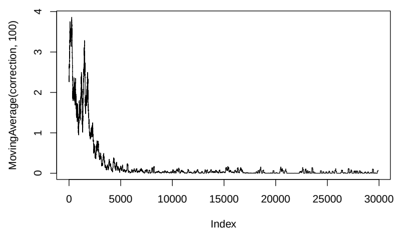








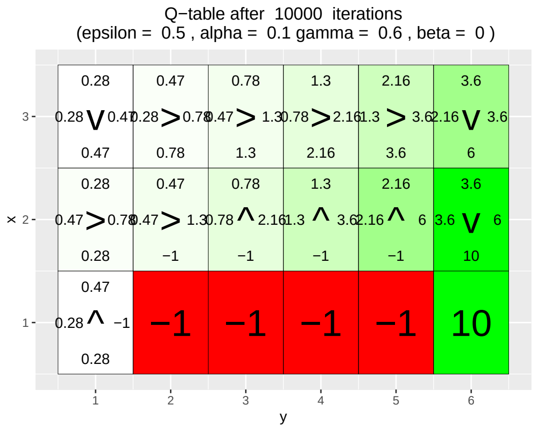


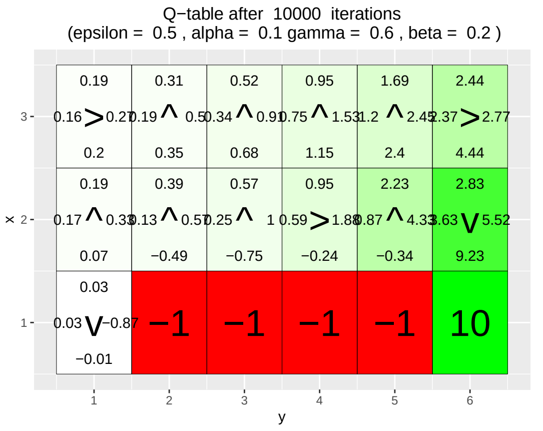


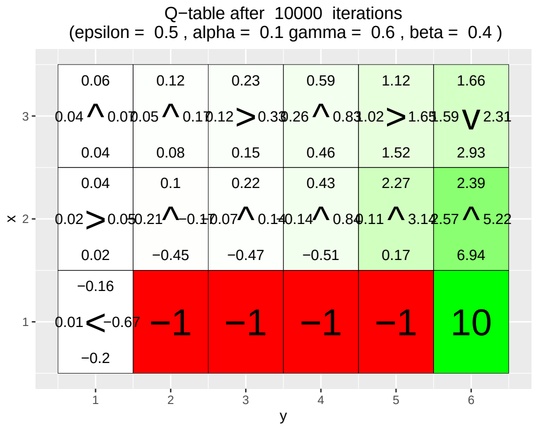
Gamma
A higher gamma means the agent looks for more long term reward, thus allowing it to find the 10 more consistently, compared to the low gamma where close rewards are valued higher thus being happy with the 5.

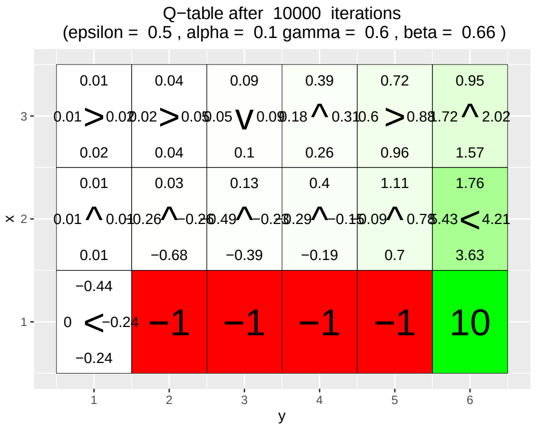
Epsilon
With a low value of epsilon, the agent will exploit the currently best policy rather than exploring and finding new, possibly better, policies. This means that if the agent finds the "5" reward first, it is likely to exploit that solution rather than exploring and finding its way to the "10" reward.

Environment C









Beta
A higher beta makes the agent more "paranoid" for random actions which makes it look for a more secure path where "bad" actions doesn't lead to too much damage. This can be seen in the path made for this environment where the agent chooses to walk as far as possible from the red areas.

REINFORCE

Environment D

Has the agent learned a good policy?

The agent does learn policies that lead to the goal. However, it is obvious that it doesn't understand the problem. The probabilities of going into the goal from an adjacent square are sometimes very clsoe to zero and the agent may heavily favor taking one route over another even though they are equal in length

Could you have used the Q-learning algorithm to solve this task?
For the Q-learning algorithm to handle new different environments the state has to be described in relative terms I.E. we are (dX, dY) squares from the goal, rather than: we are in square (x,y). This solution would, I think, work very well in these very specific circumstances where we do not have any obstacles. Although that problem is simple enough you could easily just calculate an optimal solution. An interesting thing however is that if the obstacles/red areas always would stay the same, q-learning would at least teach the agent where not to walk but it would have no idea how to find the goal state assuming it always changes.

Environment E

Has the agent learned a good policy, and how does the results from environments D and E differ?
Since the goal states in the training data are all in the top position (x=4) in environment E, the agent is not taught to move down (downwards action never gives any benefit based on the training data). The validation data does not share this similarity, and the agent can not find goal states where downward motion is necessary. In environment D, the training data has more variety than in E, and the agent can learn to adapt to more situations.
train_goals (Environment E): c(4,1), c(4,2), c(4,3), c(4,4)

Contribution

The code used has been adapted form Erik Jareman's individual submission. The questions has been answered by the whole group during a meeting