Advanced Machine Learning: Lab 3 - Individual Report

Shashi Nagarajan (shana299)

```
set.seed(42)
library(ggplot2)
```

1

The implementation of the Q learning algorithm was completed as shown below:

```
arrows <- c("^", ">", "v", "<")
action_deltas <- list(c(1,0), # up
                      c(0,1), # right
                      c(-1,0), # down
                      c(0,-1)) # left

vis_environment <- function(iterations=0, epsilon = 0.5, alpha = 0.1, gamma = 0.95, beta = 0){

  # Visualize an environment with rewards.
  # Q-values for all actions are displayed on the edges of each tile.
  # The (greedy) policy for each state is also displayed.
  #
  # Args:
  #   iterations, epsilon, alpha, gamma, beta (optional): for the figure title.
  #   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.
  #   H, W (global variables): environment dimensions.

  df <- expand.grid(x=1:H,y=1:W)
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,1],NA),df$x,df$y)
  df$val1 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,2],NA),df$x,df$y)
  df$val2 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,3],NA),df$x,df$y)
  df$val3 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,4],NA),df$x,df$y)
  df$val4 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y)
    ifelse(reward_map[x,y] == 0,arrows[GreedyPolicy(x,y)],reward_map[x,y]),df$x,df$y)
  df$val5 <- as.vector(foo)
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,max(q_table[x,y,]),
                                     ifelse(reward_map[x,y]<0,NA,reward_map[x,y])),df$x,df$y)
  df$val6 <- as.vector(foo)

  print(ggplot(df,aes(x = y,y = x)) +
          scale_fill_gradient(low = "white", high = "green", na.value = "red", name = "") +
          geom_tile(aes(fill=val6)) +
```

```
          geom_text(aes(label = val1),size = 4,nudge_y = .35,na.rm = TRUE) +
          geom_text(aes(label = val2),size = 4,nudge_x = .35,na.rm = TRUE) +
          geom_text(aes(label = val3),size = 4,nudge_y = -.35,na.rm = TRUE) +
          geom_text(aes(label = val4),size = 4,nudge_x = -.35,na.rm = TRUE) +
          geom_text(aes(label = val5),size = 10) +
          geom_tile(fill = 'transparent', colour = 'black') +
          ggtitle(paste("Q-table after ",iterations," iterations\n",
                        "(epsilon = ",epsilon,", alpha = ",alpha,"gamma = ",gamma,", beta = ",beta,")")) +
          theme(plot.title = element_text(hjust = 0.5)) +
          scale_x_continuous(breaks = c(1:W),labels = c(1:W)) +
          scale_y_continuous(breaks = c(1:H),labels = c(1:H)))

}

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.
  possible_actions = 1:4

  if (x == 1) {

    possible_actions = setdiff(possible_actions, 3)

  }

  if (x == H) {

    possible_actions = setdiff(possible_actions, 1)

  }

  if (y == 1) {

    possible_actions = setdiff(possible_actions, 4)

  }

  if (y == W) {

    possible_actions = setdiff(possible_actions, 2)

  }

  optimal_actions = possible_actions[
```

```
    which(q_table[x, y, possible_actions] == max(q_table[x, y, possible_actions]))]

  if (length(optimal_actions) > 1) {

    return(sample(optimal_actions, 1,
                  prob = rep(1/length(optimal_actions), length(optimal_actions))))

  } else {

    return(optimal_actions)

  }

}

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.
  possible_actions = 1:4

  if (x == 1) {

    possible_actions = setdiff(possible_actions, 3)

  }

  if (x == H) {

    possible_actions = setdiff(possible_actions, 1)

  }

  if (y == 1) {

    possible_actions = setdiff(possible_actions, 4)

  }

  if (y == W) {

    possible_actions = setdiff(possible_actions, 2)

  }
```

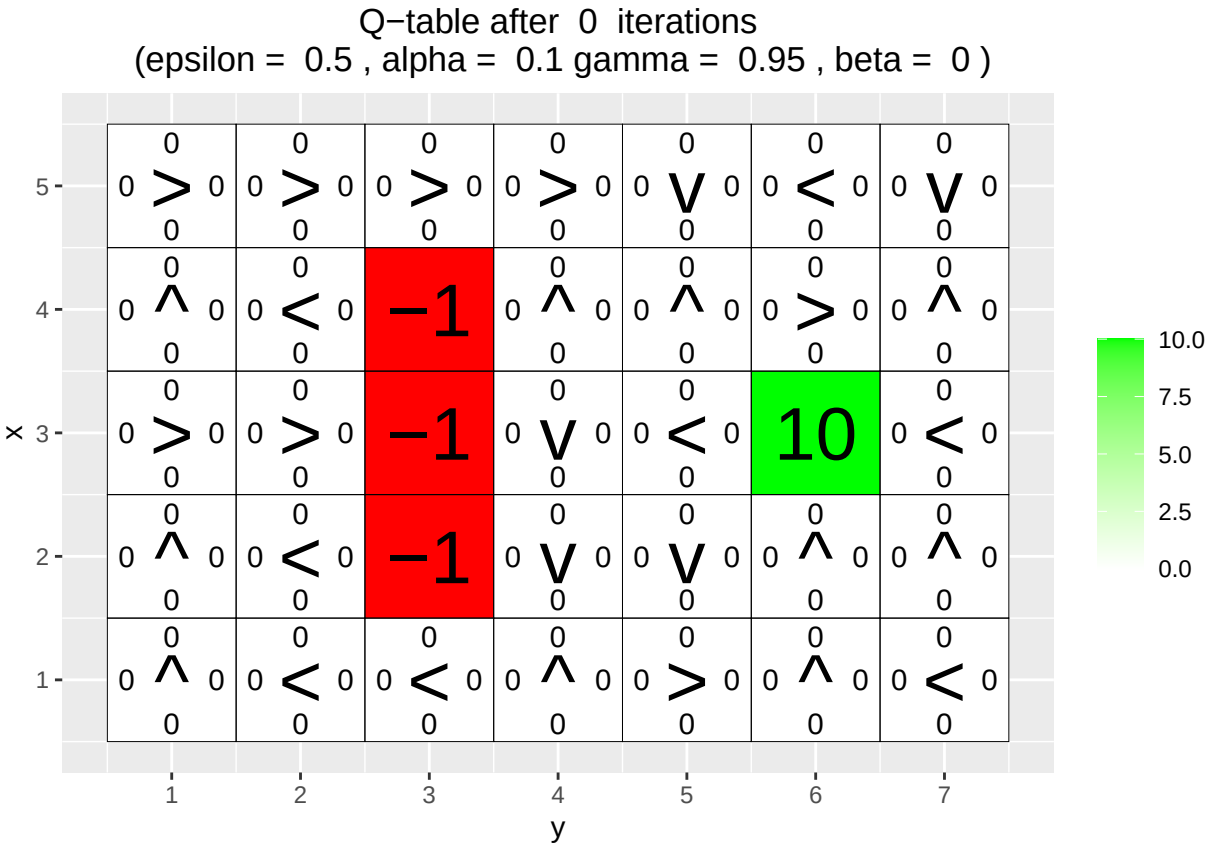
```
  optimal_actions = possible_actions[
    which(q_table[x, y, possible_actions] == max(q_table[x, y, possible_actions]))]

  if (length(optimal_actions) > 1) {

    optimal_action =
      sample(optimal_actions, 1, prob = rep(1/length(optimal_actions), length(optimal_actions)))

  } else {

    optimal_action = optimal_actions

  }

  other_actions = setdiff(possible_actions, optimal_action)

  if (length(other_actions) == 0) {

    return(optimal_action)

  } else {

    return(sample(c(optimal_action, other_actions), 1,
              prob = c(1 - epsilon, rep(epsilon/length(other_actions), length(other_actions)))))

  }

}

transition_model <- function(x, y, action, beta){

  # Computes the new state after given action is taken. The agent will follow the action
  # with probability (1-beta) and slip to the right or left with probability beta/2 each.
  #
  # Args:
  #   x, y: state coordinates.
  #   action: which action the agent takes (in {1,2,3,4}).
  #   beta: probability of the agent slipping to the side when trying to move.
  #   H, W (global variables): environment dimensions.
  #
  # Returns:
  #   The new state after the action has been taken.

  delta <- sample(-1:1, size = 1, prob = c(0.5*beta,1-beta,0.5*beta))
  final_action <- ((action + delta + 3) %% 4) + 1
  foo <- c(x,y) + unlist(action_deltas[final_action])
  foo <- pmax(c(1,1),pmin(foo,c(H,W)))

  return (foo)
}

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

```r
# 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.

current_state <- start_state # initialise
reward <- 0 # initialise
episode_correction <- 0 # initialise

repeat{
  # Follow policy, execute action, get reward.

  # follow policy to get action to execute
  if (epsilon > 0) {

    action <-
      EpsilonGreedyPolicy(current_state[1], current_state[2], epsilon)

  } else {

    action <- GreedyPolicy(current_state[1], current_state[2], epsilon)
  }

  # execute action
  new_state <-
    transition_model(current_state[1], current_state[2], action, beta)

  # get reward
  curr_action_reward <- reward_map[new_state[1], new_state[2]]
  curr_action_correction <- curr_action_reward +
    gamma*max(q_table[new_state[1], new_state[2], ]) -
    q_table[current_state[1], current_state[2], action]

  reward <- reward + curr_action_reward
  episode_correction <- episode_correction + curr_action_correction

  # Q-table update.
```

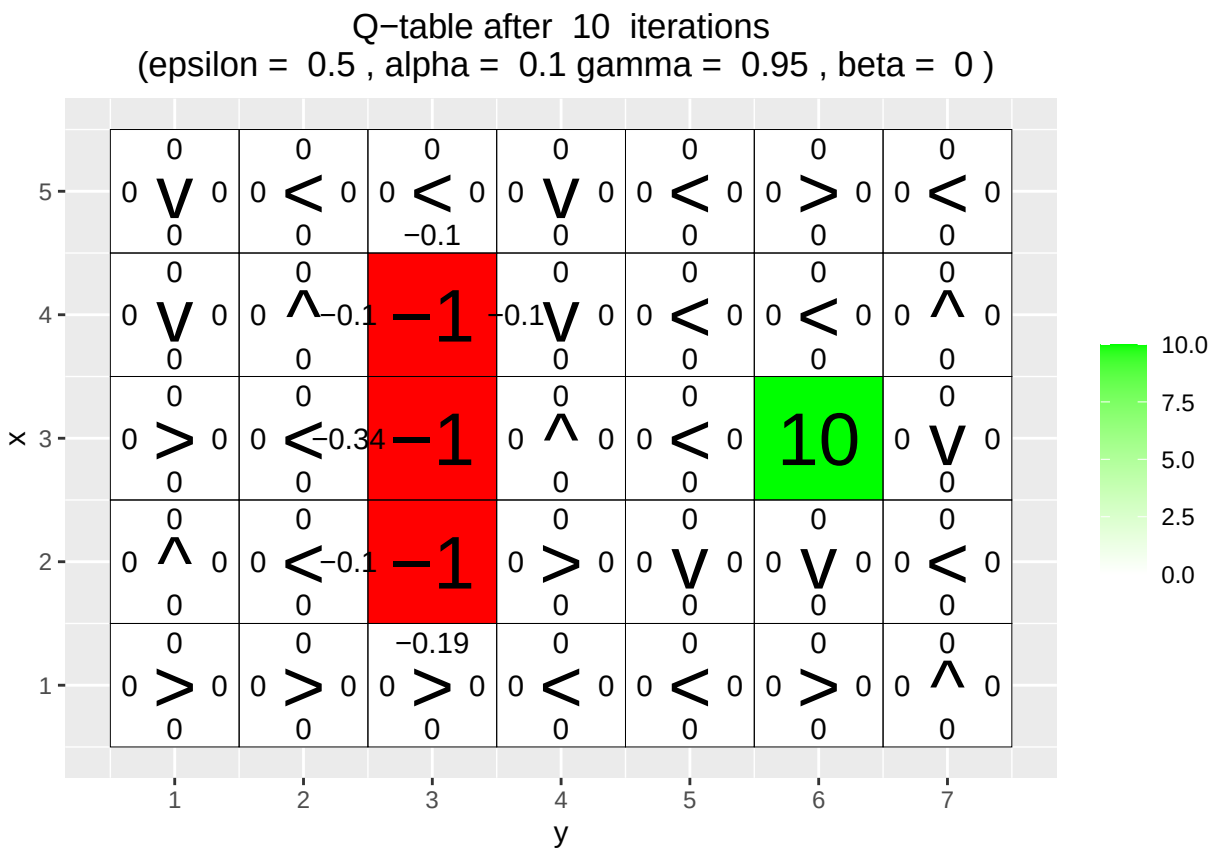
```
    q_table[current_state[1], current_state[2], action] <<-
      q_table[current_state[1], current_state[2], action] +
      alpha*(curr_action_correction)

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

    current_state <- new_state

  }

}
```

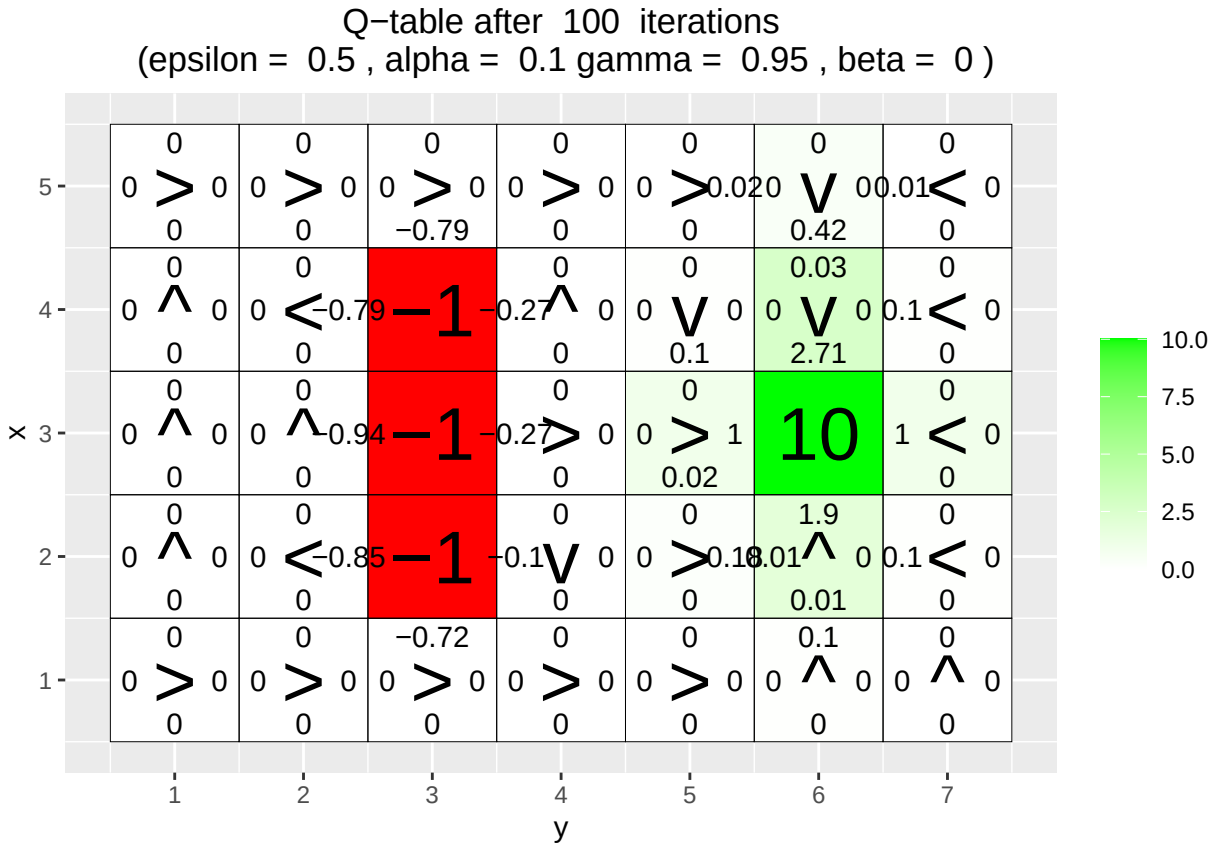
2

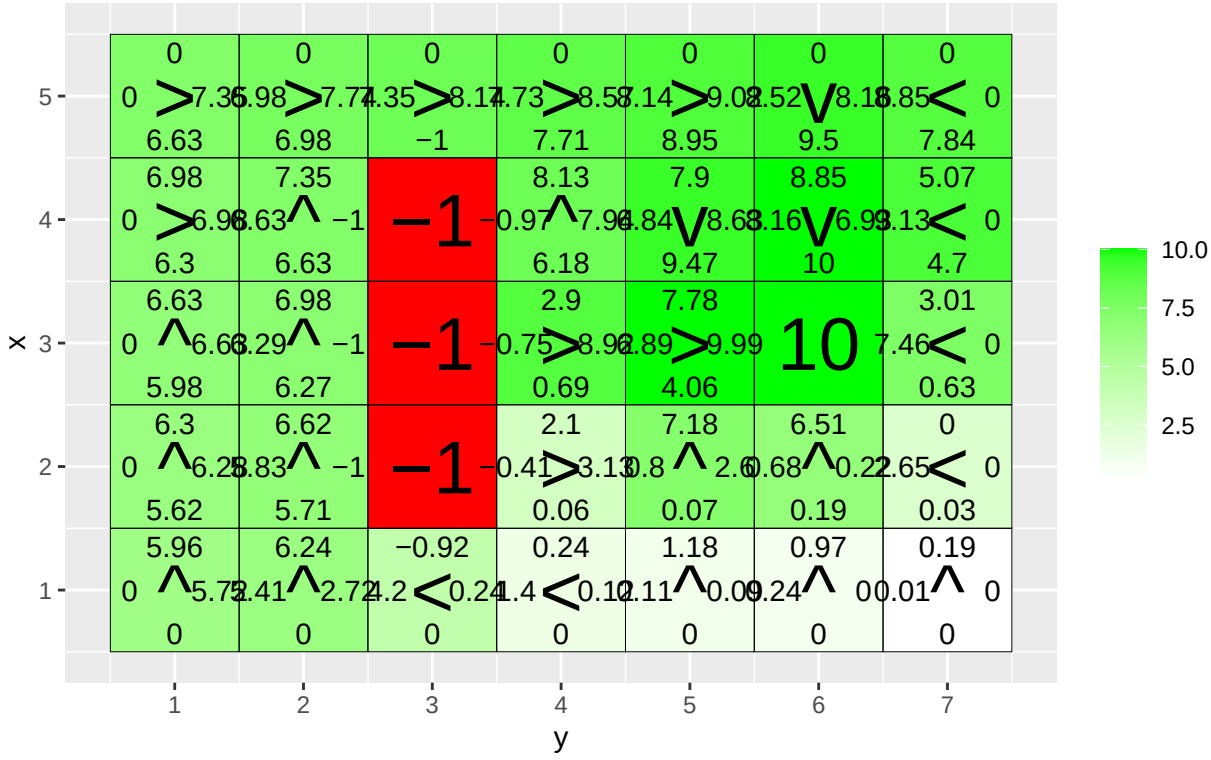
```
H <- 5
W <- 7

reward_map <- matrix(0, nrow = H, ncol = W)
reward_map[3,6] <- 10
reward_map[2:4,3] <- -1

q_table <- array(0,dim = c(H,W,4))

vis_environment()
```

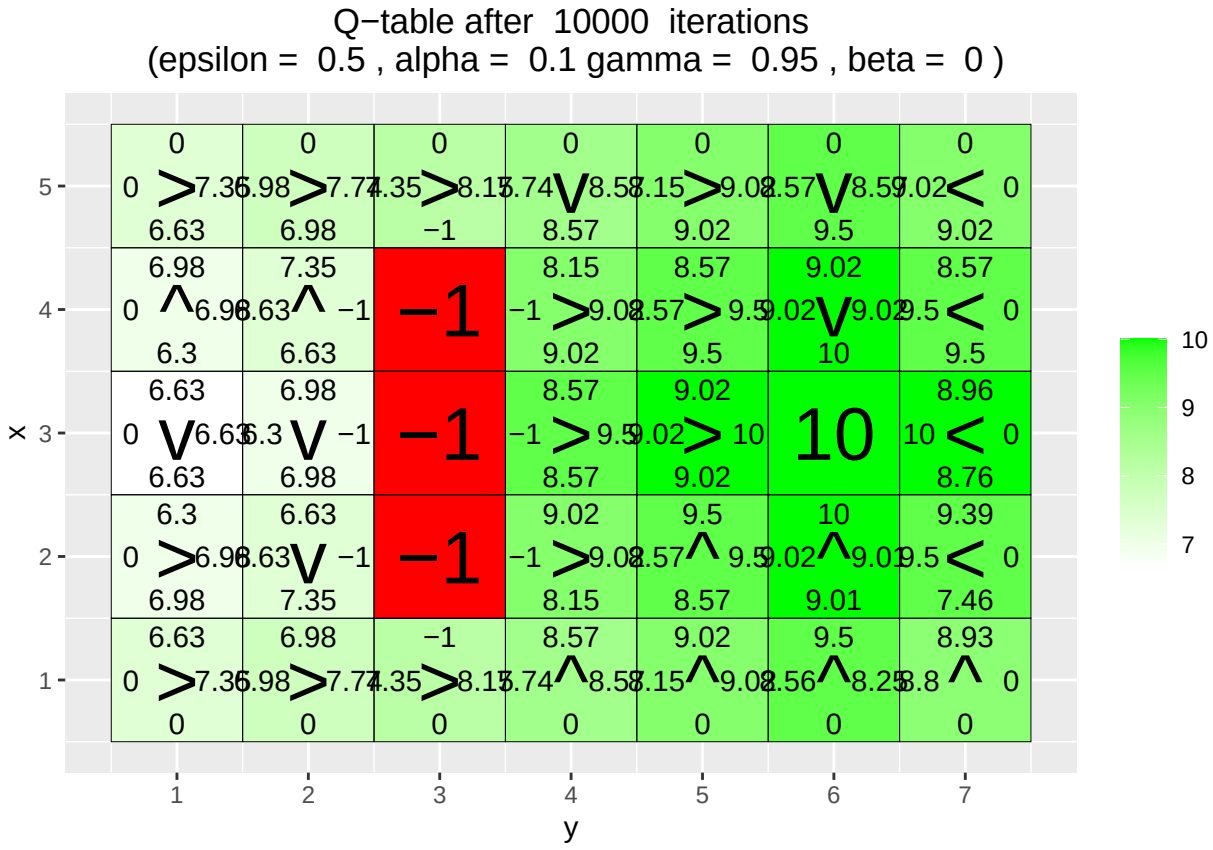


```
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))

  if(any(i==c(10,100,1000,10000)))
    vis_environment(i)
}
```




Q−table after 100 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0


Q−table after 1000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5



2.1.

Given that the start state was (3, 1), after ten iterations, one may conclude that the agent has indeed begun learning that the red terminal states (-1 reward states) are to be avoided

2.2.

Since β is 0 in this case (i.e. transitions are deterministic), the policy learnt can be deemed to be optimal. There are no loops, and all paths to the green terminal state (+10 reward state) from any start state are the shortest, if we account for the constraint that all such paths must avoid red terminal states.

If, for some reason, this weren't the case, we could choose to increase epsilon (or make epsilon dynamic, for e.g. large early on, and then small later on in the training stage) or increase the number of training iterations, so as to observe each state/action pair and update their corresponding q-value. Increasing the learning rate (α) and tuning the discount rate (γ) could have also helped.

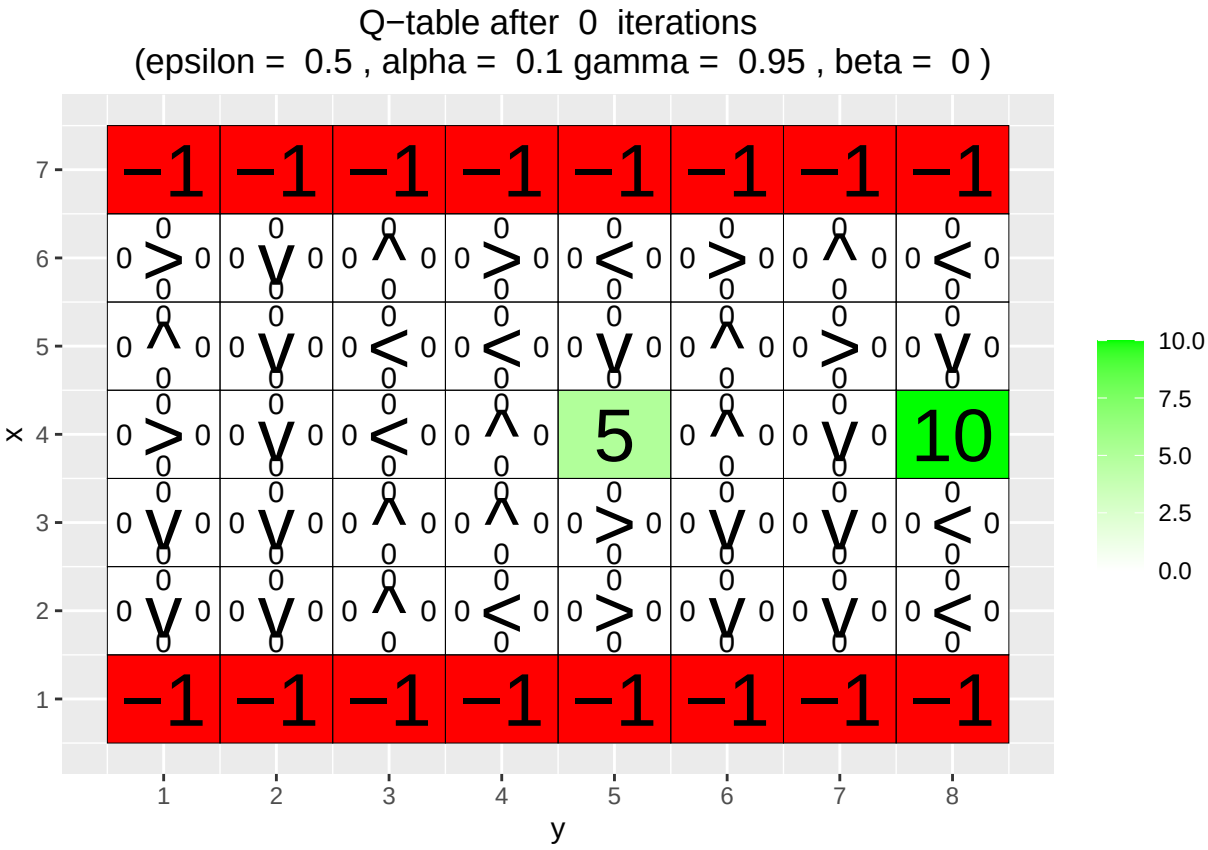
2.3.

Yes, learnt values in the Q-table do reflect the fact that there are multiple paths to get the positive reward. For instance, in state (3, 1), the q-value corresponding to any legal action (up, right or down) are equal, to 6.63, making all of those three actions equally favourable for the agent. Moreover, from the chart above, one can observe that each of those actions would set the agent on some legitimate path towards the green terminal state.

For starting states along the line x = 3, except those equivalent or adjacent to the green terminal state, the symmetry about the x = 3 line inherently makes for at least two paths costing the least number of steps to the green terminal state. If there should be unique paths from each starting state, this symmetry must be removed, and that can only be done by changing the reward scheme. Also, a non-zero γ could help the agent avoid paths near the red terminal states.

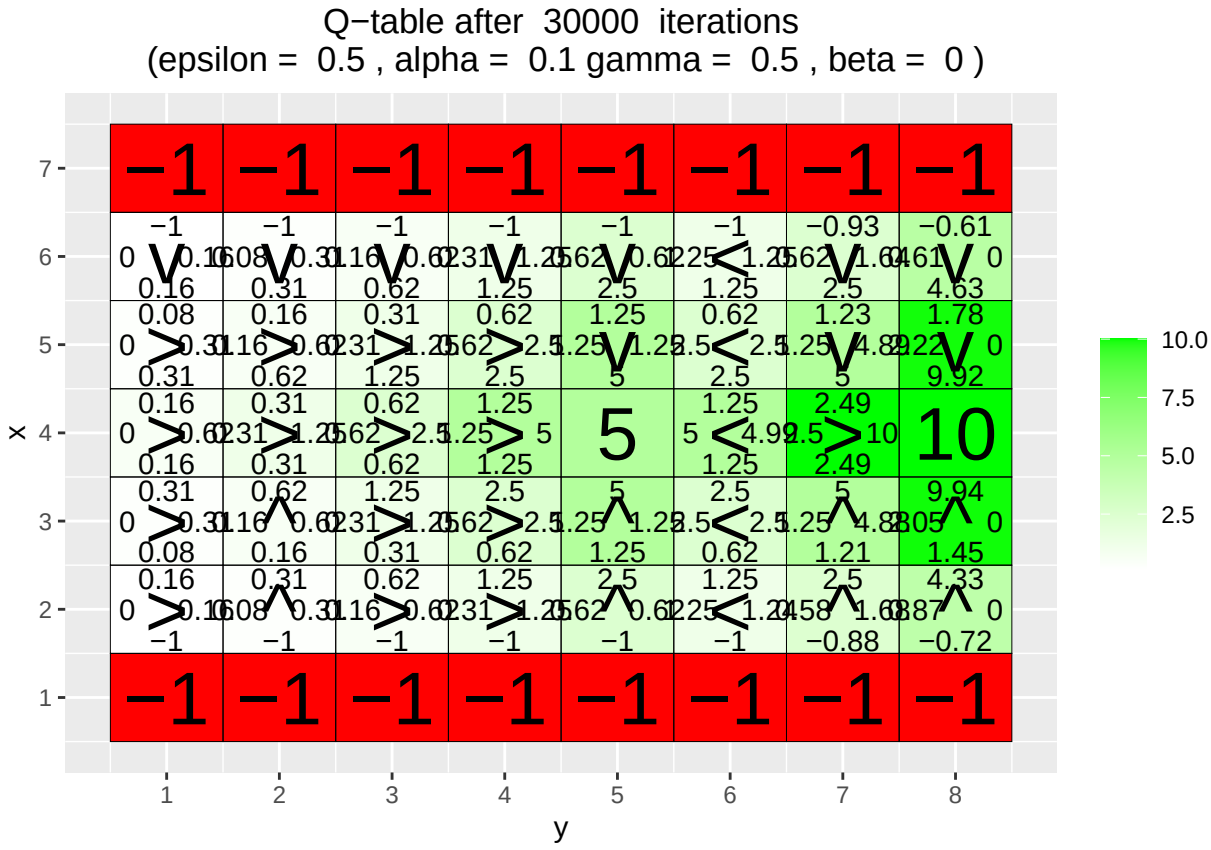
3

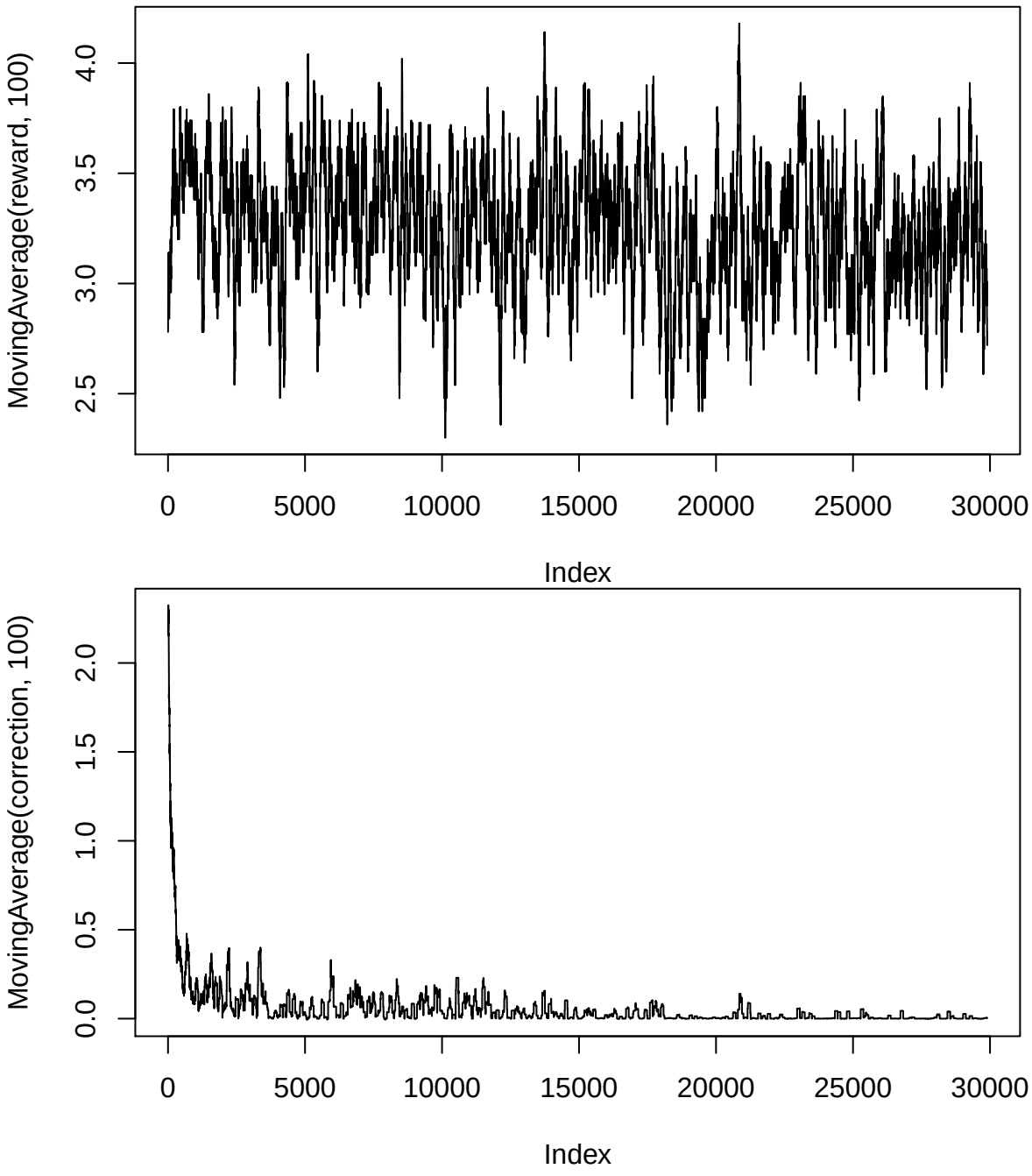
```
# Environment B (the effect of epsilon and gamma)

H <- 7
W <- 8

reward_map <- matrix(0, nrow = H, ncol = W)
reward_map[1,] <- -1
reward_map[7,] <- -1
reward_map[4,5] <- 5
reward_map[4,8] <- 10

q_table <- array(0,dim = c(H,W,4))

vis_environment()
```

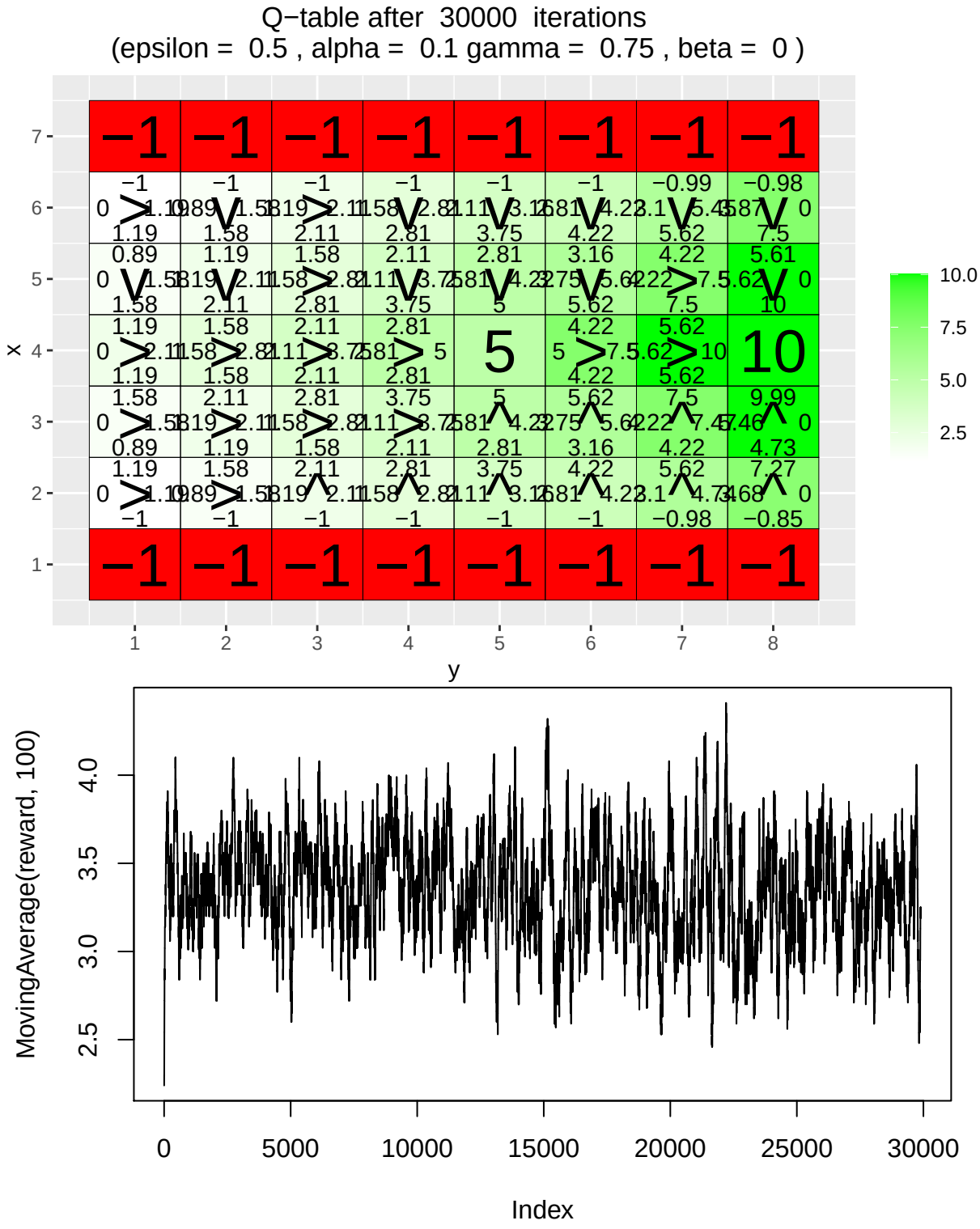


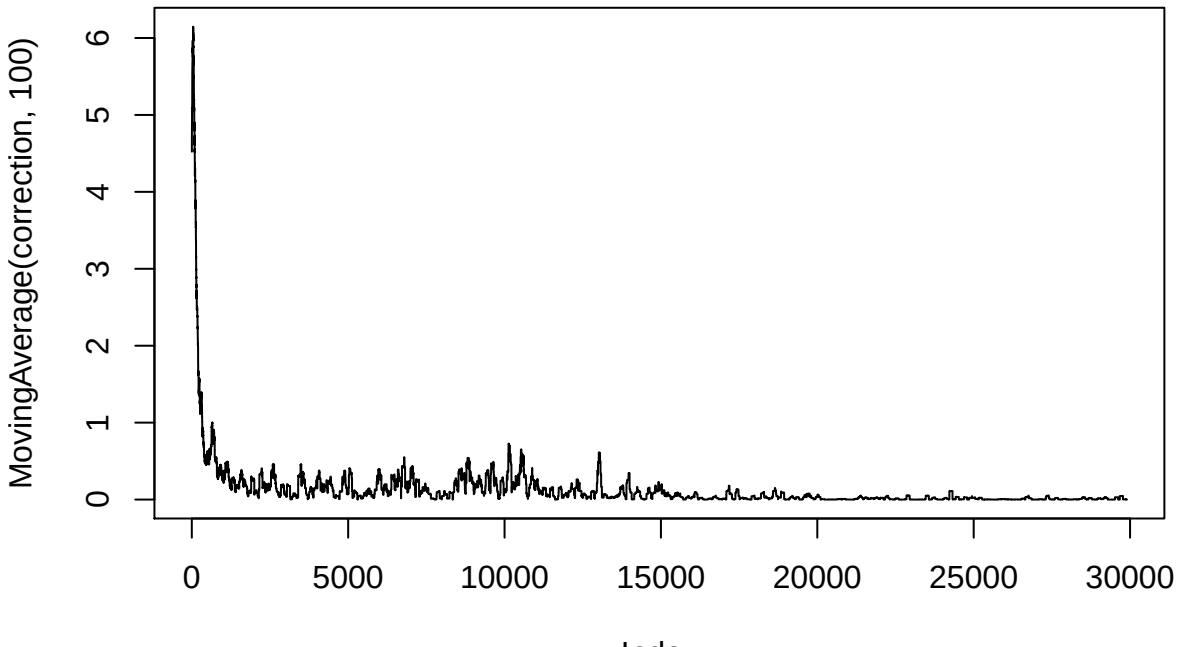

```
MovingAverage <- function(x, n){

  cx <- c(0,cumsum(x))
  rsum <- (cx[(n+1):length(cx)] - cx[1:(length(cx) - n)]) / n

  return (rsum)
}

for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
```

```
  reward <- NULL
  correction <- NULL

  for(i in 1:30000){
    foo <- q_learning(gamma = j, start_state = c(4,1))
    reward <- c(reward,foo[1])
    correction <- c(correction,foo[2])
  }

  vis_environment(i, gamma = j)
  plot(MovingAverage(reward,100),type = "l")
  plot(MovingAverage(correction,100),type = "l")
}
```

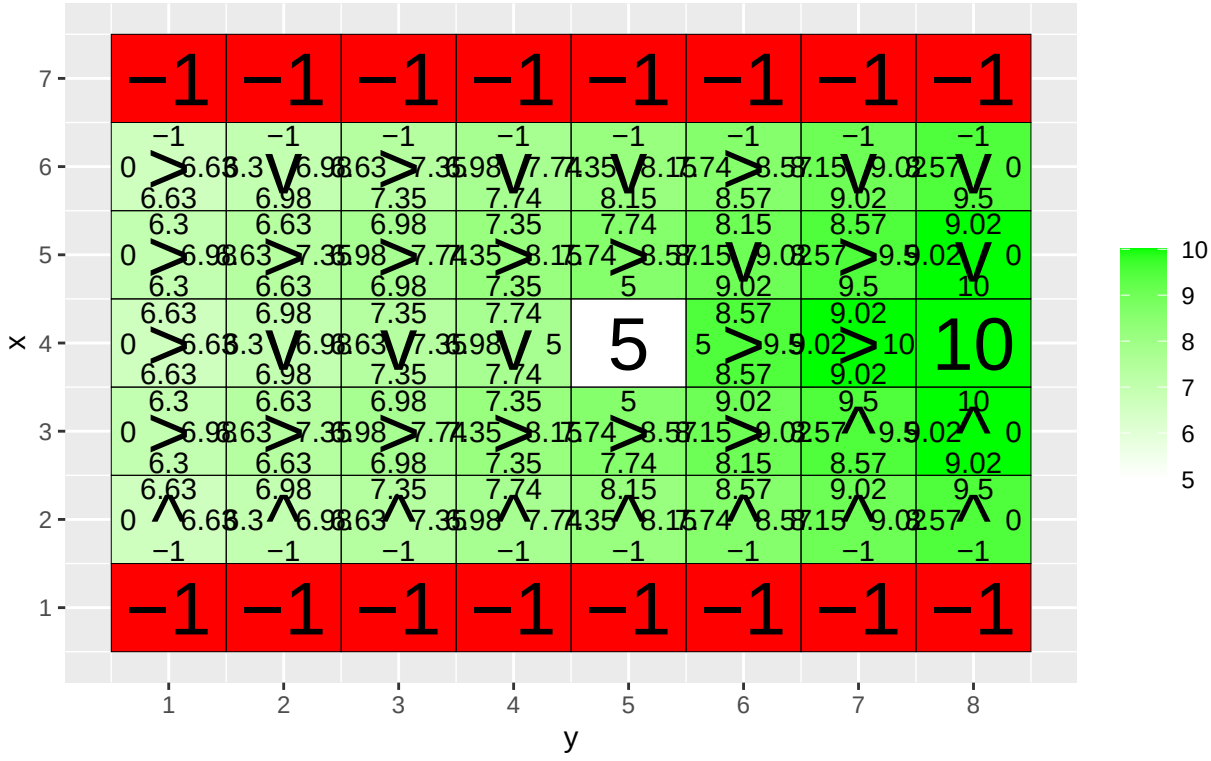


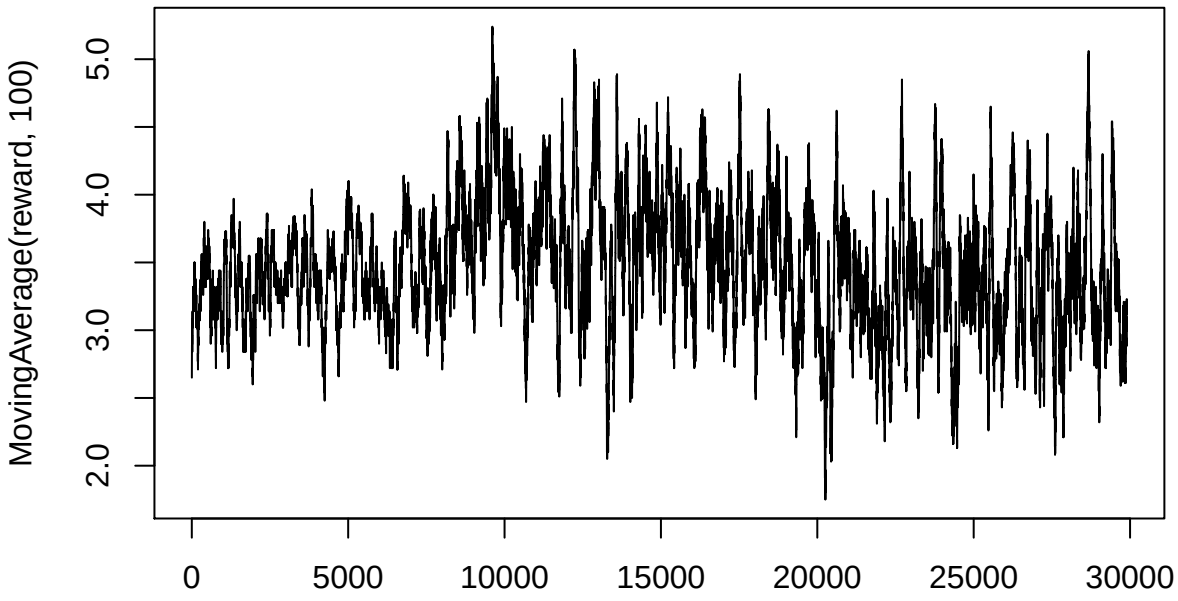

MovingAverage(reward, 100)
2.5
3.0
3.5
4.0
0
5000
10000
15000
20000
25000
30000
Index
MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index

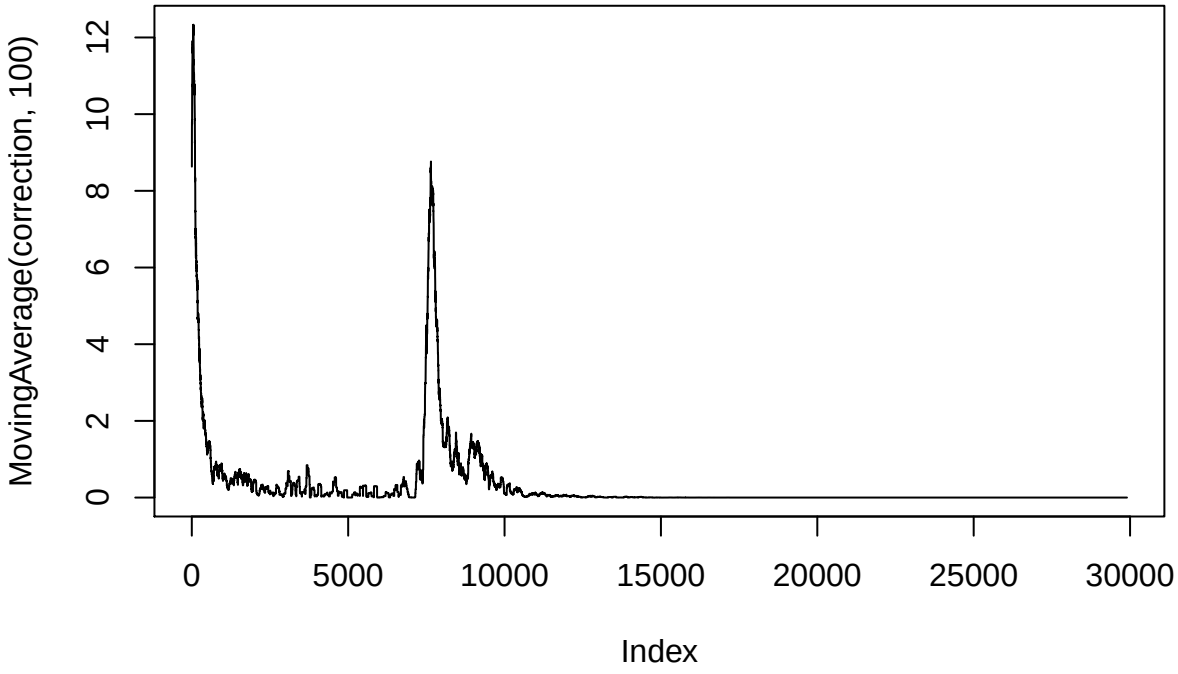

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5
MovingAverage(reward, 100)
Index
0
5000
10000
15000
20000
25000
30000
2.5
3.0
3.5
4.0



Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)

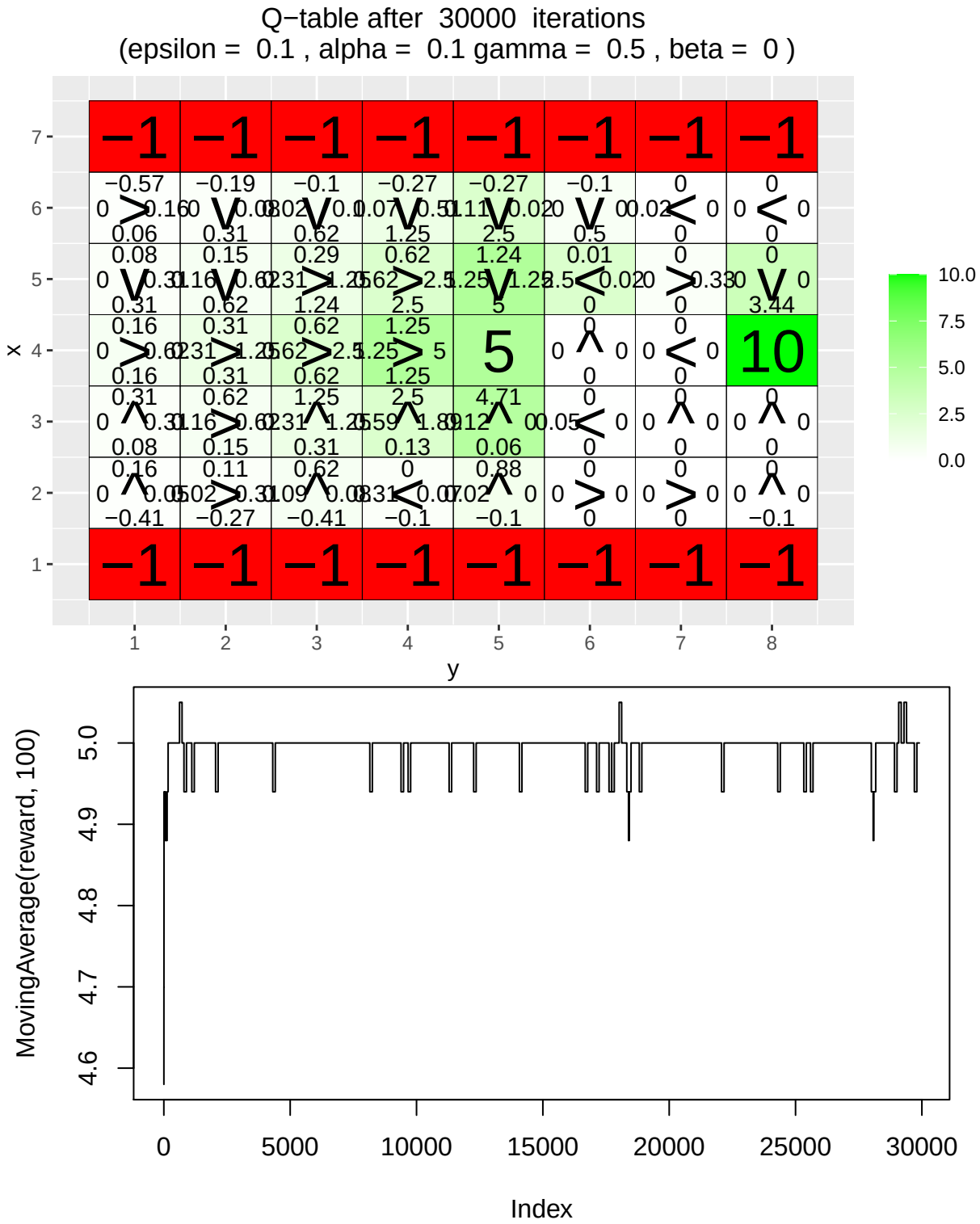


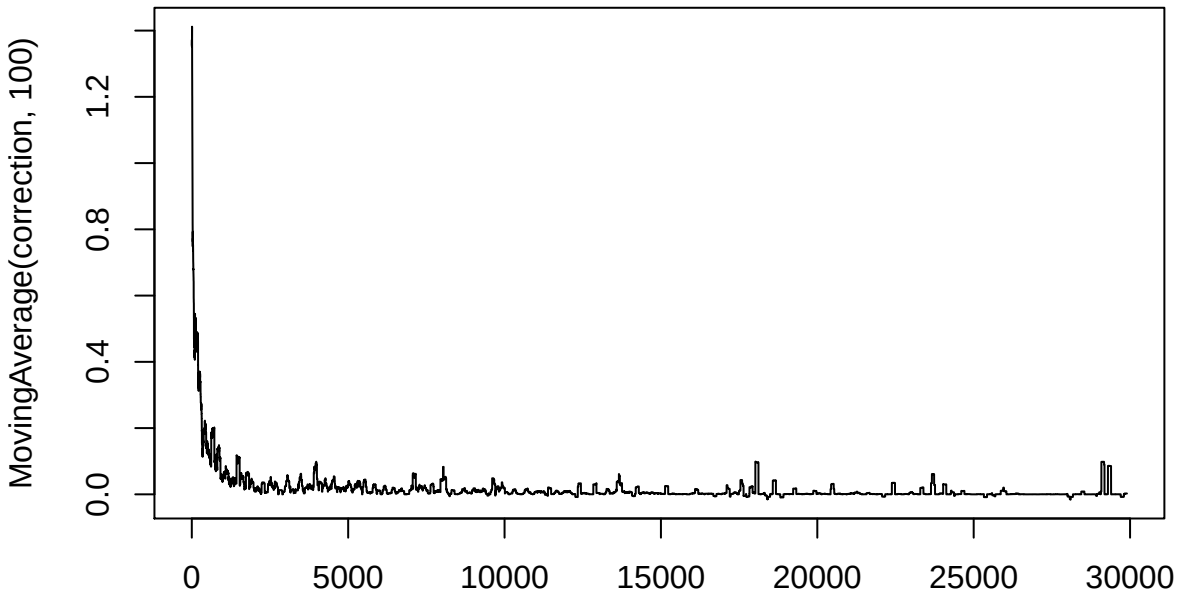


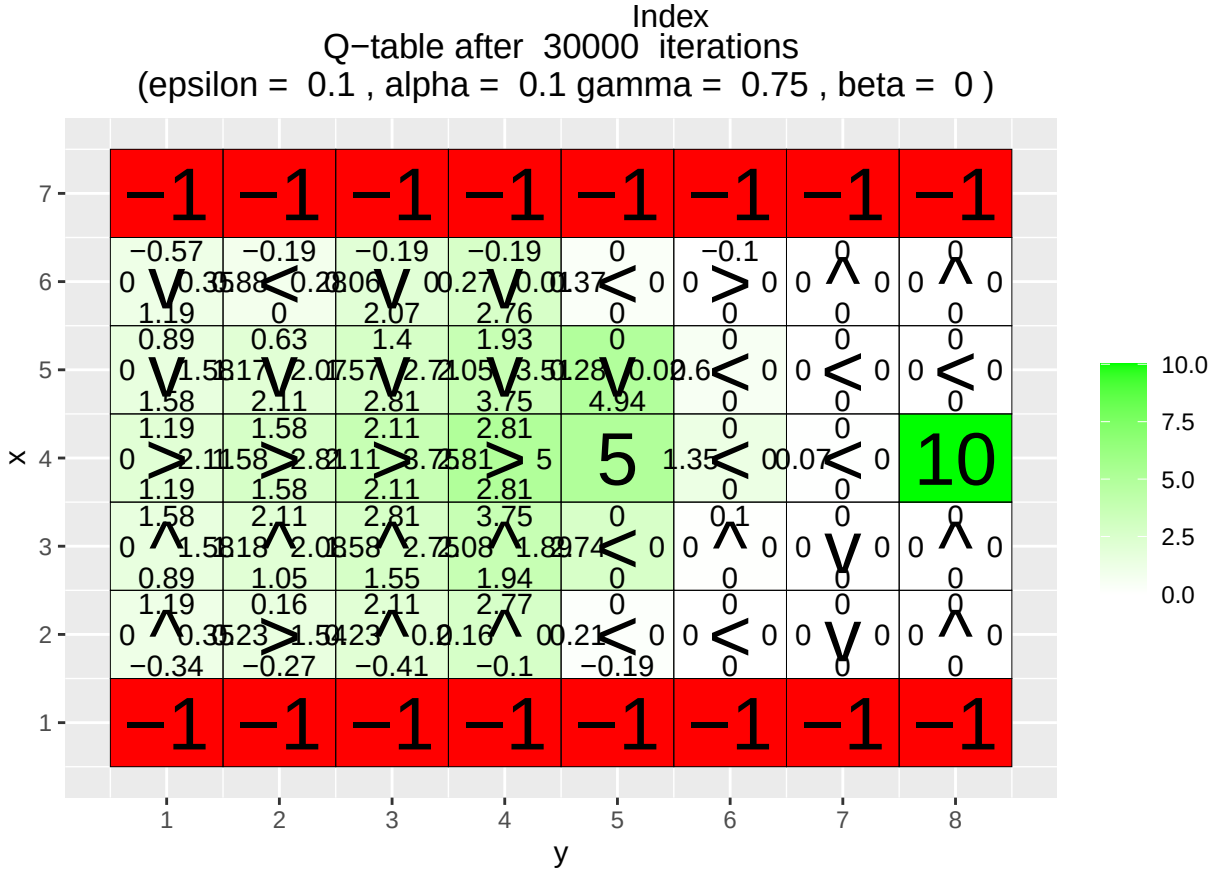


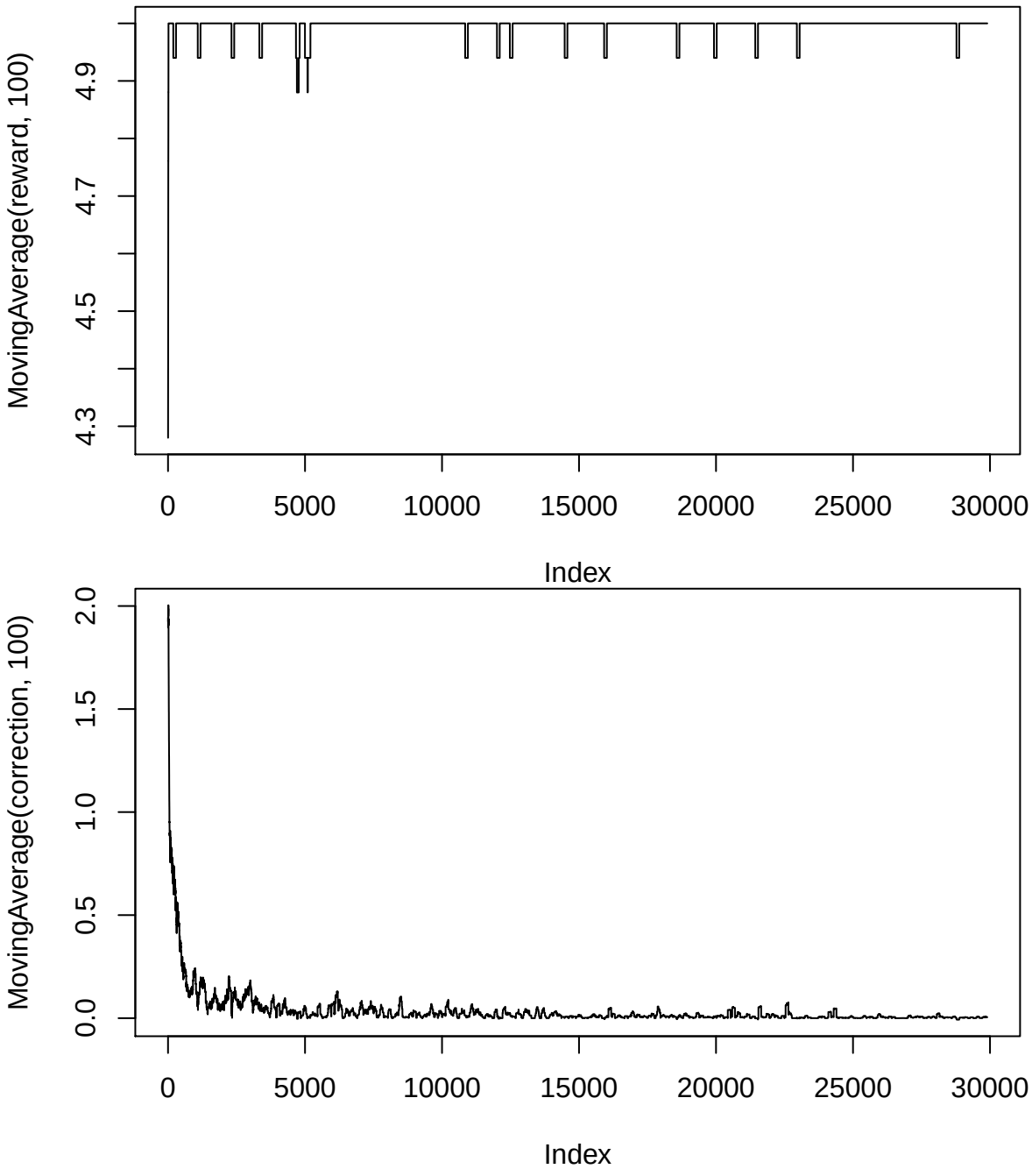

```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  for(i in 1:30000){
    foo <- q_learning(epsilon = 0.1, gamma = j, start_state = c(4,1))
    reward <- c(reward,foo[1])
    correction <- c(correction,foo[2])
  }

  vis_environment(i, epsilon = 0.1, gamma = j)
```

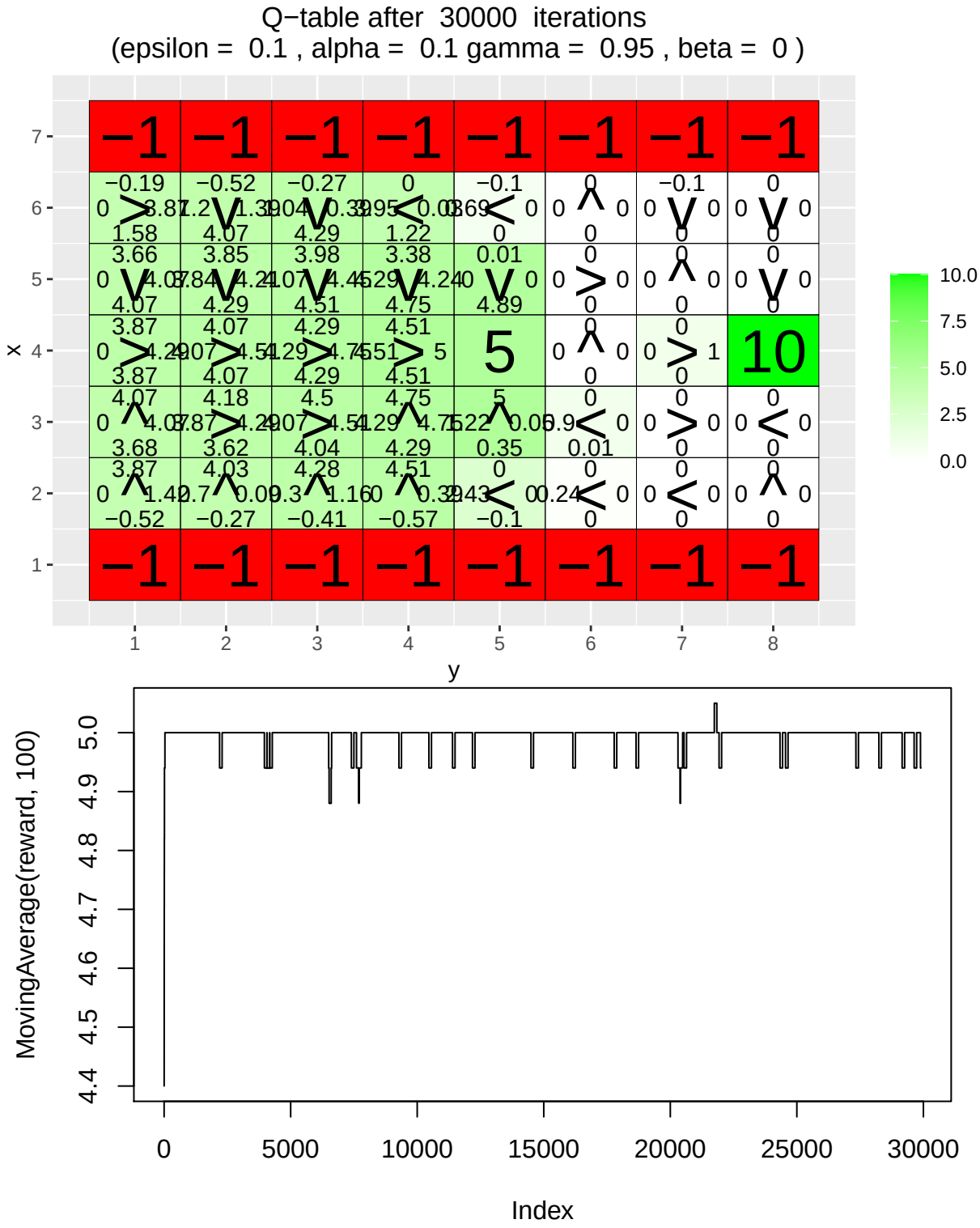
```
  plot(MovingAverage(reward,100),type = "l")
  plot(MovingAverage(correction,100),type = "l")
}
```

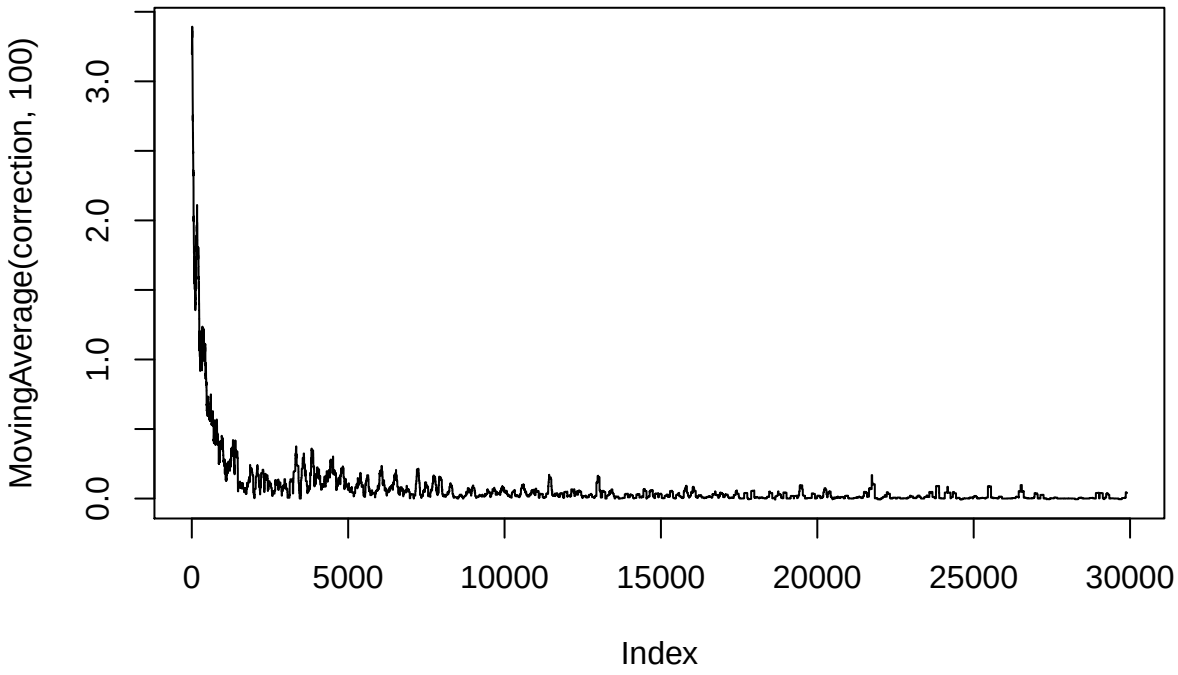



MovingAverage(correction, 100)
0.0
0.4
0.8
1.2
0
5000
10000
15000
20000
25000
30000
Index


Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0


MovingAverage(reward, 100)
4.3
4.5
4.7
4.9
0
5000
10000
15000
20000
25000
30000
Index
MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index


Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0
MovingAverage(reward, 100)
Index



Impact of ϵ on policy learnt

ϵ controls the degree of exploration vs. exploitation. All other hyper-parameters being the same, higher the value of ϵ, more the agent explores (and less it exploits the learnings up until then). To observe the impact of ϵ on the policy learnt in this specific problem, we can compare the three pairs of the policy plots above, where the graphs within each pair differ only in that the ϵ values used to learn the policies plotted are different.

In each such pair, we can observe that only that policy corresponding to the higher ϵ value has paths leading to the terminal state with +10 reward. Further, noting that the starting state of (4, 1), we can infer that the policies corresponding to lower epsilon values were learnt from episodes wherein the agent has not had the opportunity to explore states far beyond the locally rewarding state of (4, 5) (the terminal state with +5 reward).

Whenever ϵ is high in the plots above, we can see that the moving average reward plots show much oscillation, reflecting the fact that there is much ongoing exploration (despite seeing a rewarding terminal state and a path to the same). On the other hands, when ϵ is low, once a path to the nearby positive reward terminal state (+5) is found, the graphs are more or less stationary around a mean value of +5.

The moving average correction plots show that when ϵ is small, the tendency for the moving average correction figure to become stationary around mean 0 is faster than when ϵ is larger.

Impact of γ on policy learnt

γ controls how much the agent but discount future rewards. All other hyper-parameters being the same, higher the value of γ, the less discounted future rewards will be considered. To observe the impact of γ on the policy learnt in this specific problem, we can compare first three policy plots above, wherein the graphs differ only in that the γ values used to learn the policies plotted are different.

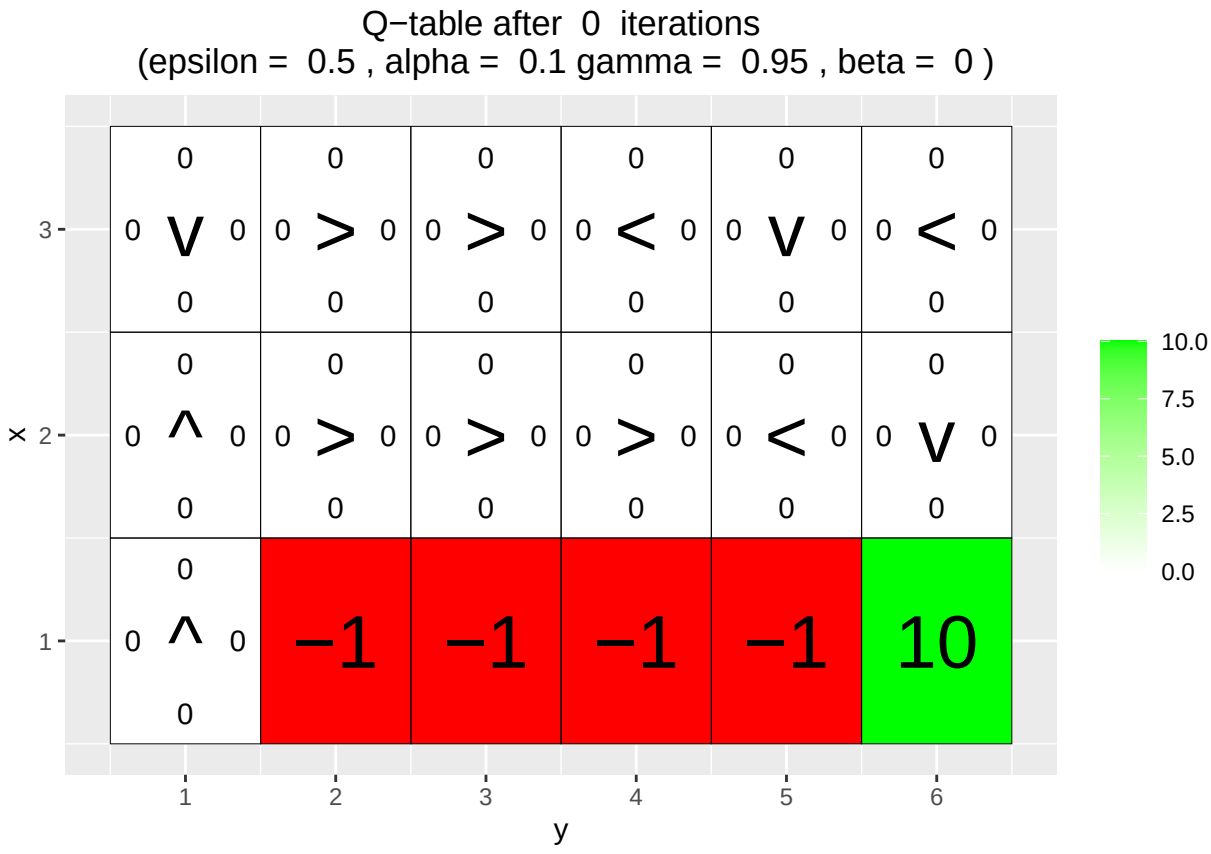
As γ increases, we can note the tendency to learn policies that can take the agent to the highest reward state. Since learning with high γ involves valuing future rewards strongly, the reward differential between the +5 and +10 reward terminal states propels the agent towards the +10 state. This propulsion happens if, by chance (ϵ must be reasonably high) a path to the +10 state is found; when that happens a high value of γ coupled with a reasonable large reward differential (10 - 5 = 5) results in the agent accumulating higher reward for paths to the +10 state rather than +5.

Also, particularly in the third moving average reward plot, we can see a spike between iterations 10,000 and 15,000, possibly corresponding to the first discovery of a path to the +10 reawrd state.

Similarly, as γ increases, we can start observing a secondary spike in the first three moving average reward plots, likely reflecting the discovery of a new path to the +10 state, which results in significant change (and therefore, correction) from previously stable paths to the +5 state.

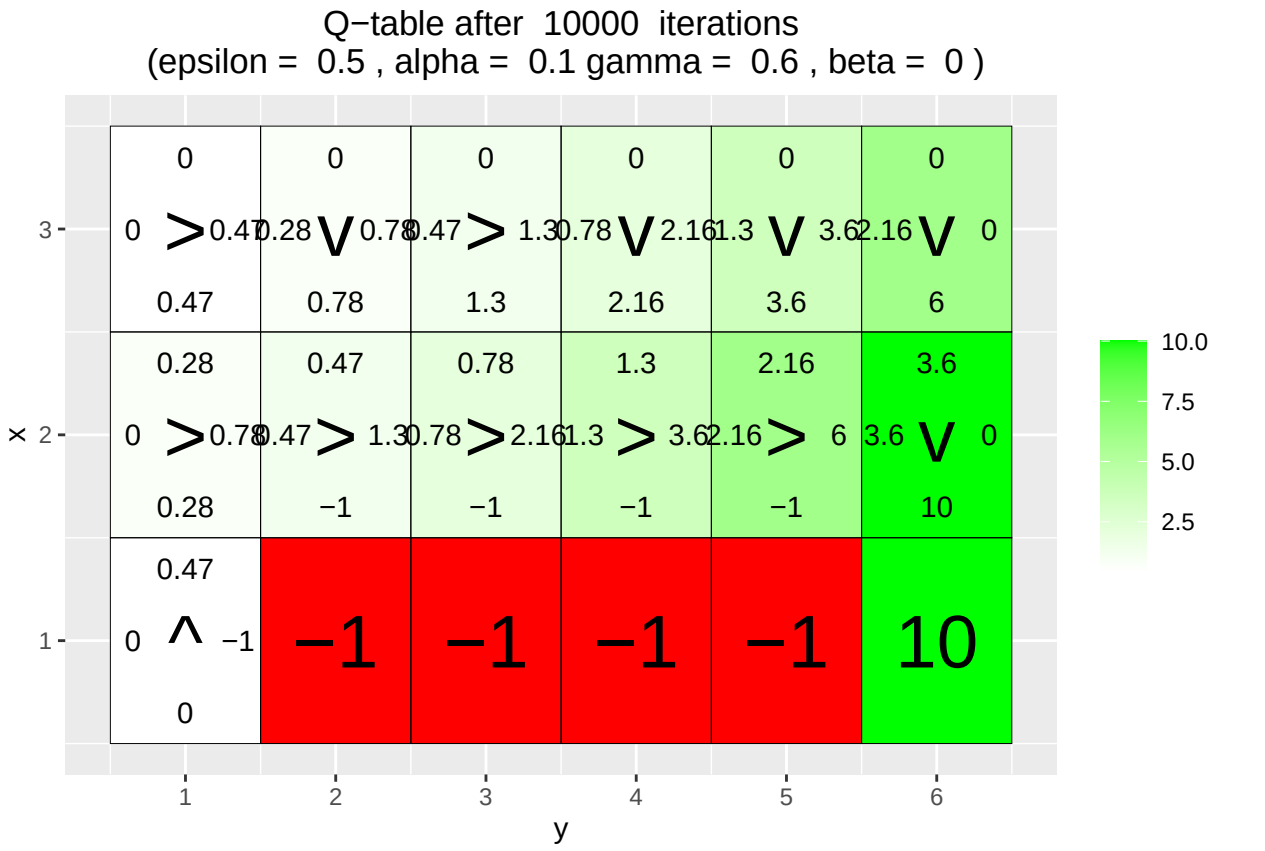
4

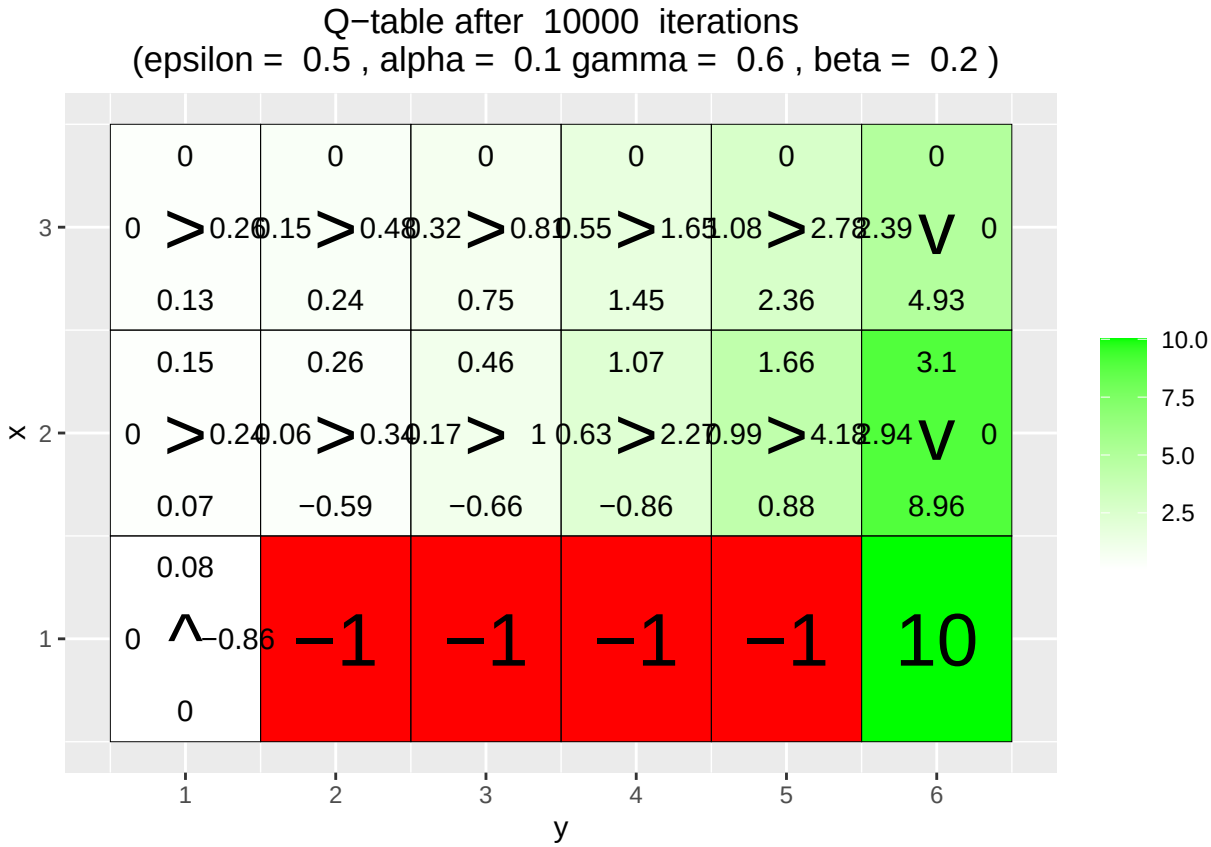
```
H <- 3
W <- 6

reward_map <- matrix(0, nrow = H, ncol = W)
reward_map[1,2:5] <- -1
reward_map[1,6] <- 10

q_table <- array(0,dim = c(H,W,4))

vis_environment()
```

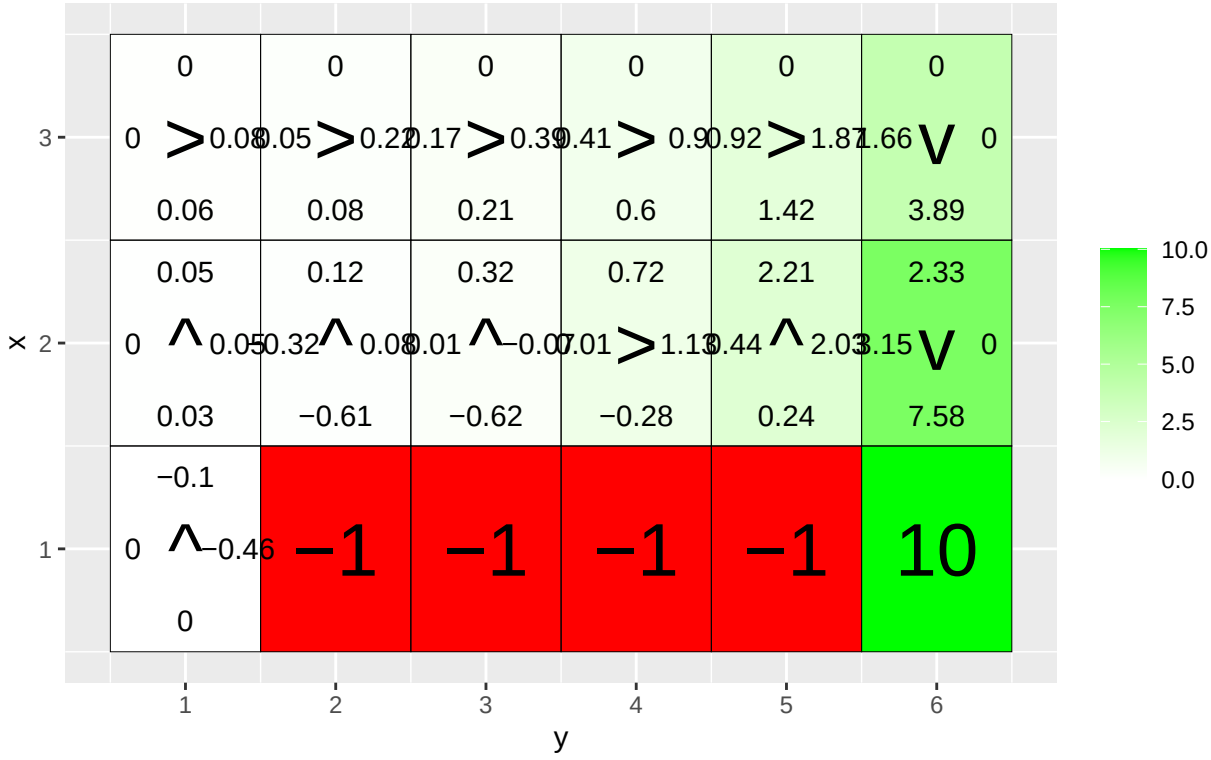


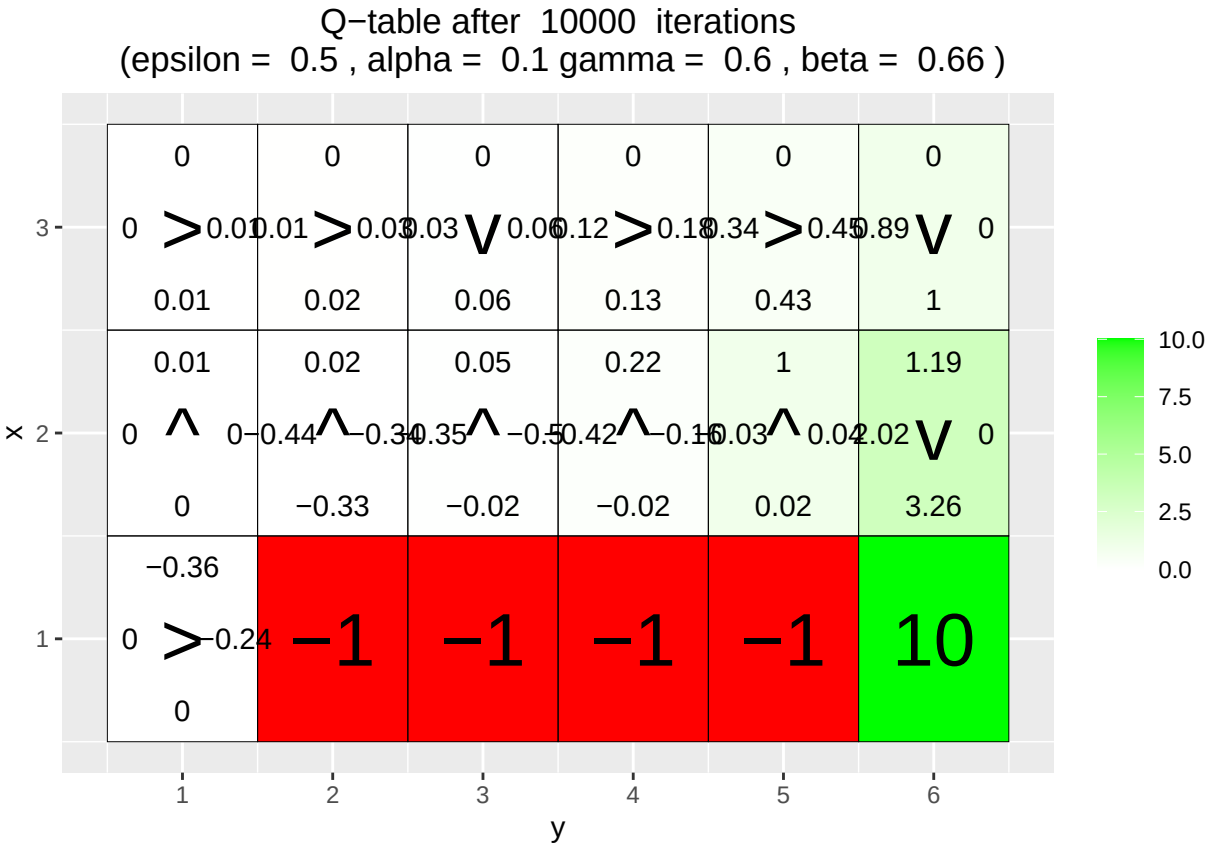

```
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

  for(i in 1:10000)
    foo <- q_learning(gamma = 0.6, beta = j, start_state = c(1,1))

  vis_environment(i, gamma = 0.6, beta = j)
```

```
}
```


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0)
0 0 0 0 0 0
0 > 0.47 0.28 V 0.78 0.47 > 1.3 0.78 V 2.16 1.3 V 3.6 2.16 V 0
0.47 0.78 1.3 2.16 3.6 6
0.28 0.47 0.78 1.3 2.16 3.6
0 > 0.78 0.47 > 1.3 0.78 > 2.16 1.3 > 3.6 2.16 > 6 3.6 V 0
0.28 −1 −1 −1 −1 10
0.47
0 ^ −1 −1 −1 −1 −1 10
0
x
y
1 2 3
1 2 3 4 5 6
10.0
7.5
5.0
2.5


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)
0 0 0 0 0 0
0 > 0.26 0.15 > 0.48 0.32 > 0.81 0.55 > 1.65 1.08 > 2.72 2.39 V 0
0.13 0.24 0.75 1.45 2.36 4.93
0.15 0.26 0.46 1.07 1.66 3.1
0 > 0.24 0.06 > 0.34 0.17 > 1 0.63 > 2.27 0.99 > 4.18 2.94 V 0
0.07 −0.59 −0.66 −0.86 0.88 8.96
0.08
0 ^ −0.86 −1 −1 −1 −1 10
0
x
y
1 2 3 4 5 6
1 2 3
10.0
7.5
5.0
2.5


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.4)
0 0 0 0 0 0
0 > 0.08 0.05 > 0.22 0.17 > 0.39 0.41 > 0.9 0.92 > 1.87 1.66 V 0
0.06 0.08 0.21 0.6 1.42 3.89
0.05 0.12 0.32 0.72 2.21 2.33
0 ^ 0.05 0.32 ^ 0.08 0.01 ^ −0.07 0.01 > 1.13 0.44 ^ 2.03 3.15 V 0
0.03 −0.61 −0.62 −0.28 0.24 7.58
−0.1
0 ^ −0.46 −1 −1 −1 −1 10
0
x
y
1 2 3 4 5 6
1 2 3
10.0
7.5
5.0
2.5
0.0



It is clear from the graphs above that as β increases, all else being the same, the agent learns to avoid states adjacent to the -1 reward state. This is because in such a state, the risk of terminating with -1 reward increases with β.

5

The code referred to was studied as asked.

6

6.1.

I would not say that the agent has learnt a good policy.

- A number of the policy plots do not show arrows pointing toward the goal in goal-adjacent states
- One would expect to see (some) symmetry in the learned policy whenever the goal is not on one of the edges of the square, but this too is not noticeable

Things to change for potentially better results include: increasing the number of training iterations and simplifying the NN architecture (reducing the number of parameters) to reduce over-fitting and improve generalisation.

6.2.

No, we could not have used Q-learning. The stochastic nature of the terminal state could nullify learnings from one episode in another if goals in the two episodes are 'oppositely' located.

7

7.1.

I would not say that the agent has learnt a good policy. Apart from the points mentioned in 6.1, not choosing a goal from any line other than the x = 4 line would result in no tuning of the weights corresponding to that co-ordinate of the goal.

7.2.

Validation policy plots from both environments, where goal states are both either at the locations (2, 3) or (1, 1) can be used for comparison.

In both cases, it is clear that results obtained for Environment D are better than those obtained for Environment E.

Reason: as indicated above, training with goal states from different rows of the grid-world have enabled the learning network in Environment D to tune the parameters using the x-coordinate of the goal. In Environment E, however, restricted training using goal states from row 1 alone essentially ended up making the learning network treat the x-coordinate of the goal as a bias, resulting in poor generalisation.