Lab 3 - Reinforcement Learning

Daniel Kouznetsov

2022-09-28

Environment A

In the first assignment you had to implement code to get the Greedy policies working. You also had to write the q_learning algorithm. Code is shown below.

```r
GreedyPolicy <- function(x, y){
  q_values <- 1:4
  for(i in 1:4){
    q_values[i] <- q_table[x, y, i]
  }
  return(which.max(q_values))
}
```

```r
EpsilonGreedyPolicy <- function(x, y, epsilon){
  rng <- runif(1)
  if (rng < epsilon){
    return(floor(runif(1, 1, 5)))
  }
  else {
    return(GreedyPolicy(x, y))
  }
}
```

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

  state <- start_state
  episode_correction <- 0
  repeat{
    # Follow policy, execute action, get reward.
    # We need current state and future state & actions
    future_action <- EpsilonGreedyPolicy(state[1], state[2], epsilon)
    future_state <- transition_model(state[1], state[2], future_action, beta)
    reward <- reward_map[future_state[1], future_state[2]]

    # Q-table update.
    temporal_difference <- reward + gamma * max(q_table[future_state[1], future_state[2], ]) -
      q_table[state[1], state[2], future_action]

    q_table[state[1], state[2], future_action] <<- q_table[state[1], state[2], future_action] +
      alpha*(temporal_difference)
    episode_correction <- episode_correction + temporal_difference

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

Question 1 What has the agent learned after the first 10 episodes?

Not much. We can see that the agent learned that it will get some negative rewards when it enters the red tiles.

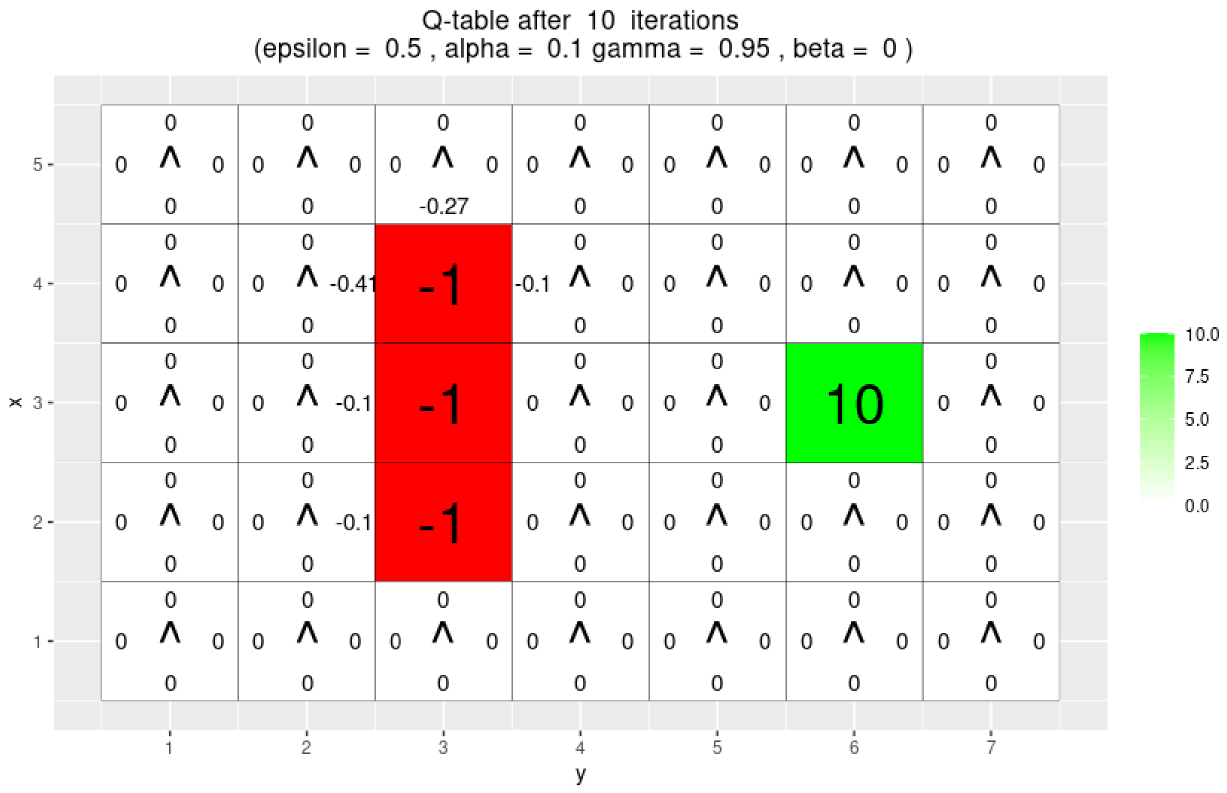


Figure 1: After 10 iterations

Question 2 Is the final greedy policy (after 10000 episodes) optimal for all states, i.e. not only for the initial state? Why / Why not?

No. For example we have tile [1, 2] which gives the greatest reward for moving upwards when clearly the most optimal route for the agent to take is to the right. We also have a high reward for going down on the same tile which results in going into a wall.

Question 3 Do the learned values in the Q-table reflect the fact that there are multiple paths (above and below the negative rewards) to get to the positive reward? If not, what could be done to make it happen?

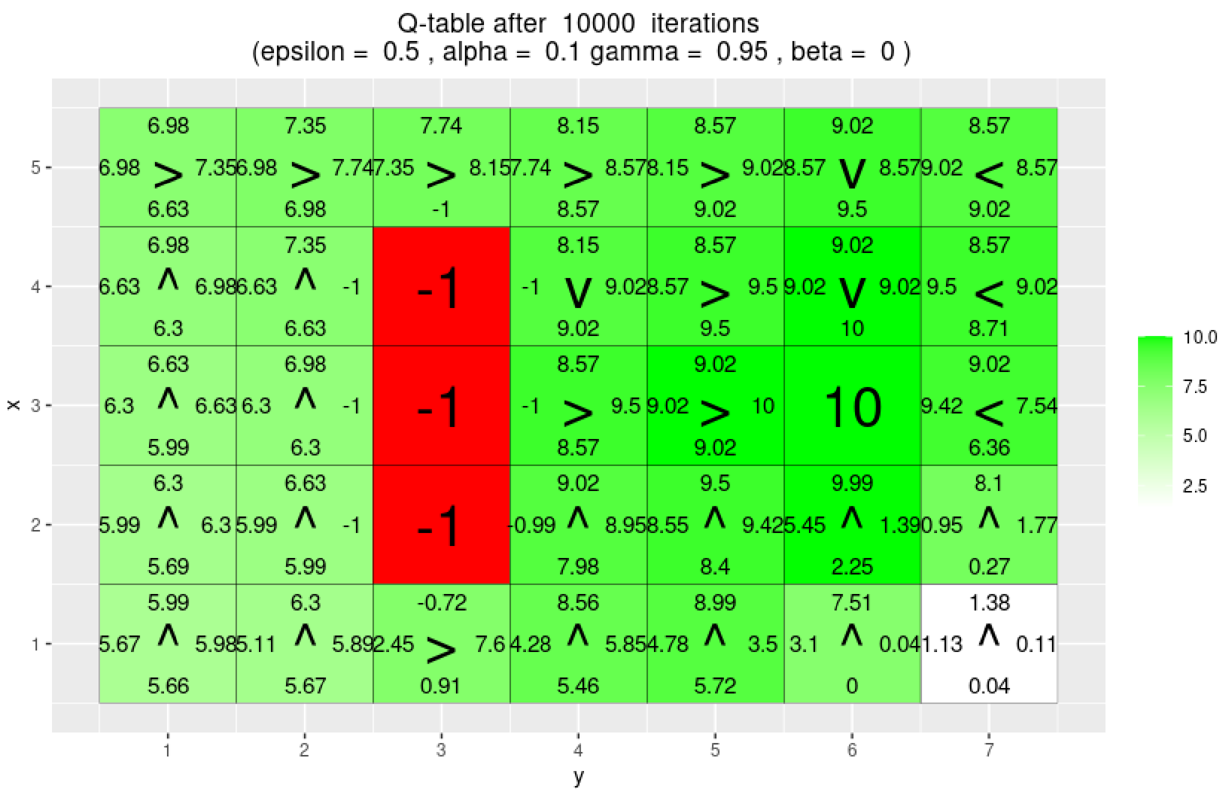


Figure 2: After 10000 iterations

Yes the learned values reflect this. For example the tile [1, 3] has a high reward from the previous tile [1, 2], the same goes for the tile above the negative rewards, [5, 3] which also has a high reward from the previous tile. The tile at [1, 3] does not have the same negative reward for stepping onto a red tile though, this could be because the agent has not gone this route as many times as the others and therefore it has not learnt that this should yield a reward of -1.

Environment B

Epsilon = 0.1 Gamma = 0.5

Reward starts at around 7-8 and quickly goes down to 4-5 and then it stays nearly at exactly 5 for the rest of the iterations (after 7500~). High correction in the beginning and then nearly 0, especially after 10000 iterations, we got some spike at around 7500.

Epsilon = 0.1 Gamma = 0.75

Reward value is steady around 7-9 until 7500 iterations when it drops down to a steady 5~. Correction starts large for the first few iterations to then drop down to around 0 for all of the other iterations except for some smaller spikes, largest one occurring at 10500~.

Epsilon = 0.1 Gamma = 0.95

Reward oscillates between 7-9 for all iterations except for the first one. Correction really large for the first few iterations and then steady at around 0 for the rest of the iterations.

Epsilon = 0.5 Gamma = 0.5

The Q-table visualization bugs for me so it does not show any updates, just the starting table for 0 iterations. The plots work though and show a steady reward rate of around 4.5 after the first few iterations. The correction value behaves as you should expect from the reward plot and is a large value for the first few iterations to then have a steady value at around 0-0.1.

Epsilon = 0.5 Gamma = 0.75

Pretty much the same as for the one with gamma = 0.5. The only noticeable difference is that the values get to a slight larger value for the reward after around 5000 iterations, also the correction variable is slightly larger for the first 5000 iterations.

Epsilon = 0.5 Gamma = 0.95

Here we got something different. The reward is stable at around 5 but after 5000 iterations we get a higher average at around 6. We also get a correction spike at 5000.

Conclusion

A low epsilon value is going to lower the chances that the agent takes a random step. This means that when we also have a low gamma value it the q-table is going to train the model to quickly finish and mostly just take the safe route to the high but lower reward of 5. If we have a higher gamma value it is going to favor the longer route to postpone the reward but get a higher value.

A high epsilon value is going to increase the chances for the agent to take a random step in either direction. This results in more oscillations for the reward plot because where the agent lands is random for every other step. The same principle goes for the gamma value, a higher value is going to yield more rewards. A lower gamma value is more short-sighted.

Environment C

With a slipping factor of zero we get that the agent can comfortably after 10000 iterations walk straight to the tile with the highest reward. If the slipping factor increases the agent will behave in such a way that slipping is not is a big risk, therefore it will take a route far from the tiles with a reward of -1.

Environment D

The agent has learned a good policy. If we compare the first run to the last one we can see that for the first one we have sub-optimal reward values for walking to our goal when it is right beside it, the latter does not have this and all tiles will make the agent walk towards our goal.

The agent does not have any memory mechanism, therefore we cannot use Q-Learning to solve this task.

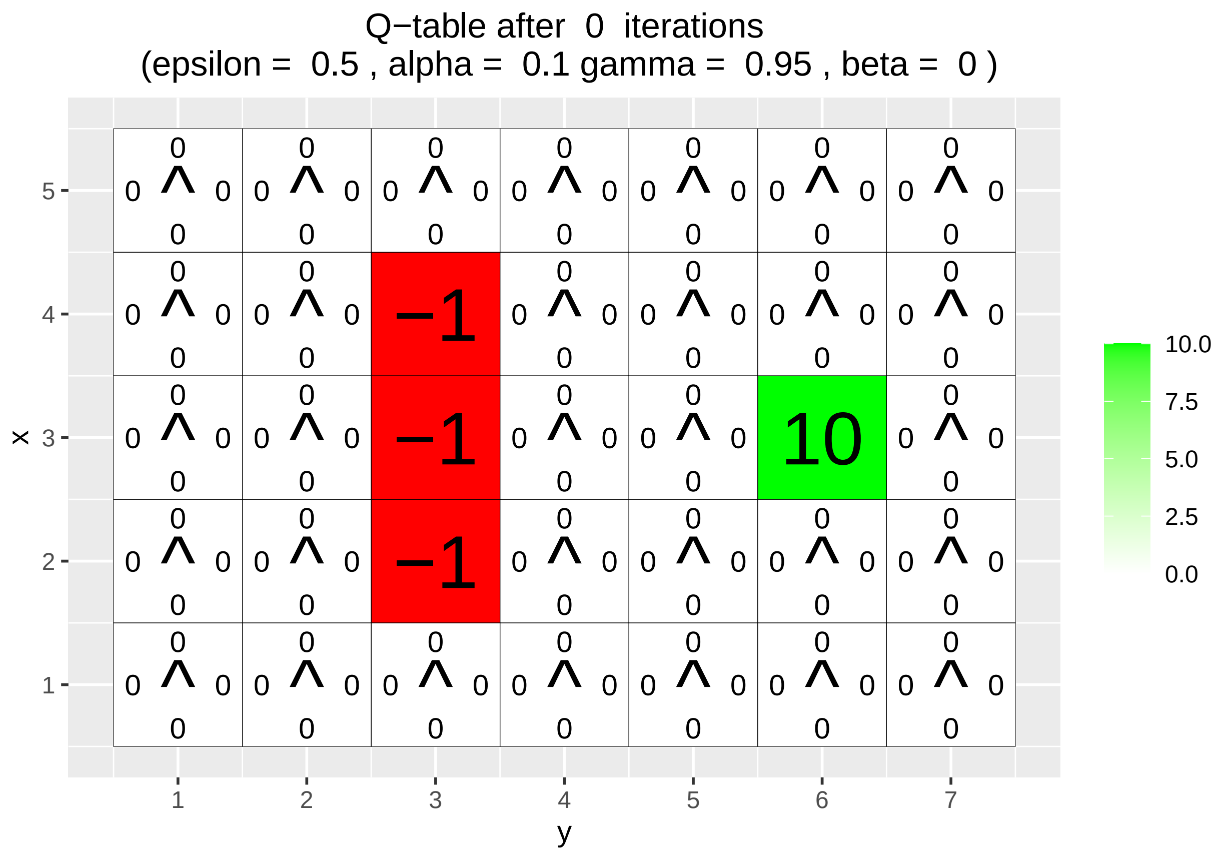
Environment E

It has not learnt a good policy. If the goal is not in the top row then all paths cannot lead to the goal.

The results differ because environment E is not only trained on having the top row as a goal whilst environment D is trained on random positions which gives the agent more flexibility depending on where the generated random goal is. hej

Code Appendix with plots for all environments

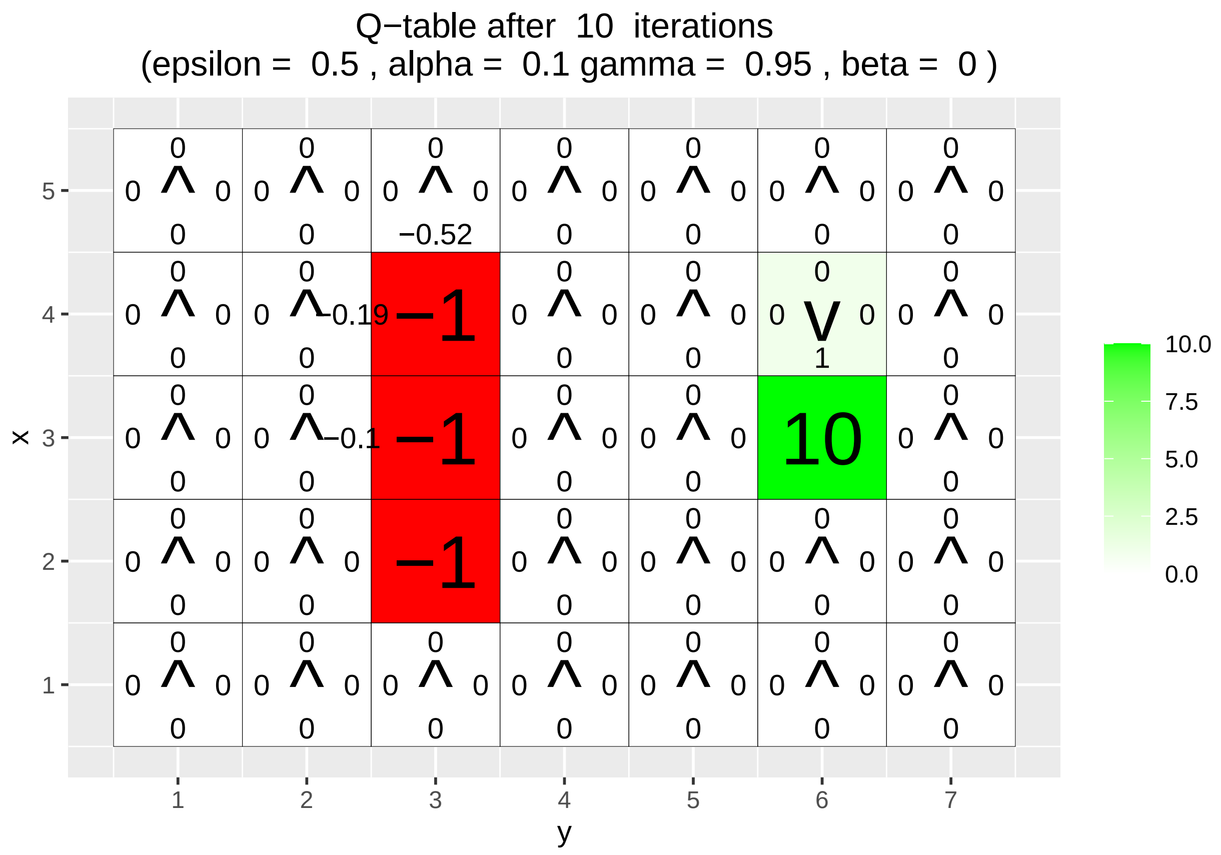
```r
# By Jose M. Peña and Joel Oskarsson.
# For teaching purposes.
# jose.m.pena@liu.se.

#####################################################################################################
# Q-learning
#####################################################################################################

#install.packages("ggplot2")
#install.packages("vctrs")
library(ggplot2)

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.
  q_values <- 1:4
  for(i in 1:4){
    q_values[i] <- q_table[x, y, i]
  }
  return(which.max(q_values))
}

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.
  rng <- runif(1)
  if (rng < epsilon){
    return(floor(runif(1, 1, 5)))
  }
  else {
    return(GreedyPolicy(x, y))
  }
}

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){

  # 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{
    # Follow policy, execute action, get reward.
    # We need current state and future state & actions
    future_action <- EpsilonGreedyPolicy(state[1], state[2], epsilon)
    future_state <- transition_model(state[1], state[2], future_action, beta)
    reward <- reward_map[future_state[1], future_state[2]]

    # Q-table update.
    temporal_difference <- reward + gamma * max(q_table[future_state[1], future_state[2], ]) -
      q_table[state[1], state[2], future_action]

    q_table[state[1], state[2], future_action] <<- q_table[state[1], state[2], future_action] +
      alpha*(temporal_difference)

    episode_correction <- episode_correction + temporal_difference

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

}

#####################################################################################################
# Q-Learning Environments
#####################################################################################################

# Environment A (learning)

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()
```

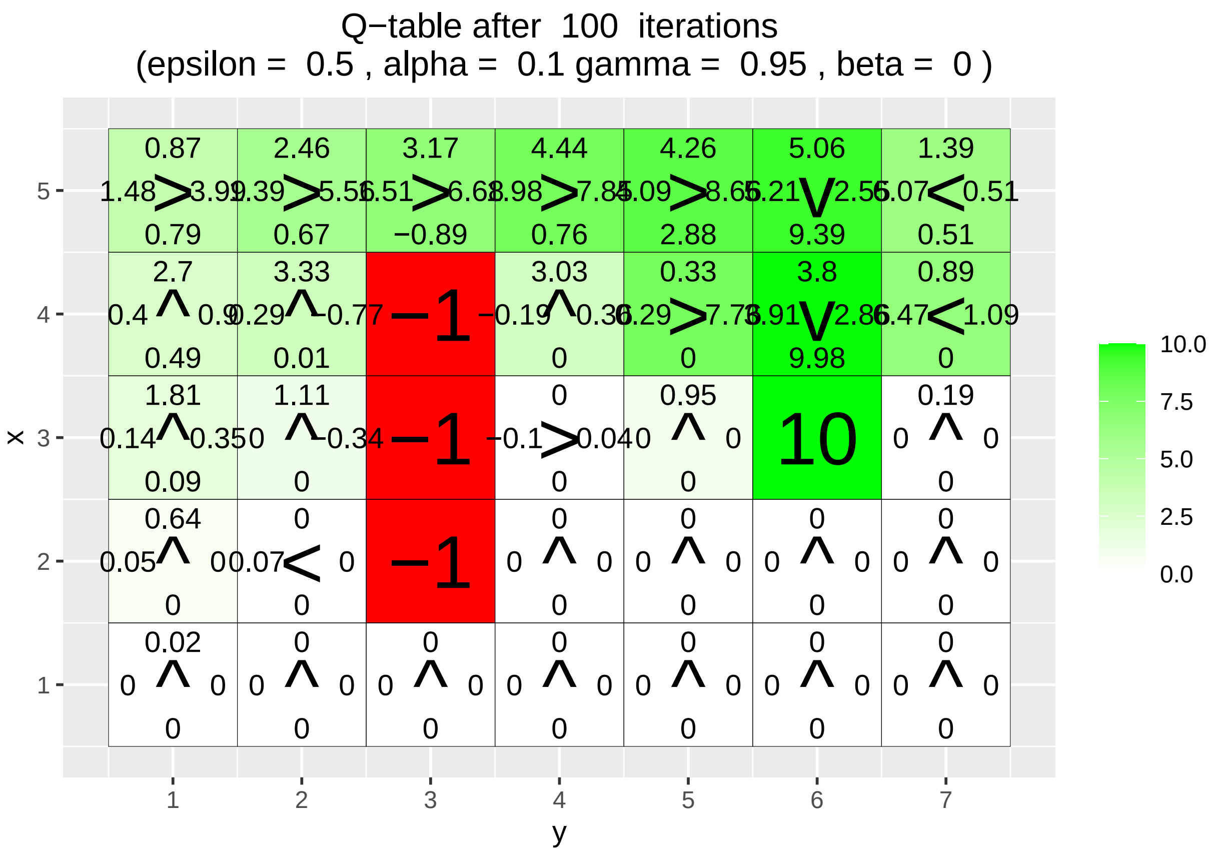


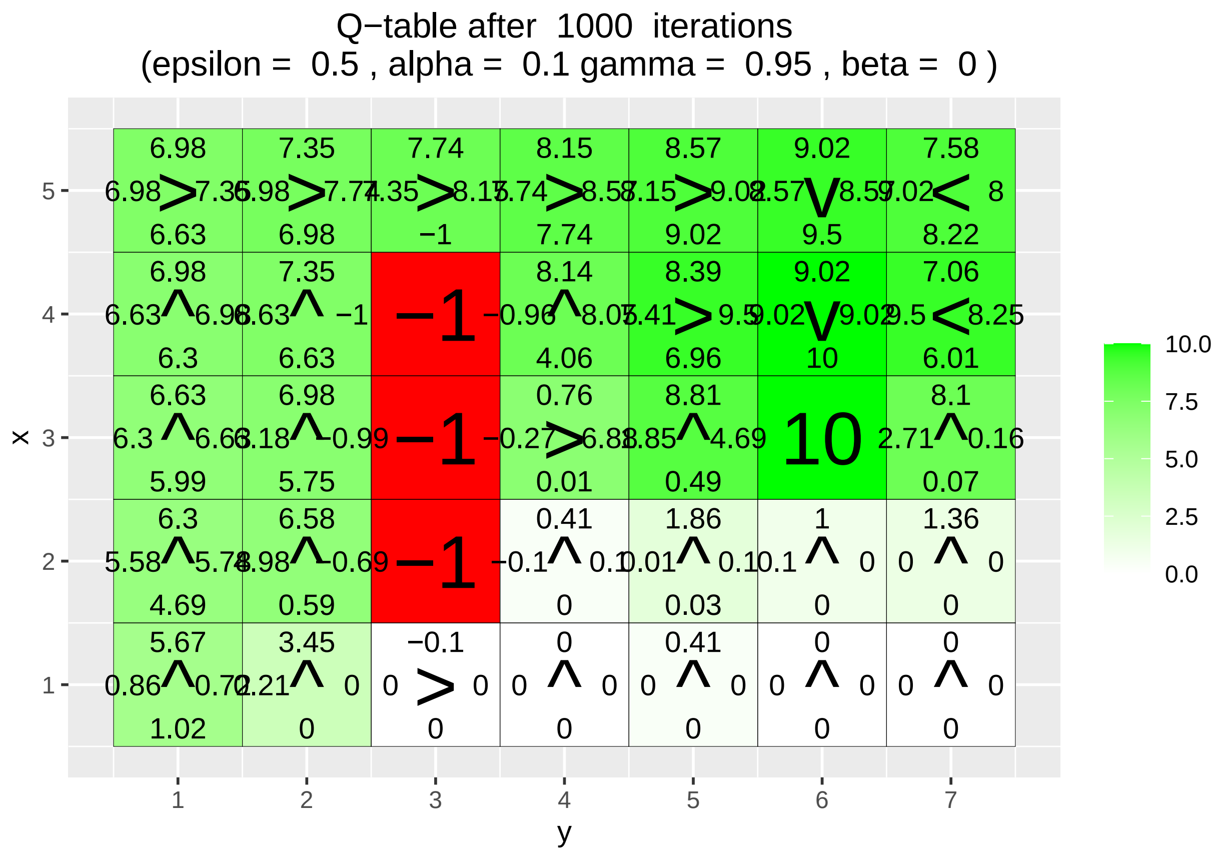
```
q_learning(start_state = c(3, 1))
```

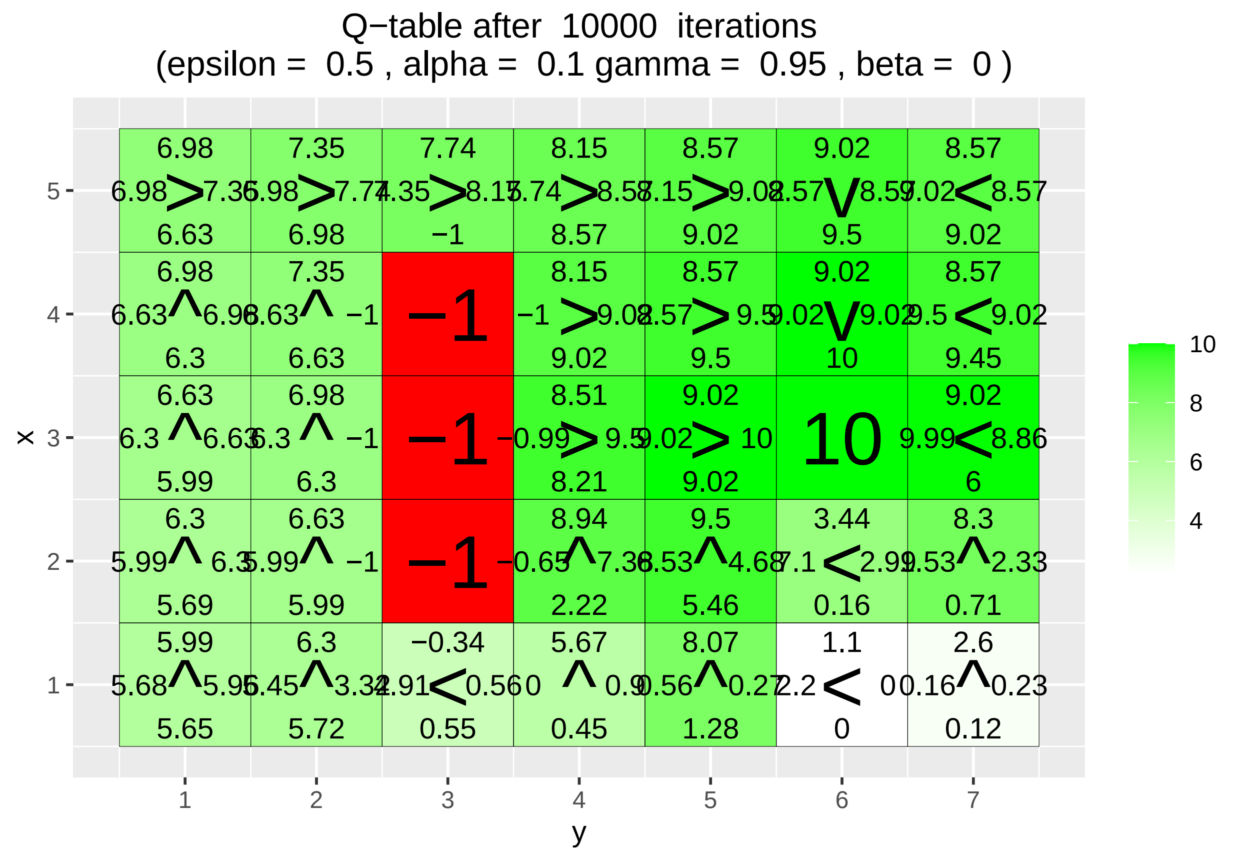
```
## [1] -1 -1
```

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

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

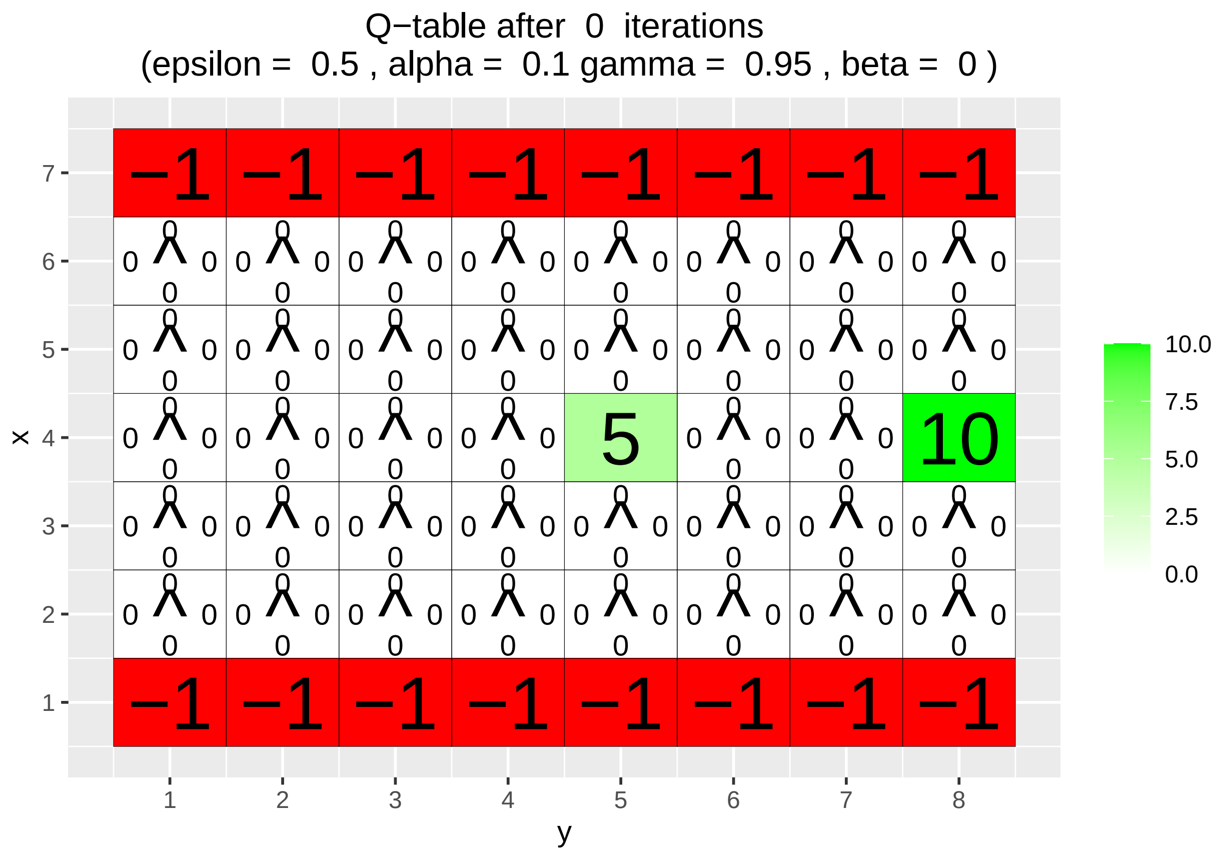

Q−table after 10 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
−1
10
−0.52
−0.19
−0.1
10.0
7.5
5.0
2.5
0.0


Q−table after 100 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
0.87 2.46 3.17 4.44 4.26 5.06 1.39
1.48 > 3.99 | 9.39 > 5.56 | 6.51 > 6.68 | 3.98 > 7.84 | 5.09 > 8.65 | 5.21 V 2.56 | 5.07 < 0.51
0.79 0.67 −0.89 0.76 2.88 9.39 0.51
2.7 3.33 3.03 0.33 3.8 0.89
0.4 ^ 0.9 | 0.29 ^ −0.77 | −1 | −0.19 ^ 0.36 | 6.29 > 7.73 | 3.91 V 2.86 | 6.47 < 1.09
0.49 0.01 0 0 9.98 0
1.81 1.11 0 0.95 0.19
0.14 ^ 0.35 | 0 ^ −0.34 | −1 | −0.1 > 0.04 | 0 ^ 0 | 10 | 0 ^ 0
0.09 0 0 0 0
0.64 0 0 0 0 0
0.05 ^ 0 | 0.07 < 0 | −1 | 0 ^ 0 | 0 ^ 0 | 0 ^ 0 | 0 ^ 0
0 0 0 0 0 0
0.02 0 0 0 0 0 0
0 ^ 0 | 0 ^ 0 | 0 ^ 0 | 0 ^ 0 | 0 ^ 0 | 0 ^ 0 | 0 ^ 0
0 0 0 0 0 0 0
x
1 2 3 4 5
y
1 2 3 4 5 6 7
10.0
7.5
5.0
2.5
0.0

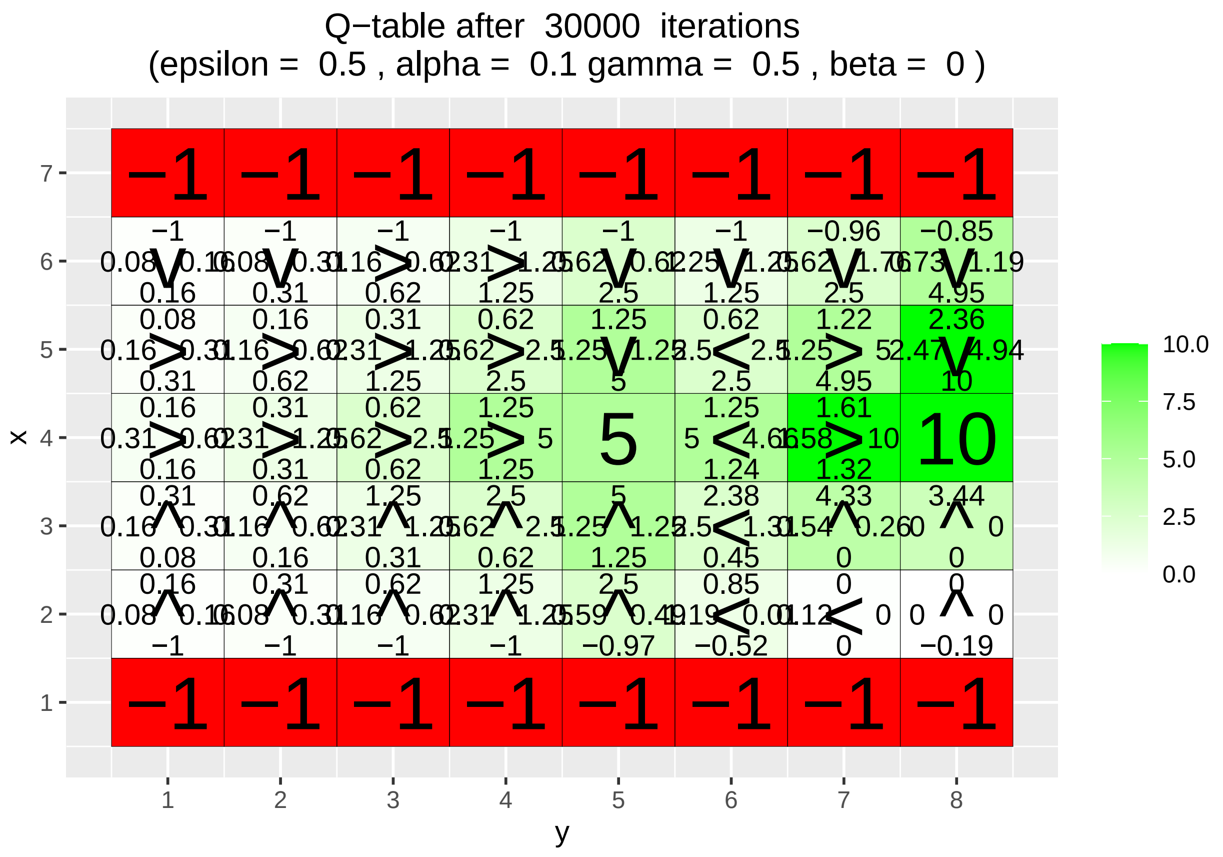

Q−table after 1000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0



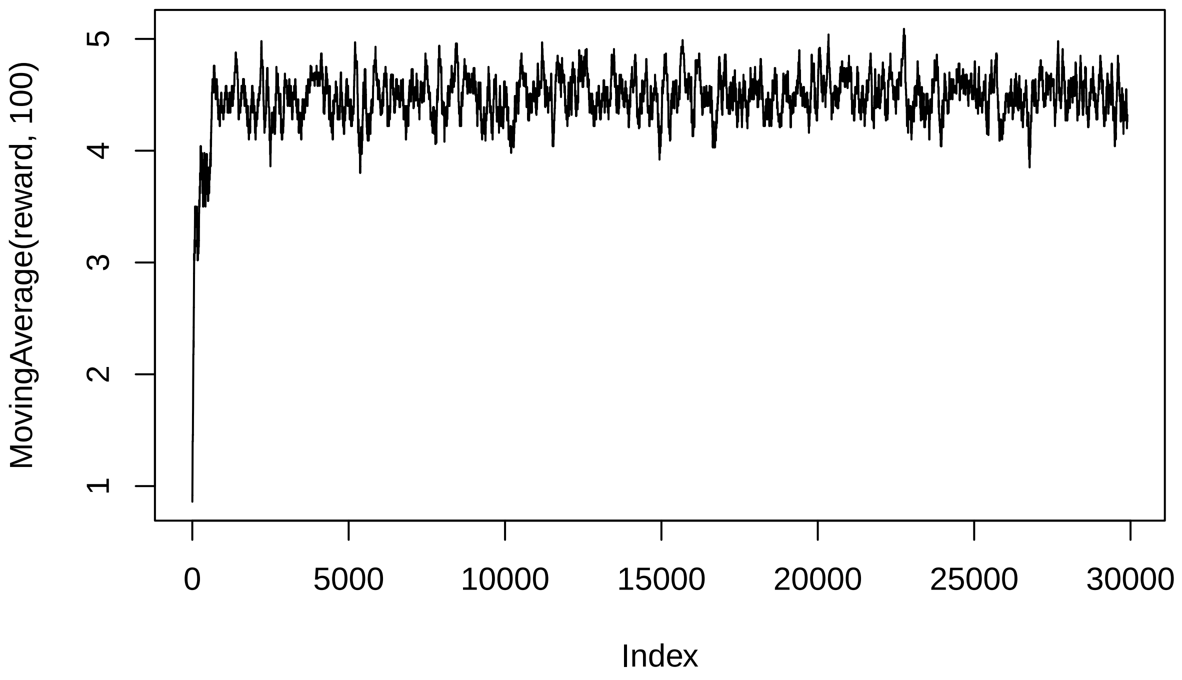
```r
# Environment B (the effect of epsilon and gamma)

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()
```

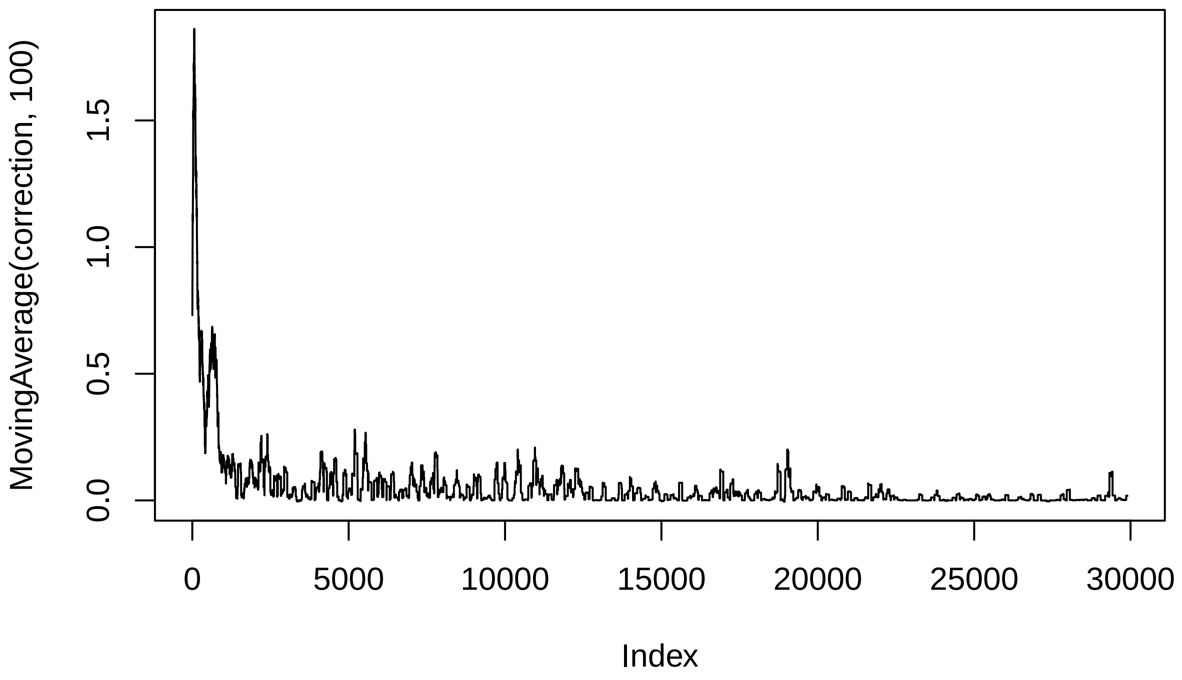


```
MovingAverage <- function(x, n){

  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")
}
```

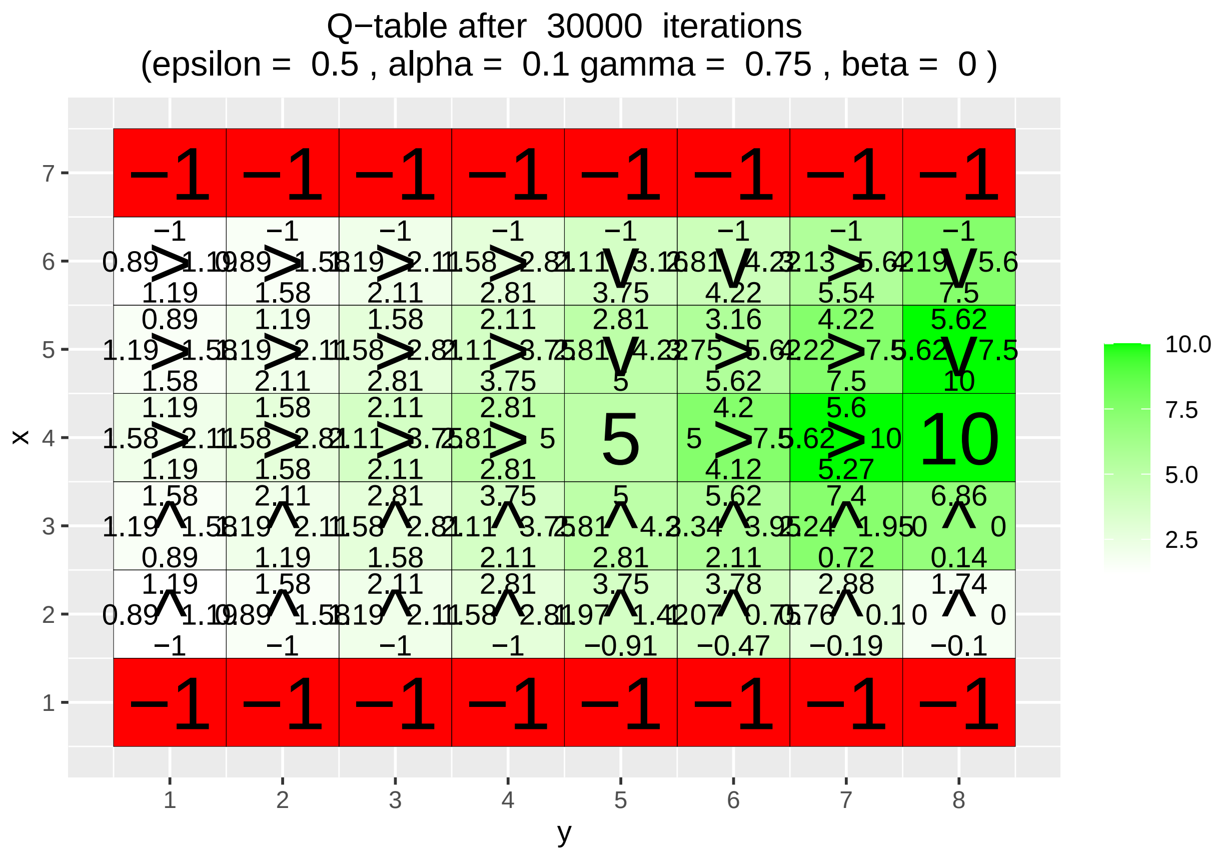


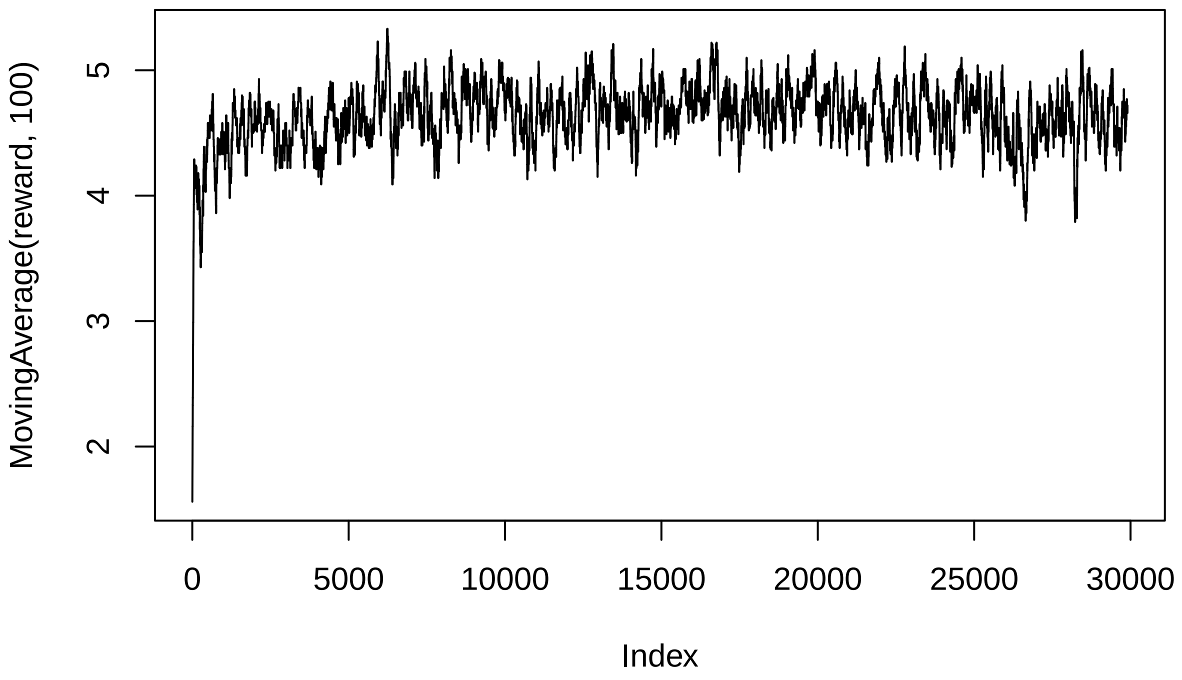
Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
1
2
3
4
5
6
7
8
−1
5
10
10.0
7.5
5.0
2.5
0.0

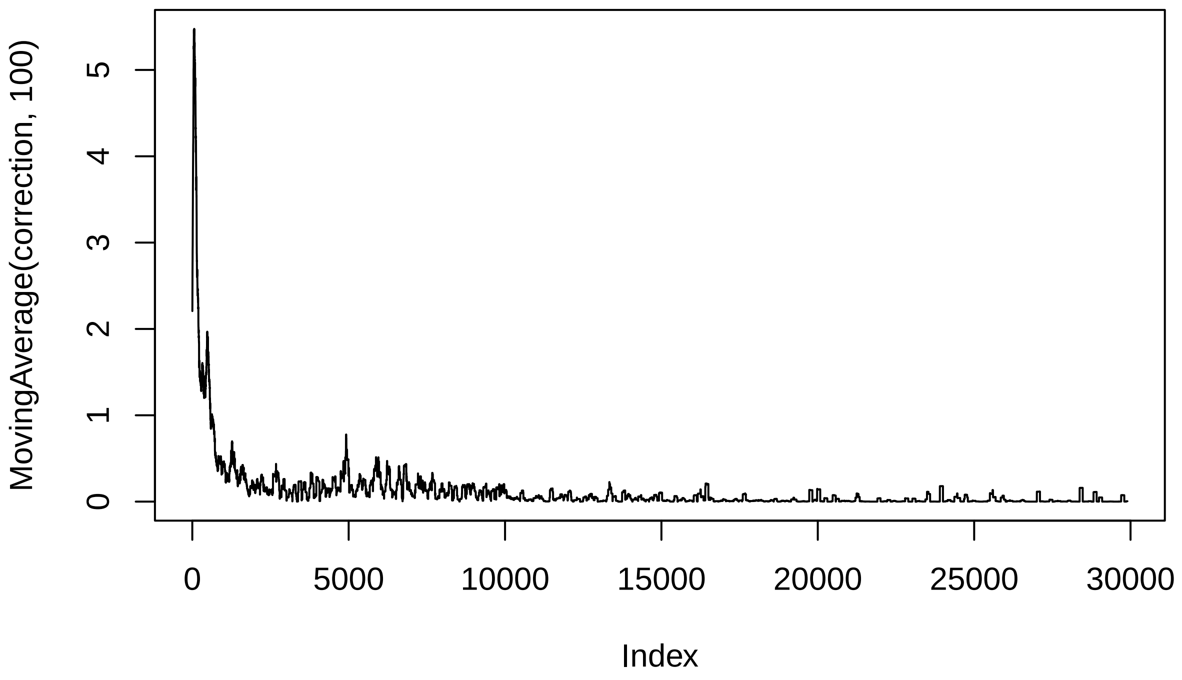

MovingAverage(reward, 100)
1
2
3
4
5
0
5000
10000
15000
20000
25000
30000
Index

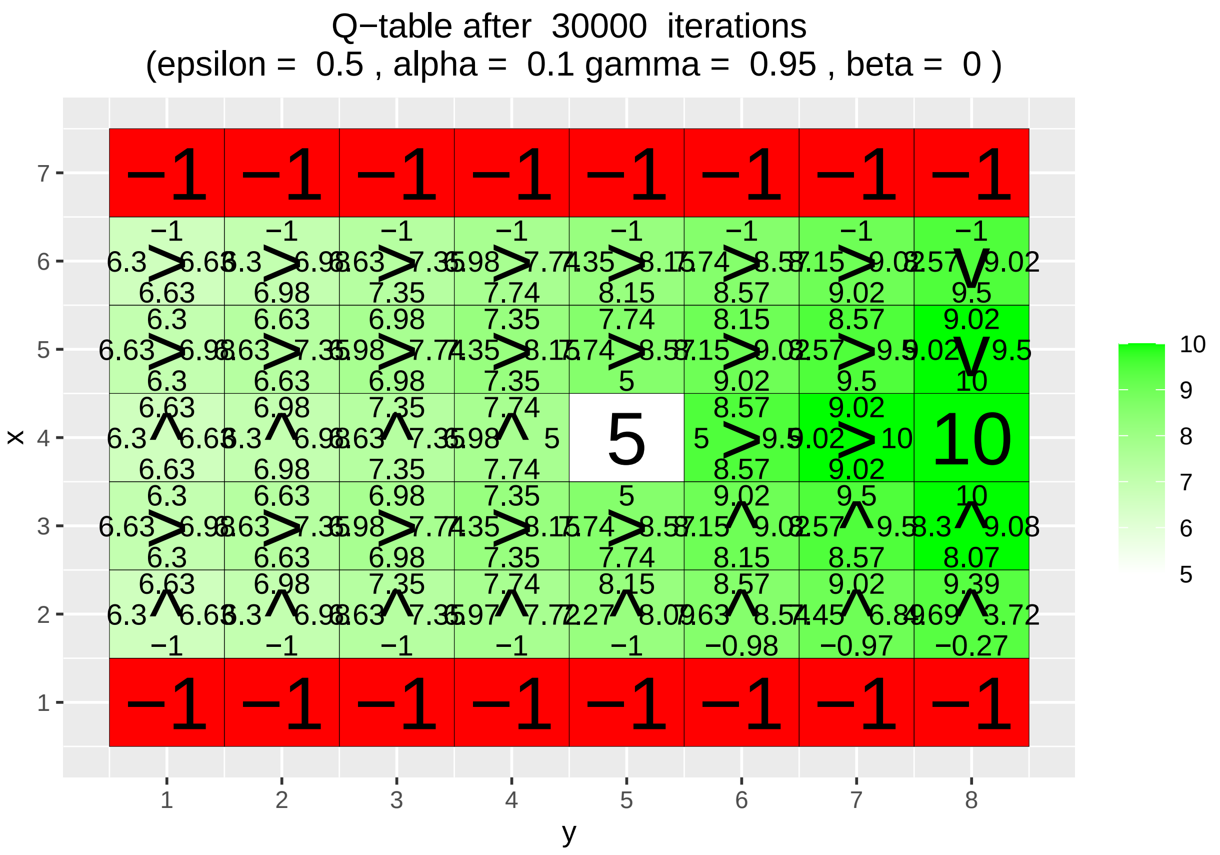


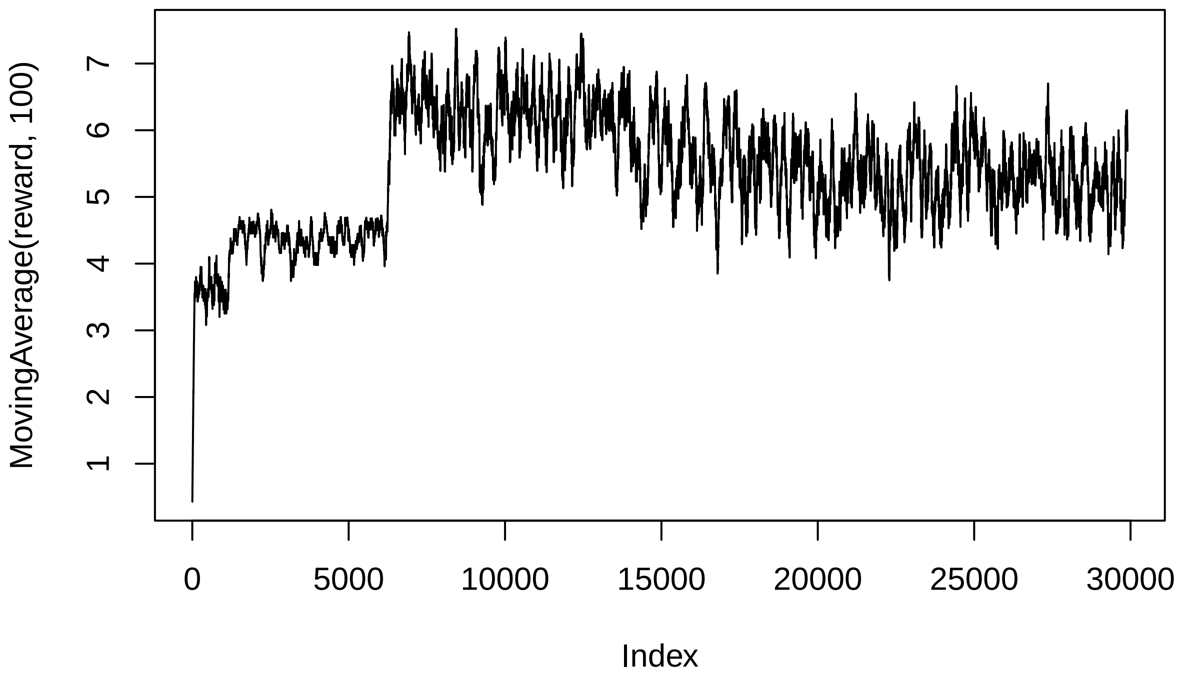
MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
0
5000
10000
15000
20000
25000
30000
Index

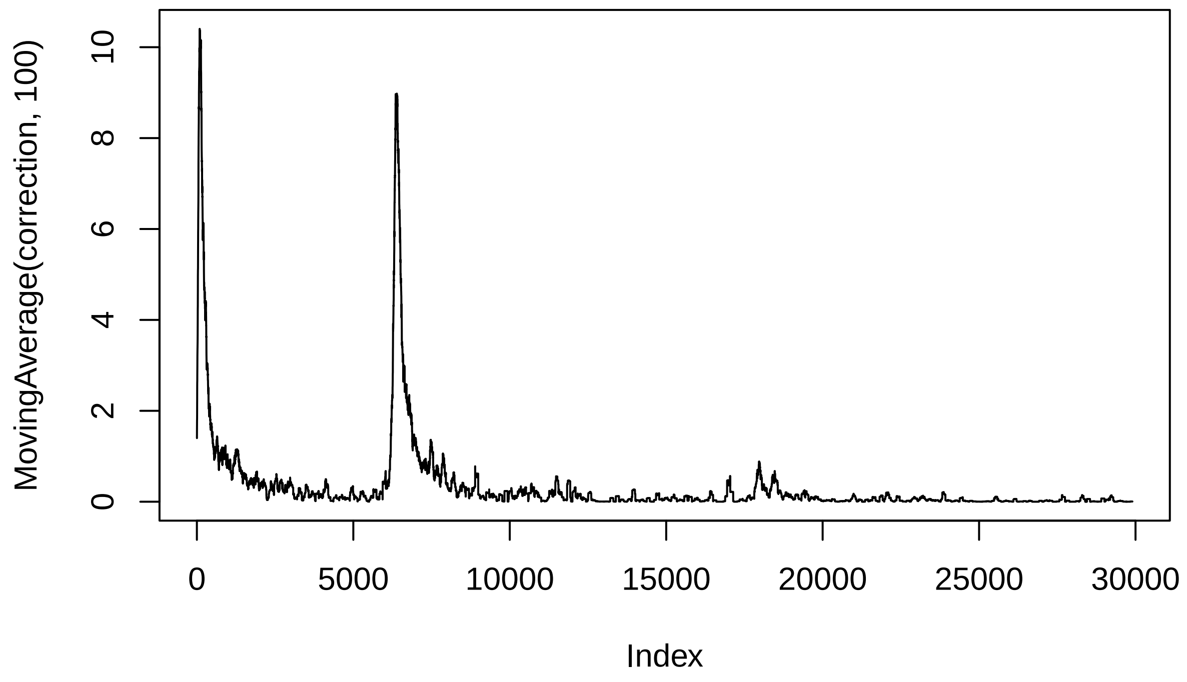

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5


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

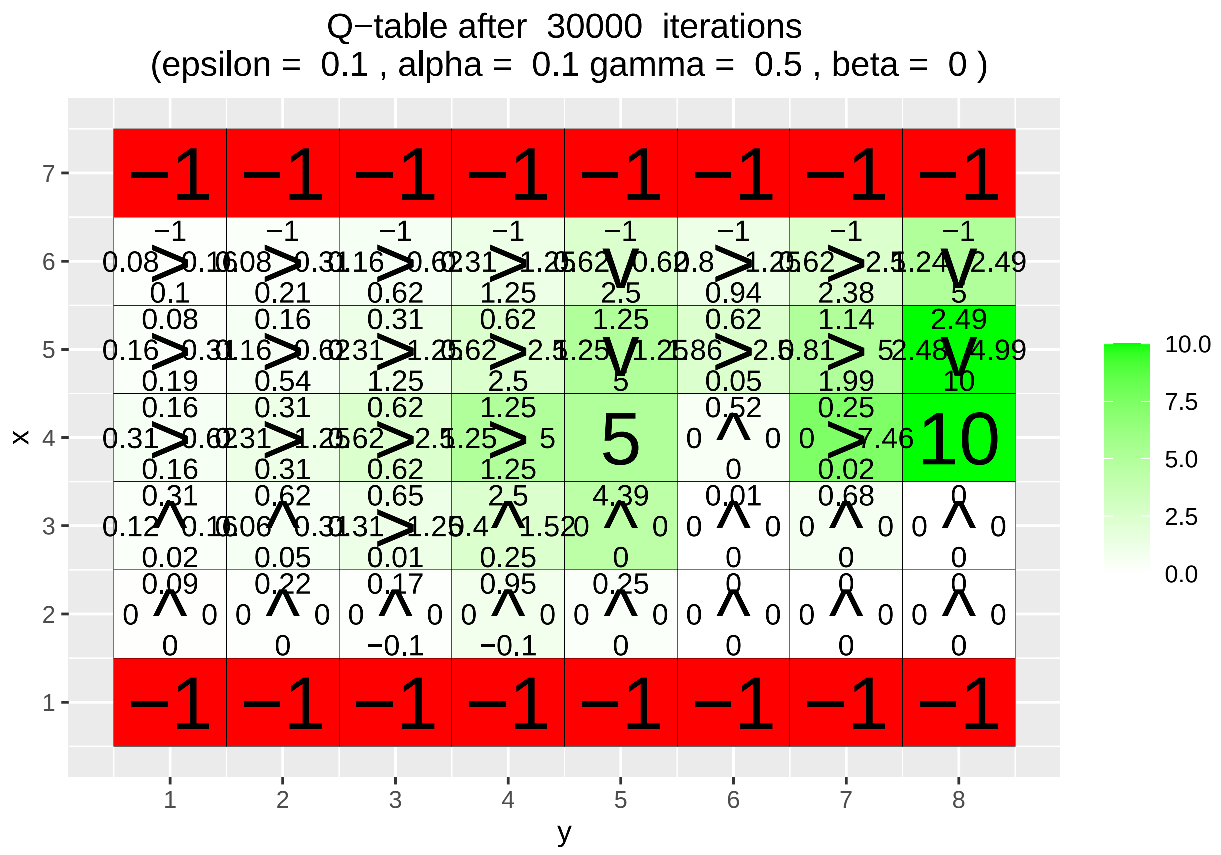

MovingAverage(correction, 100)
0
1
2
3
4
5
0
5000
10000
15000
20000
25000
30000
Index

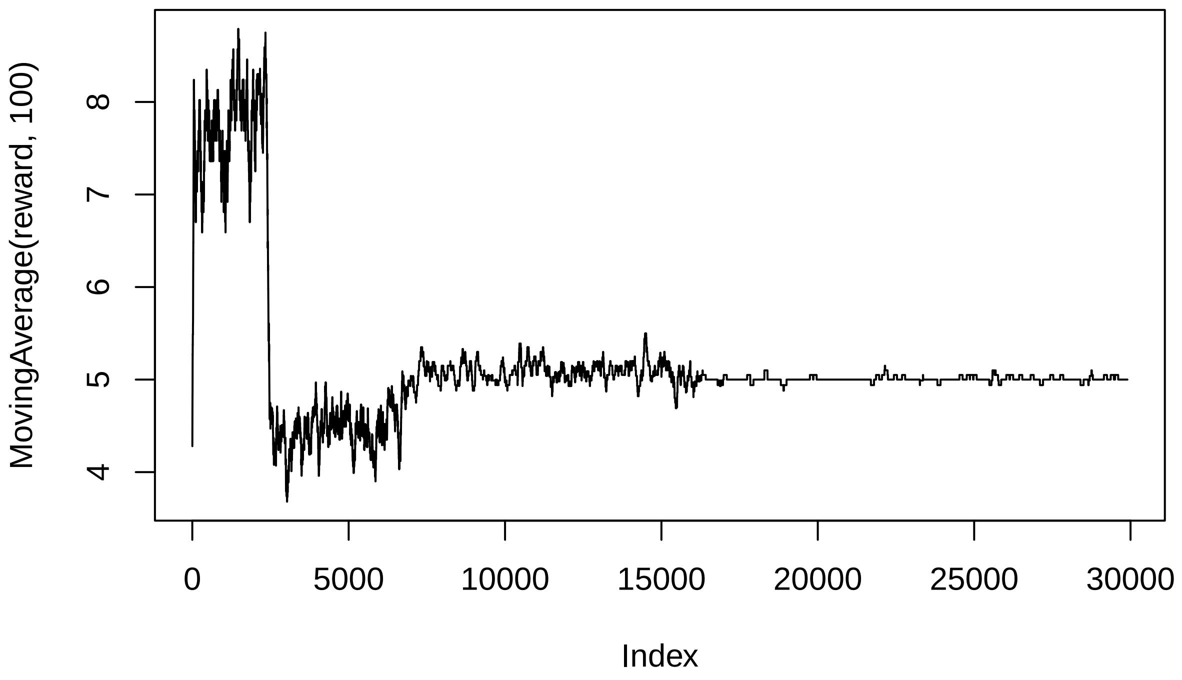

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10
9
8
7
6
5

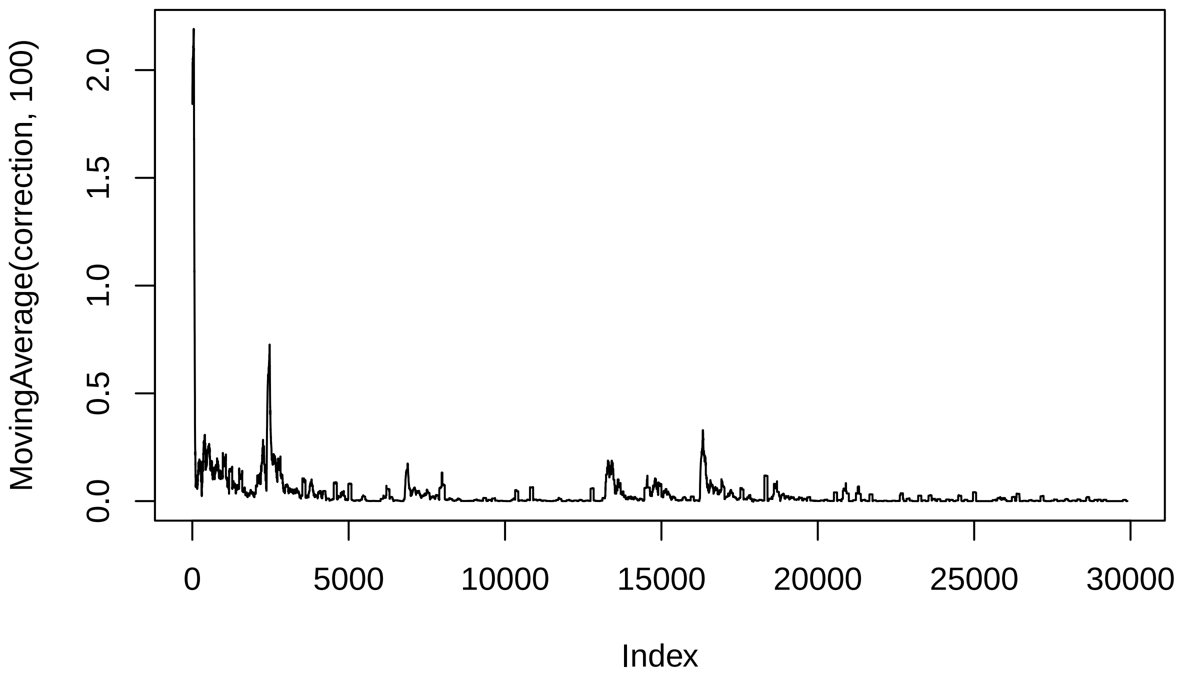

MovingAverage(reward, 100)
1
2
3
4
5
6
7
0
5000
10000
15000
20000
25000
30000
Index



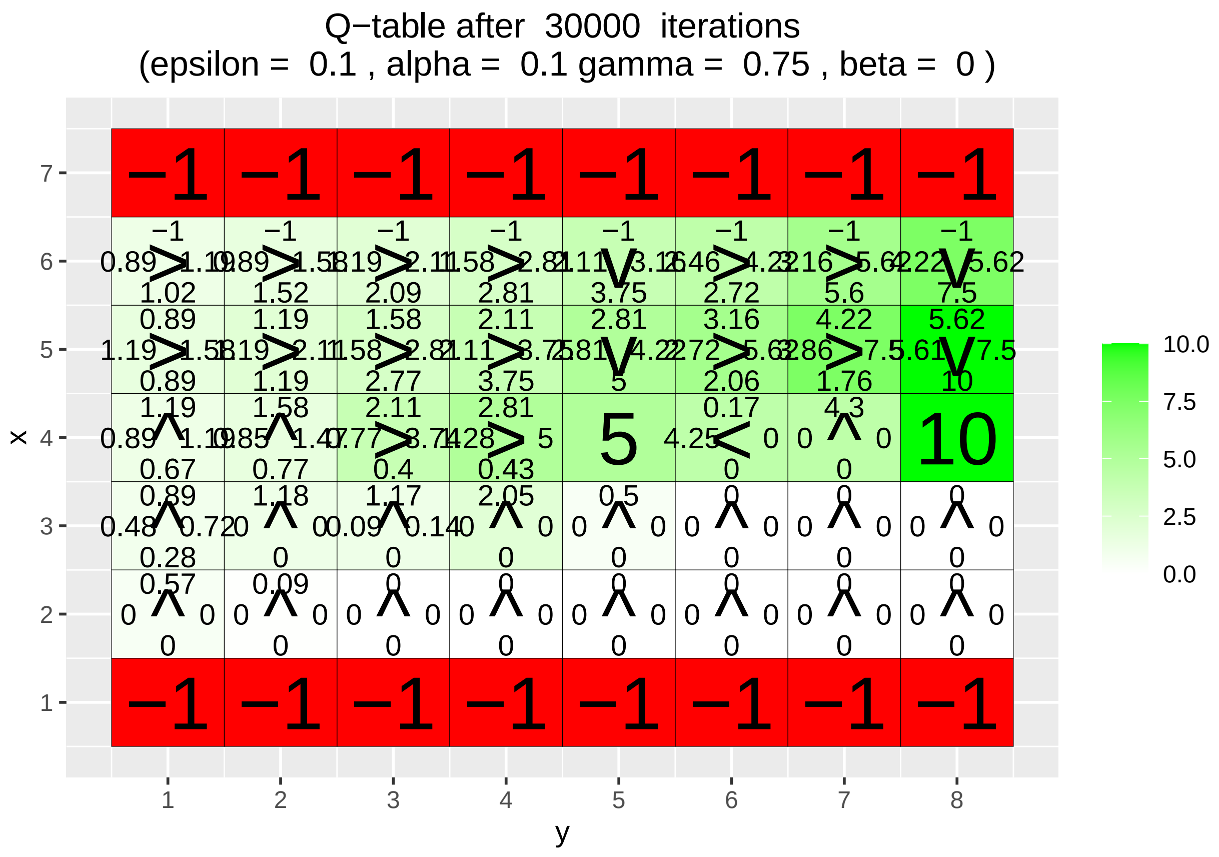
```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  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")
}
```

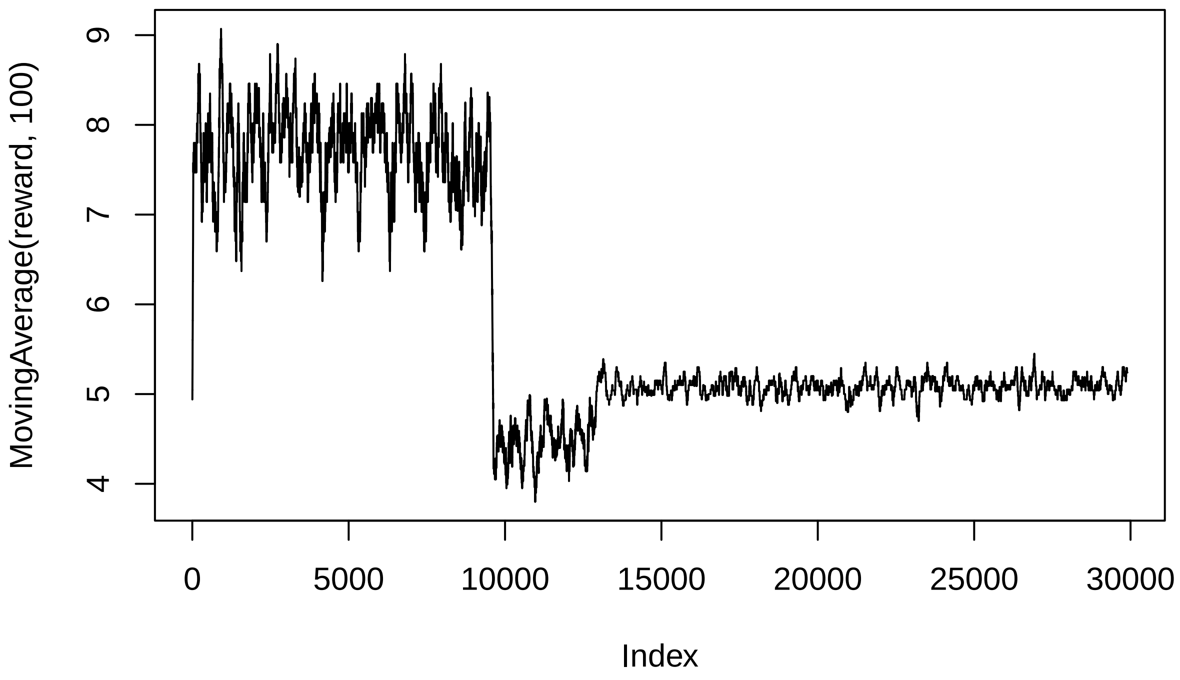

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0

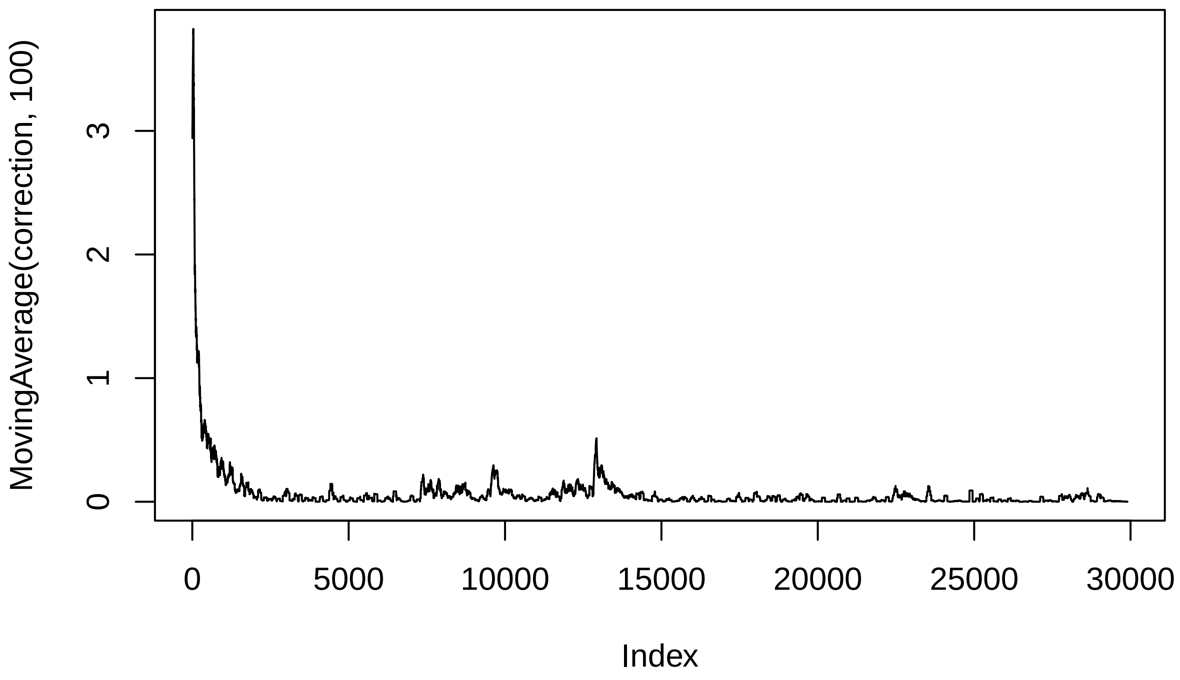

MovingAverage(reward, 100)
4
5
6
7
8
0
5000
10000
15000
20000
25000
30000
Index

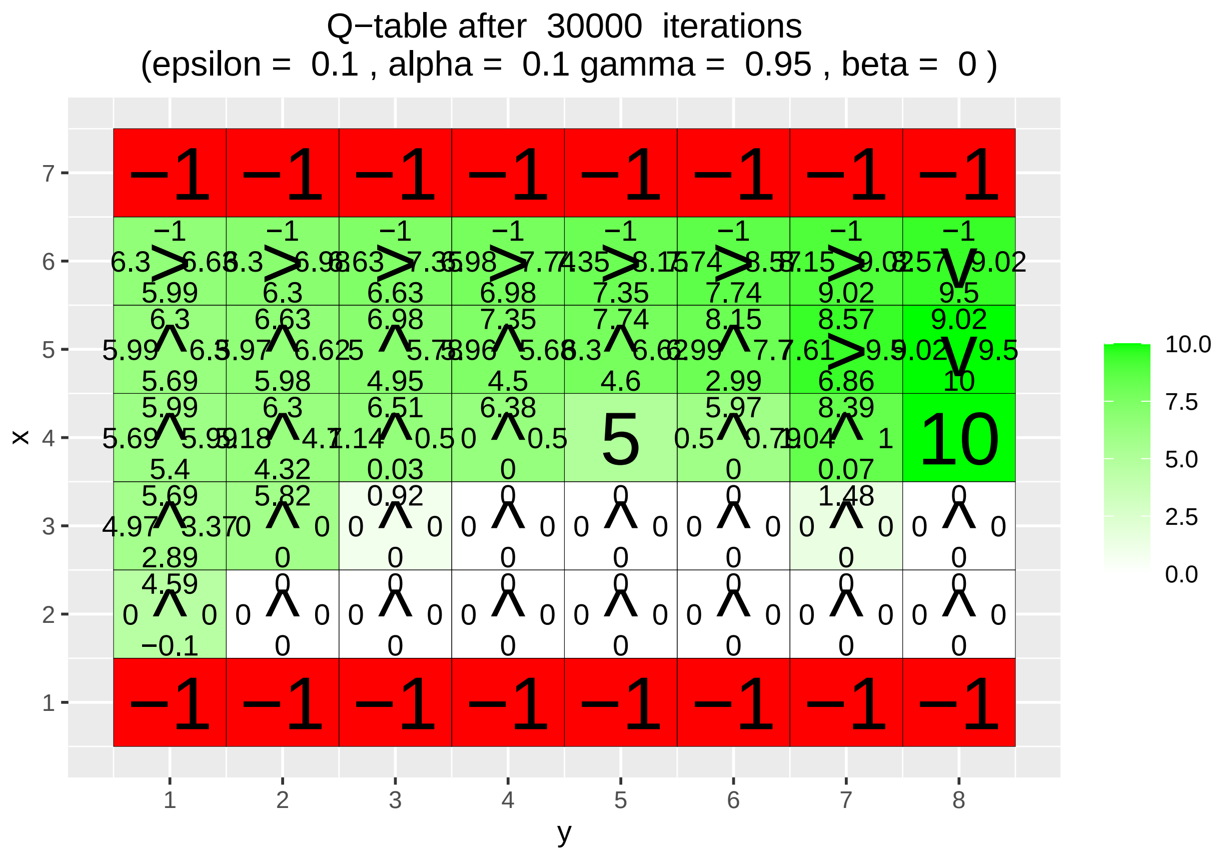


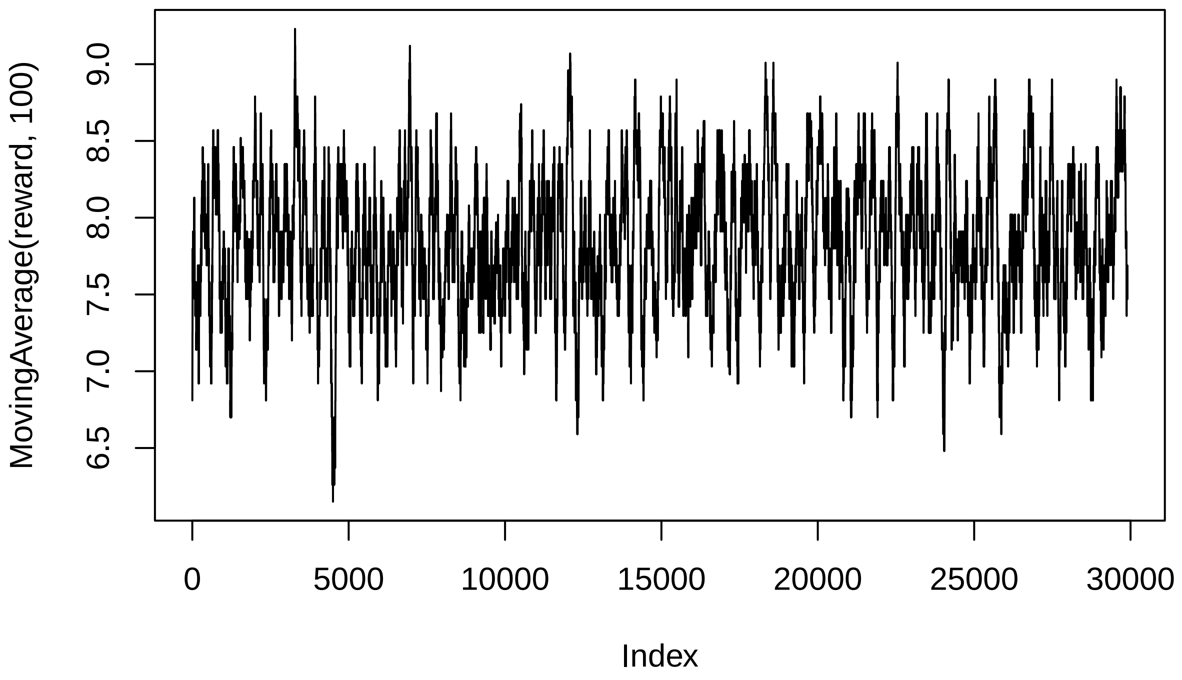
MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index

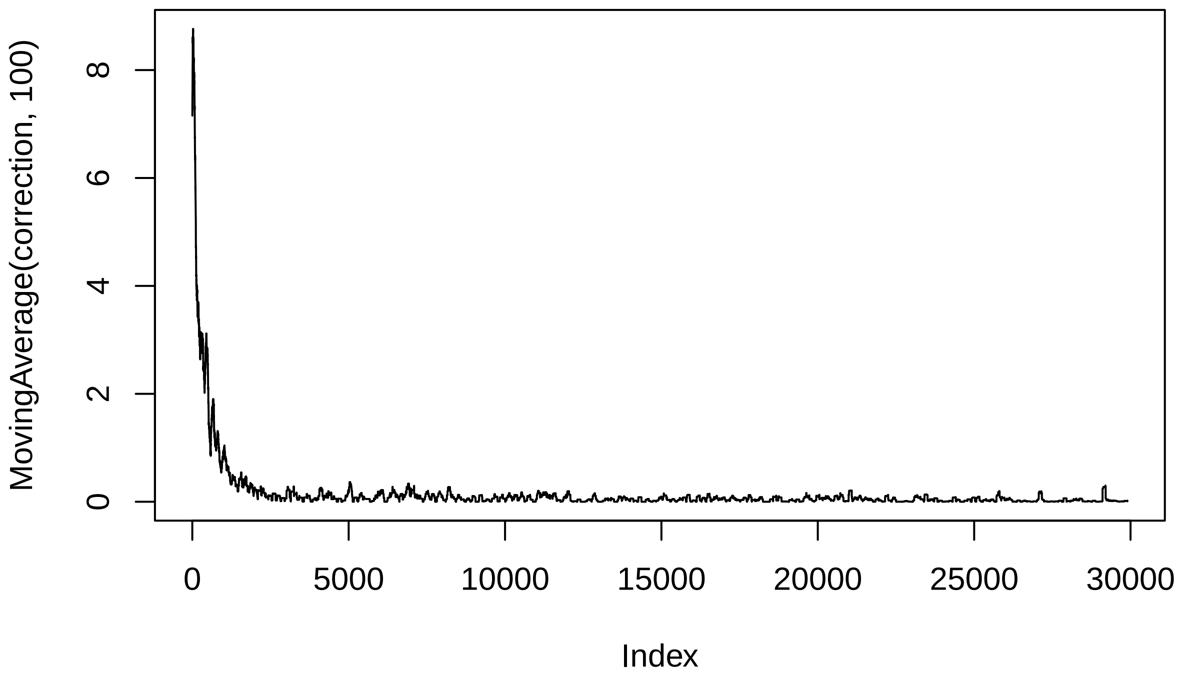

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0
−1
5
10


MovingAverage(reward, 100)
4
5
6
7
8
9
0
5000
10000
15000
20000
25000
30000
Index

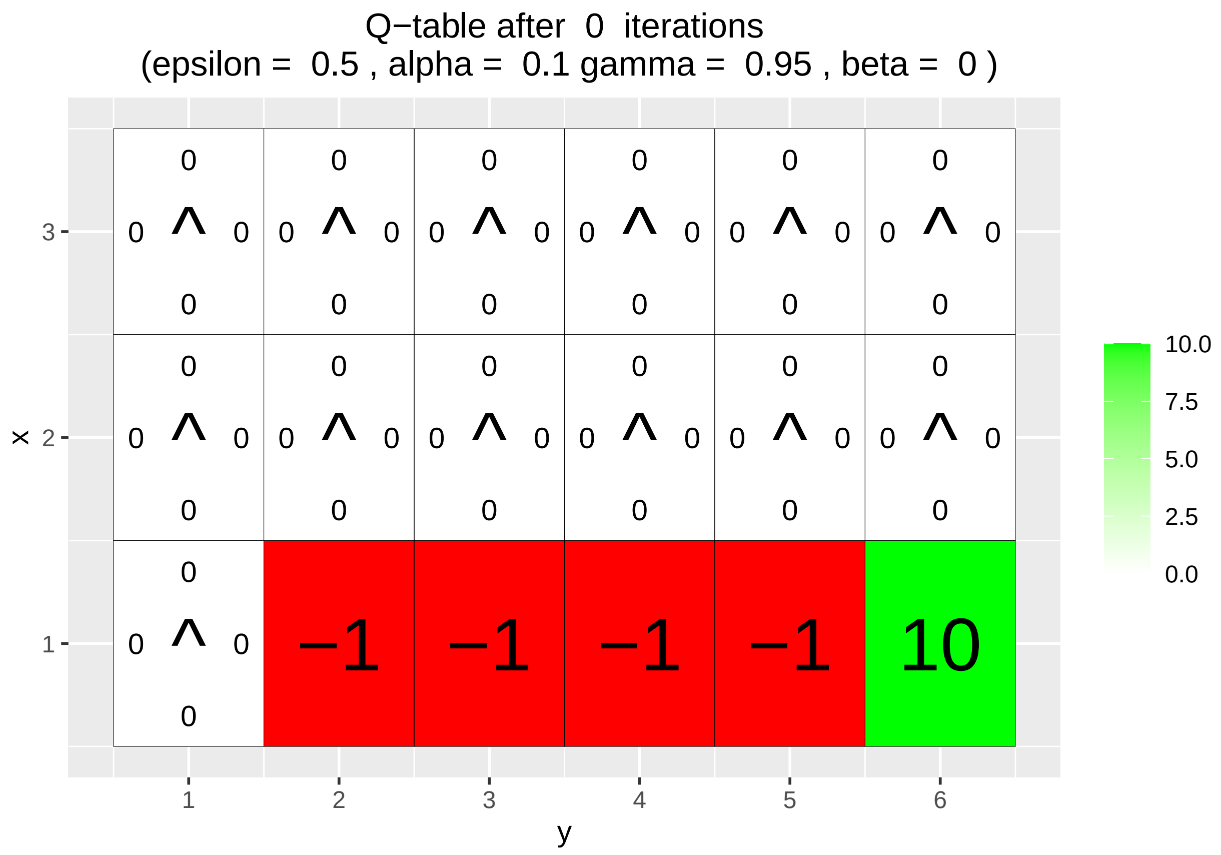

MovingAverage(correction, 100)
0
1
2
3
0
5000
10000
15000
20000
25000
30000
Index


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

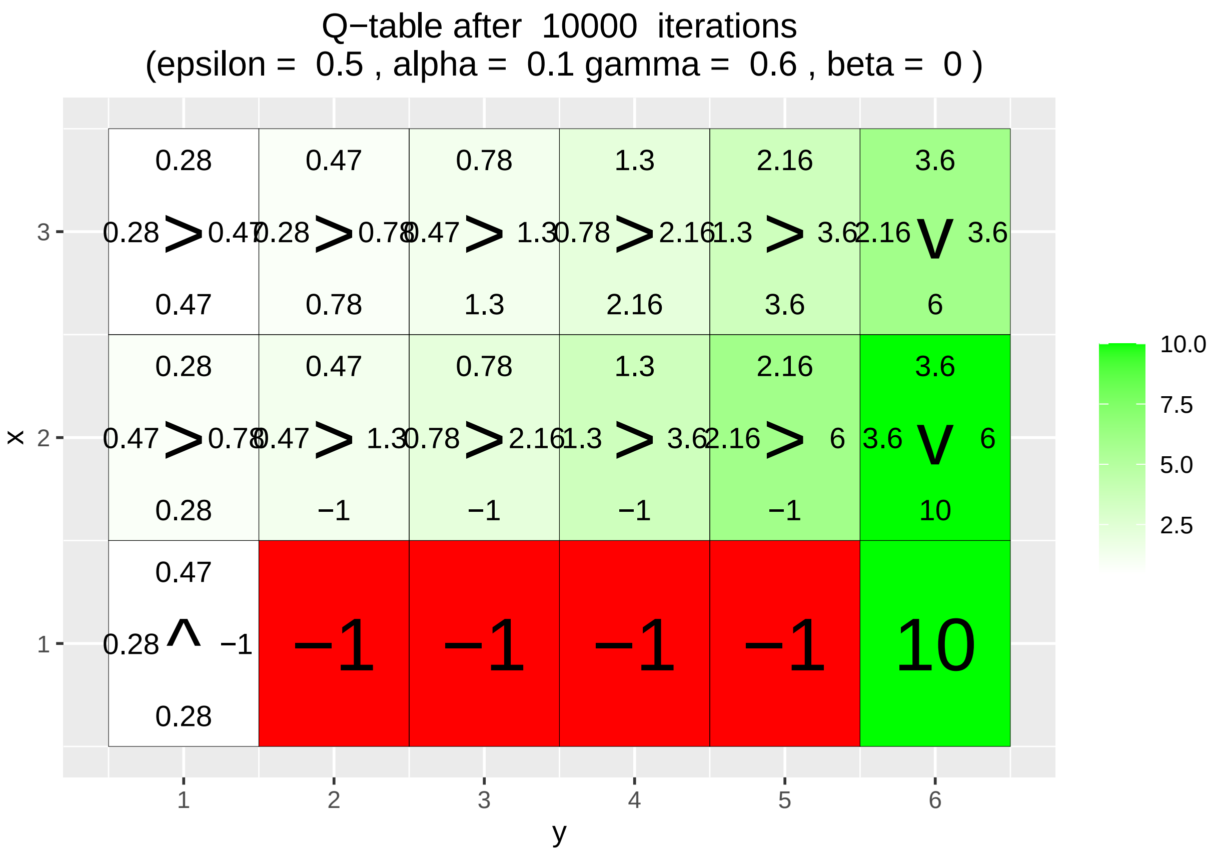

MovingAverage(reward, 100)
6.5
7.0
7.5
8.0
8.5
9.0
0
5000
10000
15000
20000
25000
30000
Index

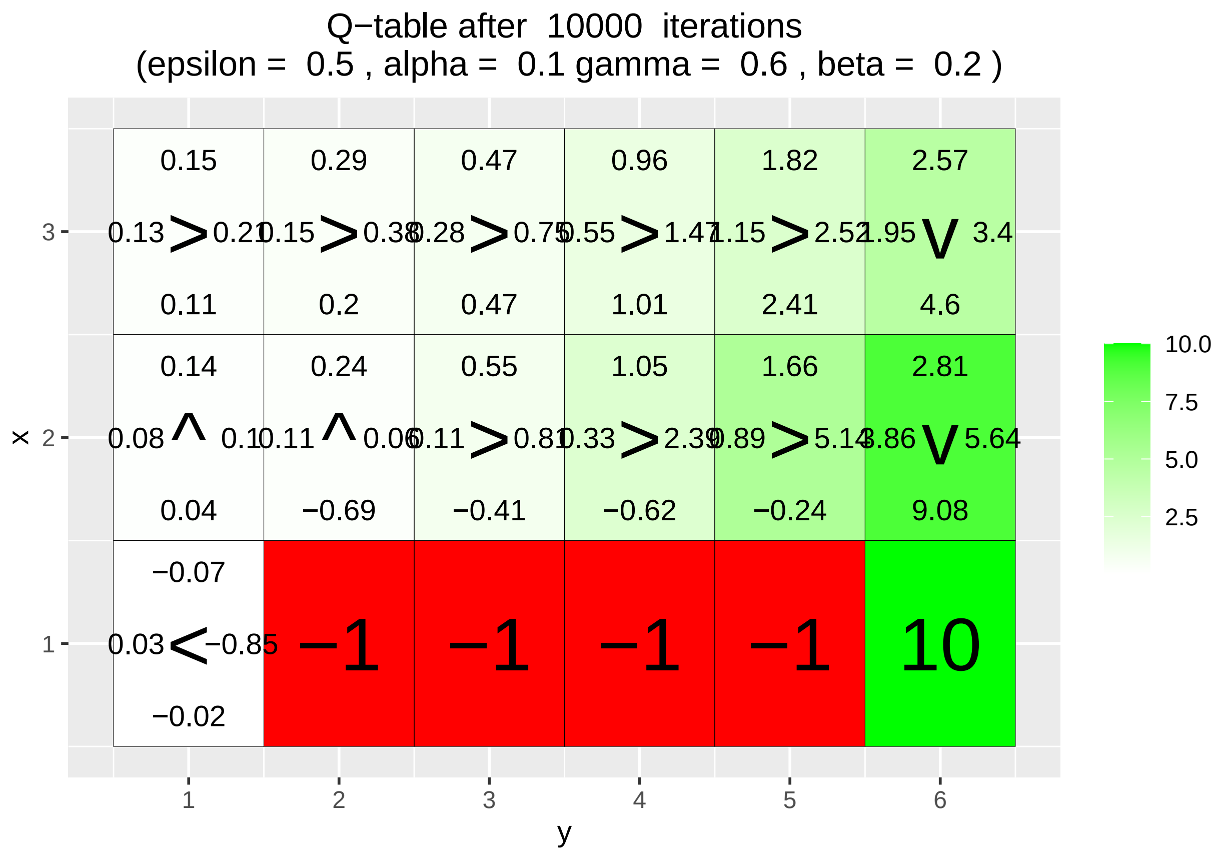


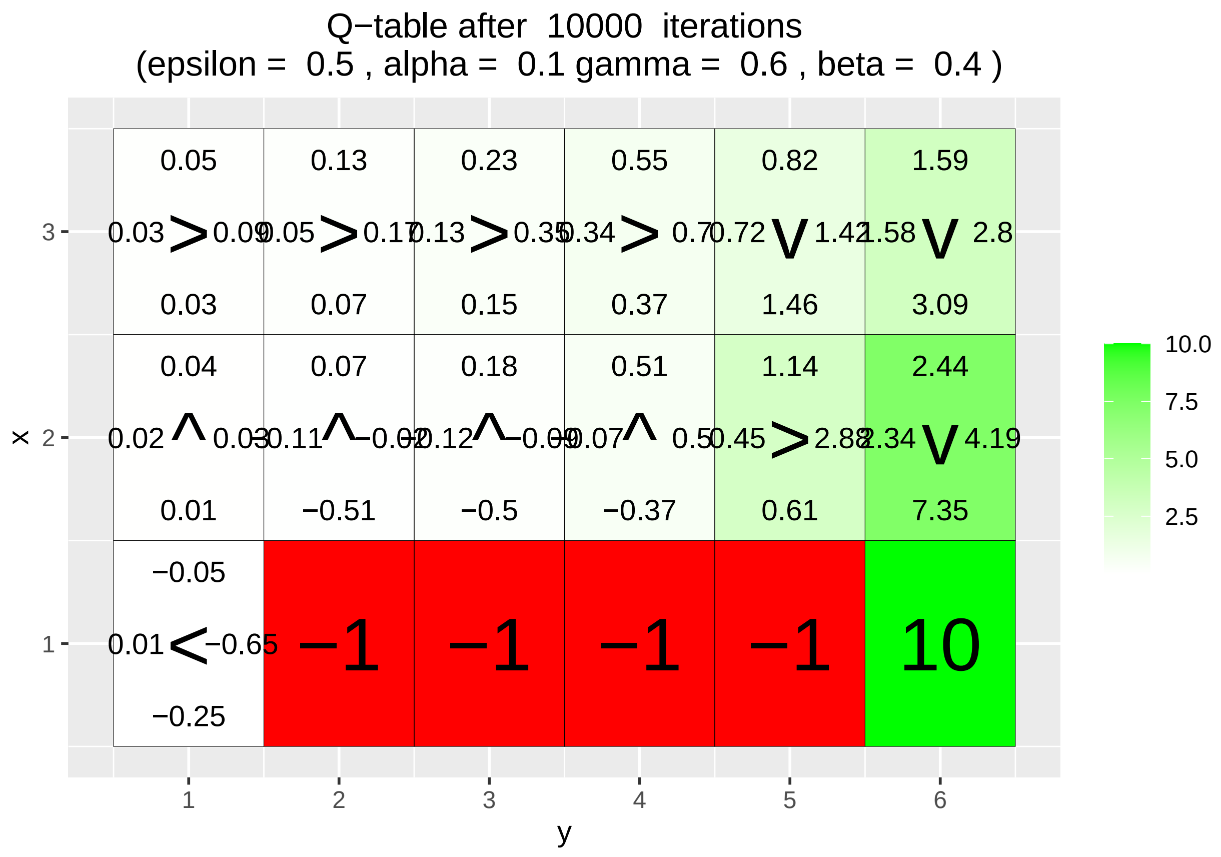
```
# Environment C (the effect of beta).

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()
```

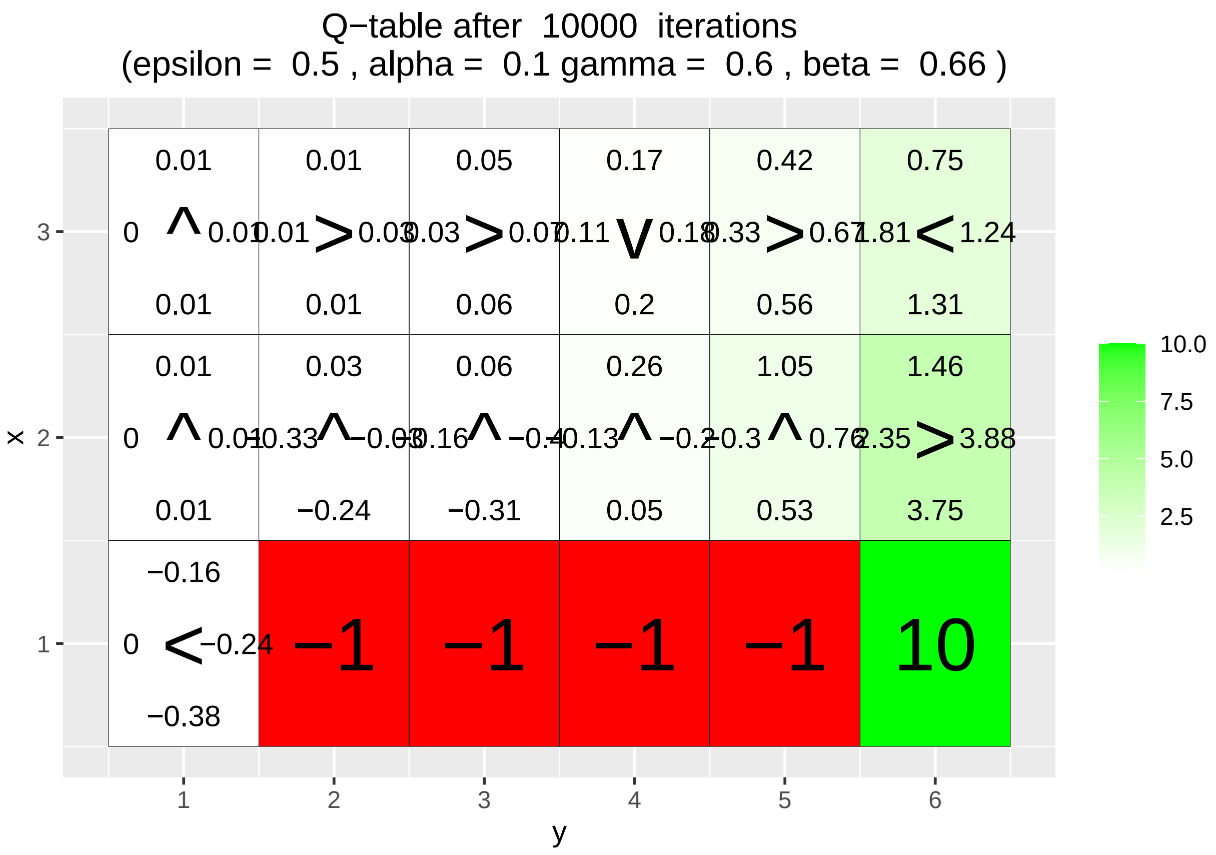


```r
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

  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)
x
y
1
2
3
4
5
6
10.0
7.5
5.0
2.5
−1
10


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)
0.15
0.13 > 0.21
0.11
0.29
0.15 > 0.38
0.2
0.47
0.28 > 0.75
0.47
0.96
0.55 > 1.47
1.01
1.82
1.15 > 2.52
2.41
2.57
1.95 V 3.4
4.6
0.14
0.08 ^ 0.1
0.04
0.24
0.11 ^ 0.06
−0.69
0.55
0.11 > 0.81
−0.41
1.05
0.33 > 2.39
−0.62
1.66
0.89 > 5.14
−0.24
2.81
3.86 V 5.64
9.08
−0.07
0.03 < −0.85
−0.02
−1
−1
−1
−1
10
x
1
2
3
y
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 = 0.4)
0.05 0.03 > 0.09 0.03
0.13 0.05 > 0.17 0.07
0.23 0.13 > 0.35 0.15
0.55 0.34 > 0.7 0.37
0.82 0.72 V 1.42 1.46
1.59 1.58 V 2.8 3.09
0.04 0.02 ^ 0.03 0.01
0.07 −0.11 ^ −0.02 −0.51
0.18 −0.12 ^ −0.09 −0.5
0.51 −0.07 ^ 0.5 −0.37
1.14 0.45 > 2.88 0.61
2.44 2.34 V 4.19 7.35
−0.05 0.01 < −0.65 −0.25
−1
−1
−1
−1
10
10.0
7.5
5.0
2.5
x
y
1
2
3
4
5
6


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.66)
0.01
0.01
0.05
0.17
0.42
0.75
0
0.01
0.01
0.03
0.03
0.07
0.11
0.18
0.33
0.67
1.81
1.24
0.01
0.01
0.06
0.2
0.56
1.31
0.01
0.03
0.06
0.26
1.05
1.46
0
0.01
−0.33
−0.03
−0.16
−0.4
−0.13
−0.2
−0.3
0.76
2.35
3.88
0.01
−0.24
−0.31
0.05
0.53
3.75
−0.16
0
−0.24
−0.38
−1
−1
−1
−1
10
10.0
7.5
5.0
2.5
x
y
1
2
3
4
5
6