TDDE15 Lab3 Q-learning

William Bergekrans

2021-09

Environment A

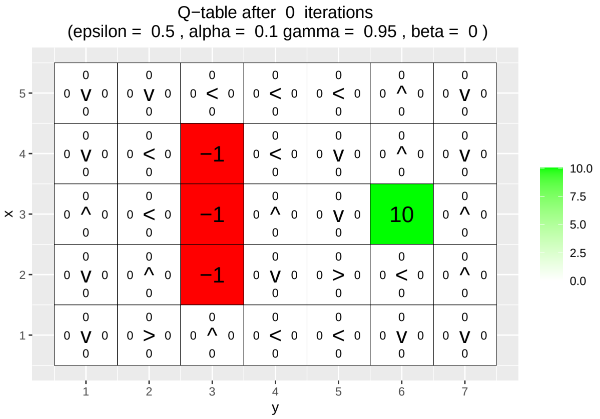
First I implement a function for epsilon-greedy and greedy policies for choosing what action to perform in a given state. The greedy policy always choose the action which has the highest expected reward while the epsilon-greedy policy chooses the best expected action with probability 1-epsilon. When it doesn't take the best action it chooses a random action to make the agent explore the world and find potential better routes. In order to break ties at random I use the function which.is.max() from the nnet package, which differs from the which.max() function because it breaks ties randomly.

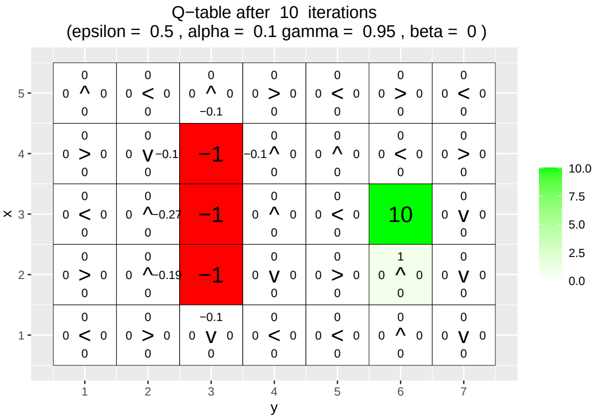
Next I implement the function q_learning, which is a function that should perform one episode of a Q-learning algorithm. The Q-learning algorithm first decides on a action with the help of the epsilon-greedy policy. The state that the agent ends up in is calculated with the given transition_model function. The function for updating the Q-table is:

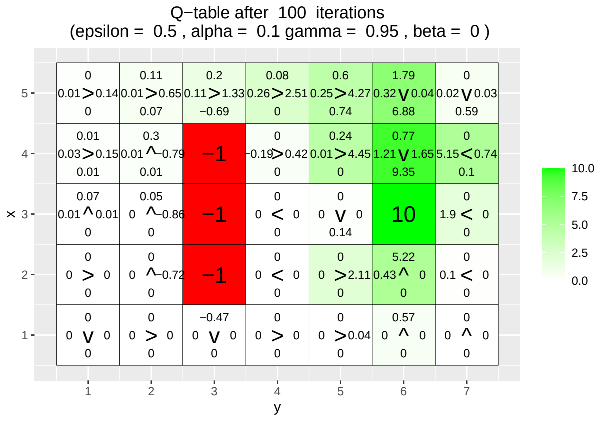
$$Q(S,A) = Q(S,A) + \alpha[R + \gamma max_a Q(S',a) - Q(S,A)]$$

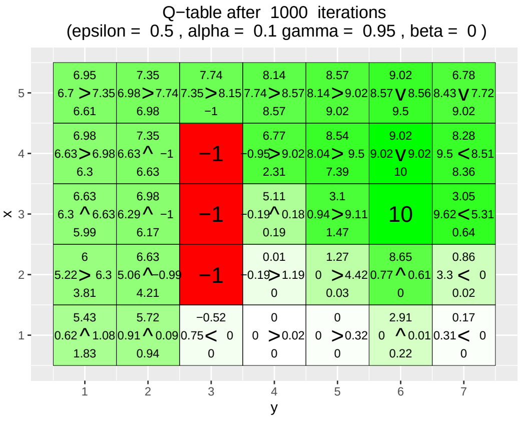
Where S is the current state, A is the chosen action, R is the reward given in the new state, S' is the new state, α is a learning parameter and γ is the discount factor for future rewards. This function is used for each iteration in the Q-learning algorithm to update the Q-table. The function is terminated when the agent reach a state with a reward that isn't zero.

When training an agent using the implemented function q_learning I get the following progress:

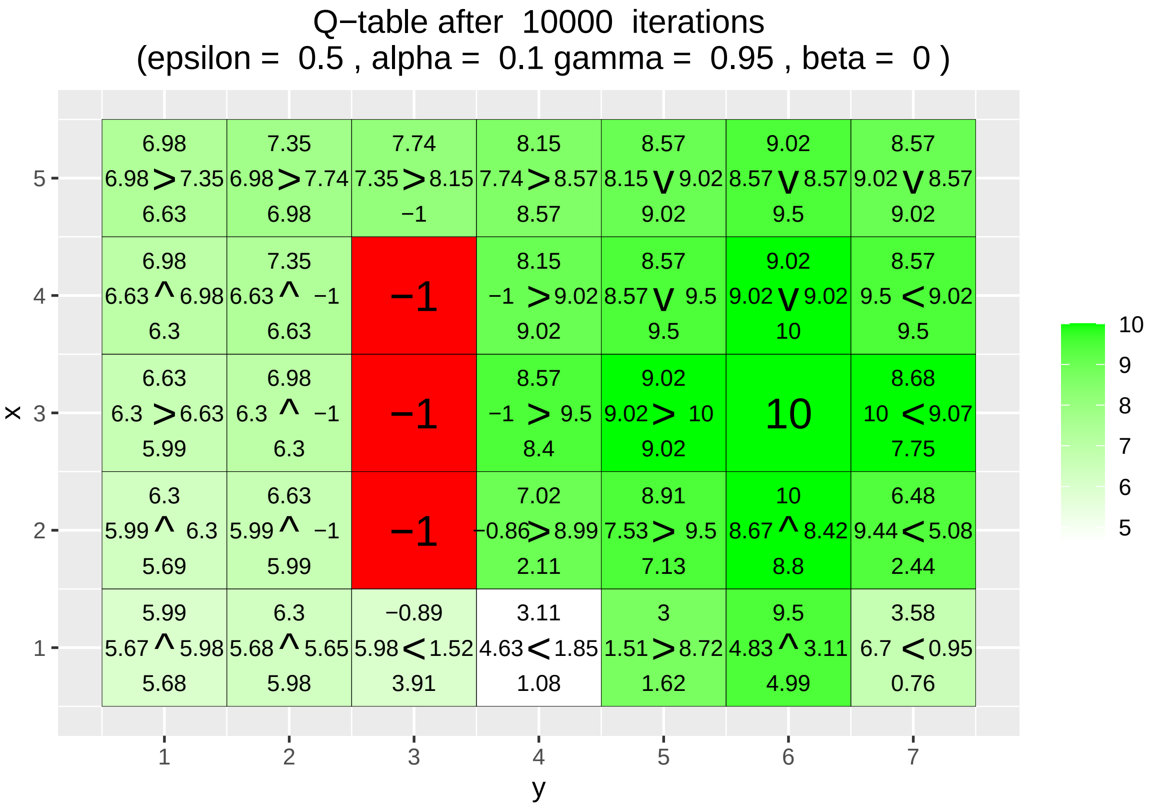








The final result after 10.000 iterations is:



After 10 iterations the agent has learnt to avoid most of the states which gives a negative reward. I would say this makes sense because before any terminal states are registered the agent explores the environment randomly, and the “bad” states happen to be closer than the “good” state.

After 10 000 iterations the policy prefers an action at each state. Even though the Q-learning algorithm with an epsilon-greedy policy eventually converges to the optimal policy I think it is clear that the current policy is not the best possible policy. One example of this is that the agent does not want to go through both gaps, it takes a longer route without any real gains. However, the Bellman equations which we enforce in the Q-learning algorithm will always be optimal compared to earlier iterations. The solution after 10 000 iterations is the best policy the agent has explored.

The model failed to detect that there are two paths that lead to the positive reward. If the agent is located in the path above the negative rewards the agent wants to go left and take the other path. This problem occurs because The agent has explored the bottom paths much more than the top ones and the expected rewards are therefore higher (and chosen more frequently in the training). One way to counter this behavior is to explore more by increasing ϵ. The policy should also become better if training runs for more episodes as it will eventually correct this mistake.

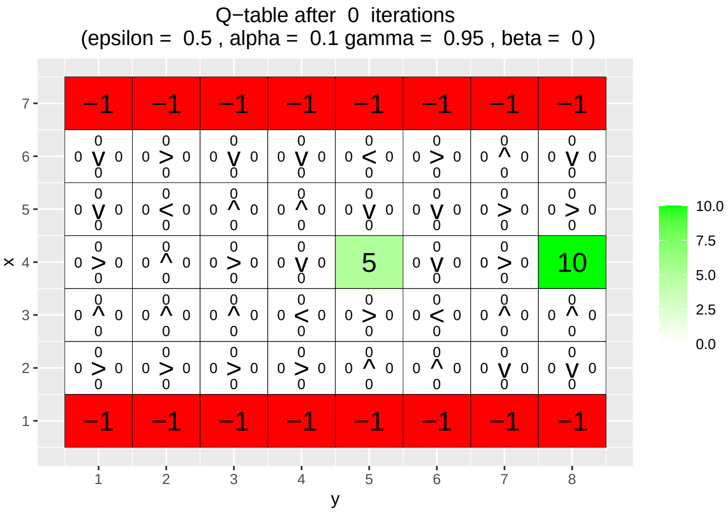
Code for the first part (environment A).

```
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.
  # Use which.is.max from nnet in order to break ties at random.
  return (which.is.max(q_table[x,y,]));
}

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.
  alpha <- runif(1,0,1)
  if (alpha > epsilon) {
    action <- GreedyPolicy(x,y)
  } else {
    action <- sample(1:4, 1)
  }
  return (action);
}


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.
  state = start_state
  episode_correction = 0

  repeat{
    # Decide what action to take
    action = EpsilonGreedyPolicy(state[1], state[2], epsilon)
    # Get the next state
    next_state = transition_model(state[1], state[2], action, beta)

    current_q = q_table[state[1], state[2], action]
    next_q = max(q_table[next_state[1], next_state[2], ])
    reward = reward_map[next_state[1], next_state[2]]

    # Calculate the temporal difference (td) correction.
    td_error = reward + gamma*next_q - current_q
    # Update the q-table.
    q_table[state[1], state[2], action] <<- current_q + alpha*(td_error)

    # Update current state to be next state.
    episode_correction = episode_correction + td_error
    state = next_state

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

Environment B

In this part the same q-learning algorithm as above is used for training different agents. The ϵ and γ parameters are changed between the different runs. β is set to 0 and α is 0.1.

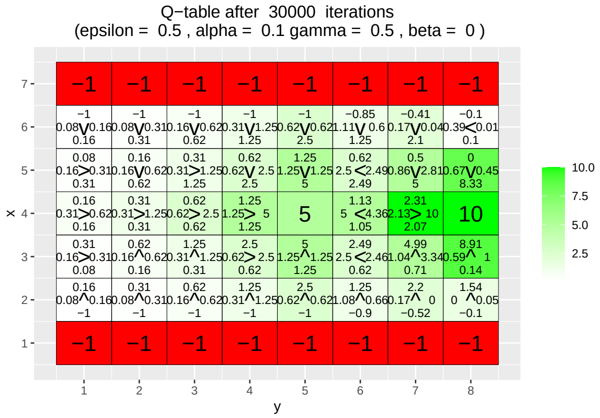
The agent always starts in state [4,1], the environment looks as follows:

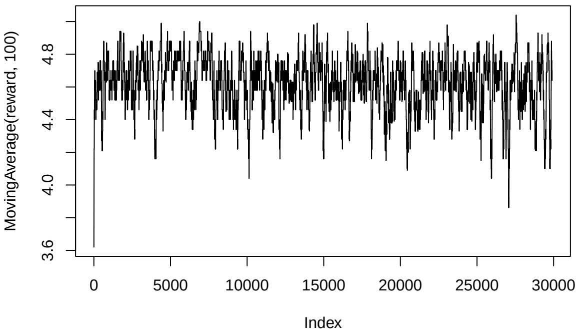


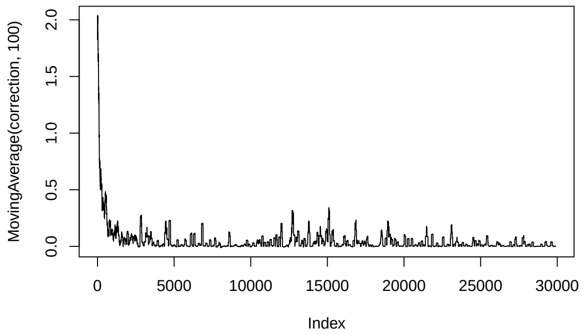
For each setting three different graphs are generated.

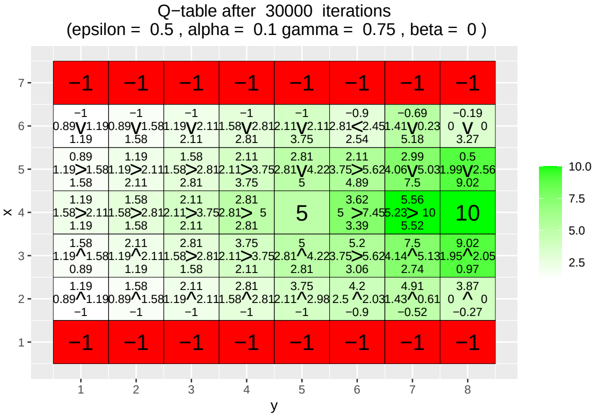
- The environment with Q-table and best action.
- The reward given.
- How much correction that was carried out.

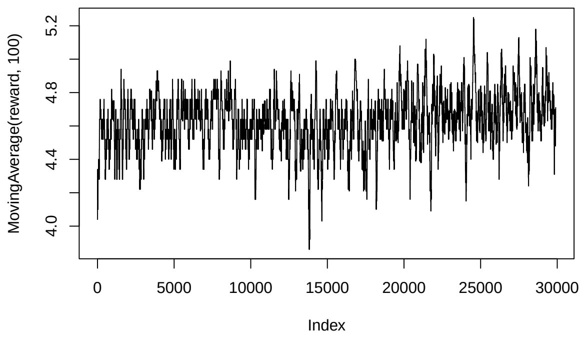
The results are:

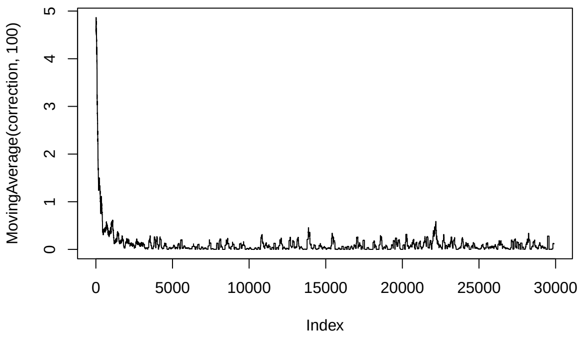


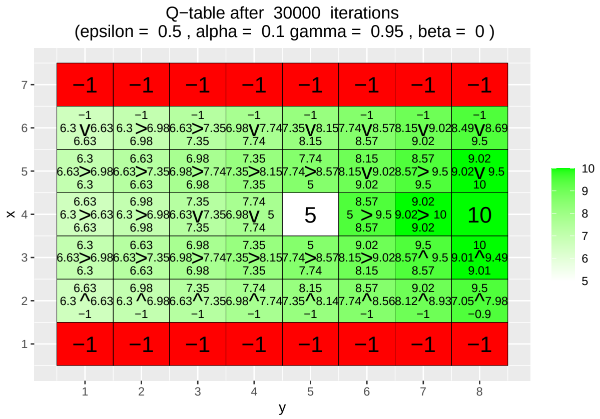


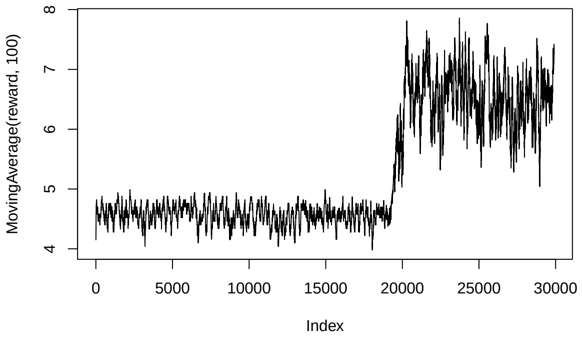


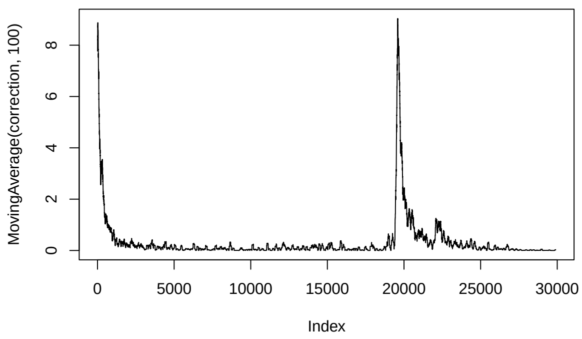


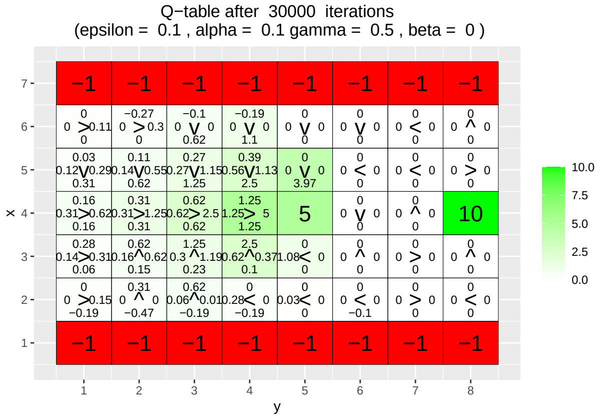

MovingAverage(reward, 100)
4.0
4.4
4.8
5.2
0
5000
10000
15000
20000
25000
30000
Index

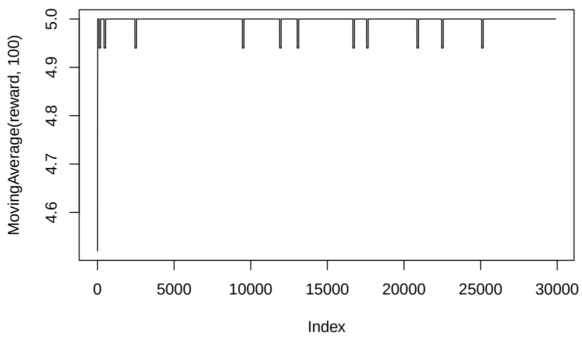

MovingAverage(correction, 100)
0
1
2
3
4
5
0
5000
10000
15000
20000
25000
30000
Index

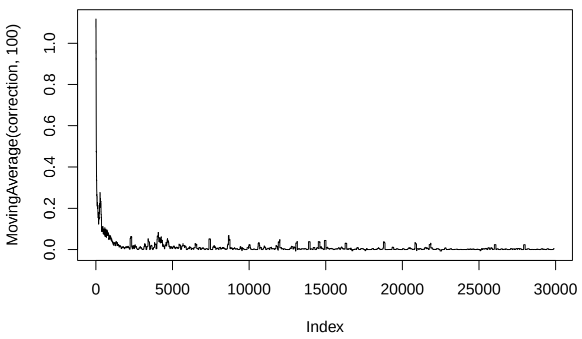

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
−1 −1 −1 −1 −1 −1 −1 −1
−1 −1 −1 −1 −1 −1 −1 −1
6.3 V6.63 6.3 >6.98 6.63>7.35 6.98>7.74 7.35V8.15 7.74V8.57 8.15V9.02 8.49V8.69
6.63 6.98 7.35 7.74 8.15 8.57 9.02 9.5
6.3 6.63 6.98 7.35 7.74 8.15 8.57 9.02
6.63>6.98 6.63>7.35 6.98>7.74 7.35>8.15 7.74>8.57 8.15V9.02 8.57> 9.5 9.02V 9.5
6.3 6.63 6.98 7.35 5 9.02 9.5 10
6.63 6.98 7.35 7.74 8.57 9.02
6.3 >6.63 6.3 >6.98 6.63V7.35 6.98V 5 5 5 > 9.5 9.02> 10 10
6.63 6.98 7.35 7.74 8.57 9.02
6.3 6.63 6.98 7.35 5 9.02 9.5 10
6.63>6.98 6.63>7.35 6.98>7.74 7.35>8.15 7.74>8.57 8.15>9.02 8.57^ 9.5 9.01^9.49
6.3 6.63 6.98 7.35 7.74 8.15 8.57 9.01
6.63 6.98 7.35 7.74 8.15 8.57 9.02 9.5
6.3 ^6.63 6.3 ^6.98 6.63^7.35 6.98^7.74 7.35^8.14 7.74^8.56 8.12^8.93 7.05^7.98
−1 −1 −1 −1 −1 −1 −1 −0.9
−1 −1 −1 −1 −1 −1 −1 −1
x
7
6
5
4
3
2
1
y
1
2
3
4
5
6
7
8
10
9
8
7
6
5

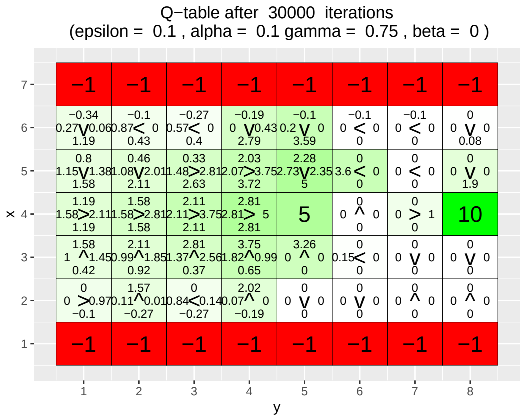

MovingAverage(reward, 100)
4
5
6
7
8
0
5000
10000
15000
20000
25000
30000
Index

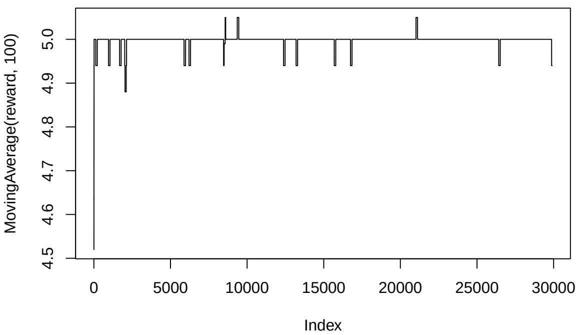

MovingAverage(correction, 100)
0
2
4
6
8
0
5000
10000
15000
20000
25000
30000
Index

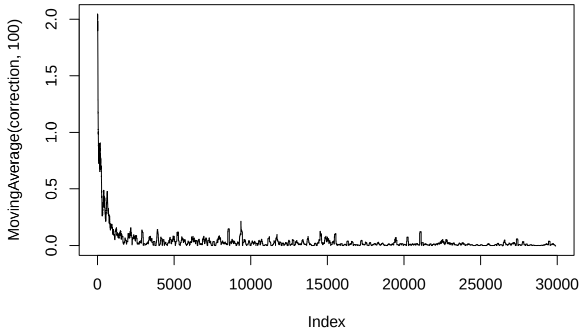

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
−1 −1 −1 −1 −1 −1 −1 −1
0 −0.27 −0.1 −0.19 0 0 0 0
0 >0.11 0 >0.3 0 V 0 0 V 0 0 V 0 0 V 0 0 < 0 0 ^ 0
0 0 0.62 1.1 0 0 0 0
0.03 0.11 0.27 0.39 0 0 0 0
0.12V0.29 0.14V0.55 0.27V1.15 0.56V1.13 0 V 0 0 < 0 0 < 0 0 > 0
0.31 0.62 1.25 2.5 3.97 0 0 0
0.16 0.31 0.62 1.25 0 0
0.31>0.62 0.31>1.25 0.62> 2.5 1.25> 5 5 0 V 0 0 ^ 0 10
0.16 0.31 0.62 1.25 0 0
0.28 0.62 1.25 2.5 0 0 0 0
0.14>0.31 0.16^0.62 0.3 ^1.19 0.62^0.37 1.08< 0 0 ^ 0 0 > 0 0 ^ 0
0.06 0.15 0.23 0.1 0 0 0 0
0 0.31 0.62 0 0 0 0 0
0 >0.15 0 ^ 0 0.06^0.01 0.28< 0 0.03< 0 0 < 0 0 > 0 0 < 0
−0.19 −0.47 −0.19 −0.19 0 −0.1 0 0
−1 −1 −1 −1 −1 −1 −1 −1
x
7
6
5
4
3
2
1
y
1
2
3
4
5
6
7
8
10.0
7.5
5.0
2.5
0.0

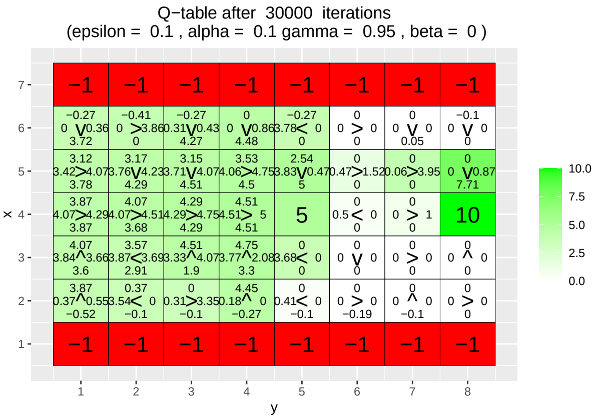

MovingAverage(reward, 100)
4.6
4.7
4.8
4.9
5.0
0
5000
10000
15000
20000
25000
30000
Index

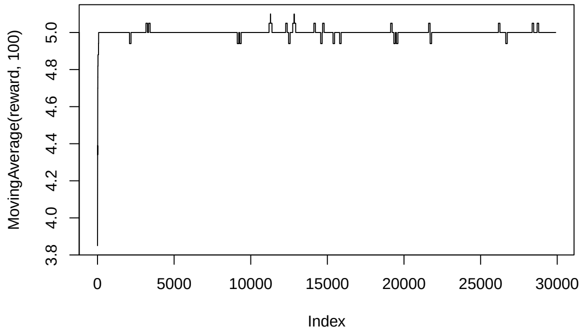

MovingAverage(correction, 100)
0.0
0.2
0.4
0.6
0.8
1.0
0
5000
10000
15000
20000
25000
30000
Index

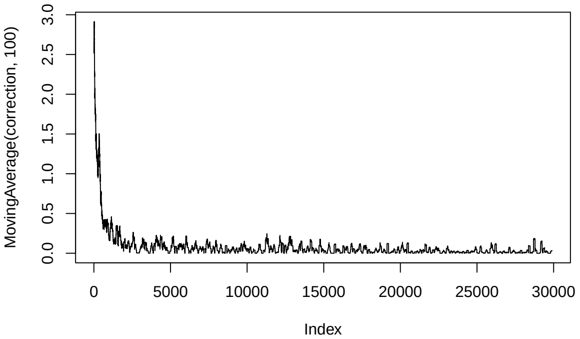












For this environment the best performance is received when $\epsilon = 0.5$ and $\gamma = 0.95$. In the two related graphs for the reward and correction averages what happened. The agent learnt to find 5 very quickly, after 15000 iterations the agent picked up on the better reward 10 and made a lot of corrections to the q-table. The movingAverage reward also increased a lot. The reason I think this is the best is because the agent has explored all states and seem to have found an optimal policy (or close to). The average reward in the end is also the highest out of all our trained agents.

When γ is decreased to 0.75 there is already a big drop in performance. The agent fails to learn a route around the 5 and only choose reward 10 if it is to the right of reward 5. This happens because γ is the discount factor and a lower γ make future rewards worse than instant rewards. Compared to when the discount factor was higher the agent no longer thinks it is worth to wait for reward 10 if it is close to 5 on any other side than the right, where it still chose to take one action extra for a higher reward. The average reward stays constant around 4.5 and there is very little correction after the first 2000 iterations. This behavior is intensified even more when γ is decreased to 0.5. This discount factor is environment B makes the agent go to it's closest reward. However if reward 10 was changed to 30 it would still be worth to wait another time step if standing in state [4,7].

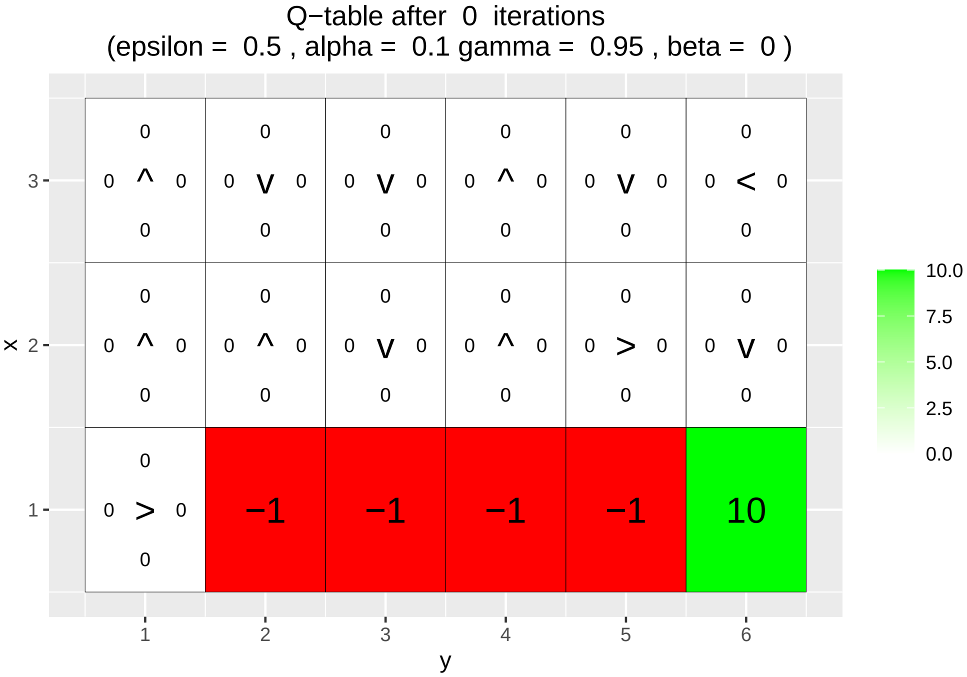
ϵ is the probability that the agent chooses an action randomly compared to taking the greedy action. The agents with an ϵ set to 0.1 all have a lot of states without any expected returns. A lower γ makes it even worse. When $\gamma = 0.5$ and $\epsilon = 0.1$ the agent have only updated a few values to the right of reward 5. Most states to the left of 5 have also been visited to few times for the agent to reach a good policy. When γ increases the agents become a bit better, but they still doesn't explore the space to the right of reward 5. It is clear that exploration is very important in order for the agent to learn a good policy. If an ϵ equal to 0.1 is to be used many more iterations are needed to reach an acceptable policy.

To summarize, the discount factor γ influences the optimal path and makes the agent prefer shorter routes, which can be both good and bad depending on the specific situation and environment. The exploration parameter ϵ greatly influences the agent's ability to learn its environment. Exploration is obviously very important and in the examples with a smaller ϵ the performance was much worse compared to when ϵ was bigger.

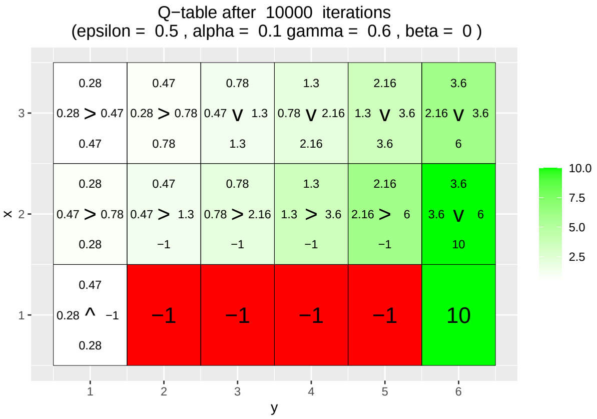
Environment C

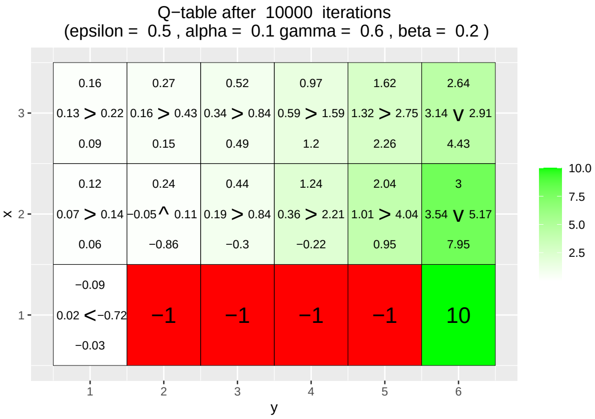
The β parameter is the probability that the chosen action is performed and therefore that the agent ends up in the intended state. I higher β adds uncertainty to the agent and actions next to bad rewards are riskier. I shall now explore what the agent learns for different β-values.

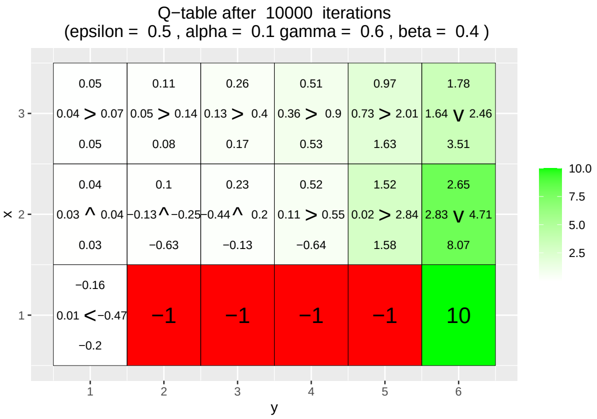
The new environment is:

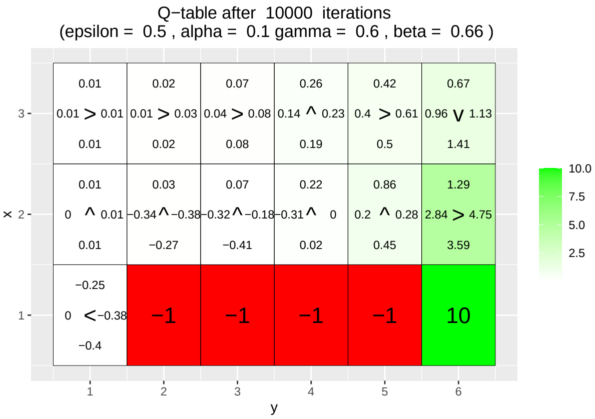


What the agent has learnt for different β configurations:


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0)


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.4)


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.66)

When $\beta = 0$ the agent always chooses an action on the shortest path to reward 10. There is no risk in doing this because the agent will always end up in the intended state after an action. When β increases the agent becomes more risk-averse. For lower values of β the agent starts to go as far away from negative rewards as it can, but only for the states furthest away from reward 10. When β is 0.66 most actions point away from the negative rewards because we are more likely to go right or left than up if "up" is the chosen action.

The agent does seem to do a good job of finding the best policy and compensating for uncertainties in the effect of actions.

Environment D

In this environment the goal positions for learning are randomly distributed across the grid. The resulting policy seem to do a great job, because after 5000 iterations the agent always chooses an action on the closest path to the validation goals. This is the same for all the different validation goals which means that the agent can adapt which is desired.

The main advantage with REINFORCE compared with Q-learning is that policies can be created for states that the agent hasn't visited. In this problem the goal parameters are also state variables, the Q-learning algorithm would not be able to create policies for states (goals) it has not visited. To use Q-learning for this task all goals need to be in the training set in order to find the best policy.

Enviroment E

In the environment the goal positions for learning are all located in the top row of the grid. Here the agent clearly failed to learn a good policy, as its primary focus is to go to the top row and not to the goal. The agent tries to take the new goal into account but only a few states seem to switch the most likely action in a good direction. For most goals the most probable action lead only to the top row, and it does not matter where the starting position is.

The results are very different between environment D end E. There are two big differences between the runs:

- All training goals being located approximate to each other vs. randomly scattered.
- 4 vs. 8 training goals.

I believe that the biggest factor is the first of the two. Even though the REINFORCE algorithm does try to set a policy for every state the model gives to much weight to the top part of the grid-world in environment E.

When the goal position is [1,1] the preferred action for the entire world in E is switched to ‘left’. Still, even in that trial there is not a single state that chooses an action that ends up in the goal state.

The training of the model seems to rely on a good distribution of the goal states across the grid in order to update the policies for unknowns states in a good way.