LAB 3: Reinforcement Learning

Martynas Lukosevicius, Alejo Pérez, Joel Nilsson

10/09/2021

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

```
## Warning: package 'vctrs' was built under R version 4.0.5
```

Statement of Contribution

- code: Alejo Perez
- Answers to questions: Joel Nilsson, Alejo Perez Gomez, Martynas Lukosevicius

2.2

Implementation, as requested, of the GreedyPolicy and EpsilonGreedyPolicy.

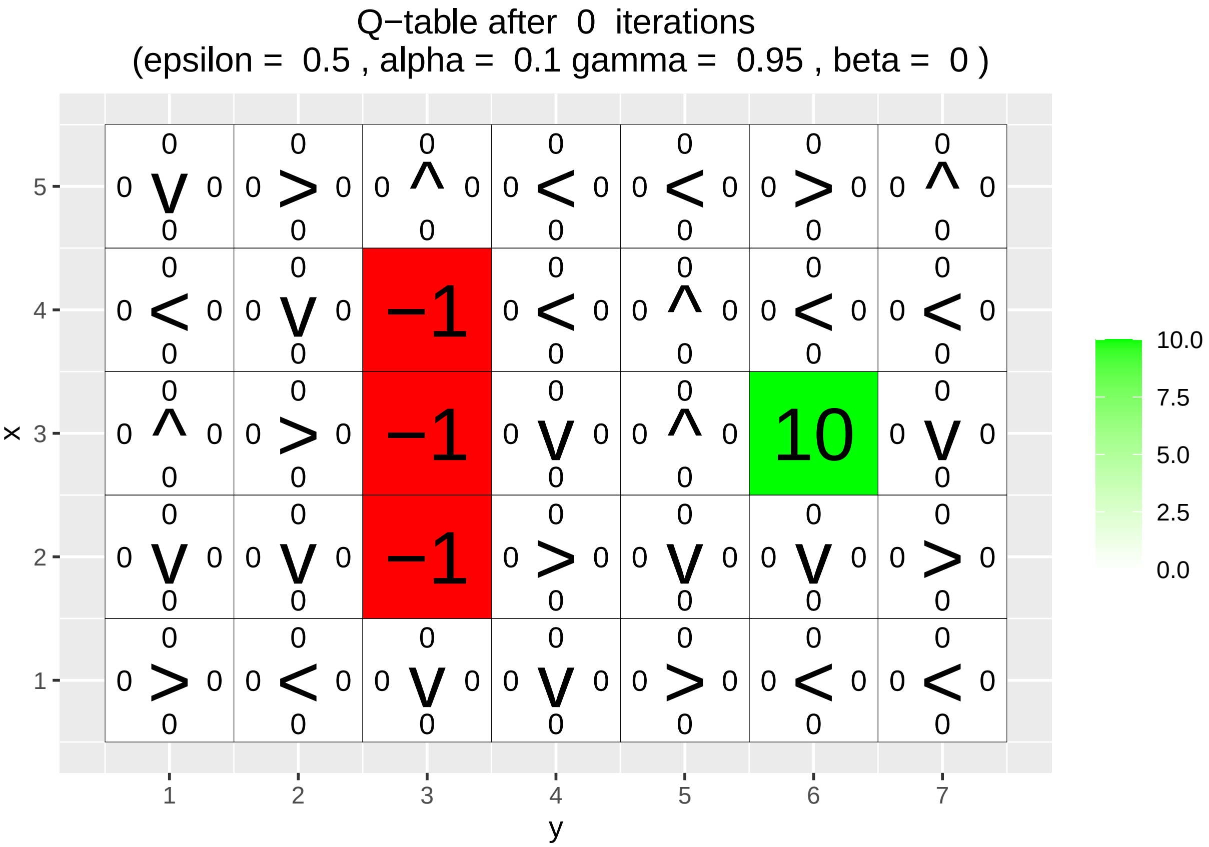
```
GreedyPolicy <- function(x, y){

  # 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}.

  # Locating the max q-value for all actions
  idx_max_val <- which(max(q_table[x,y,]) == q_table[x,y,])

  if (length(idx_max_val)==1){

    res<-idx_max_val

  # if there is more than 1 action with max q, break ties at random
  } else {res <- sample(idx_max_val, 1)}

  res
}

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}.

  # Set random behavior with probability epsilon
  random <- sample(c(0,1), 1, prob=c((1-epsilon), epsilon))

  if (random){res<- sample(1:4, 1)}

  else{res <- GreedyPolicy(x,y)}

  res

  }
```

Implementation of Q-learning algorithm

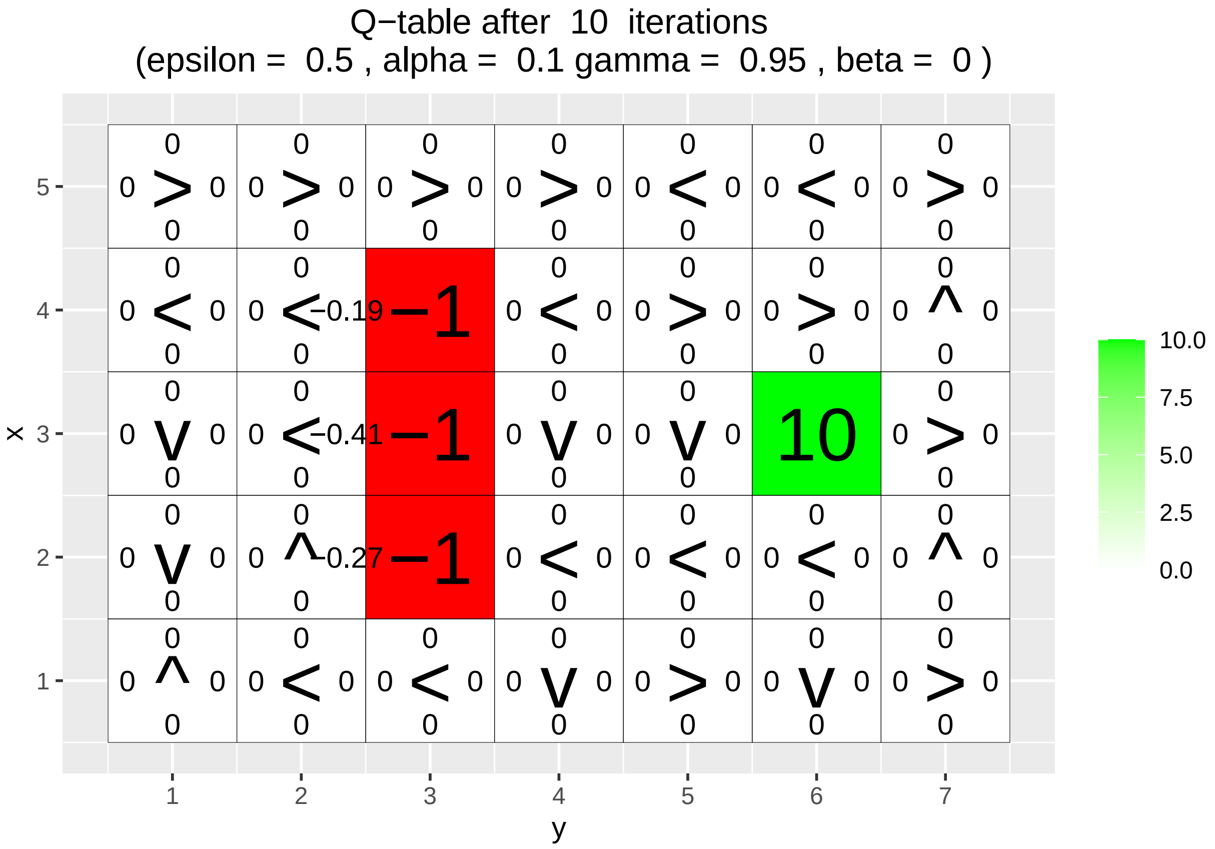
```
q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95,
                       beta = 0){

  # 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.

  #Initialise the episode correction
  current_state <- start_state
  episode_correction <- 0

  repeat{

    # Follow policy, execute action, get reward.
    # Action defined as the Epsilon Greedy output
    action <- EpsilonGreedyPolicy(current_state[1], current_state[2], epsilon)

    # We get the new State accounting for the possible slip and random action
    new_state <- transition_model(current_state[1], current_state[2],
                                  action, beta)

    # Catch the reward in that coordinate
    reward <- reward_map[new_state[1],new_state[2]]

    # Calculate the correction
    corr <- reward +
      gamma * max(q_table[new_state[1],
      new_state[2],]) - q_table[current_state[1],current_state[2], action]

    # Accumulate correction
    episode_correction <- episode_correction + corr

    # Q-table update
    q_table[current_state[1],
            current_state[2], action] <<- q_table[current_state[1],
                                                  current_state[2],
                                                  action] + alpha*corr
    #set new state
    current_state <- new_state

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

}
```

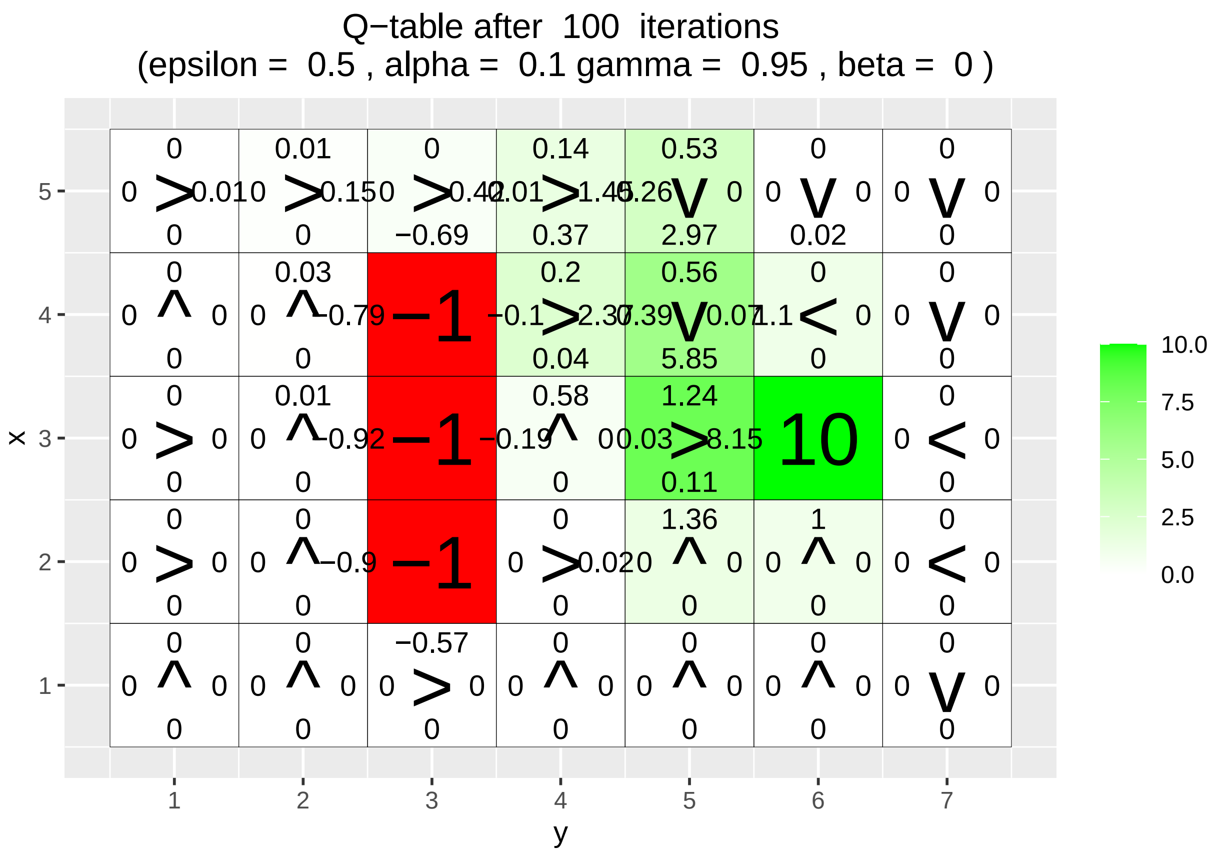
Now by running 10000 episodes of Q-learning with $\epsilon = 0.5$, $beta = 0$, $alpha = 0.1$ and $gamma = 0.95$ we will answer the following questions and visualize the policies.

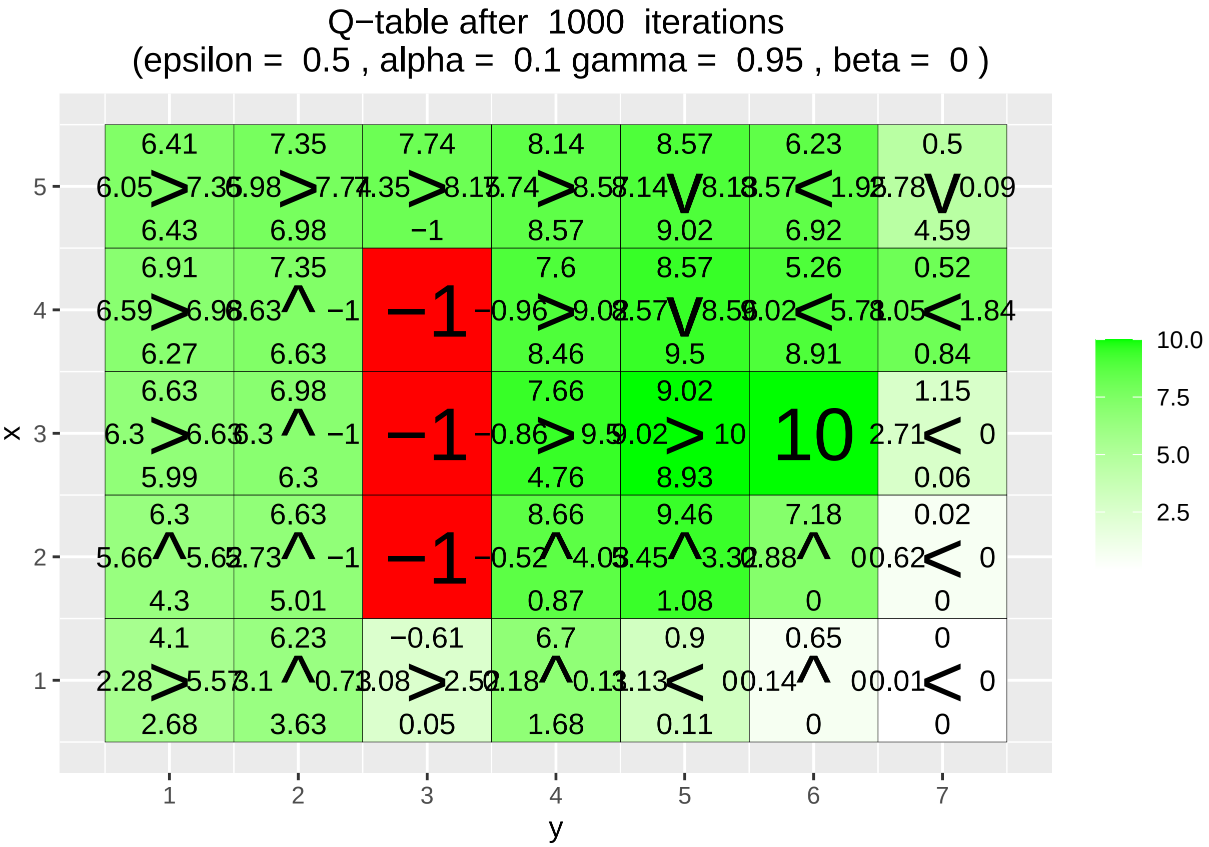
```
#####################################################################################################
# Q-Learning Environments
#####################################################################################################

# 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()
```

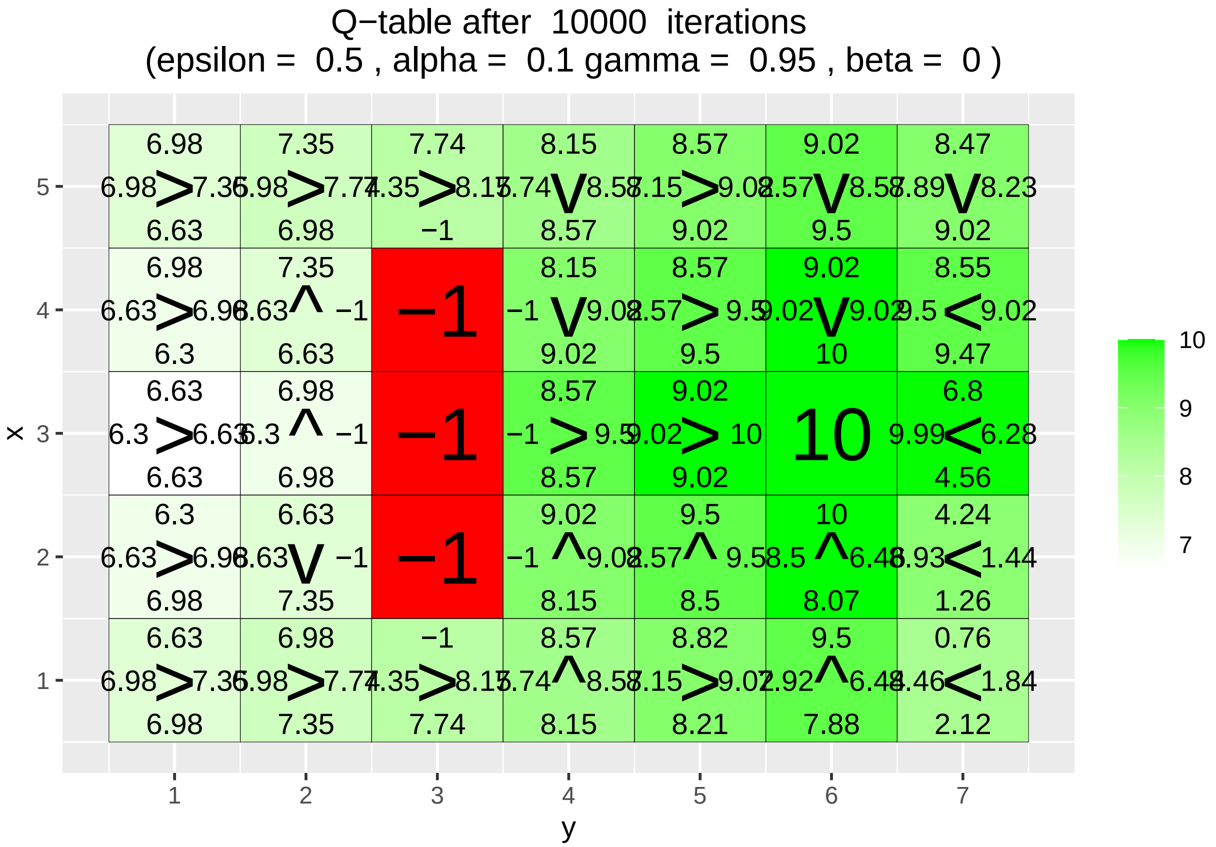


```r
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))

  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
2
3
4
5
6
7
−0.19
−0.41
−0.27
−1
−1
−1
10
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 0.01 0 0.14 0.53 0 0
0 >0.010 >0.150 >0.42.01 >1.45.26 V 0 0 V 0 0 V 0
0 0 −0.69 0.37 2.97 0.02 0
0 0.03 0.2 0.56 0 0
0 ^ 0 0 ^−0.79 −1 −0.1 >2.37.39 V 0.07.1 < 0 0 V 0
0 0 0.04 5.85 0 0
0 0.01 0.58 1.24 0
0 > 0 0 ^−0.92 −1 −0.19 ^ 00.03 >8.15 10 0 < 0
0 0 0 0.11 0
0 0 0 1.36 1 0
0 > 0 0 ^−0.9 −1 0 >0.020 ^ 0 0 ^ 0 0 < 0
0 0 0 0 0 0
0 0 −0.57 0 0 0 0
0 ^ 0 0 ^ 0 0 > 0 0 ^ 0 0 ^ 0 0 ^ 0 0 V 0
0 0 0 0 0 0 0
x
5
4
3
2
1
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)
6.41 7.35 7.74 8.14 8.57 6.23 0.5
6.05 > 7.36.98 > 7.74.35 > 8.15.74 > 8.58.14 V 8.18.57 < 1.95.78 V 0.09
6.43 6.98 −1 8.57 9.02 6.92 4.59
6.91 7.35 7.6 8.57 5.26 0.52
6.59 > 6.98.63 ^ −1 −1 −0.96 > 9.08.57 V 8.59.02 < 5.78.05 < 1.84
6.27 6.63 8.46 9.5 8.91 0.84
6.63 6.98 7.66 9.02 1.15
6.3 > 6.63.3 ^ −1 −1 −0.86 > 9.59.02 > 10 10 2.71 < 0
5.99 6.3 4.76 8.93 0.06
6.3 6.63 8.66 9.46 7.18 0.02
5.66 ^ 5.62.73 ^ −1 −1 −0.52 ^ 4.03.45 ^ 3.32.88 ^ 00.62 < 0
4.3 5.01 0.87 1.08 0 0
4.1 6.23 −0.61 6.7 0.9 0.65 0
2.28 > 5.57.1 ^ 0.73.08 > 2.52.18 ^ 0.13.13 < 00.14 ^ 00.01 < 0
2.68 3.63 0.05 1.68 0.11 0 0
x
5
4
3
2
1
y
1
2
3
4
5
6
7
10.0
7.5
5.0
2.5



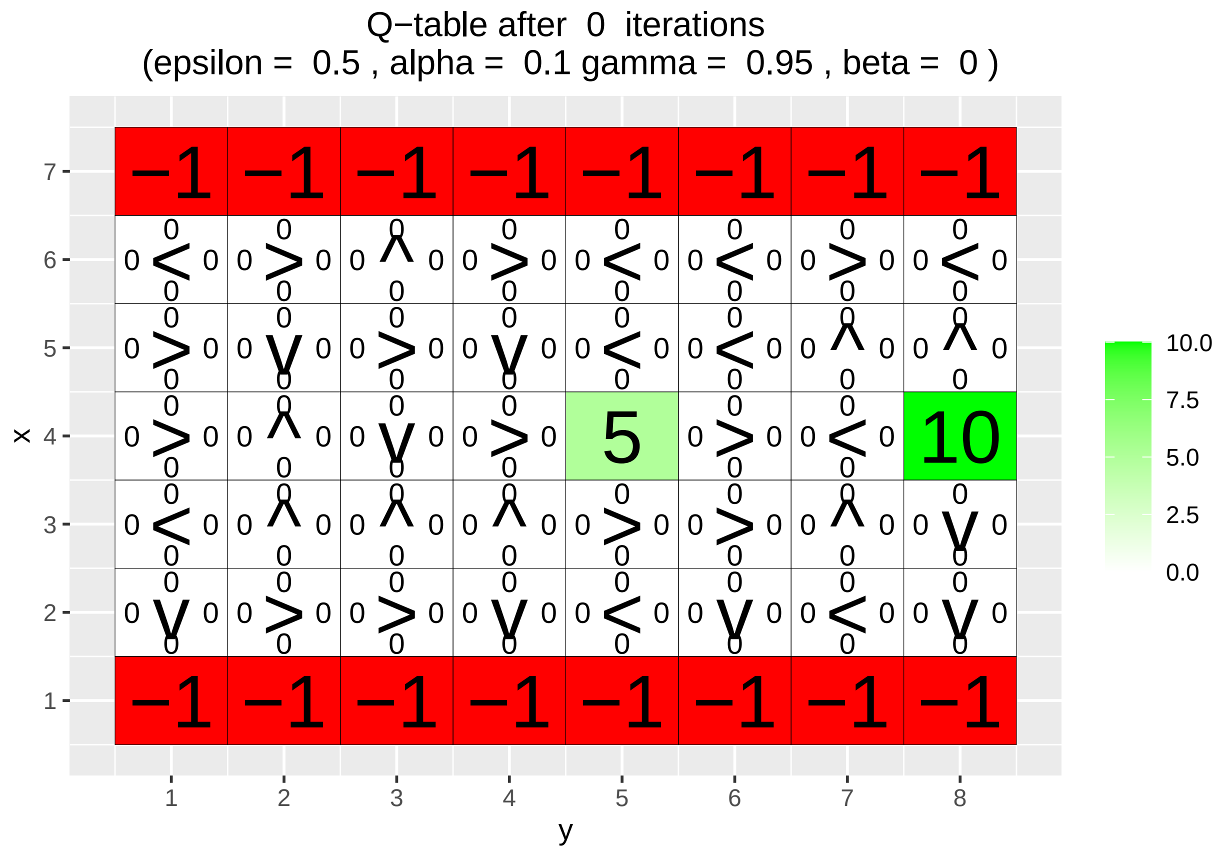
– *What has the agent learned after the first 10 episodes ?* It has just learn not to fall to the terminal negative states. For the neighboring states a negative q-value for the action that leads to the -1 rewarded state has been set. This applies to the states that are on the path coming from the initial state, x,y = (3,1). In overall, to avoid the traps.

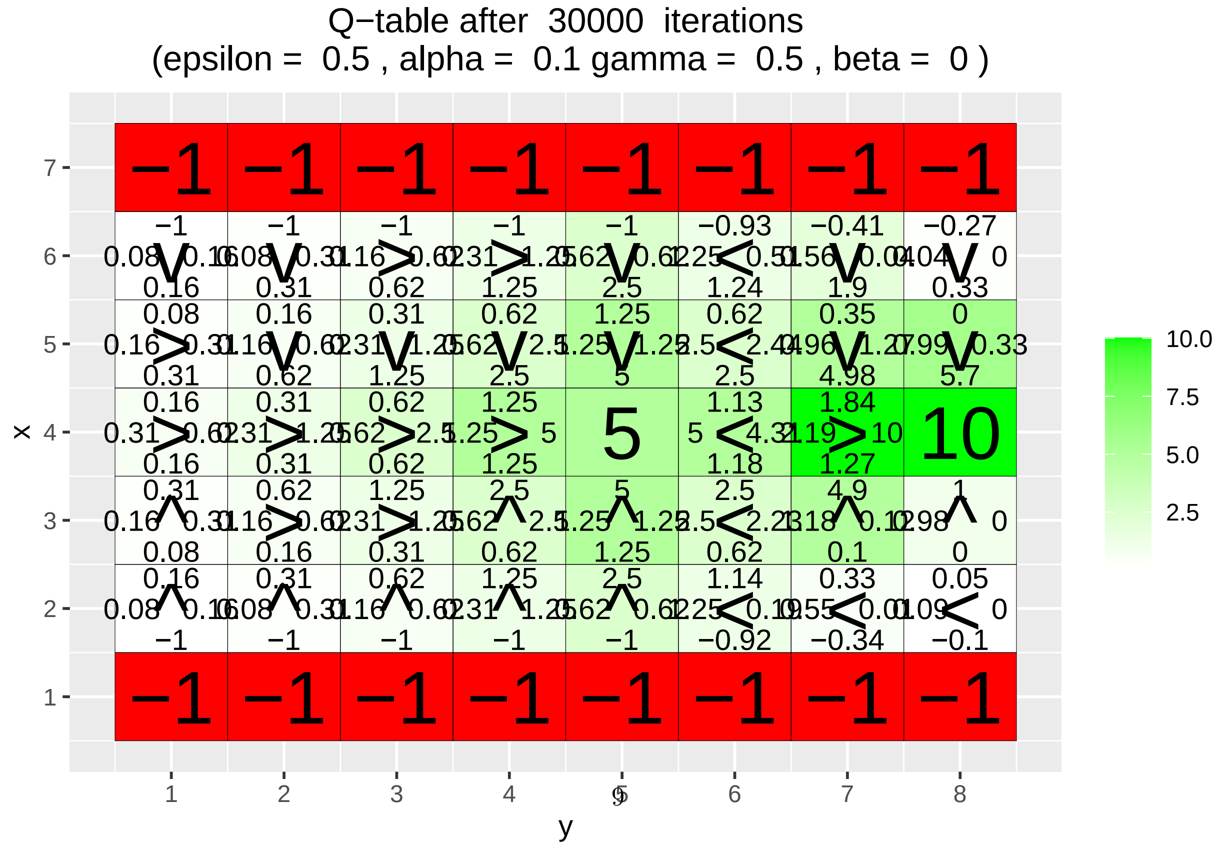
– *Is the final greedy policy (after 10000 episodes) optimal for all states, i.e. not only for the initial state ? Why / Why not ?* It avoids falling in the traps, but it could get to shorter paths to the goal starting from different states other than the current initial one.

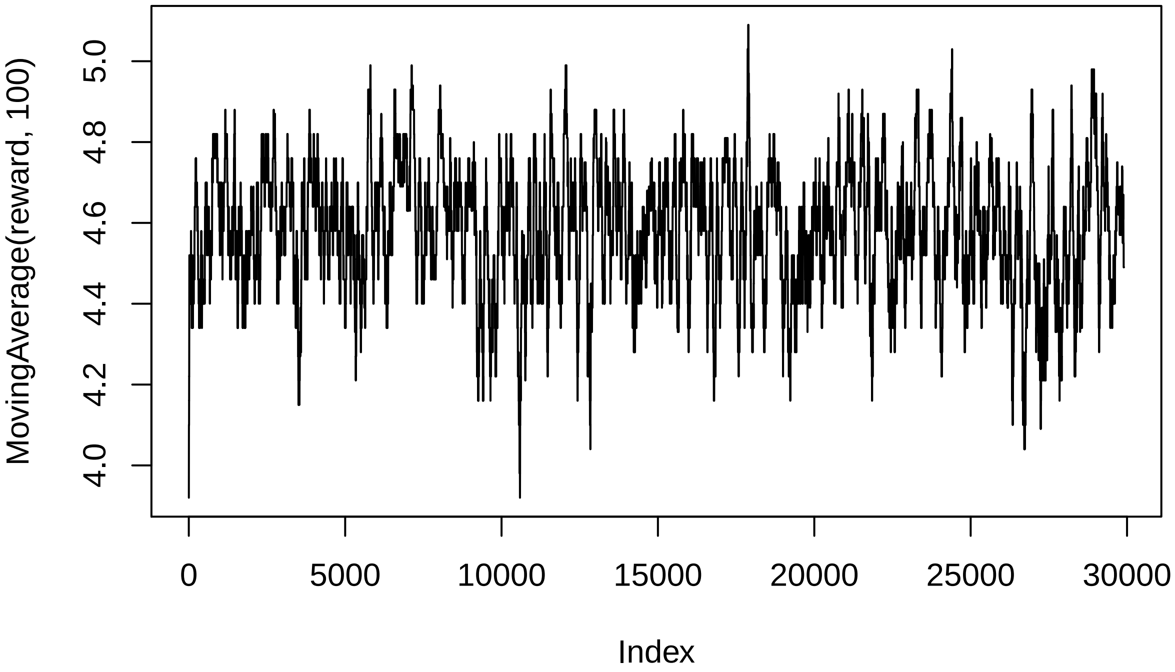
– *Do the learned values in the Q-table reflect the fact that there are multiple paths (above and below the negative rewards) to get to the positive reward ? If not, what could be done to make it happen ?* No, it could be solved by increasing the exploration rate through parameter */epsilon*, less greediness. We can also do it by raising the number of iterations.

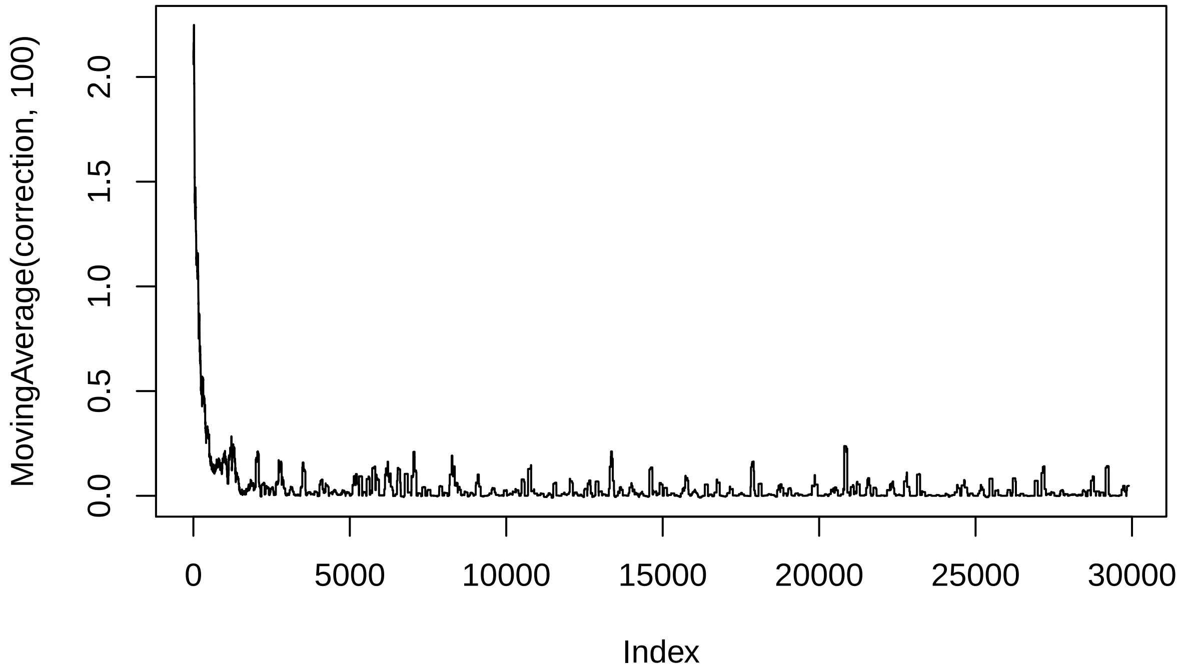
2.3

Environment B (the effect of epsilon and gamma)

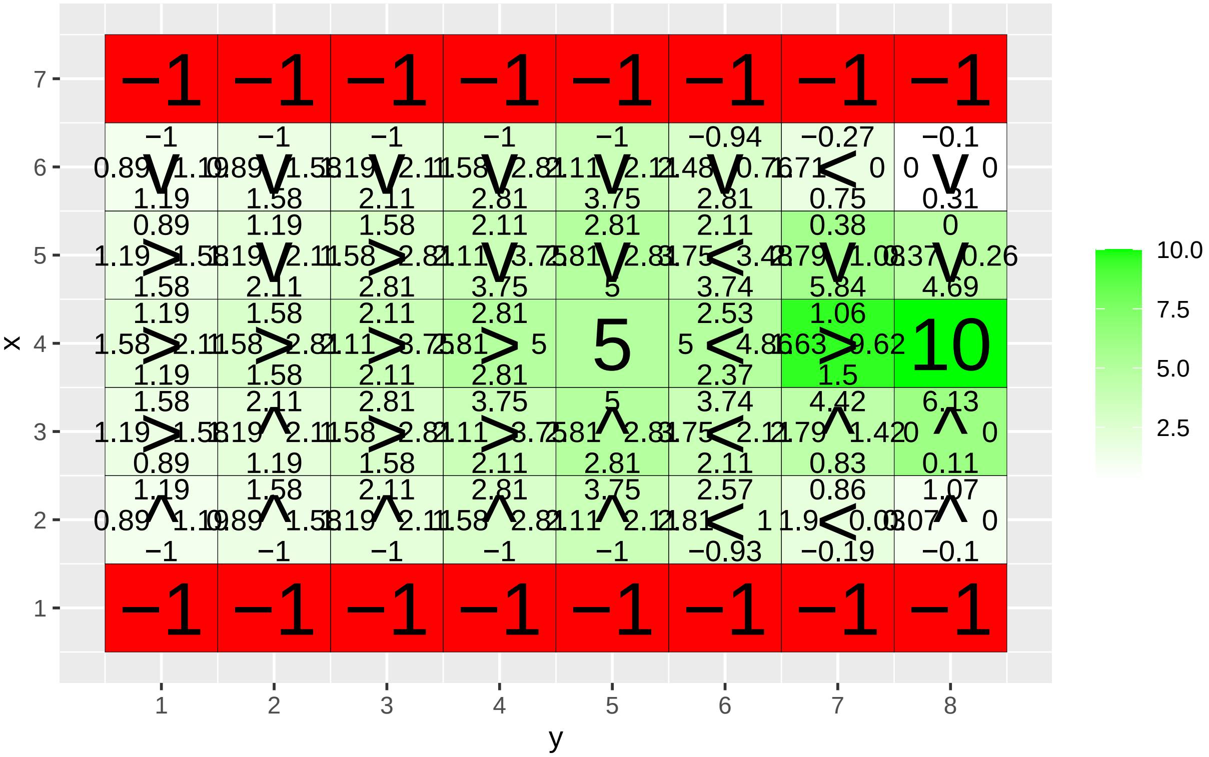


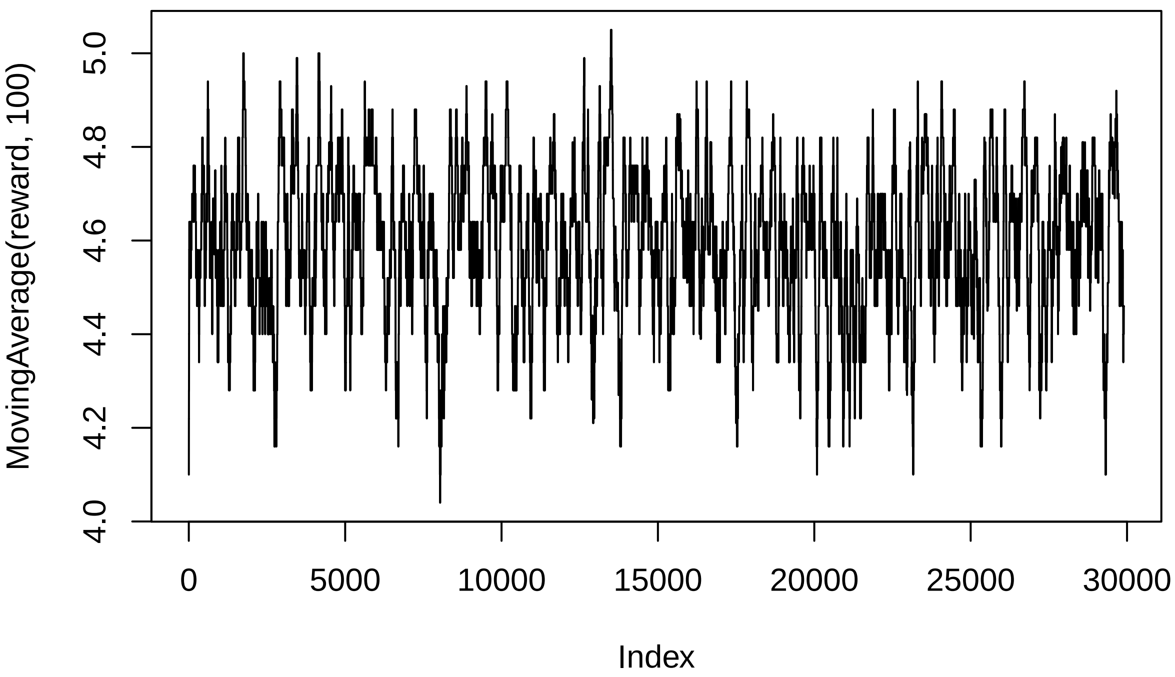


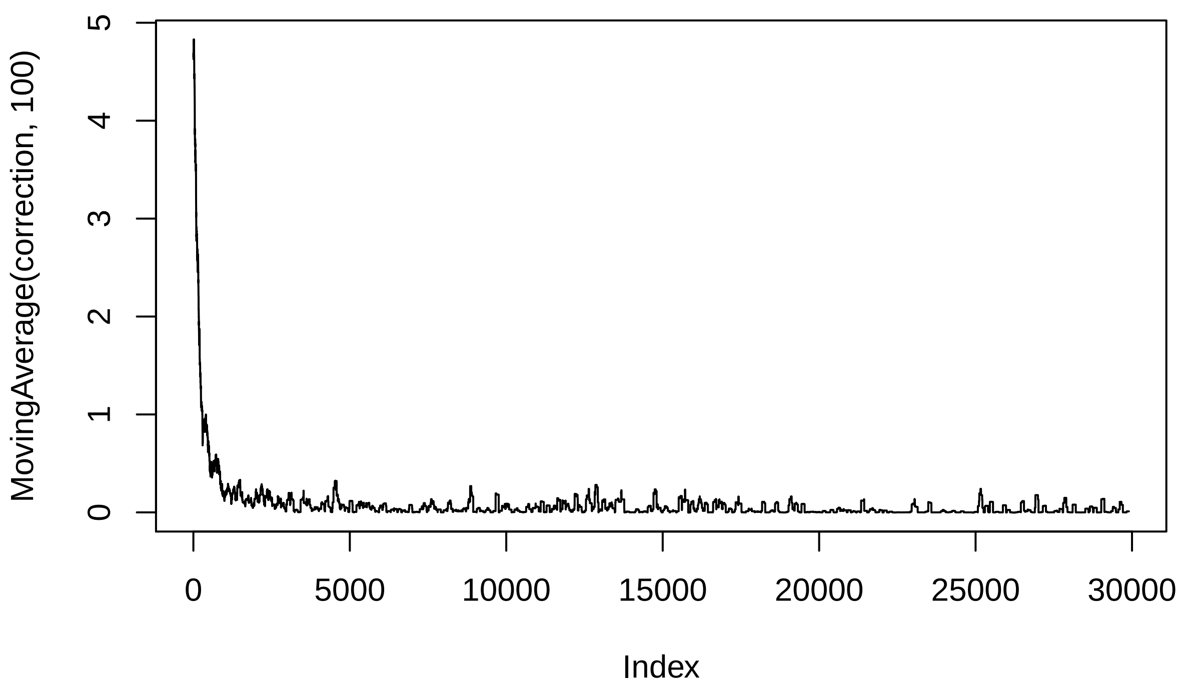

MovingAverage(reward, 100)
4.0
4.2
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index


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

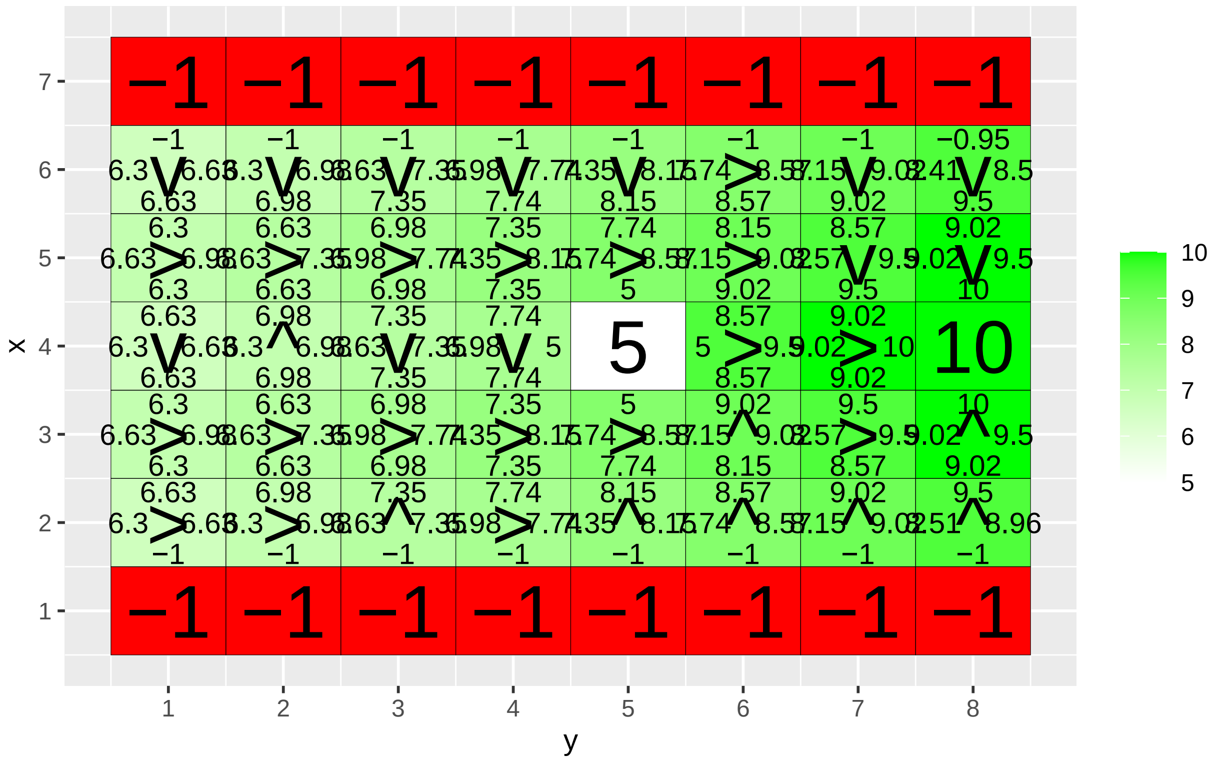
Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)

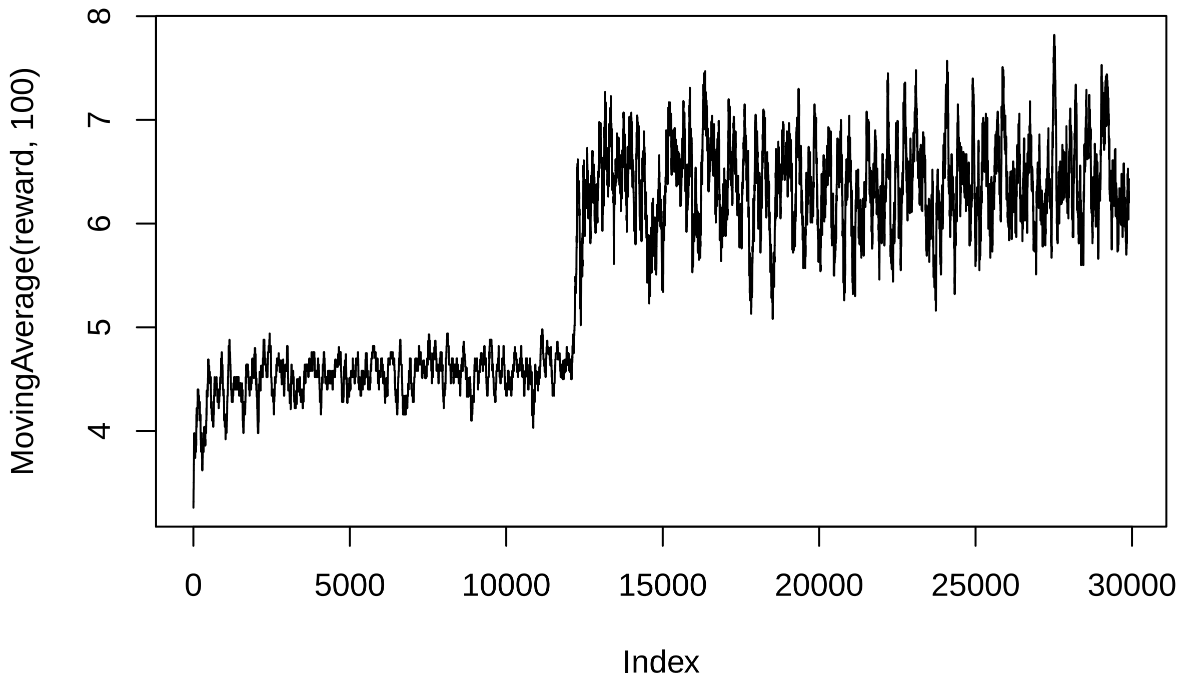


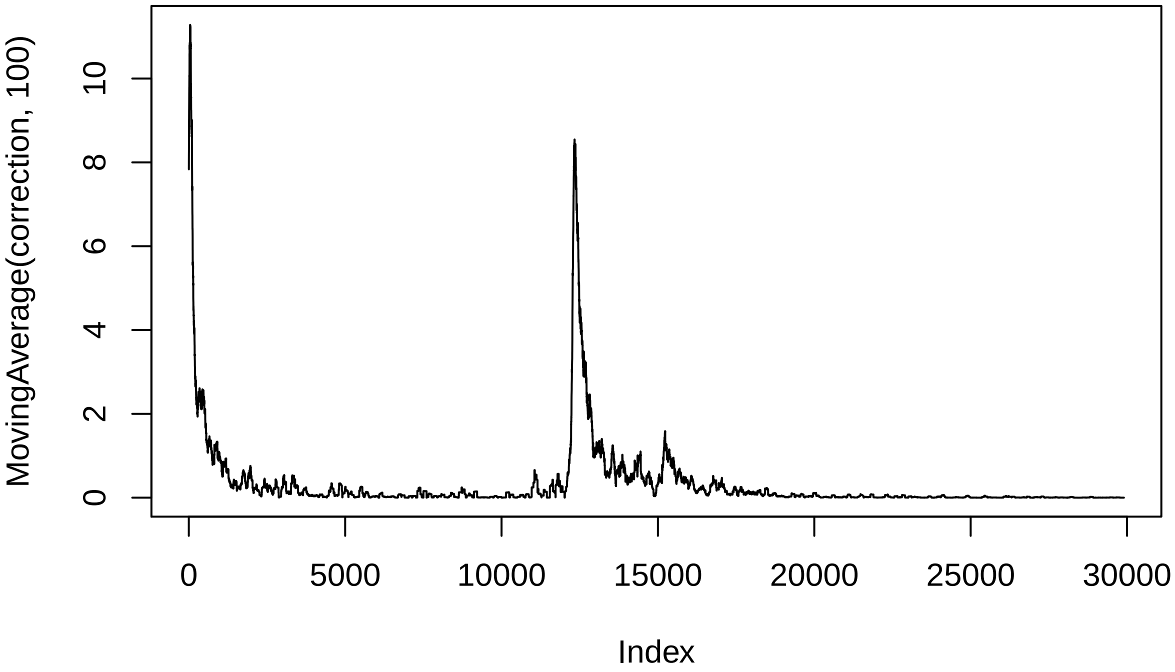

MovingAverage(reward, 100)
4.0
4.2
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index

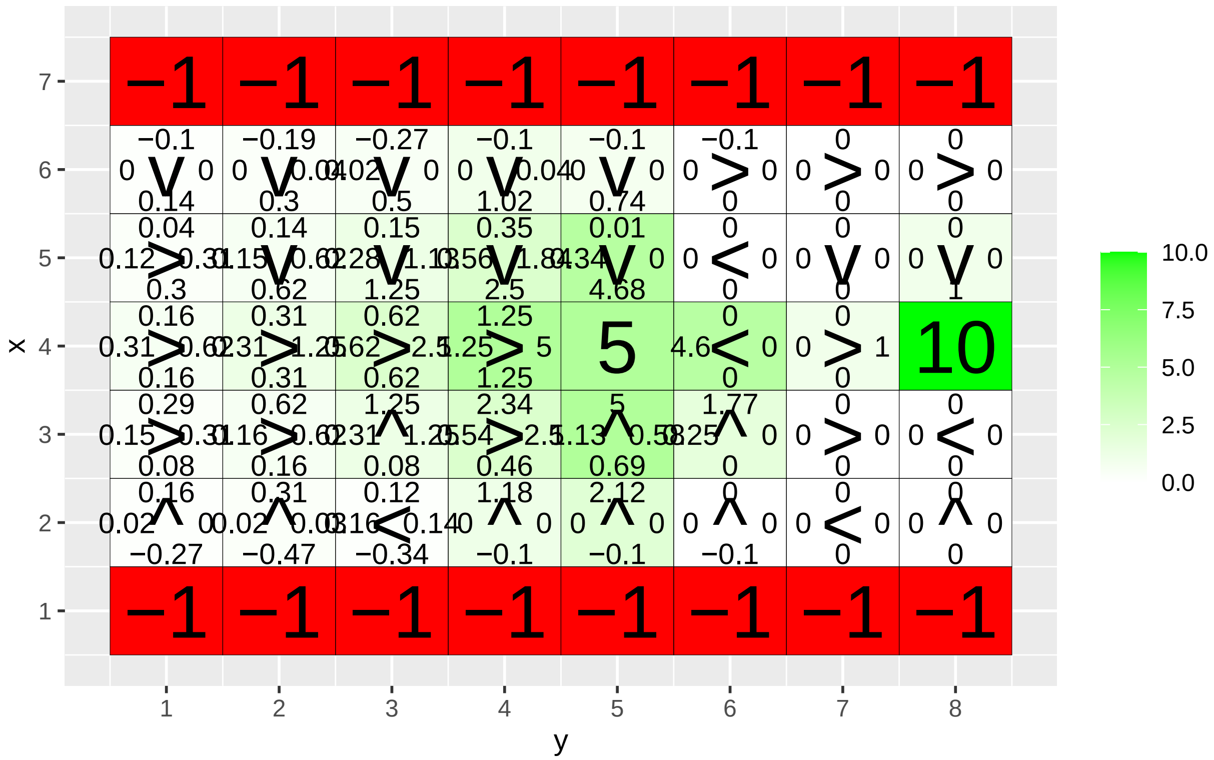

MovingAverage(correction, 100)
0
1
2
3
4
5
0
5000
10000
15000
20000
25000
30000
Index

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

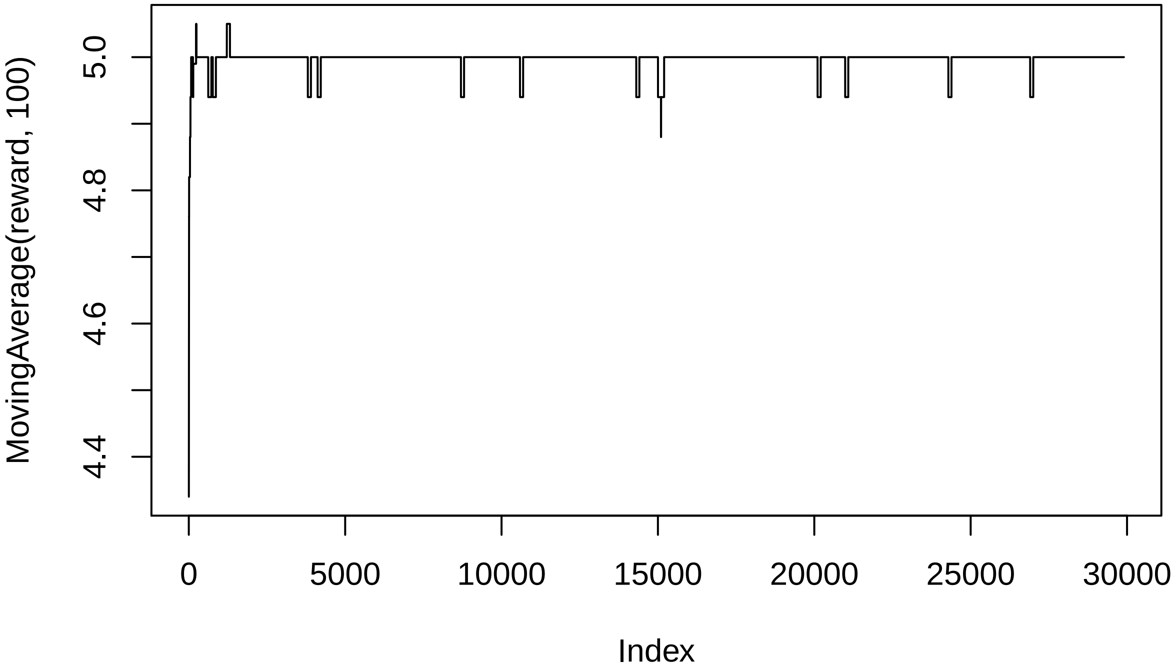


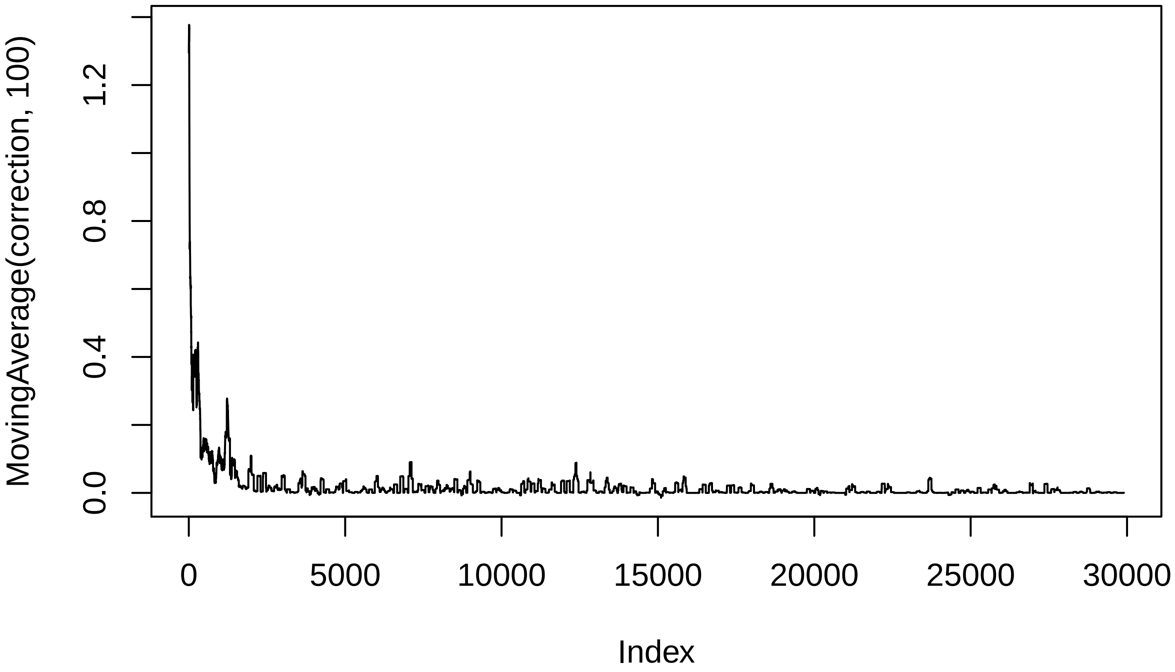

MovingAverage(reward, 100)
8
7
6
5
4
0
5000
10000
15000
20000
25000
30000
Index

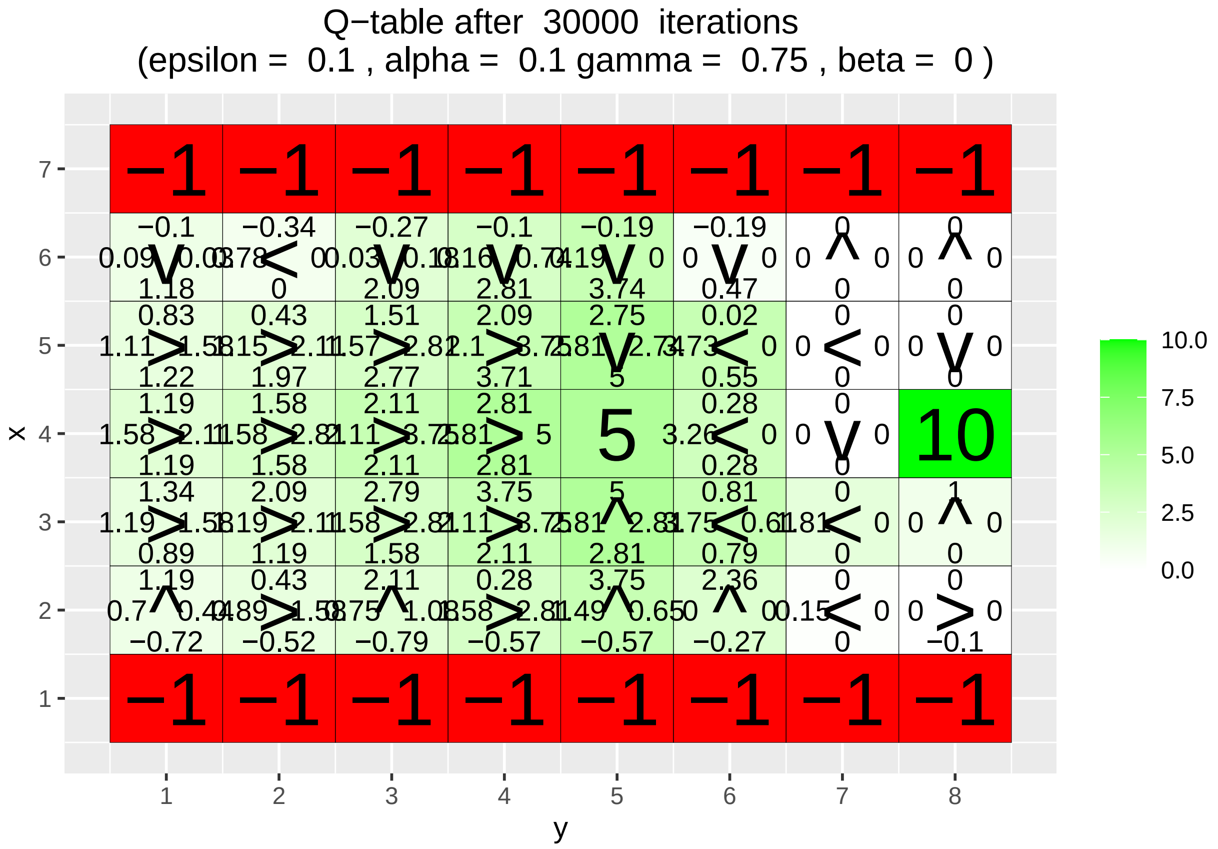

MovingAverage(correction, 100)
0
2
4
6
8
10
0
5000
10000
15000
20000
25000
30000
Index

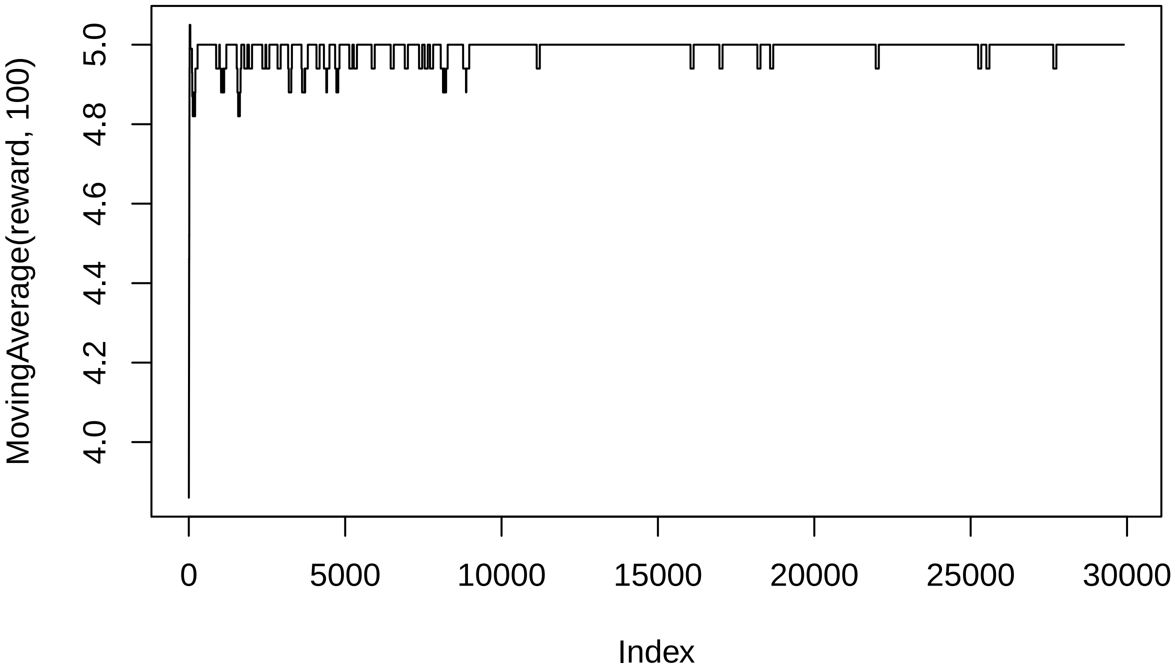


Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)

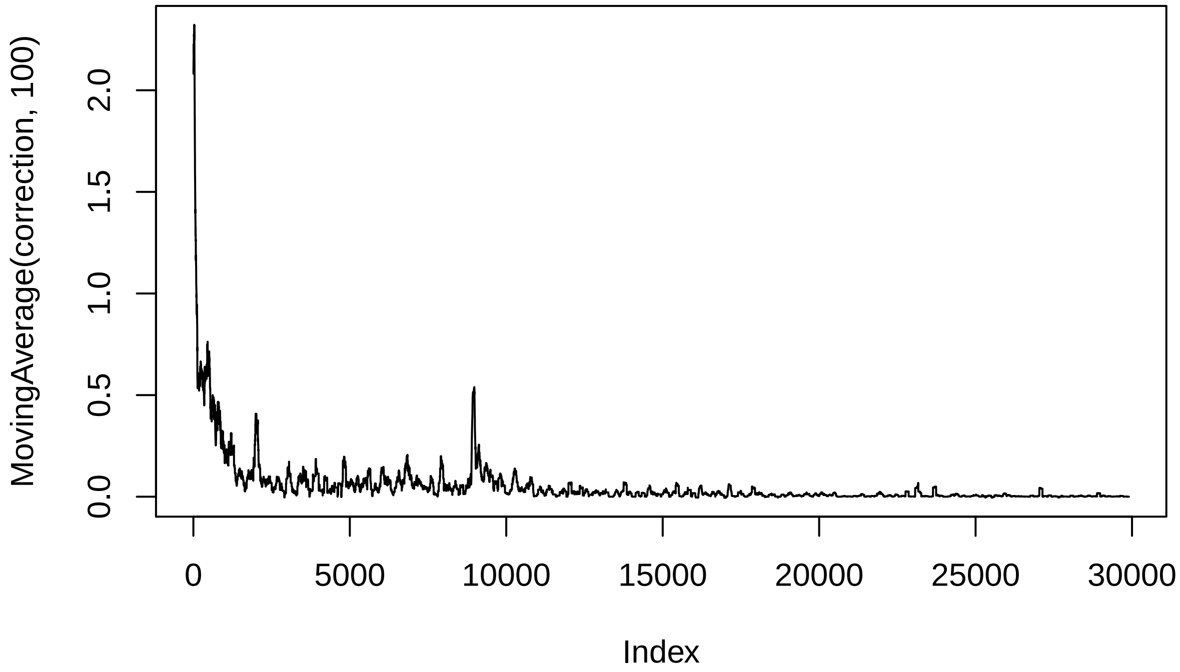

MovingAverage(reward, 100)
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index

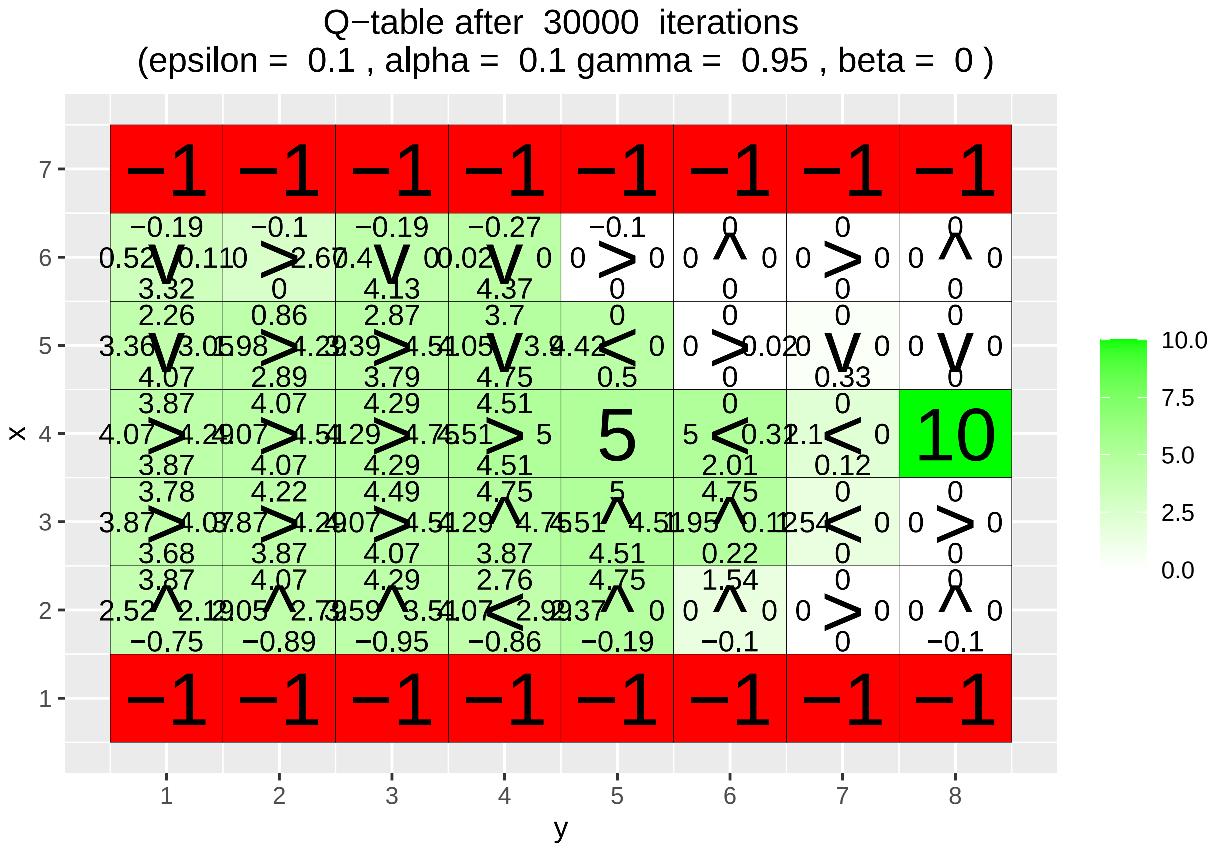

MovingAverage(correction, 100)
0.0
0.4
0.8
1.2
0
5000
10000
15000
20000
25000
30000
Index

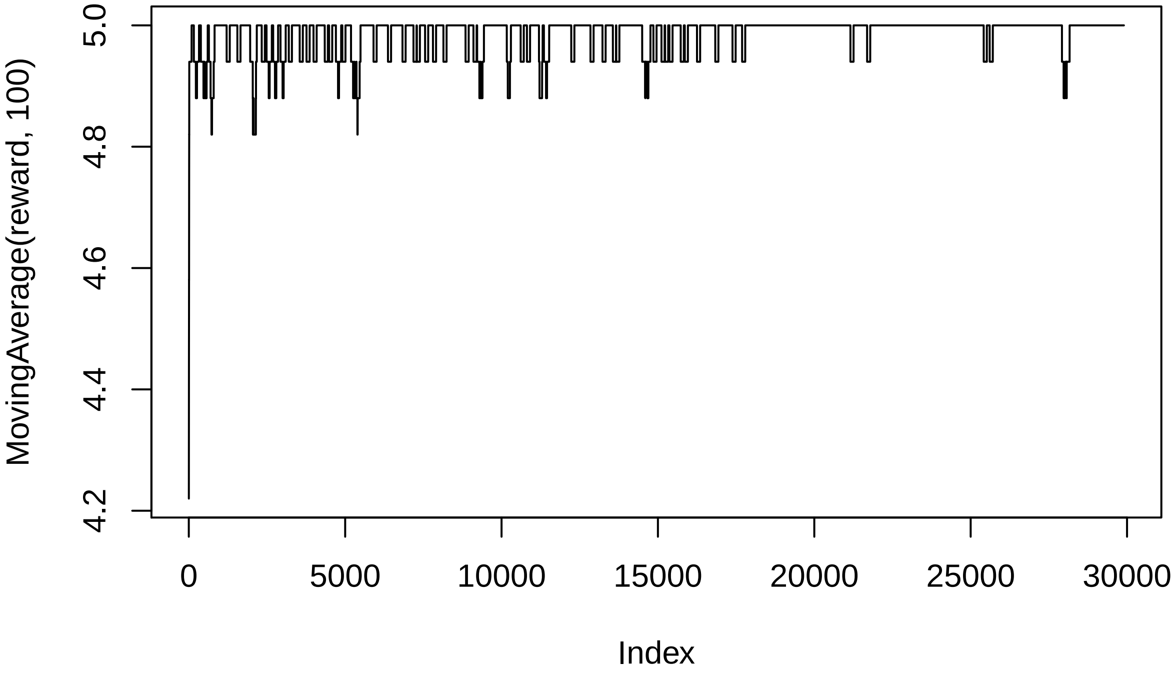

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
1
2
3
4
5
6
7
8
−1
5
10
10.0
7.5
5.0
2.5
0.0

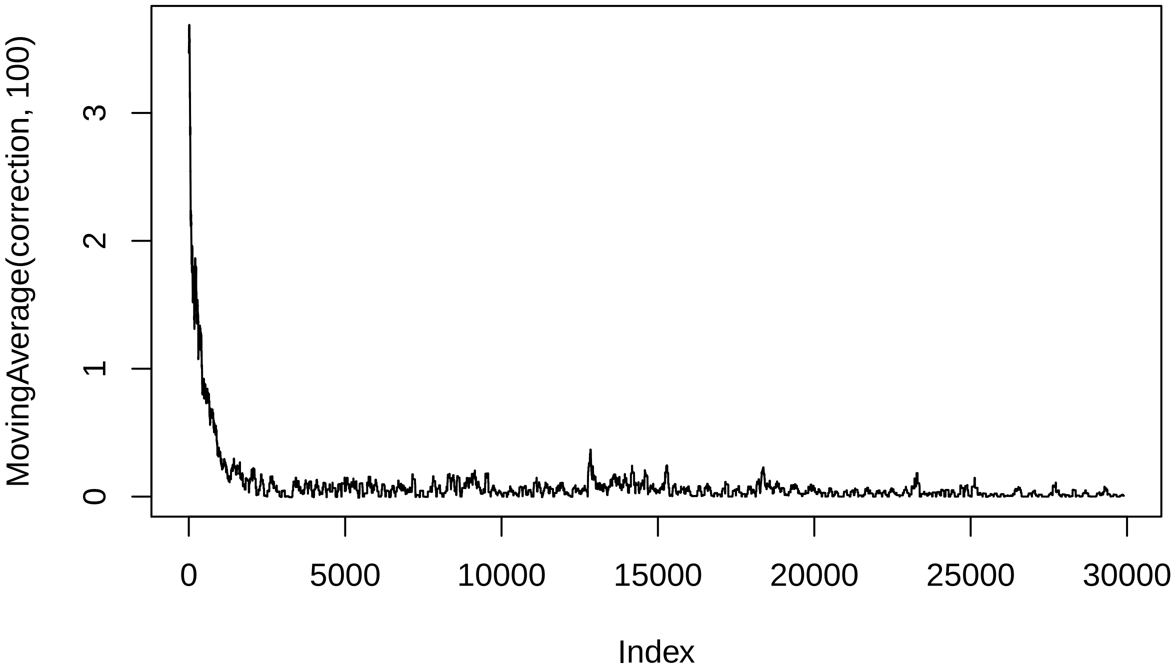


MovingAverage(reward, 100)
4.0
4.2
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index


MovingAverage(correction, 100)
0.0
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)
7
6
5
4
3
2
1
x
1
2
3
4
5
6
7
8
y
−1 −1 −1 −1 −1 −1 −1 −1
−0.19 0.52 V 0.11 3.32
−0.1 0 > 2.67 0
−0.19 0.4 V 0 4.13
−0.27 0.02 V 0 4.37
−0.1 0 > 0 0
0 0 ^ 0 0
0 0 > 0 0
0 0 ^ 0 0
2.26 3.36 V 3.05 4.07
0.86 1.98 > 4.23 2.89
2.87 3.39 > 4.51 3.79
3.7 4.05 V 3.9 4.75
0 4.42 < 0 0.5
0 0 > 0.02 0
0 0 V 0 0.33
0 0 V 0 0
3.87 4.07 > 4.29 3.87
4.07 4.07 > 4.51 4.07
4.29 4.29 > 4.75 4.29
4.51 4.51 > 5 4.51
5
0 5 < 0.31 2.01
0 2.1 < 0 0.12
10
3.78 3.87 > 4.07 3.68
4.22 3.87 > 4.29 3.87
4.49 4.07 > 4.51 4.07
4.75 4.29 ^ 4.74 3.87
5 4.51 ^ 4.51 4.51
4.75 1.95 ^ 0.11 0.22
0 2.54 < 0 0
0 0 > 0 0
3.87 2.52 ^ 2.12 −0.75
4.07 2.05 ^ 2.79 −0.89
4.29 3.59 ^ 3.54 −0.95
2.76 4.07 < 2.92 −0.86
4.75 2.37 ^ 0 −0.19
1.54 0 ^ 0 −0.1
0 0 > 0 0
0 0 ^ 0 −0.1
−1 −1 −1 −1 −1 −1 −1 −1
10.0
7.5
5.0
2.5
0.0


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



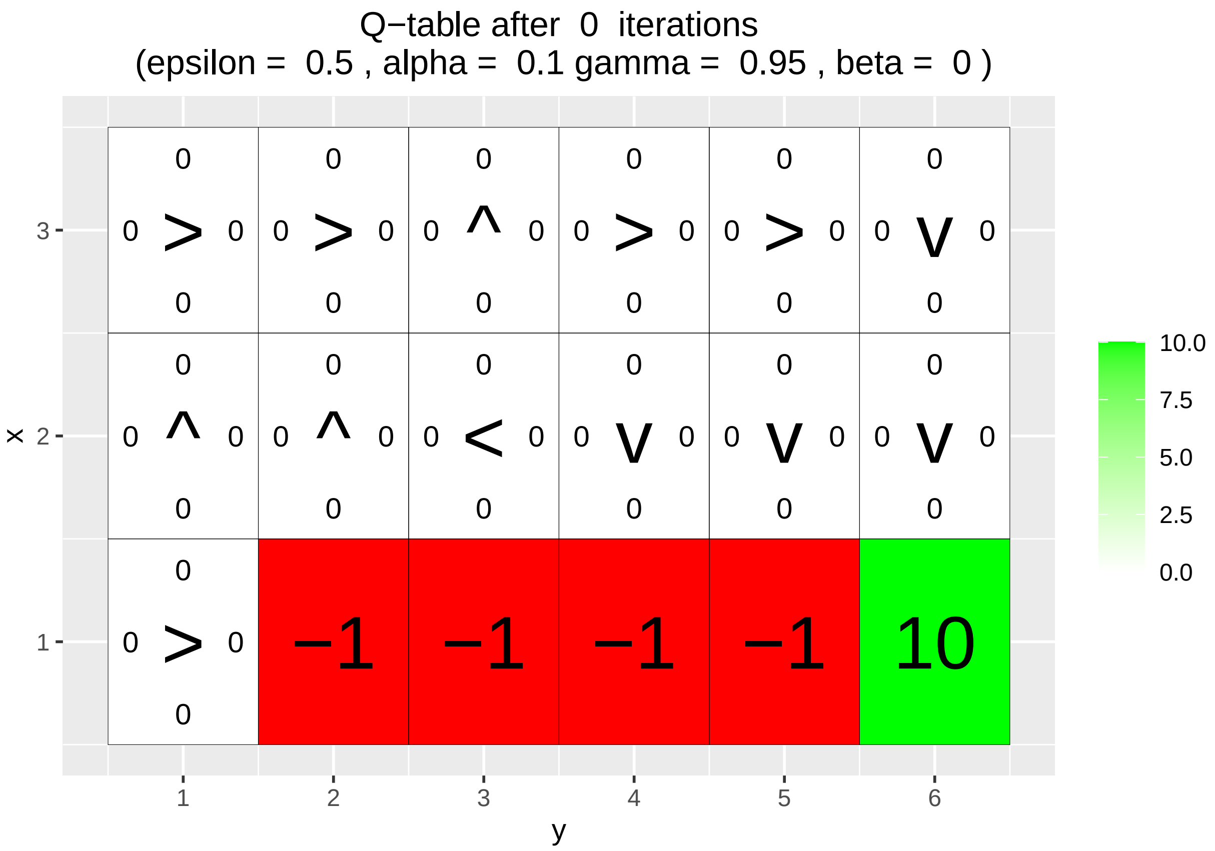
For the first experiment, there are not much incentives to get pass the goal “5”, although there’s a fair amount of exploration rate. For gamma=0.75 we don’t discount the reward for the future state as before. We can see in the 6th y column then that the policy got points to the right instead. In its corresponding moving average plot for the rewards, we can see that as soon as we reach the goal 10, the signal shifts immediately. With respect to the correction plot, there’s a well pronounced spike when the same happened.

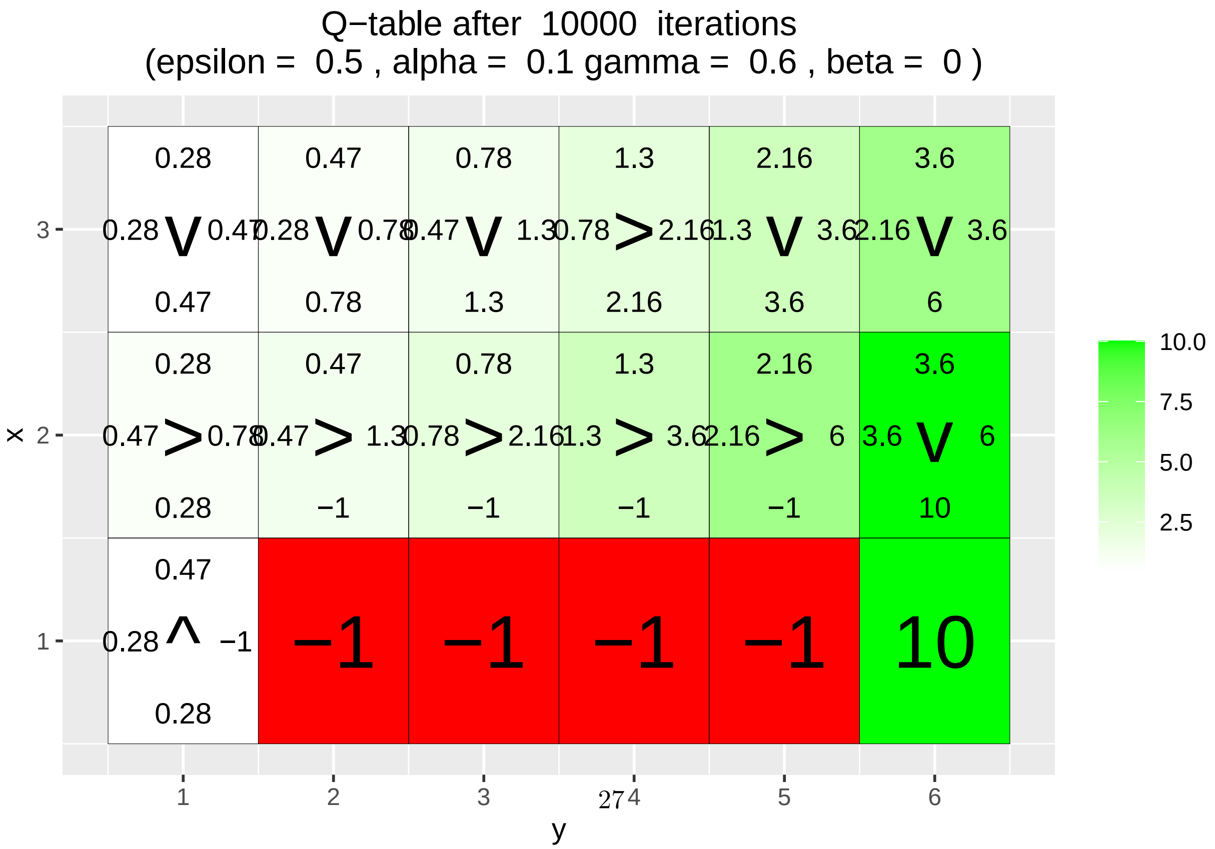
When lowering ϵ to 0.1, there is no exploration rate going on, therefore it doesn’t reach the goal “10”, it conforms to reaching goal “5”. In the reward plot we see clearly that it mostly follows the same path with not many changes. When increased γ to 0.75, it didn’t change abruptly, it’s not able to find the goal “10”. When gamma is set to 0.95 it’s able to get to goal “10” again. That’s due to, although there’s a scarce exploration rate, the little discount of γ puts more stress to future rewards than to the current ones.

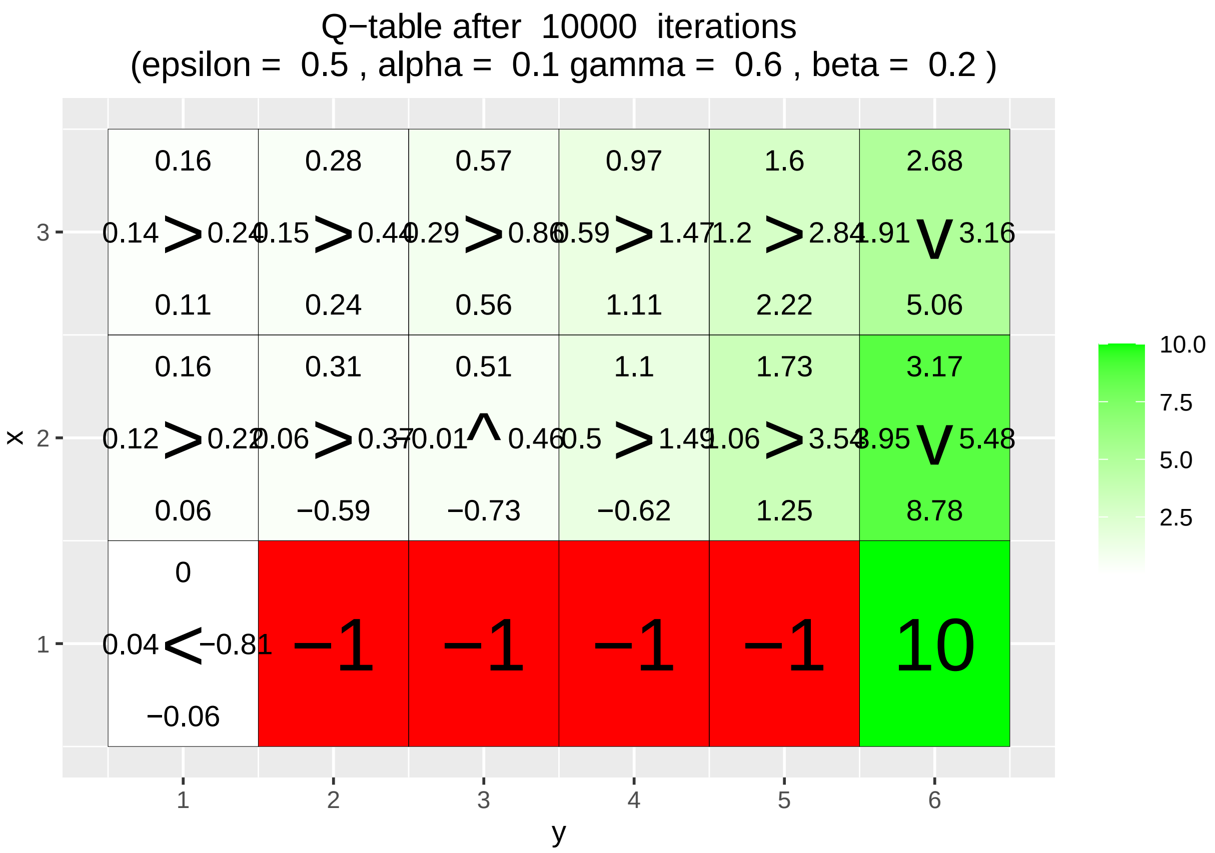
By observing the plotted policies, we can conclude that increasing γ (discount factor)and ϵ (exploration rate) chances of the agent finding better optima and doesn’t getting stuck in local optimum rise.

2.4

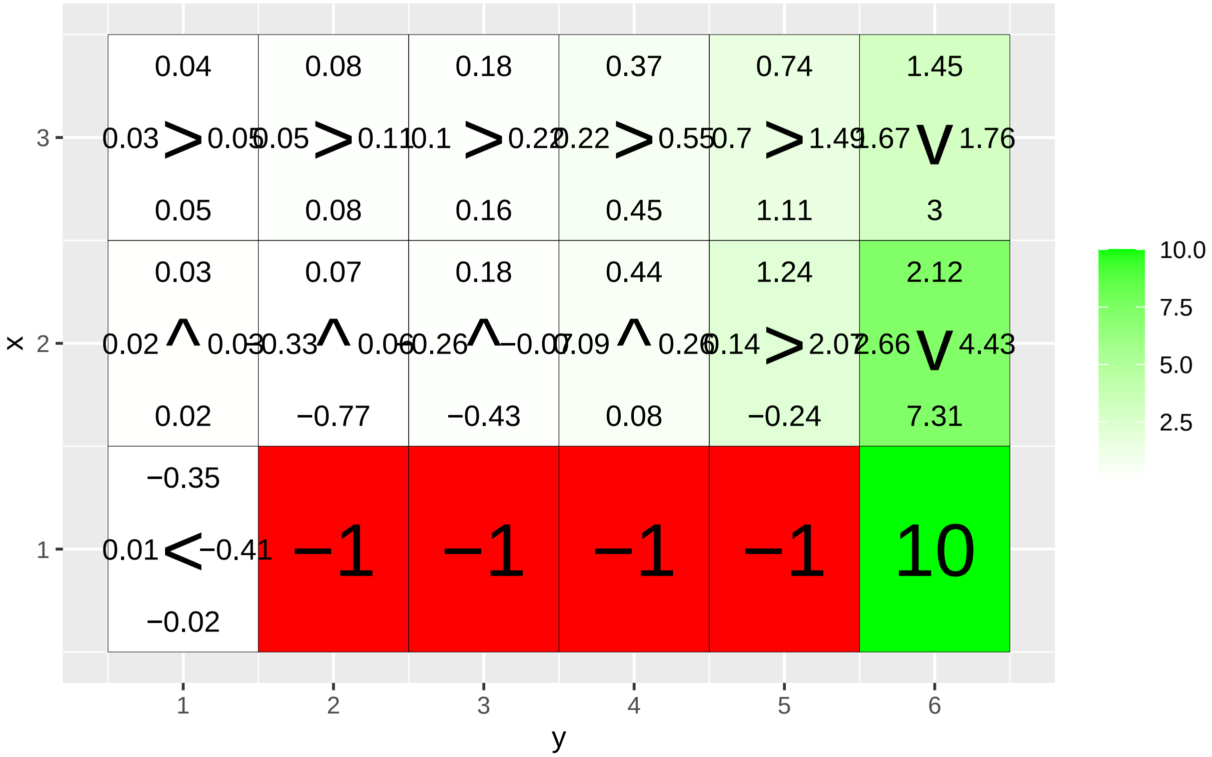
Environment C (the effect of beta).

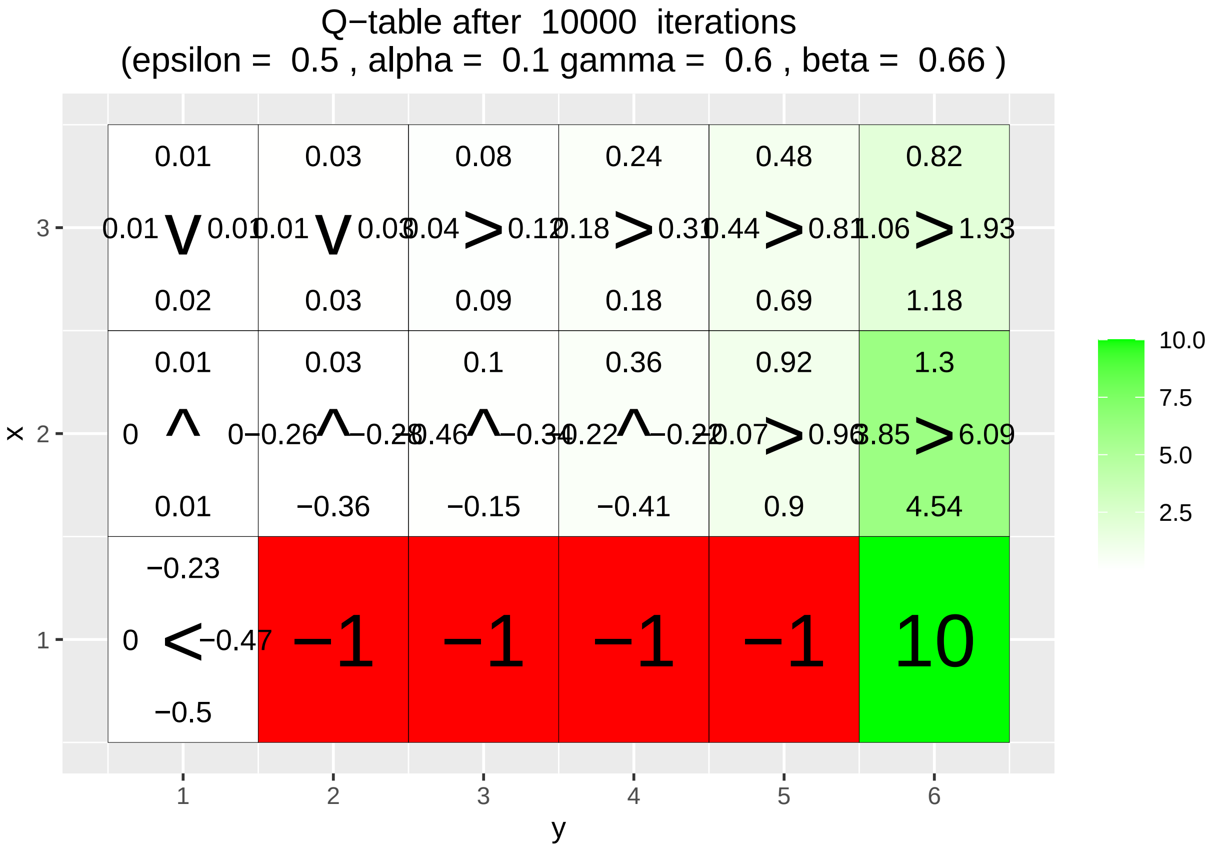





Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)
0.16
0.14 > 0.24
0.11
0.28
0.15 > 0.44
0.24
0.57
0.29 > 0.86
0.56
0.97
0.59 > 1.47
1.11
1.6
1.2 > 2.84
2.22
2.68
1.91 V 3.16
5.06
0.16
0.12 > 0.22
0.06
0.31
0.06 > 0.37
−0.59
0.51
−0.01 ^ 0.46
−0.73
1.1
0.5 > 1.49
−0.62
1.73
1.06 > 3.54
1.25
3.17
3.95 V 5.48
8.78
0
0.04 < −0.81
−0.06
−1
−1
−1
−1
10
x
3
2
1
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)





As we can see based on the plots, the higher β is, negative q-values at the edge states to the negative reward become less negative and the positive q-values related to actions that lead to the goal become closer to 0. That's because the agent can slip following the shorter path and fall in the trap.

It is also of important to claim that the agent learns to follow paths that are longer but further to the traps. However, when beta is too high the policy, the agent will just learn to avoid the traps, although it will be able to reach the goal accounting for the slip.

REINFORCE

2.6

- *Has the agent learned a good policy? Why / Why not ?* Yes, in all cases it would reach a goal. - *Could you have used the Q-learning algorithm to solve this task ?* Yes, but Q table would be huge. Lots of combinations relating for the different goals unlike as if we just had one goal.

2.7

- *Has the agent learned a good policy? Why / Why not ?* No, it learned to go to top row, because it learned just the x coordinates because in the training the goal had the same x-coordinate. Therefore, the learning has been biased, not exploring all possible states, the coordinate x was deemed not important for the agent and learning process.

- *If the results obtained for environments D and E differ, explain why.*

It looks like agent learned to ignore x value of a target, as in training task targets were always in top row meaning x was a constant.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
#install.packages("ggplot2")
#install.packages("vctrs")
library(ggplot2)
library(vctrs)
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}.

  # Locating the max q-value for all actions
  idx_max_val <- which(max(q_table[x,y,]) == q_table[x,y,])

  if (length(idx_max_val)==1){

    res<-idx_max_val

  # if there is more than 1 action with max q, break ties at random
  } else {res <- sample(idx_max_val, 1)}

  res
}

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}.

  # Set random behavior with probability epsilon
  random <- sample(c(0,1), 1, prob=c((1-epsilon), epsilon))

  if (random){res<- sample(1:4, 1)}

  else{res <- GreedyPolicy(x,y)}

  res

  }
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)
}
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)))

}

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.

  #Initialise the episode correction
  current_state <- start_state
  episode_correction <- 0

  repeat{

    # Follow policy, execute action, get reward.

    # Action defined as the Epsilon Greedy output
    action <- EpsilonGreedyPolicy(current_state[1], current_state[2], epsilon)

    # We get the new State accounting for the possible slip and random action
    new_state <- transition_model(current_state[1], current_state[2],
                                  action, beta)

    # Catch the reward in that coordinate
    reward <- reward_map[new_state[1],new_state[2]]

    # Calculate the correction
    corr <- reward +
      gamma * max(q_table[new_state[1],
      new_state[2],]) - q_table[current_state[1],current_state[2], action]

    # Accumulate correction
    episode_correction <- episode_correction + corr

    # Q-table update
    q_table[current_state[1],
            current_state[2], action] <<- q_table[current_state[1],
                                                  current_state[2],
                                                  action] + alpha*corr

    #set new state
    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)
}
```

```r
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")
}

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")
}
```

```r
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)
}
```