TDDE15_Lab3_oskhi827

Oskar Hidén - oskhi827

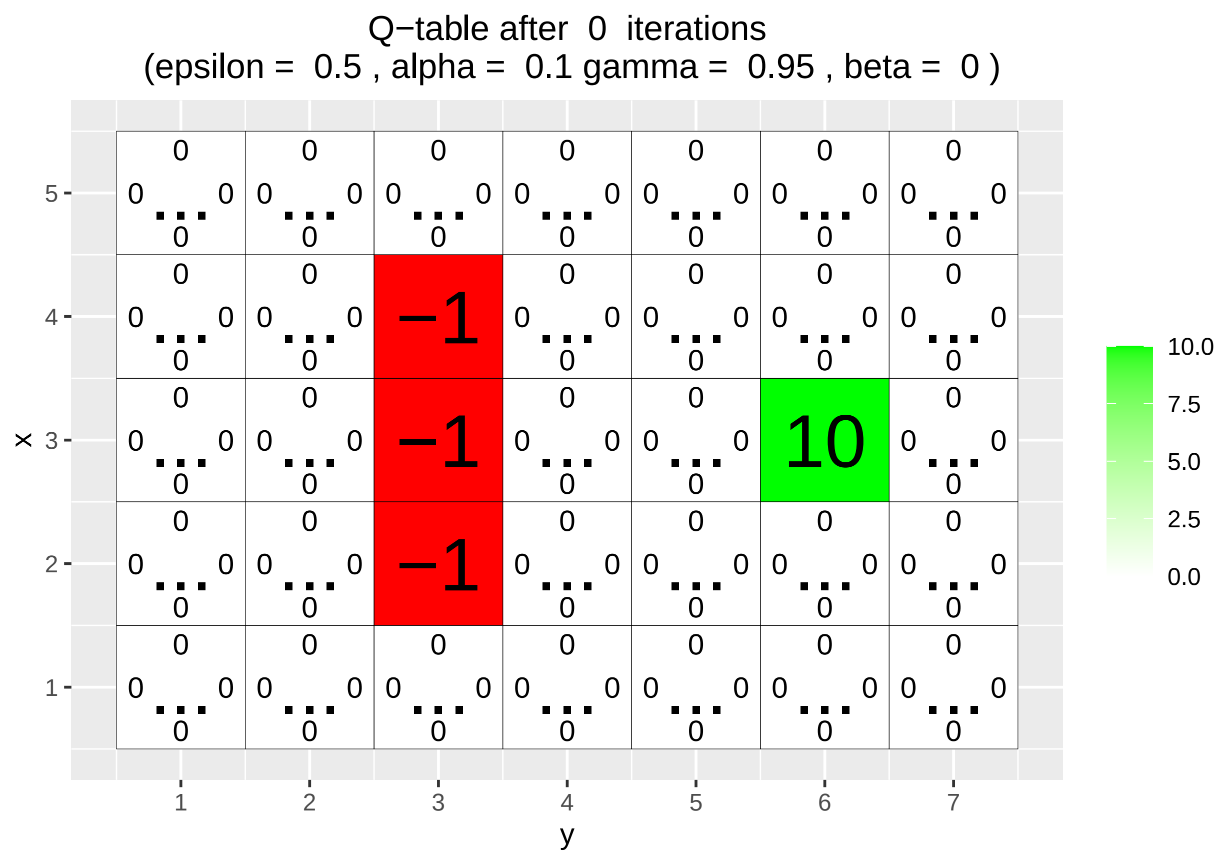
10/5/2020

```r
require(extrafont)
    # need only do this once!
    font_import(pattern="[A/a]rial", prompt=FALSE)
# 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)

# If you do not see four arrows in line 16, then do the following:
# File/Reopen with Encoding/UTF-8

arrows <- c("↑", "→", "↓", "←")
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)
```

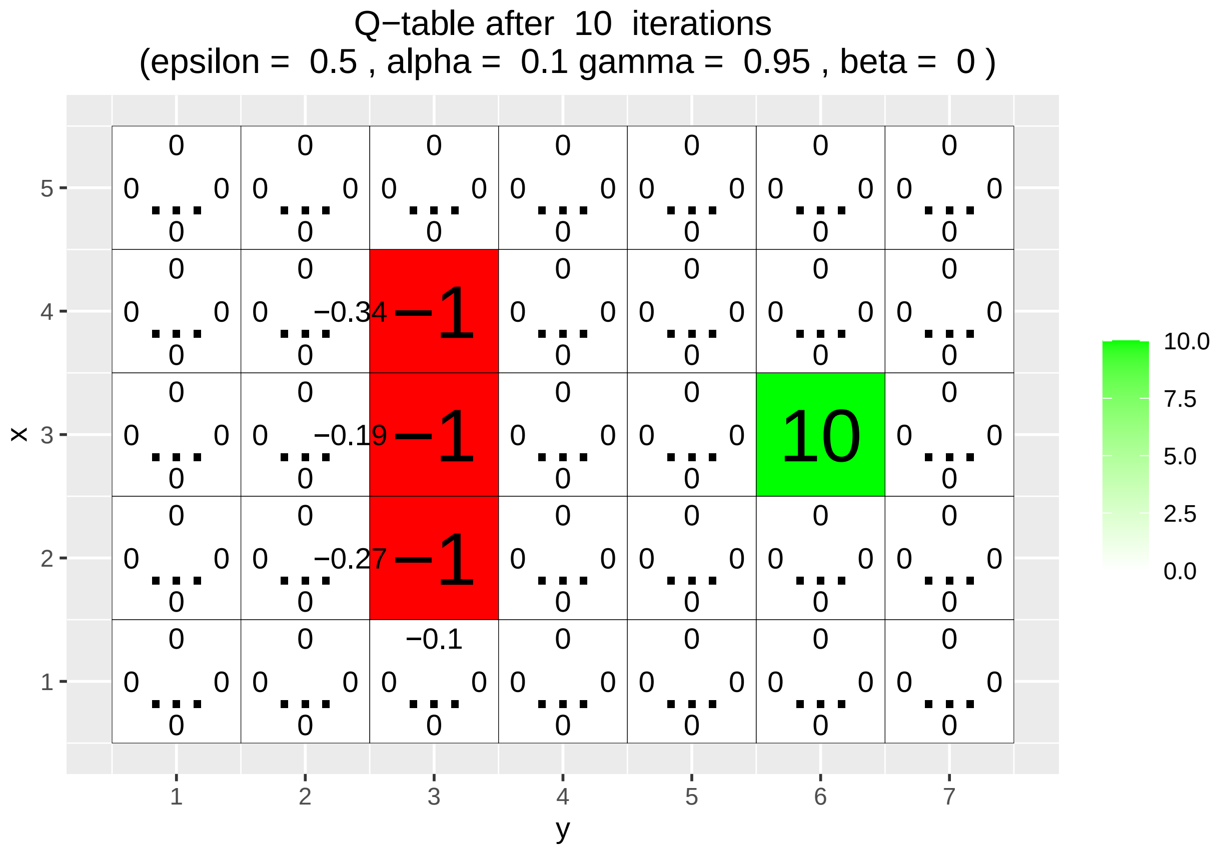
```r
  df$val3 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,4],NA),df$x,df$y)
  df$val4 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y)
    ifelse(reward_map[x,y] == 0,arrows[GreedyPolicy(x,y)],reward_map[x,y]),df$x,df$y)
  df$val5 <- as.vector(foo)
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,max(q_table[x,y,]),
                                     ifelse(reward_map[x,y]<0,NA,reward_map[x,y])),df$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.
  return(which.max(rank(q_table[x,y,], ties.method = "random")))
}

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}.
```

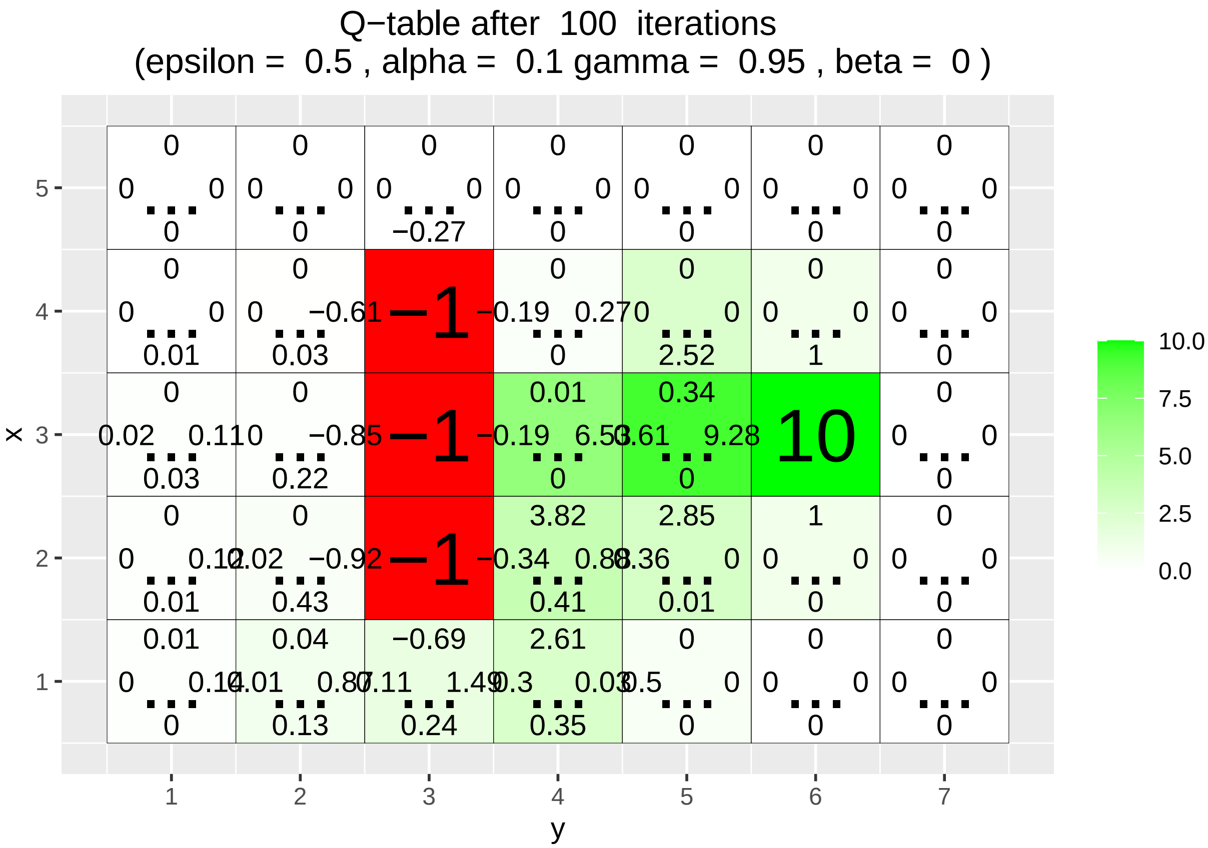
```r
  # Your code here.
  if(runif(1)>epsilon){
    direction = GreedyPolicy(x,y)
  }else{
    direction = sample(1:4,1)
  }

  return(direction)
}

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 <<-.
```

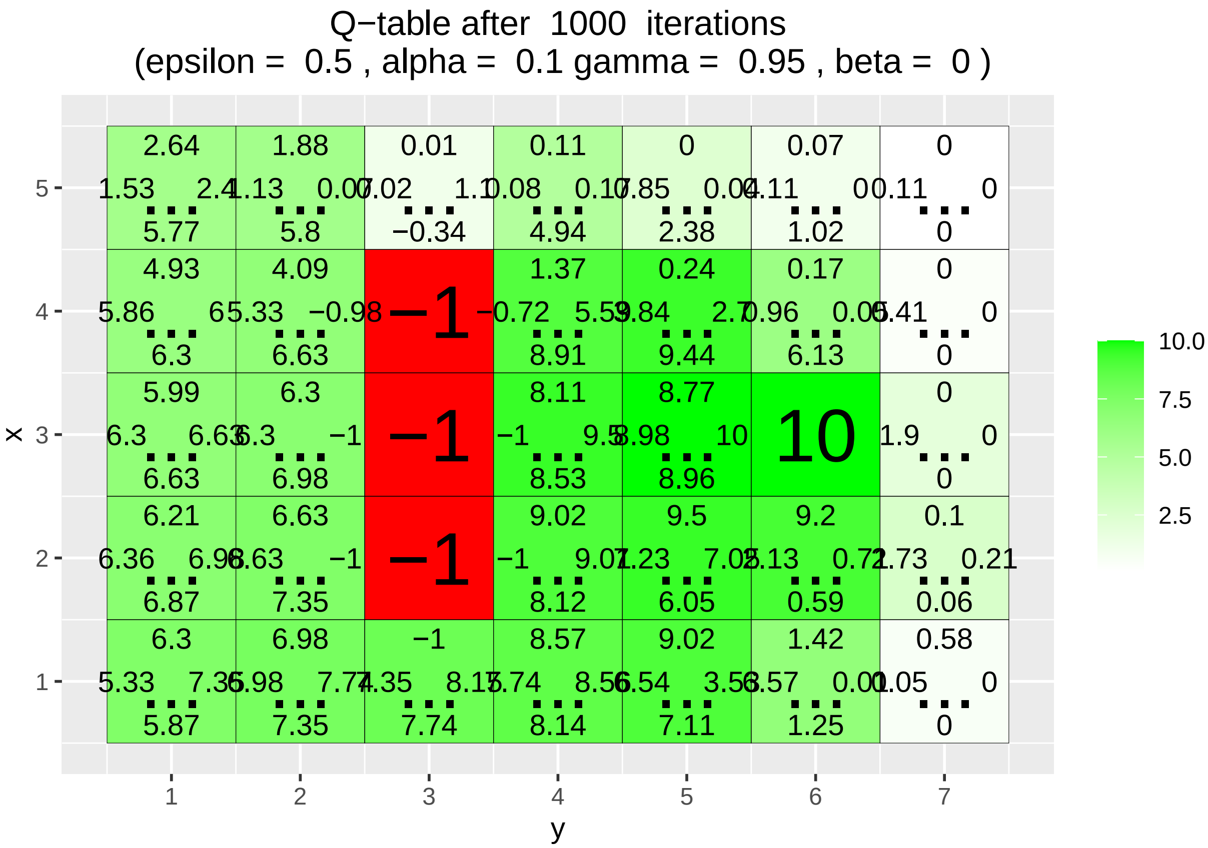
```
  # Your code here.
  current_state = start_state
  episode_correction = 0
  repeat{
    # Follow policy, execute action, get reward.
    action = EpsilonGreedyPolicy(current_state[1], current_state[2], epsilon)
    new_state = transition_model(current_state[1], current_state[2], action, beta)
    reward = reward_map[new_state[1], new_state[2]]

    # Q-table update.
    q_action_value = q_table[current_state[1], current_state[2], action]
    next_exp_r = max(q_table[new_state[1], new_state[2],])

    correction = gamma*next_exp_r - q_action_value
    q_table[current_state[1], current_state[2],action] <<- q_action_value + alpha*(reward + correction)
    episode_correction = episode_correction+correction
    current_state=new_state
    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))

  }

}

#####################################################################################################
# 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()
```



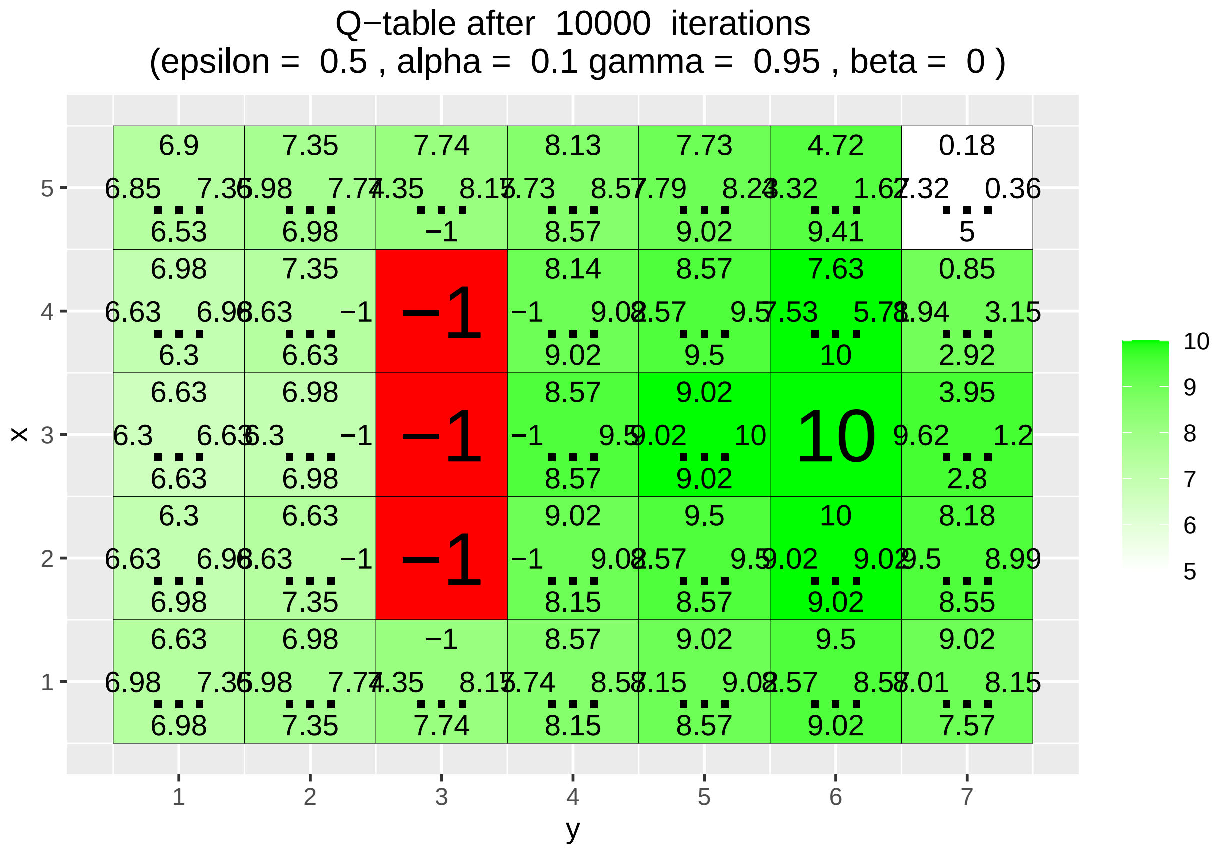

```
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))

  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)
−0.34
−0.19
−0.27
−0.1
−1
10
x
y
10.0
7.5
5.0
2.5
0.0

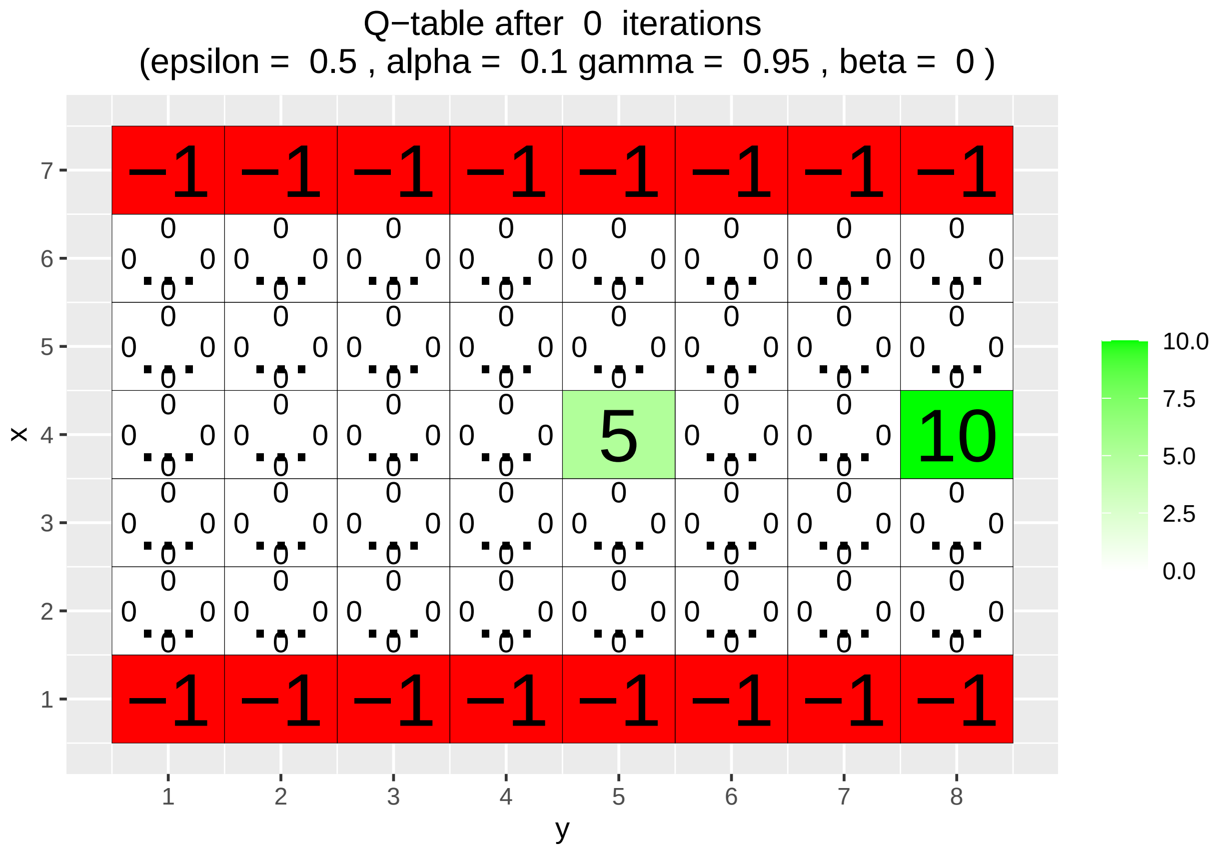

Q−table after 100 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10
−1
10.0
7.5
5.0
2.5
0.0



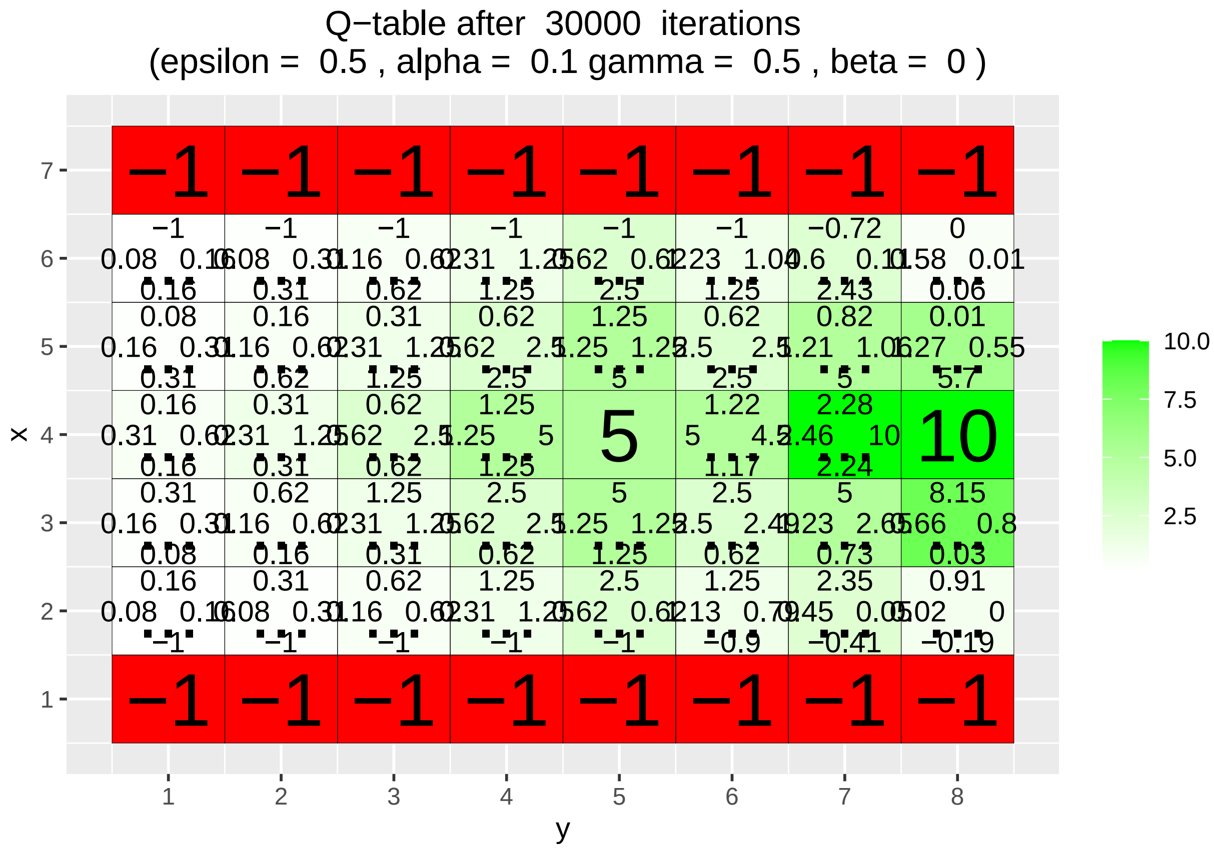
Q−table after 1000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
10
−1



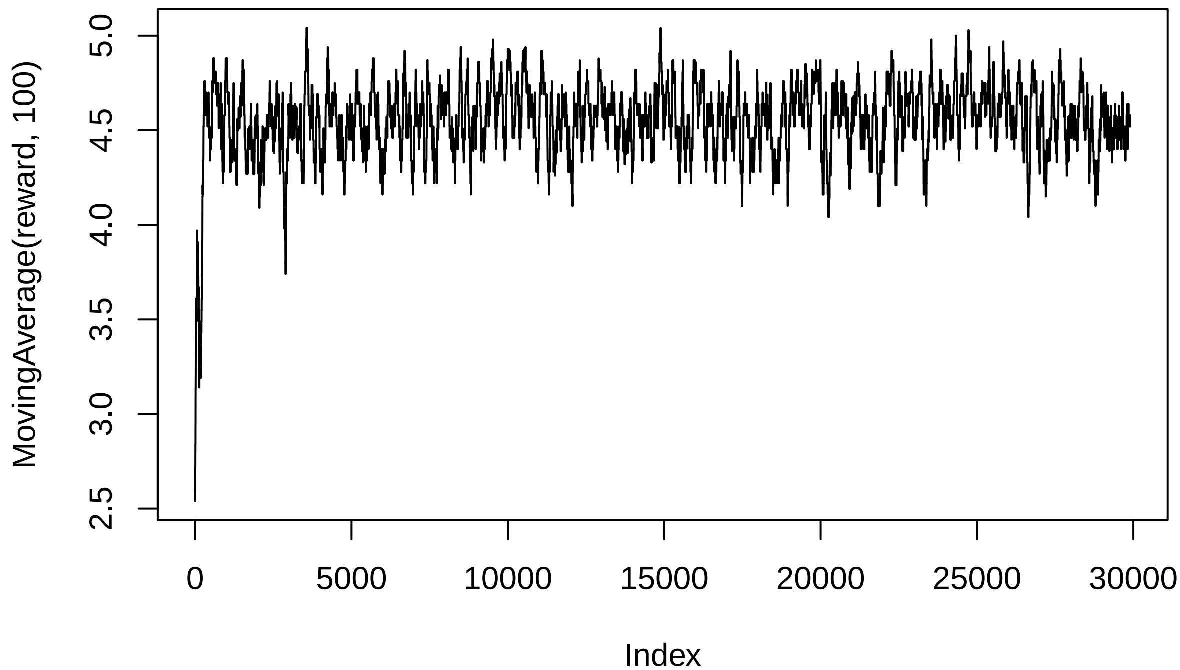

```r
# Environment B (the effect of epsilon and gamma)

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()
```



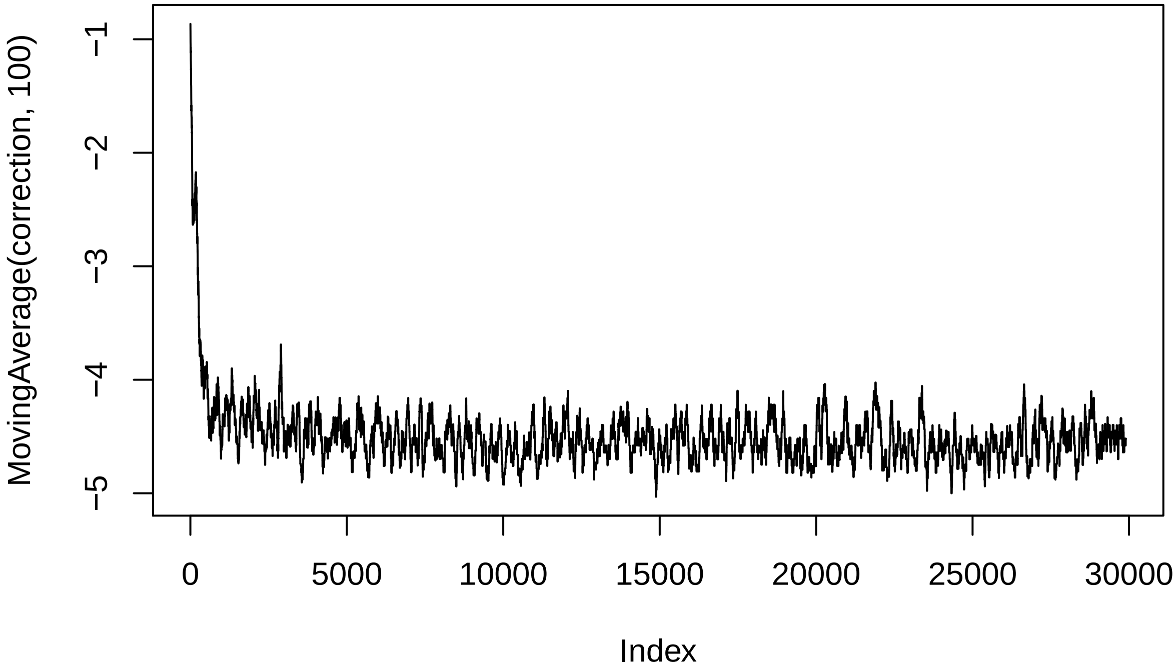

```
MovingAverage <- function(x, n){

  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")
}
```

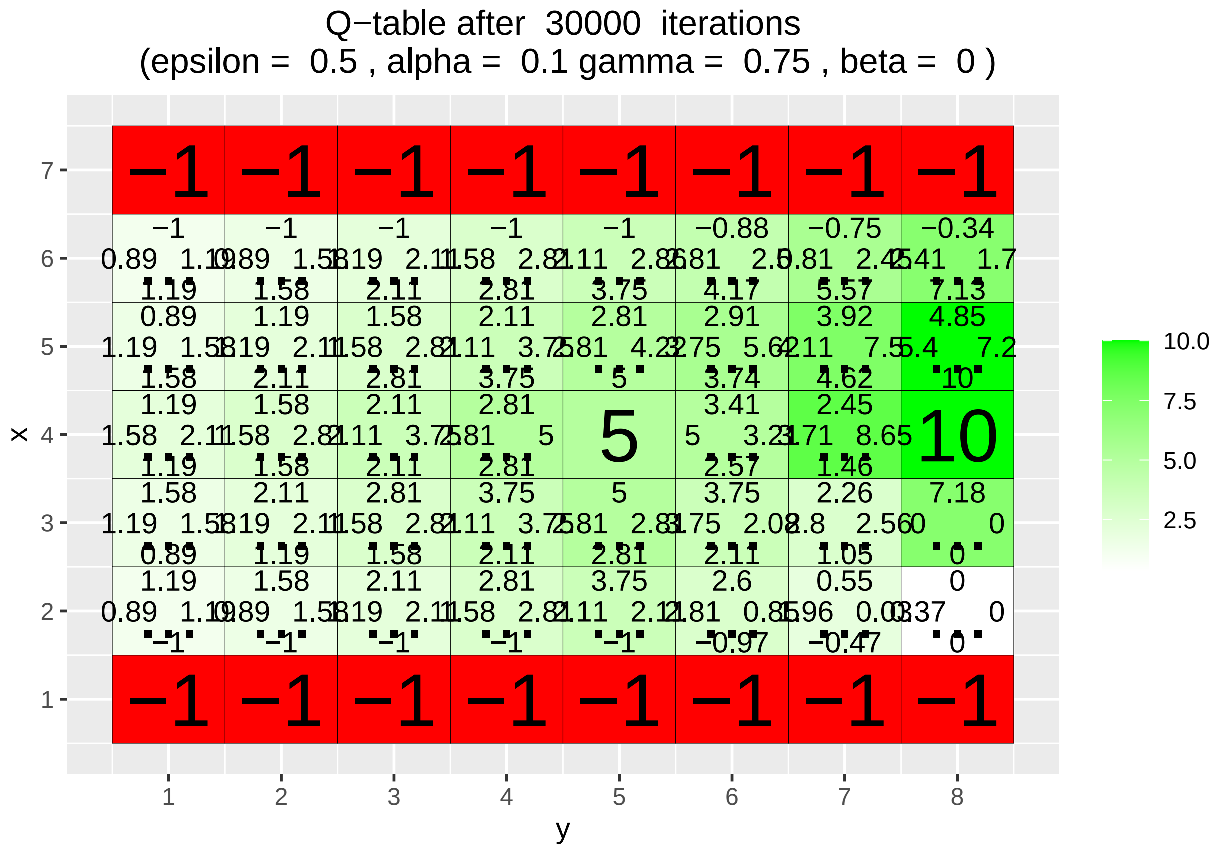


Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10.0
7.5
5.0
2.5

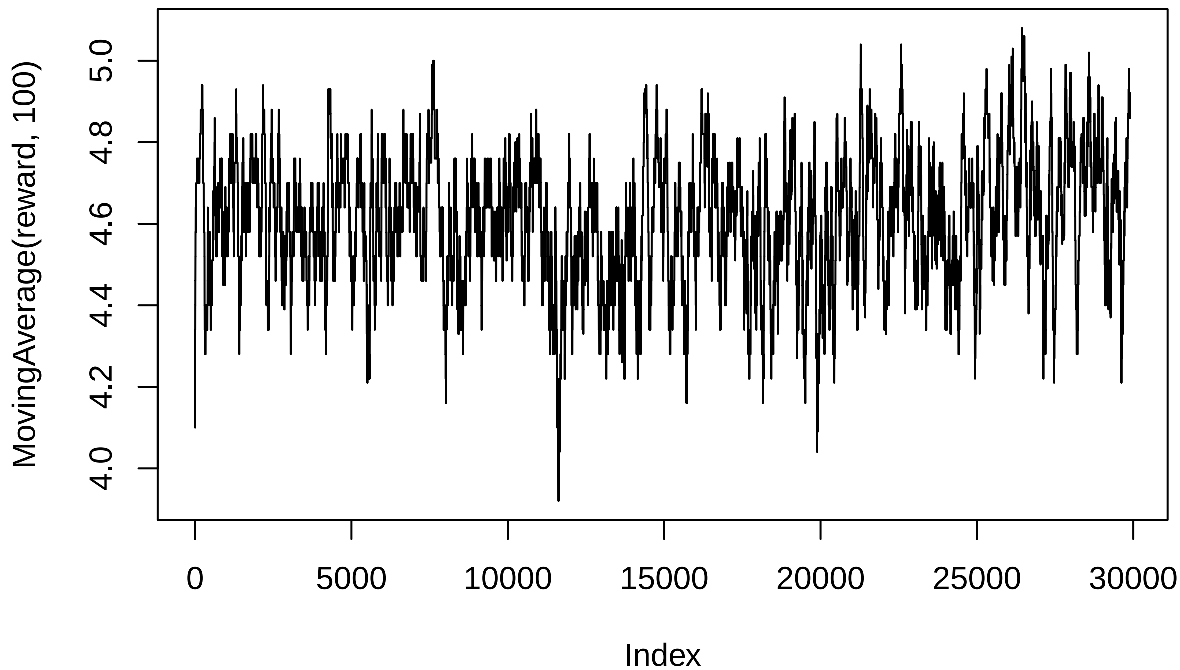


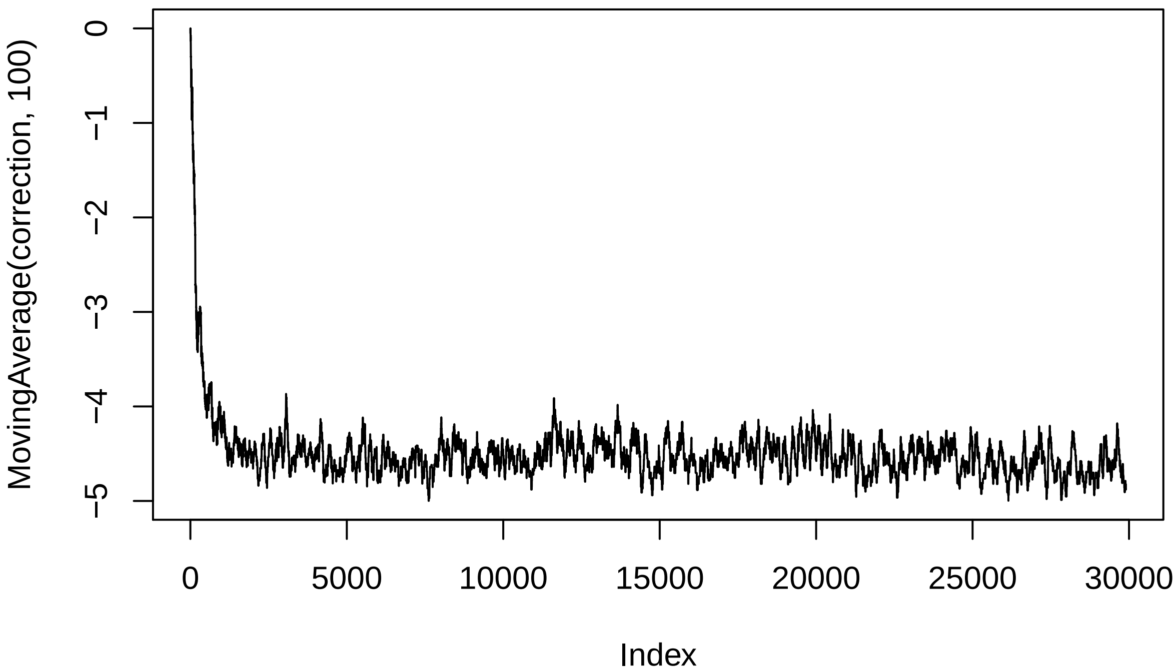
MovingAverage(reward, 100)
2.5
3.0
3.5
4.0
4.5
5.0
0
5000
10000
15000
20000
25000
30000
Index

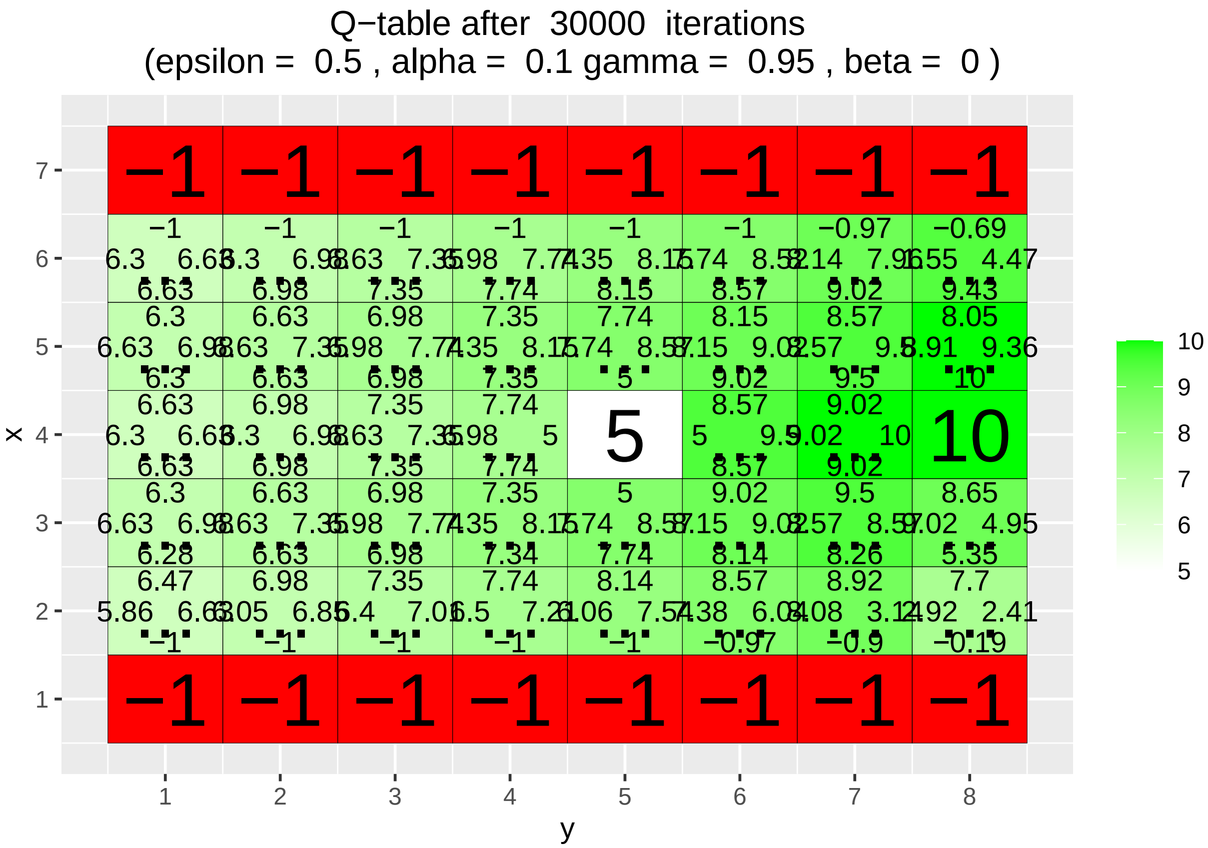

MovingAverage(correction, 100)
−1
−2
−3
−4
−5
0
5000
10000
15000
20000
25000
30000
Index



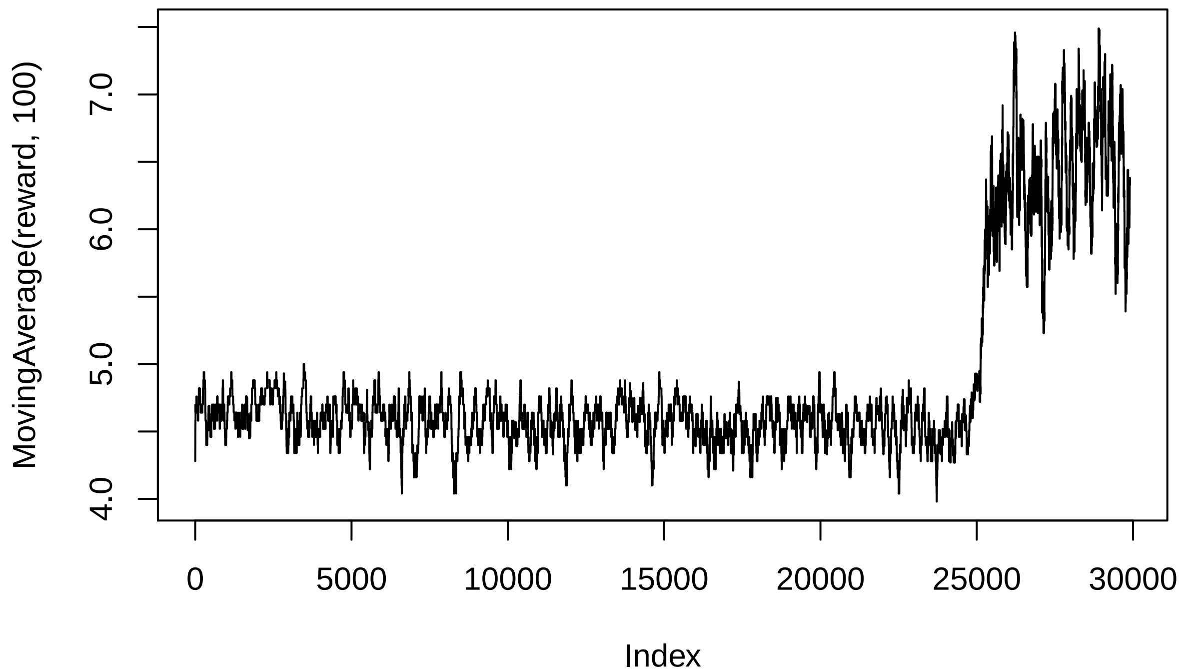
Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5


MovingAverage(reward, 100)
4.0
4.2
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index

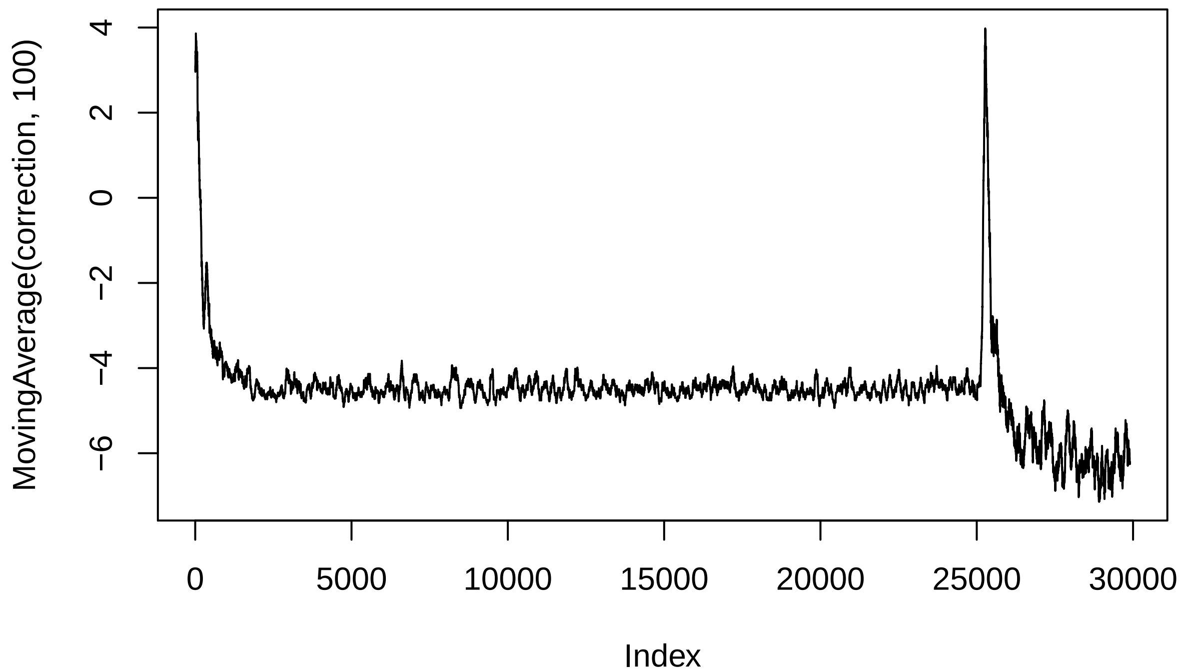

MovingAverage(correction, 100)
0
−1
−2
−3
−4
−5
0
5000
10000
15000
20000
25000
30000
Index



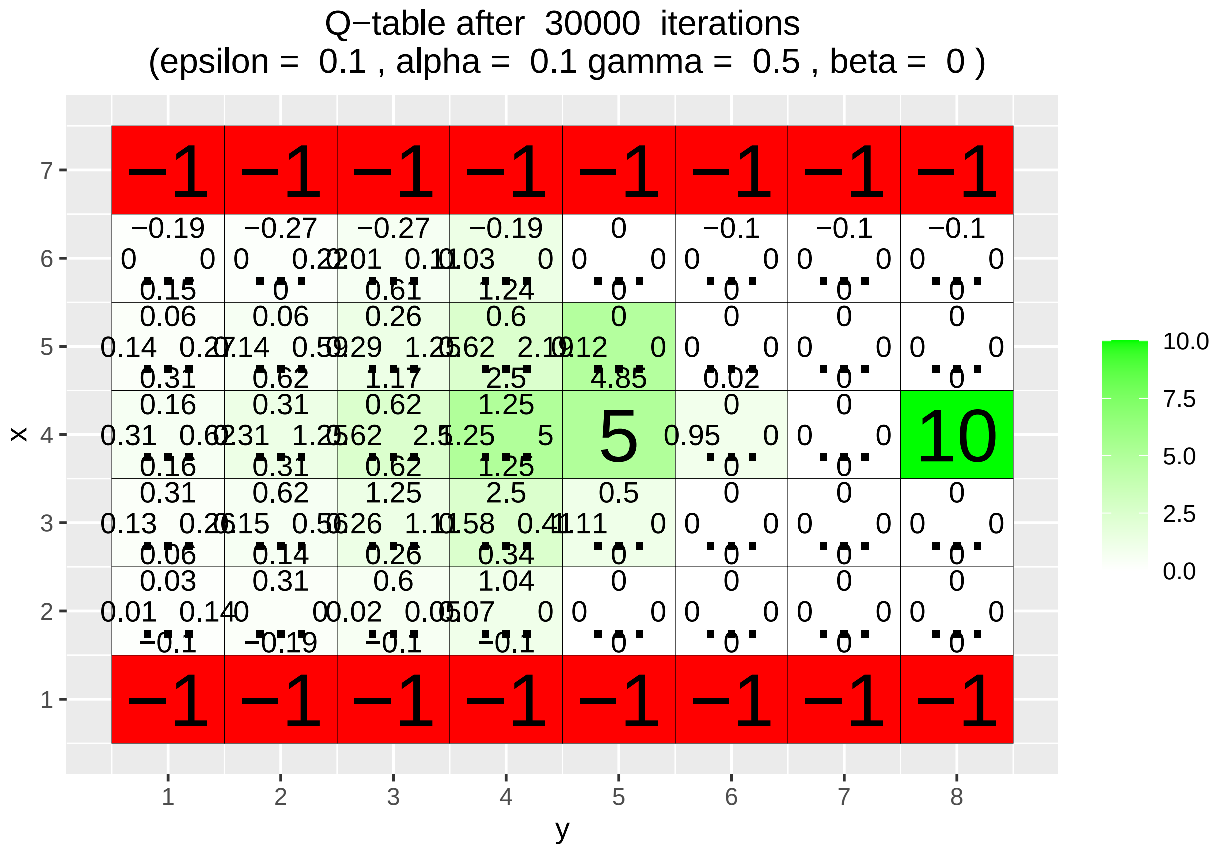
Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
5
10



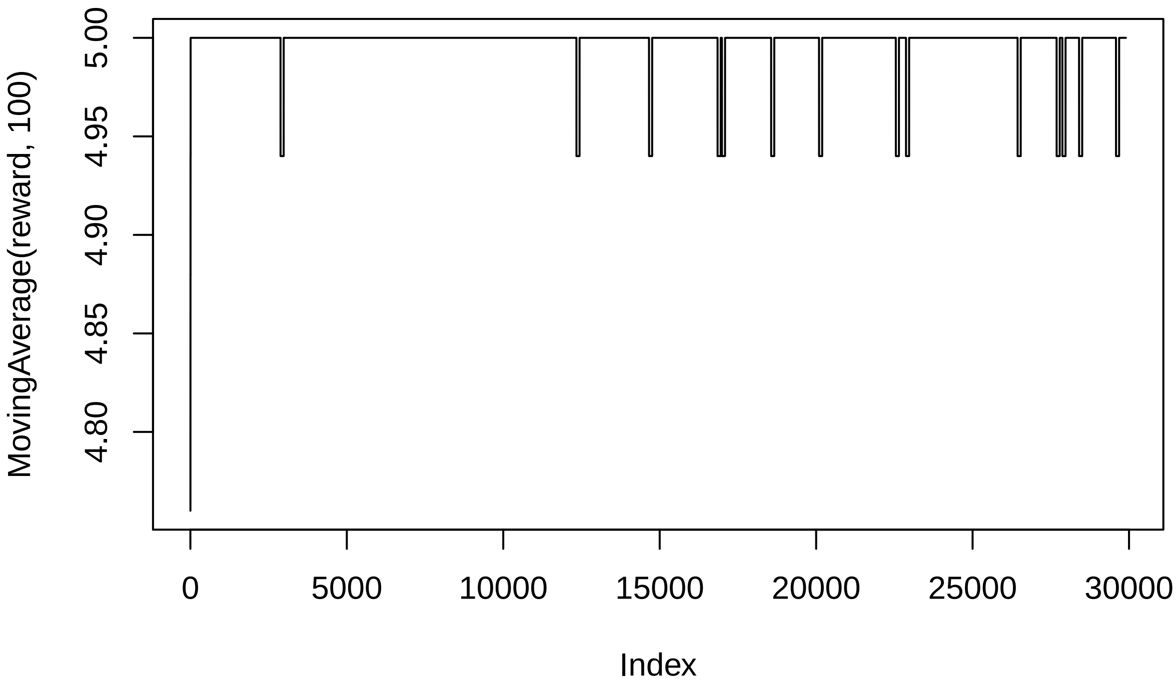
MovingAverage(reward, 100)
4.0
5.0
6.0
7.0
0
5000
10000
15000
20000
25000
30000
Index

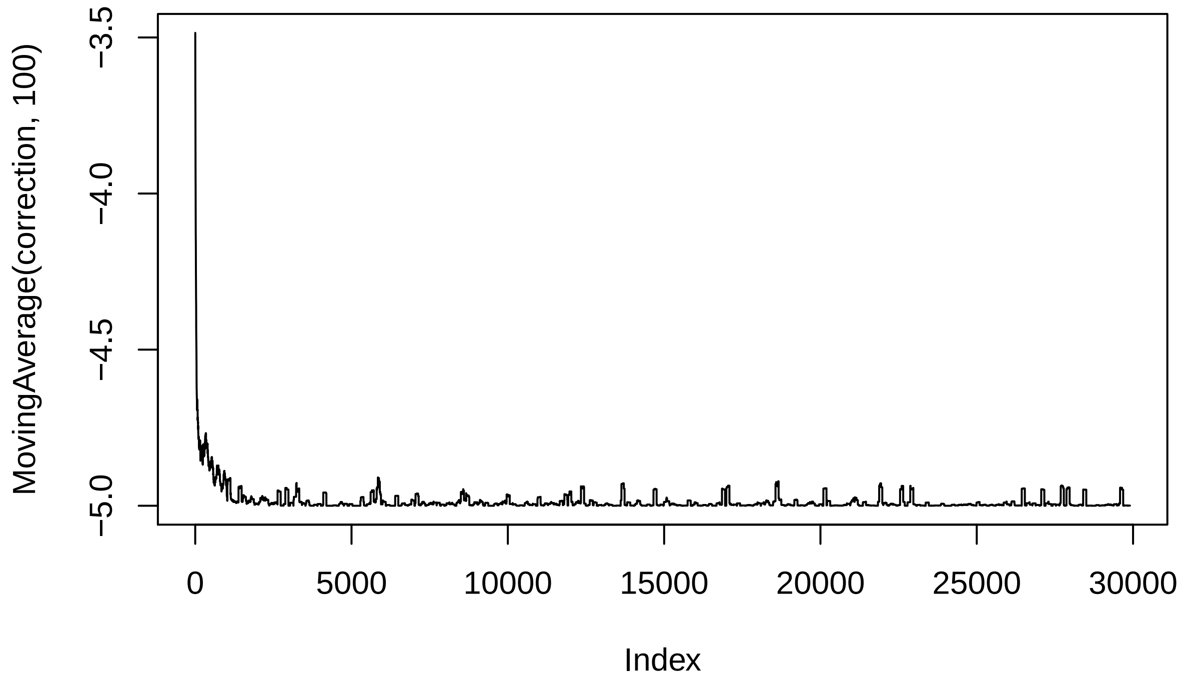



```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  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")
}
```



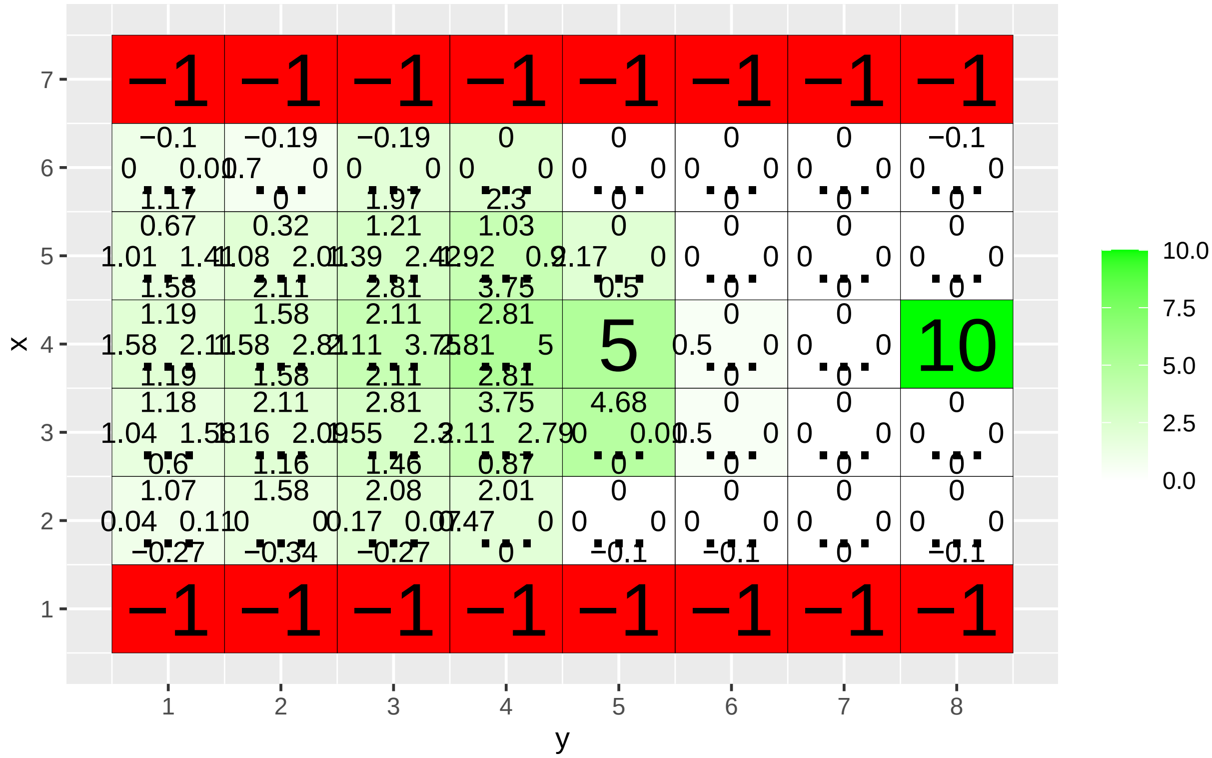
Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
5
10
10.0
7.5
5.0
2.5
0.0

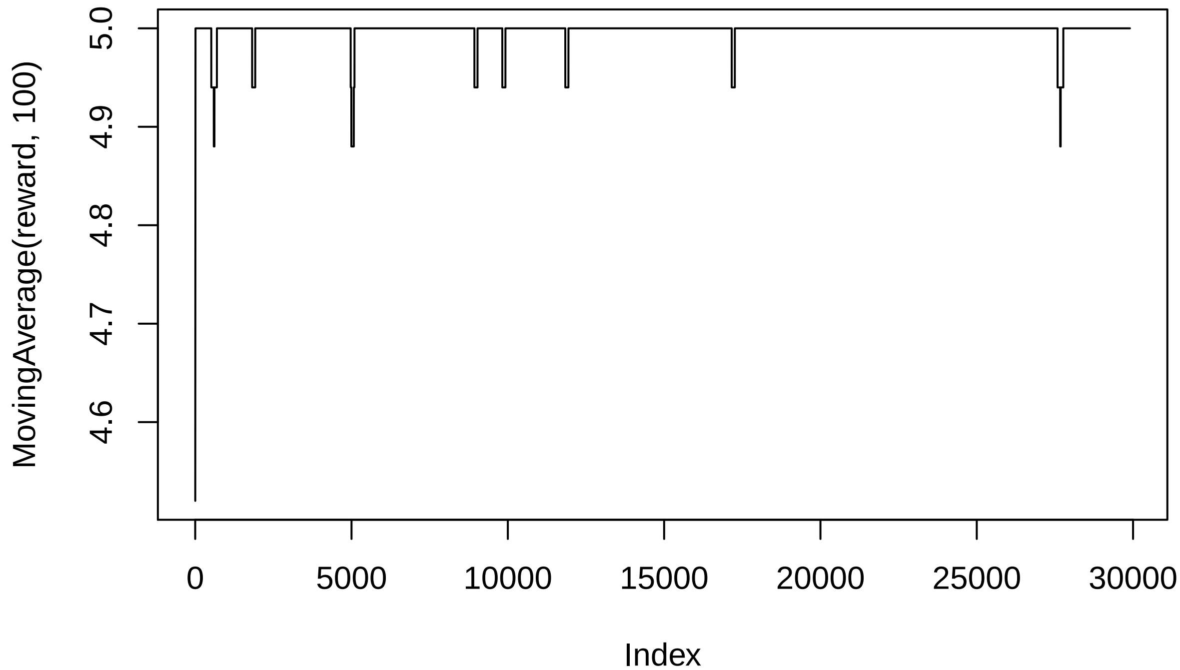

MovingAverage(reward, 100)
5.00
4.95
4.90
4.85
4.80
0
5000
10000
15000
20000
25000
30000
Index

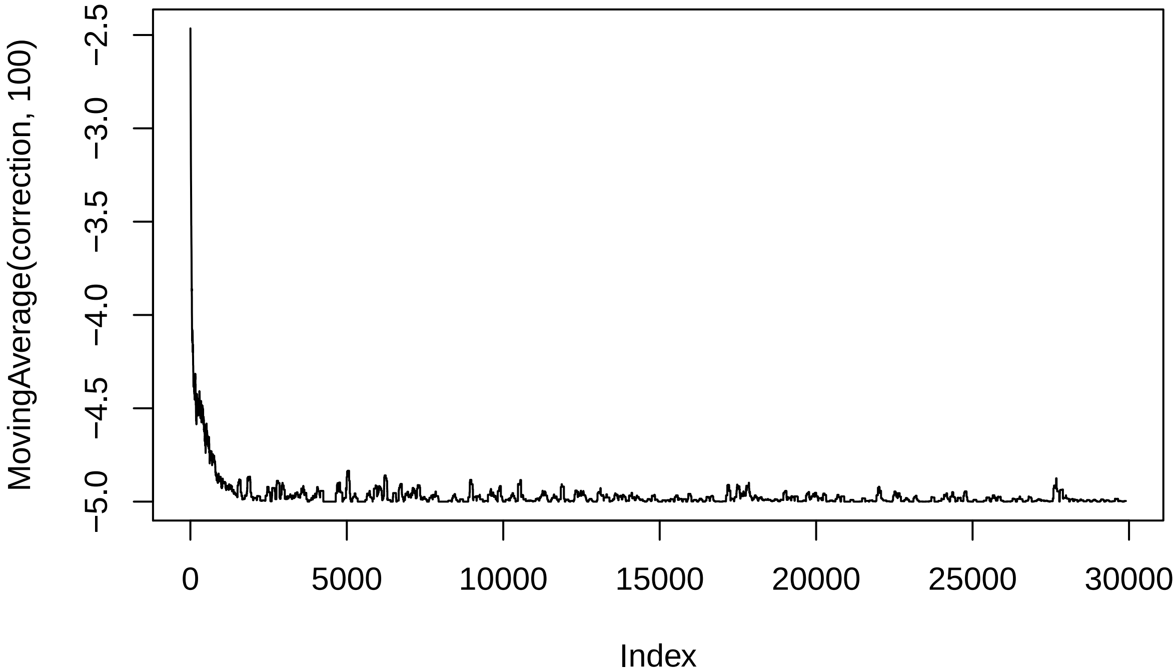


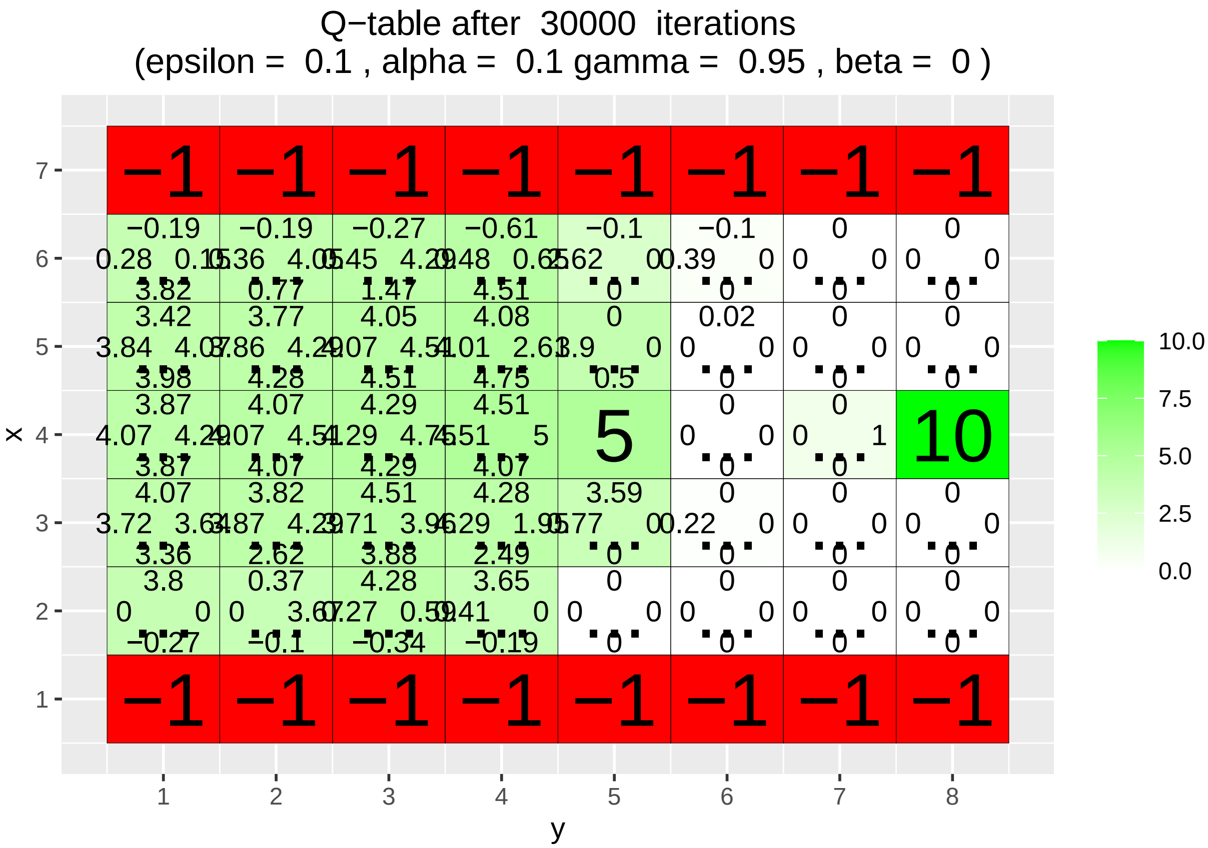
−3.5
−4.0
−4.5
−5.0
MovingAverage(correction, 100)
0
5000
10000
15000
20000
25000
30000
Index

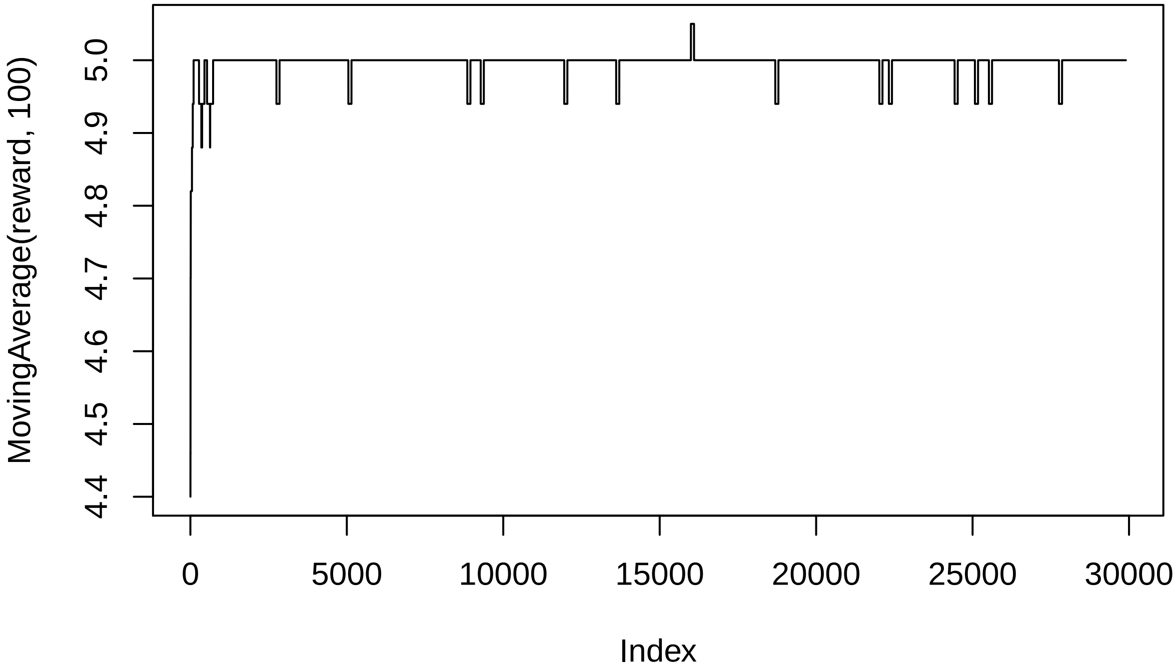
Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)

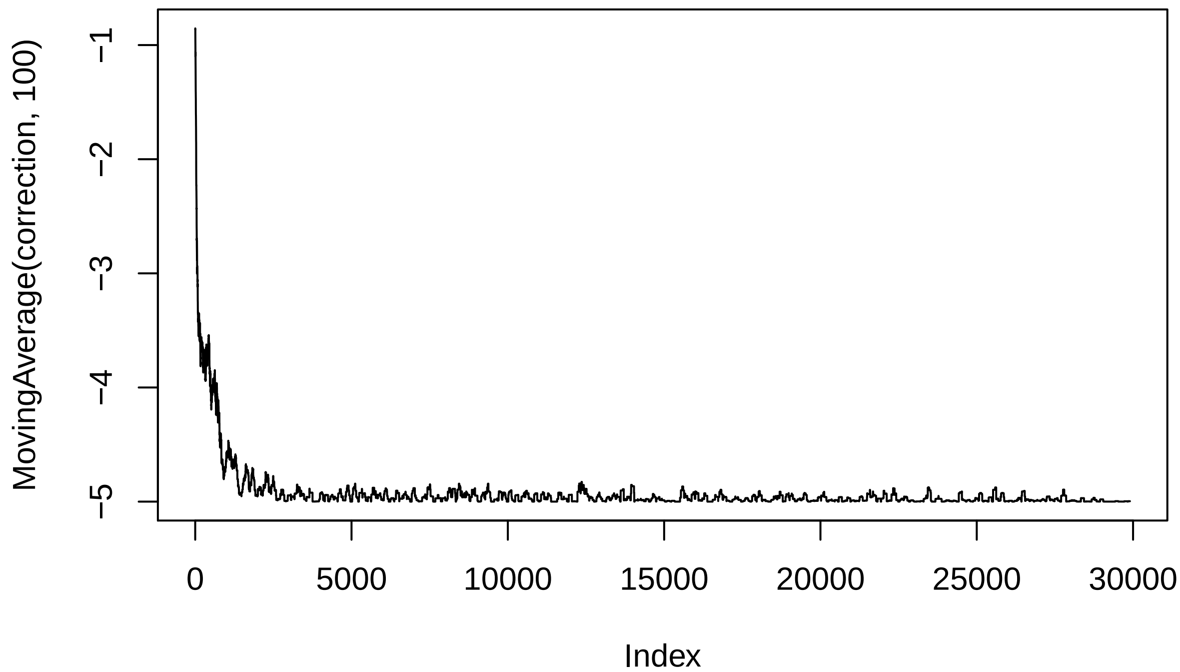



MovingAverage(reward, 100)
5.0
4.9
4.8
4.7
4.6
0
5000
10000
15000
20000
25000
30000
Index

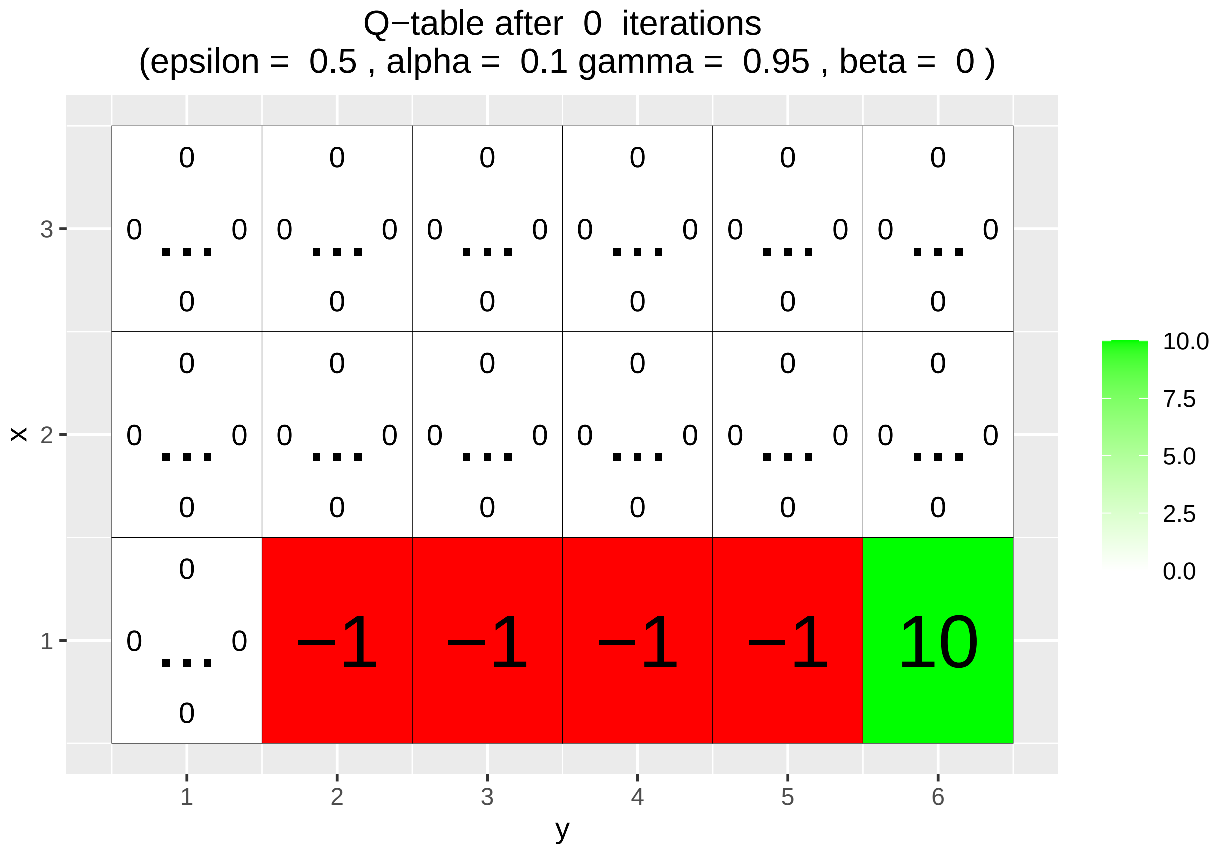

MovingAverage(correction, 100)
−2.5
−3.0
−3.5
−4.0
−4.5
−5.0
0
5000
10000
15000
20000
25000
30000
Index


Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
1
2
3
4
5
6
7
8
10.0
7.5
5.0
2.5
0.0
5
10

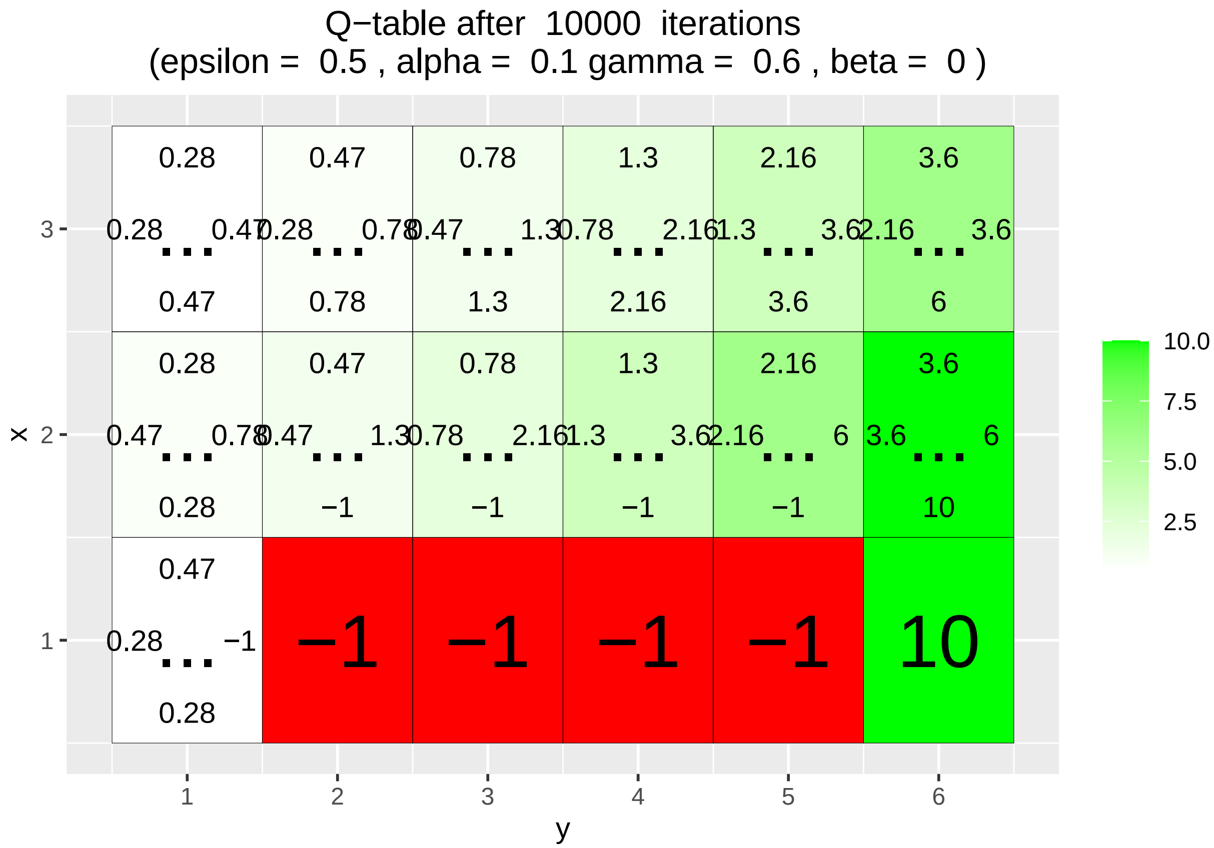

MovingAverage(reward, 100)
4.4
4.5
4.6
4.7
4.8
4.9
5.0
0
5000
10000
15000
20000
25000
30000
Index

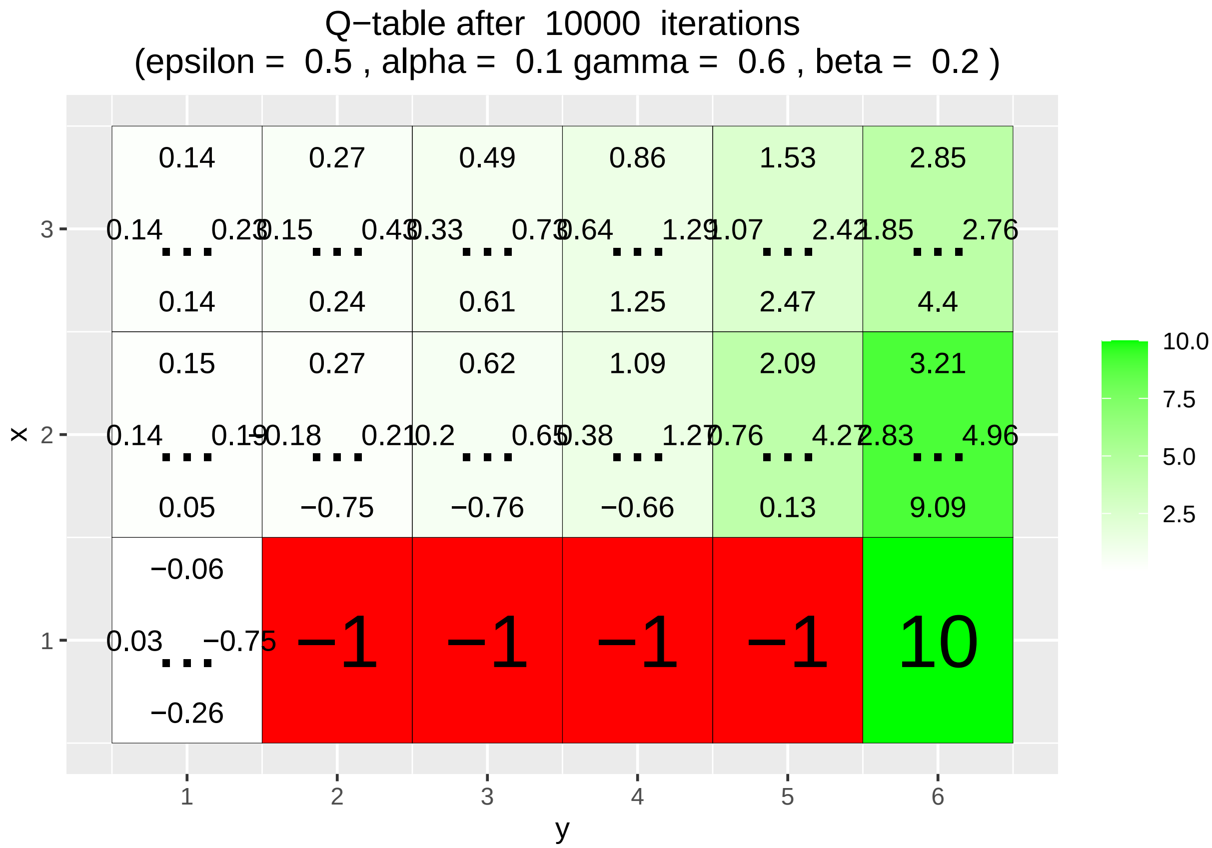


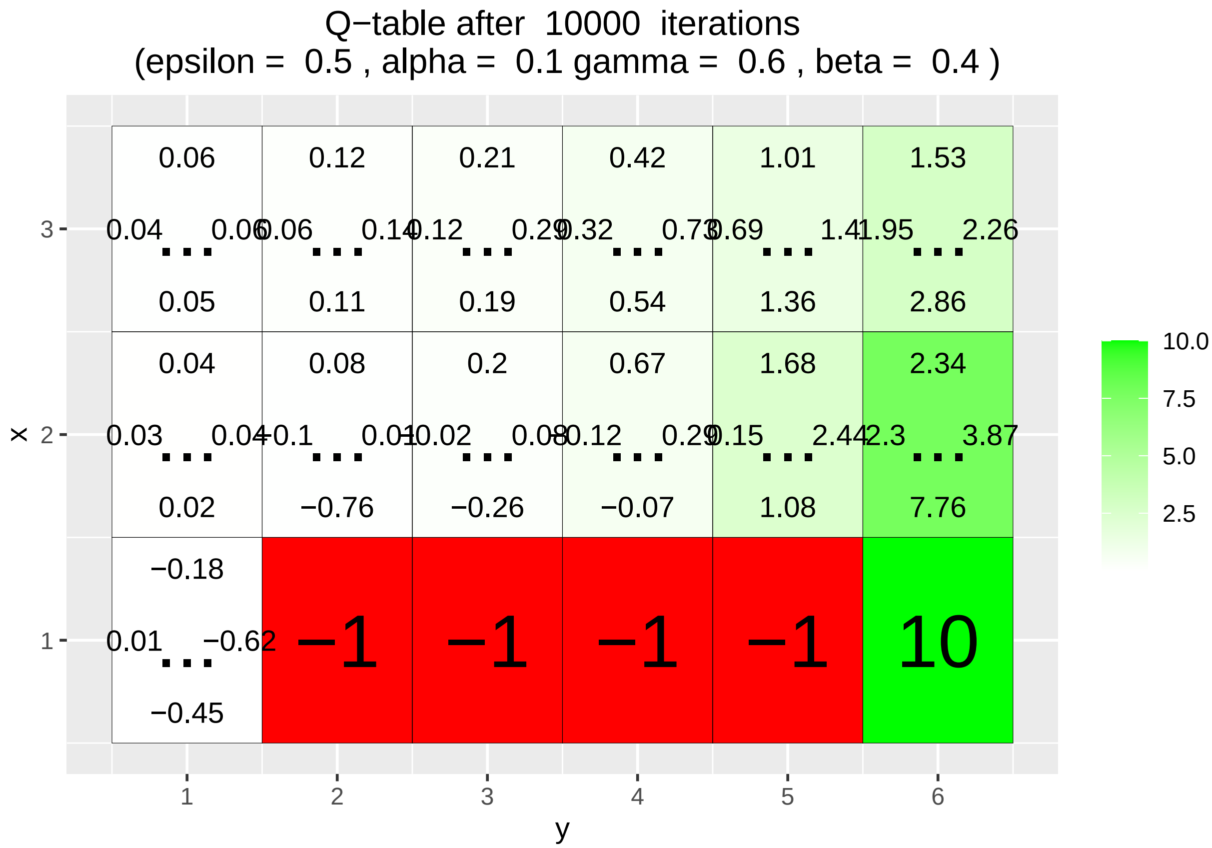

```r
# Environment C (the effect of beta).

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()
```

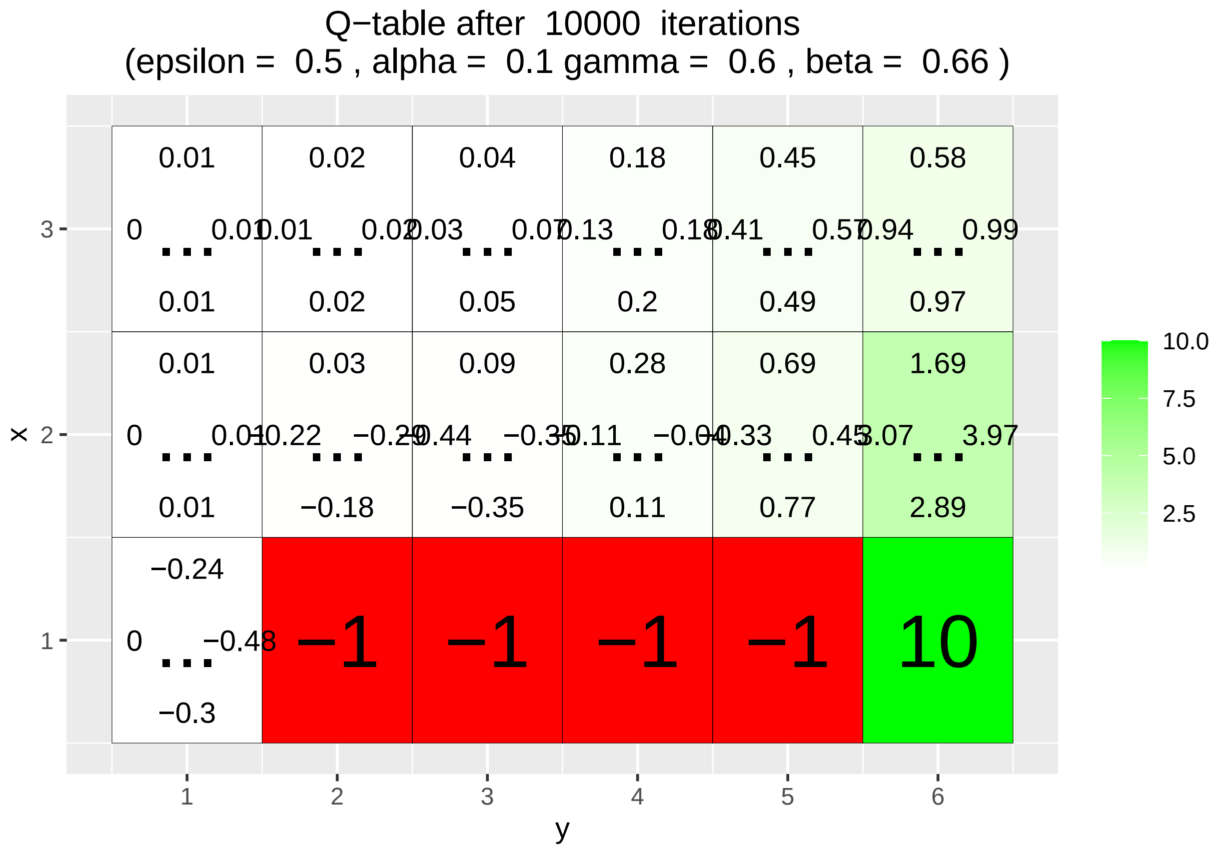



```r
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

  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
10.0
7.5
5.0
2.5
−1
−1
−1
−1
10


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)
x
y
−1
10
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)
0.06
0.04
0.06
0.05
0.12
0.06
0.14
0.11
0.21
0.12
0.29
0.19
0.42
0.32
0.73
0.54
1.01
0.69
1.4
1.36
1.53
1.95
2.26
2.86
0.04
0.03
0.04
0.02
0.08
−0.1
0.01
−0.76
0.2
−0.02
0.08
−0.26
0.67
0.12
0.29
−0.07
1.68
0.15
2.44
1.08
2.34
2.3
3.87
7.76
−0.18
0.01
−0.62
−0.45
−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
0.02
0.04
0.18
0.45
0.58
0
0.01
0.01
0.02
0.03
0.07
0.13
0.18
0.41
0.57
0.94
0.99
0.01
0.02
0.05
0.2
0.49
0.97
0.01
0.03
0.09
0.28
0.69
1.69
0
0.01
−0.22
−0.29
−0.44
−0.35
0.11
−0.04
0.33
0.45
3.07
3.97
0.01
−0.18
−0.35
0.11
0.77
2.89
−0.24
0
−0.48
−0.3
−1
−1
−1
−1
10
10.0
7.5
5.0
2.5
x
y
1
2
3
4
5
6