Lab 3 RL Individual Report

Akshay Gurudath

29/09/2021

2.1 Q-Learning Algorithm

```
# By Jose M. PeÃ±a and Joel Oskarsson.
# For teaching purposes.
# jose.m.pena@liu.se.

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

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

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

```
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.
  max <- which(q_table[x,y,] == max(q_table[x,y,]))
  if(length(max) > 1){
    max <- sample(max, 1)
  }
  return(max)

}

EpsilonGreedyPolicy <- function(x, y, epsilon){

  # Get an epsilon-greedy action for state (x,y) from q_table.
  #
  # Args:
```

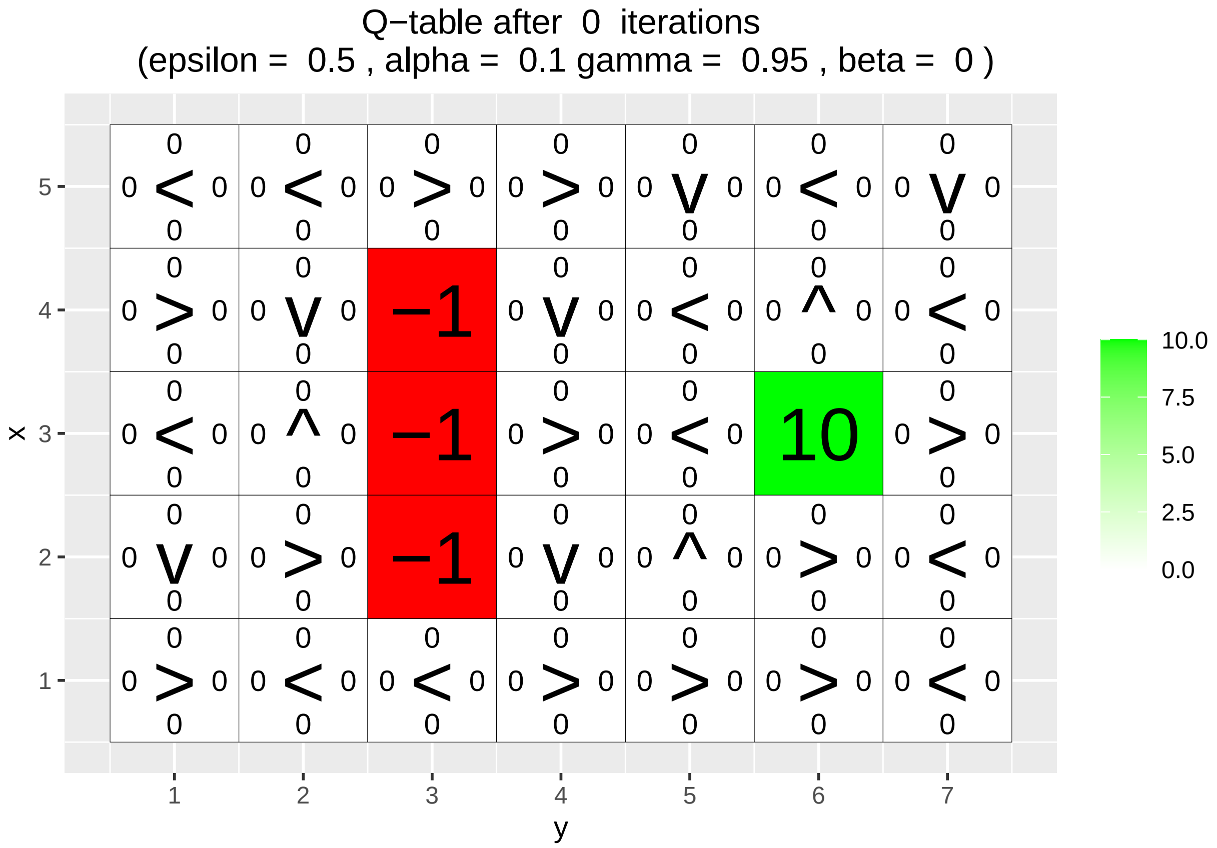
```
  #   x, y: state coordinates.
  #   epsilon: probability of acting randomly.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.

  # Your code here.
  a=GreedyPolicy(x,y)

  all=c(1,2,3,4)

  if(runif(1)<=1-epsilon){
    return(a)
  }
  else{
    return(sample(all,1))
  }

}

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.
  #start_state=c(sample.int(5,1),sample.int(7,1))
  x=start_state[1]
  y=start_state[2]

  episode_correction=0

  repeat{
    # Follow policy, execute action, get reward.
    a=EpsilonGreedyPolicy(x,y,epsilon=epsilon)
    #a_greedy=GreedyPolicy(x,y)

    new_state=transition_model(x,y,action=a,beta=beta)
   # print(new_state)
    reward=reward_map[new_state[1],new_state[2]]
    temp_diff=-q_table[x,y,a]+(reward+gamma*max(q_table[new_state[1],new_state[2],]))

    q_table[x,y,a]<<-q_table[x,y,a]+alpha*temp_diff
    # Q-table update.
    x=new_state[1]
    y=new_state[2]
    episode_correction=episode_correction+temp_diff
    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))
  }

}
```

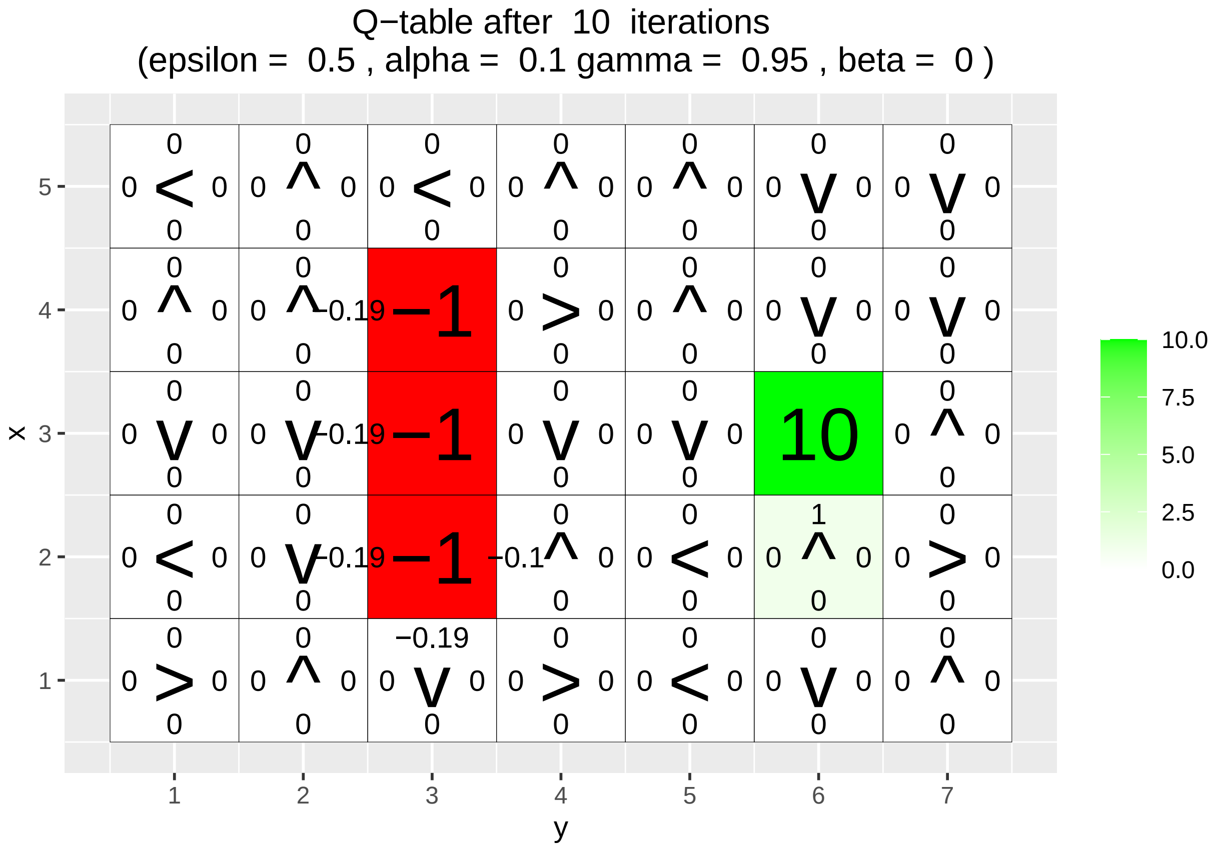
2.2 Environment A

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

# Environment A (learning)
set.seed(12345)
H <- 5
```

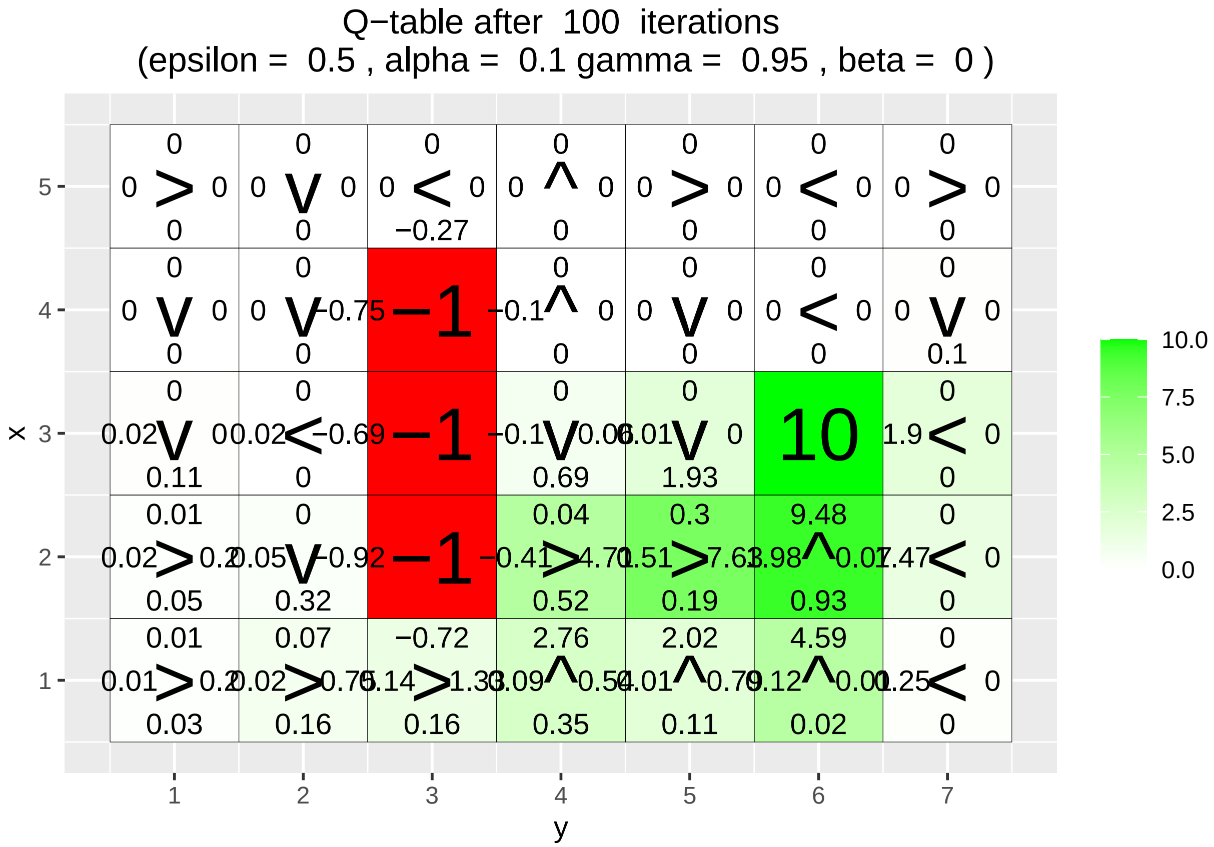
```r
W <- 7

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()
```

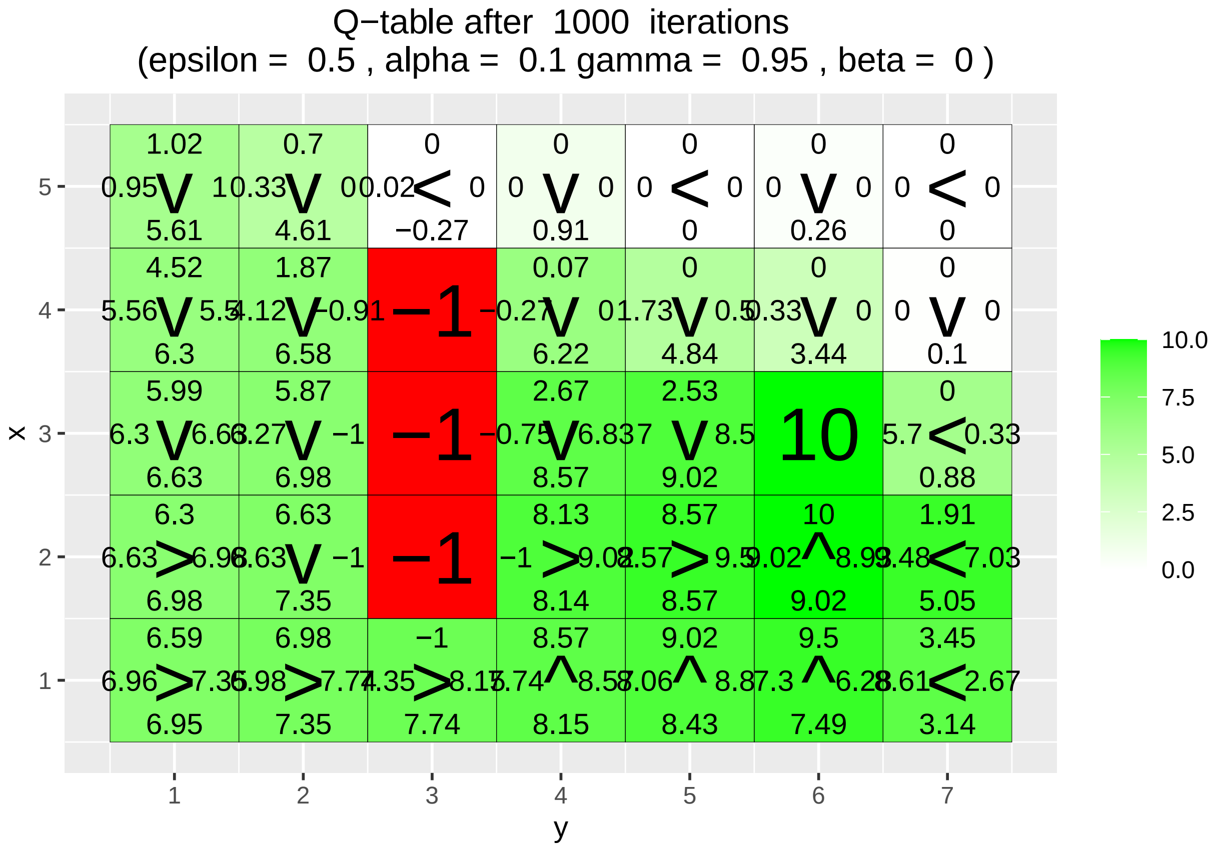


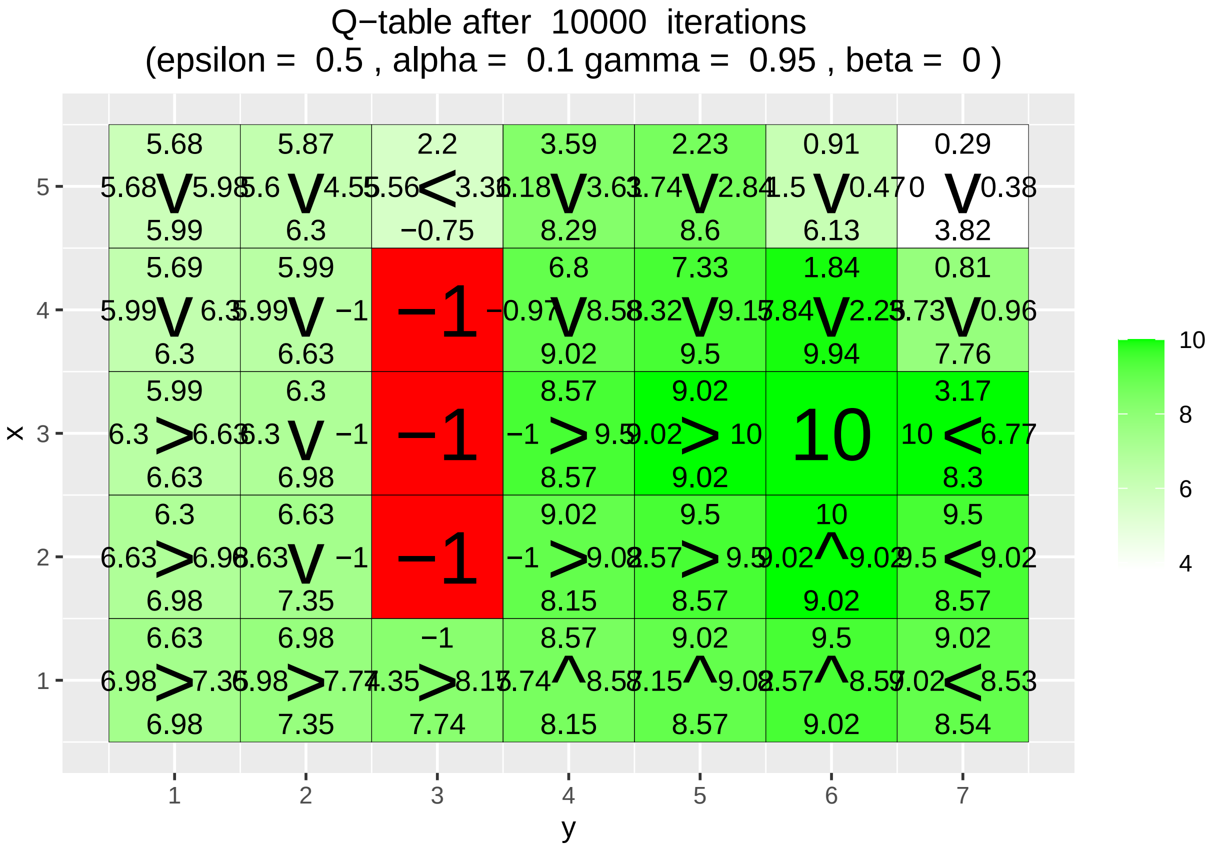
```r
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1),epsilon=0.5,gamma=0.95)

  if(any(i==c(10,100,1000,10000,20000,30000,40000)))
    vis_environment(i)
}
```



	1	2	3	4	5	6	7
5	0 0 < 0 0	0 0 ^ 0 0	0 0 < 0 0	0 0 ^ 0 0	0 0 ^ 0 0	0 0 V 0 0	0 0 V 0 0
4	0 0 ^ 0 0	0 0 ^ −0.19 0	−1	0 0 > 0 0	0 0 ^ 0 0	0 0 V 0 0	0 0 V 0 0
3	0 0 V 0 0	0 0 V −0.19 0	−1	0 0 V 0 0	0 0 V 0 0	10	0 0 ^ 0 0
2	0 0 < 0 0	0 0 V −0.19 0	−1	0 −0.1 ^ 0 0	0 0 < 0 0	1 0 ^ 0 0	0 0 > 0 0
1	0 0 > 0 0	0 0 ^ 0 0	−0.19 0 V 0 0	0 0 > 0 0	0 0 < 0 0	0 0 V 0 0	0 0 ^ 0 0


Q−table after 100 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
1
2
3
4
5
6
7
−1
−1
−1
10
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)
1.02 0.7 0 0 0 0 0
0.95 V 1 0.33 V 0 0.02 < 0 0 V 0 0 < 0 0 V 0 0 < 0
5.61 4.61 −0.27 0.91 0 0.26 0
4.52 1.87 0.07 0 0 0
5.56 V 5.5 4.12 V −0.91 −1 −0.27 V 0 1.73 V 0.5 0.33 V 0 0 V 0
6.3 6.58 6.22 4.84 3.44 0.1
5.99 5.87 2.67 2.53 0
6.3 V 6.66 3.27 V −1 −1 −0.75 V 6.83 7 V 8.5 10 5.7 < 0.33
6.63 6.98 8.57 9.02 0.88
6.3 6.63 8.13 8.57 10 1.91
6.63 > 6.98 6.63 V −1 −1 −1 > 9.08 8.57 > 9.5 9.02 ^ 8.93 3.48 < 7.03
6.98 7.35 8.14 8.57 9.02 5.05
6.59 6.98 −1 8.57 9.02 9.5 3.45
6.96 > 7.35 6.98 > 7.74 7.35 > 8.15 7.74 ^ 8.58 8.06 ^ 8.8 7.3 ^ 6.28 3.61 < 2.67
6.95 7.35 7.74 8.15 8.43 7.49 3.14
x
5
4
3
2
1
1 2 3 4 5 6 7
y
10.0
7.5
5.0
2.5
0.0



a. The agent has learned almost nothing in the first 10 episodes. It has learned that the states close to -1 are states to stay away from. However, because of the duration, it has only learned so much and not explored other paths.

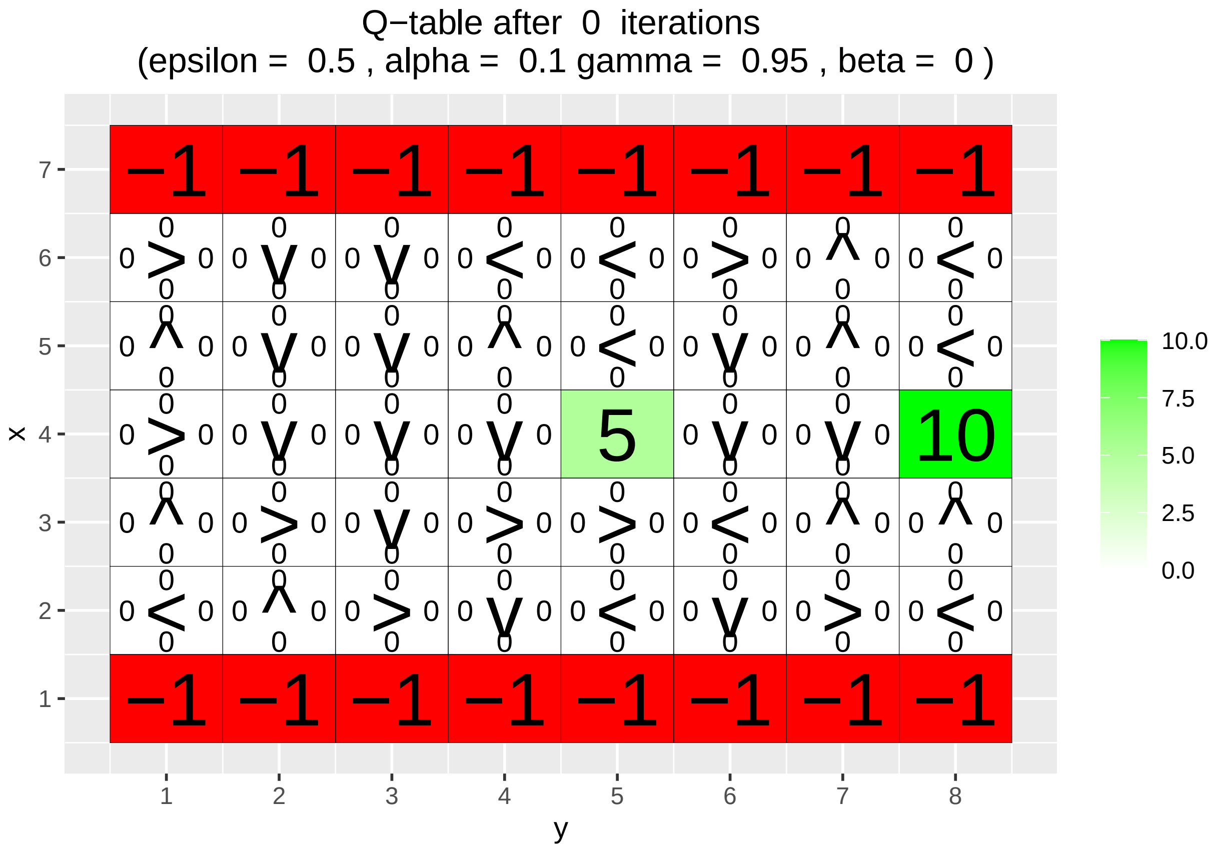
b. The final greedy policy isn't exactly optimal for all states as some states are less visited than the others and their q-values cannot be trusted with the same certainty as some other states which are visited more often.

c. No multiple paths are not reflected in the current solution (both upward and downward). These paths are not perfect and a better way to show multiple paths is by initializing the states randomly each time an episode starts. This will result in a more uniform visitation of states and thus will show other paths to the objective from every state.

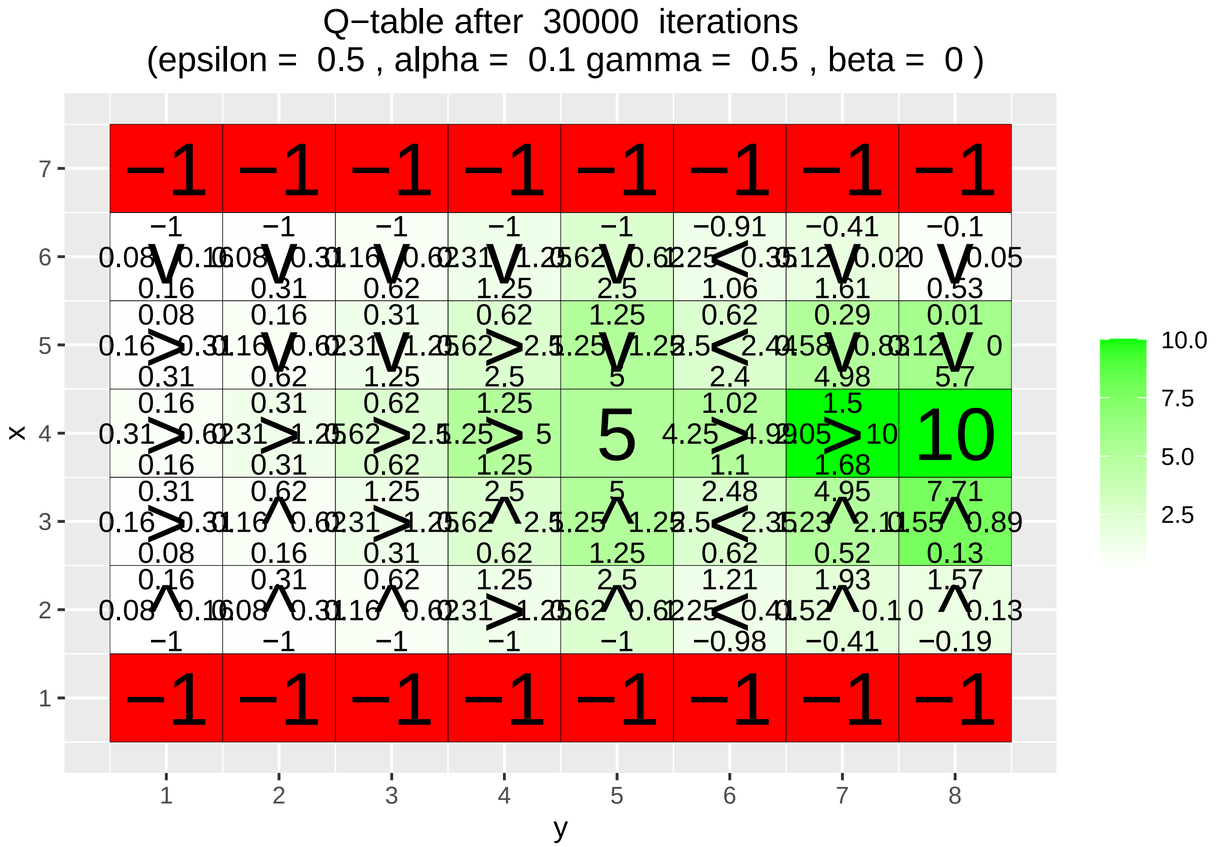
2.3 Environment B

```
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
```

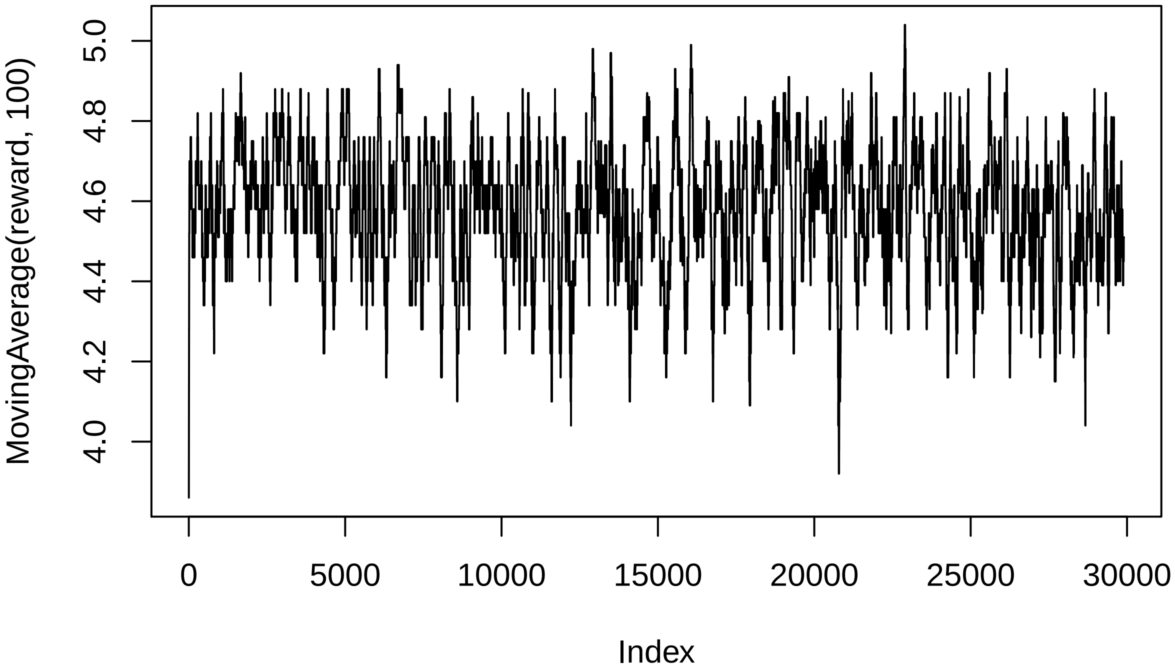
```r
q_table <- array(0,dim = c(H,W,4))

vis_environment()
```

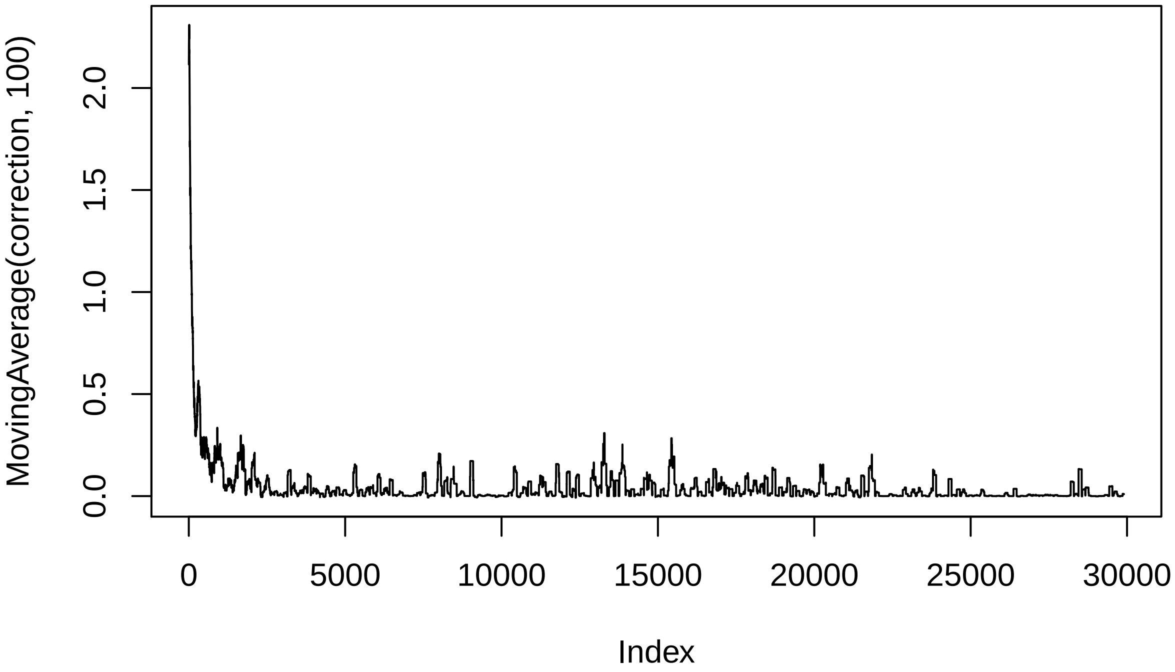


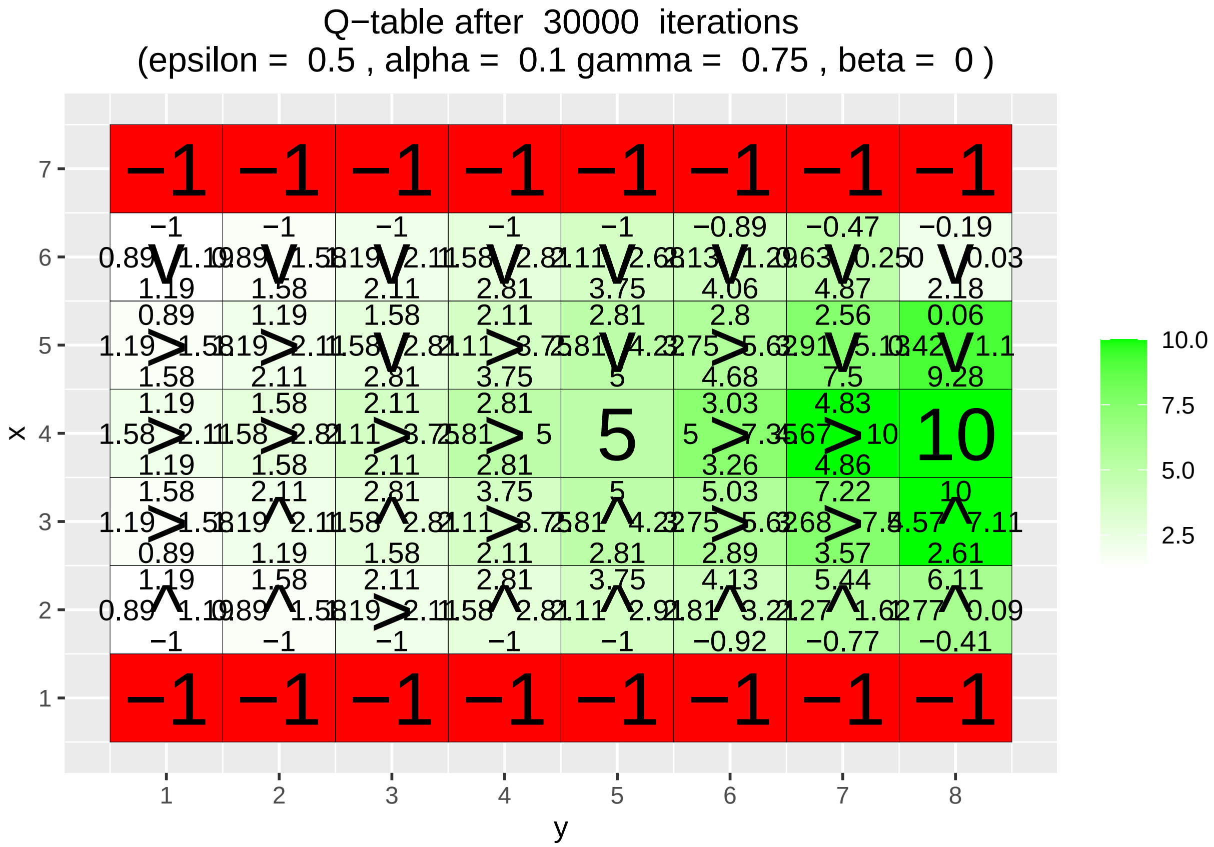
```r
MovingAverage <- function(x, n){

  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")
}
```

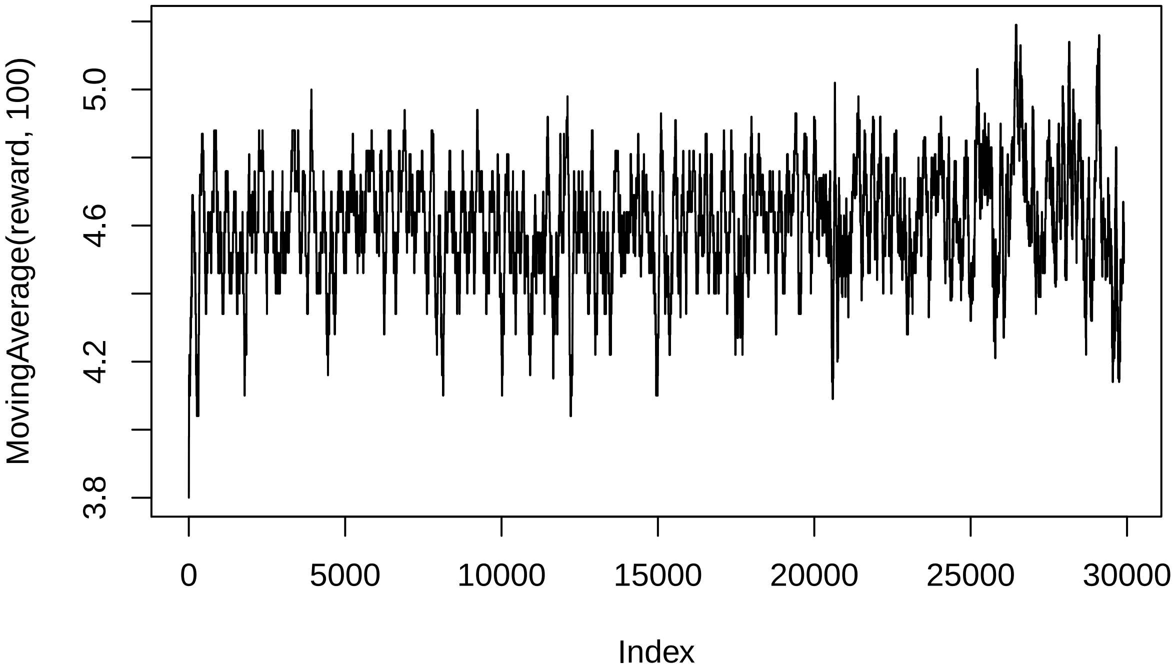


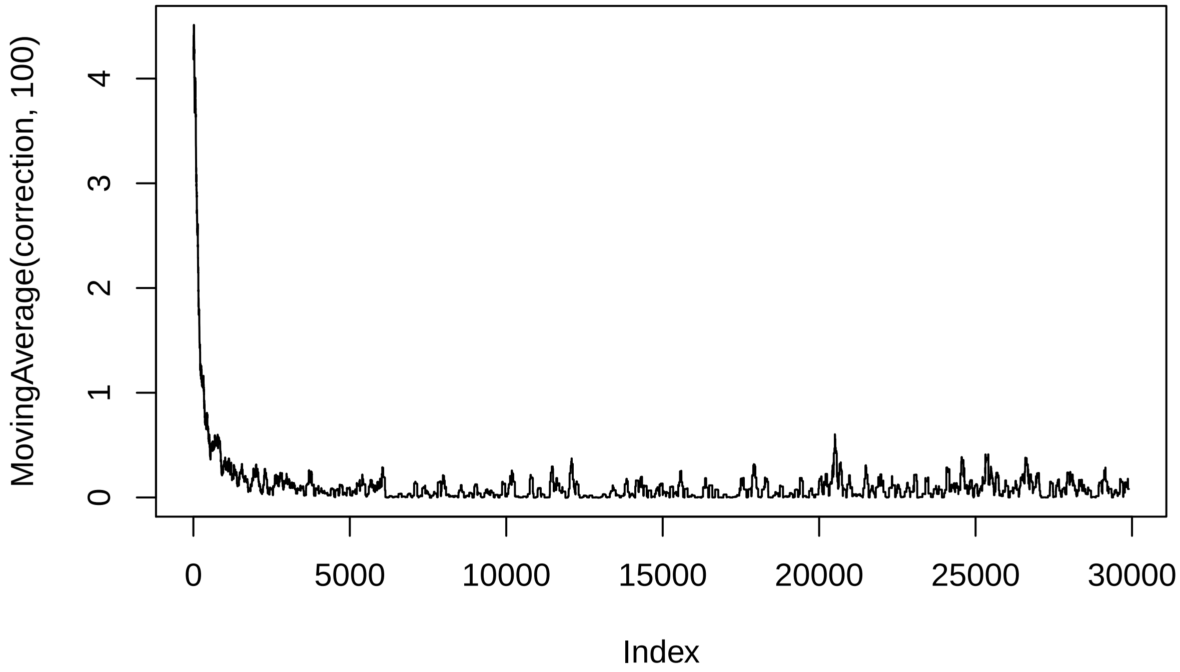
Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.5 , beta = 0)
7
6
5
4
3
2
1
x
1
2
3
4
5
6
7
8
y
10.0
7.5
5.0
2.5
−1 −1 −1 −1 −1 −1 −1 −1
−1 0.08 V 0.16 0.16
−1 0.08 V 0.31 0.31
−1 0.16 V 0.62 0.62
−1 0.31 V 1.25 1.25
−1 0.62 V 0.62 2.5
−0.91 1.25 < 0.35 1.06
−0.41 0.12 V 0.02 1.61
−0.1 0 V 0.05 0.53
0.08 0.16 > 0.31 0.31
0.16 0.16 V 0.62 0.62
0.31 0.31 V 1.25 1.25
0.62 0.62 > 2.5 2.5
1.25 1.25 V 1.25 5
0.62 2.5 < 2.4 2.4
0.29 0.58 V 0.83 4.98
0.01 0.12 V 0 5.7
0.16 0.31 > 0.62 0.16
0.31 0.31 > 1.25 0.31
0.62 0.62 > 2.5 0.62
1.25 1.25 > 5 1.25
5
1.02 4.25 > 4.92 1.1
1.5 2.05 > 10 1.68
10
0.31 0.16 > 0.31 0.08
0.62 0.16 ^ 0.62 0.16
1.25 0.31 > 1.25 0.31
2.5 0.62 ^ 2.5 0.62
5 1.25 ^ 1.25 1.25
2.48 2.5 < 2.35 0.62
4.95 1.23 ^ 2.1 0.52
7.71 1.55 ^ 0.89 0.13
0.16 0.08 ^ 0.16 −1
0.31 0.08 ^ 0.31 −1
0.62 0.16 ^ 0.62 −1
1.25 0.31 > 1.25 −1
2.5 0.62 ^ 0.62 −1
1.21 1.25 < 0.4 −0.98
1.93 0.52 ^ 0.1 −0.41
1.57 0 ^ 0.13 −0.19
−1 −1 −1 −1 −1 −1 −1 −1


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

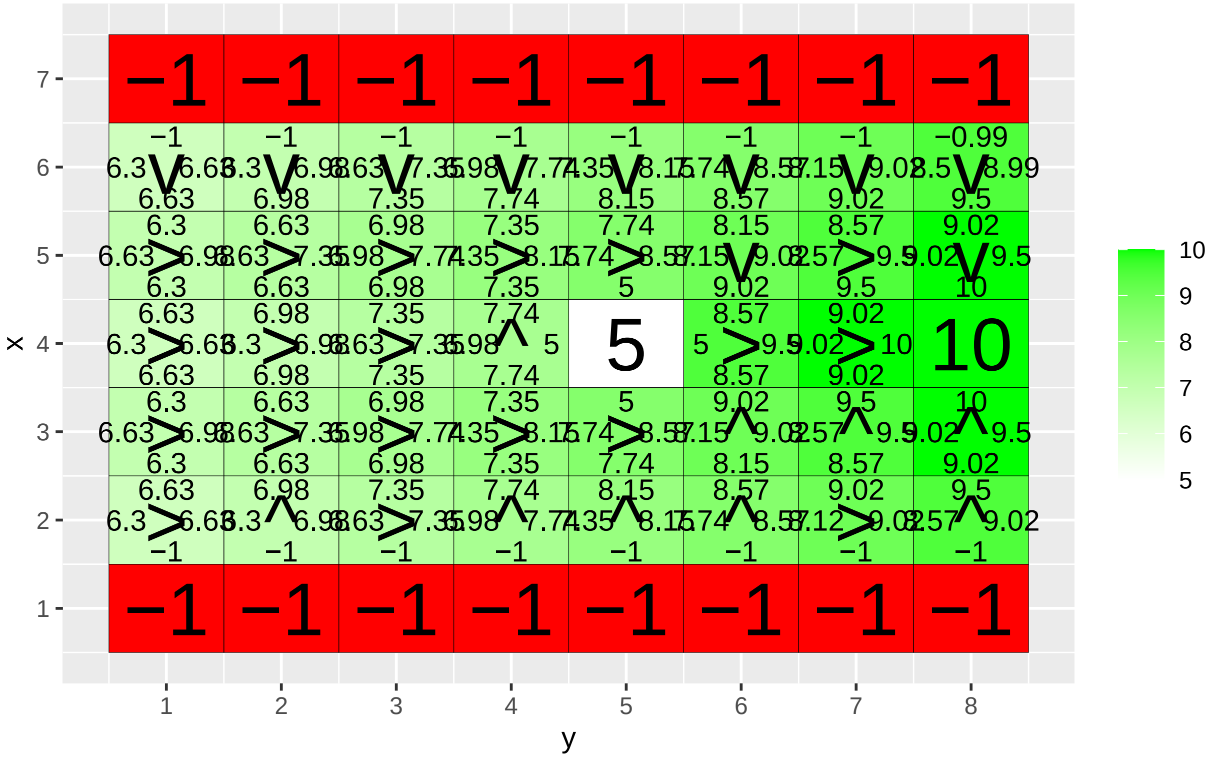

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index

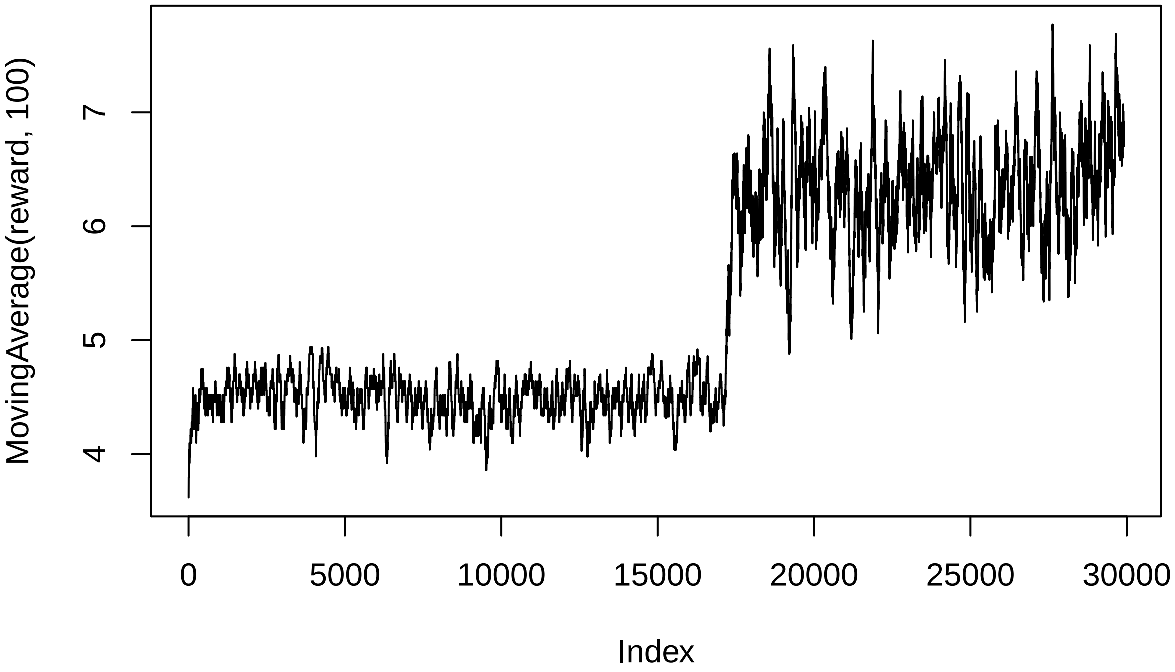

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
−1 −1 −1 −1 −1 −1 −1 −1
−1 −1 −1 −1 −1 −0.89 −0.47 −0.19
0.89 V 1.19 0.89 V 1.58 1.19 V 2.11 1.58 V 2.81 2.11 V 2.68 2.13 V 1.29 0.63 V 0.25 0 V 0.03
1.19 1.58 2.11 2.81 3.75 4.06 4.87 2.18
0.89 1.19 1.58 2.11 2.81 2.8 2.56 0.06
1.19 > 1.58 1.19 > 2.11 1.58 V 2.81 2.11 > 3.75 2.81 V 4.23 3.75 > 5.63 2.91 V 5.13 0.42 V 1.1
1.58 2.11 2.81 3.75 5 4.68 7.5 9.28
1.19 1.58 2.11 2.81 3.03 4.83
1.58 > 2.11 1.58 > 2.81 2.11 > 3.75 2.81 > 5 5 5 > 7.34 5.67 > 10 10
1.19 1.58 2.11 2.81 3.26 4.86
1.58 2.11 2.81 3.75 5 5.03 7.22 10
1.19 > 1.58 1.19 ^ 2.11 1.58 ^ 2.81 2.11 > 3.75 2.81 ^ 4.23 3.75 > 5.63 3.68 > 7.5 4.57 ^ 7.11
0.89 1.19 1.58 2.11 2.81 2.89 3.57 2.61
1.19 1.58 2.11 2.81 3.75 4.13 5.44 6.11
0.89 ^ 1.19 0.89 ^ 1.58 1.19 > 2.11 1.58 ^ 2.81 2.11 ^ 2.91 2.81 ^ 3.21 2.27 ^ 1.61 1.77 ^ 0.09
−1 −1 −1 −1 −1 −0.92 −0.77 −0.41
−1 −1 −1 −1 −1 −1 −1 −1
x
7
6
5
4
3
2
1
y
1 2 3 4 5 6 7 8
10.0
7.5
5.0
2.5

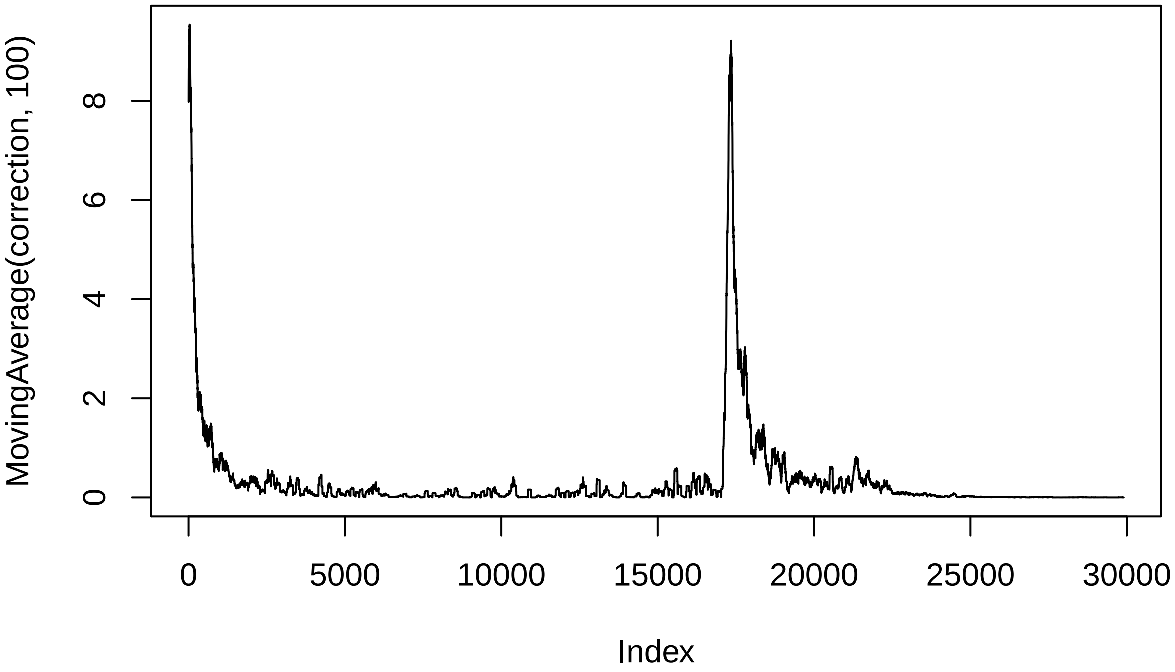

MovingAverage(reward, 100)
3.8
4.2
4.6
5.0
0
5000
10000
15000
20000
25000
30000
Index


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

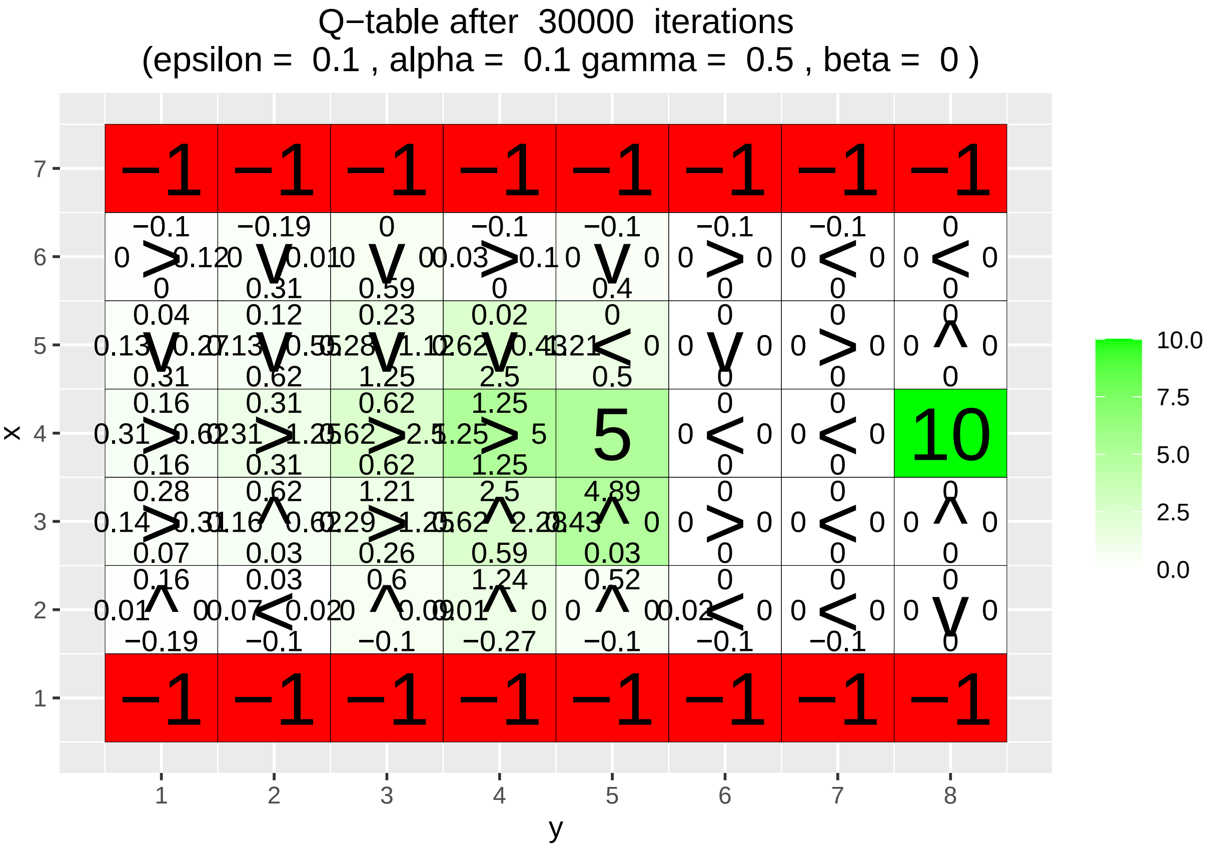


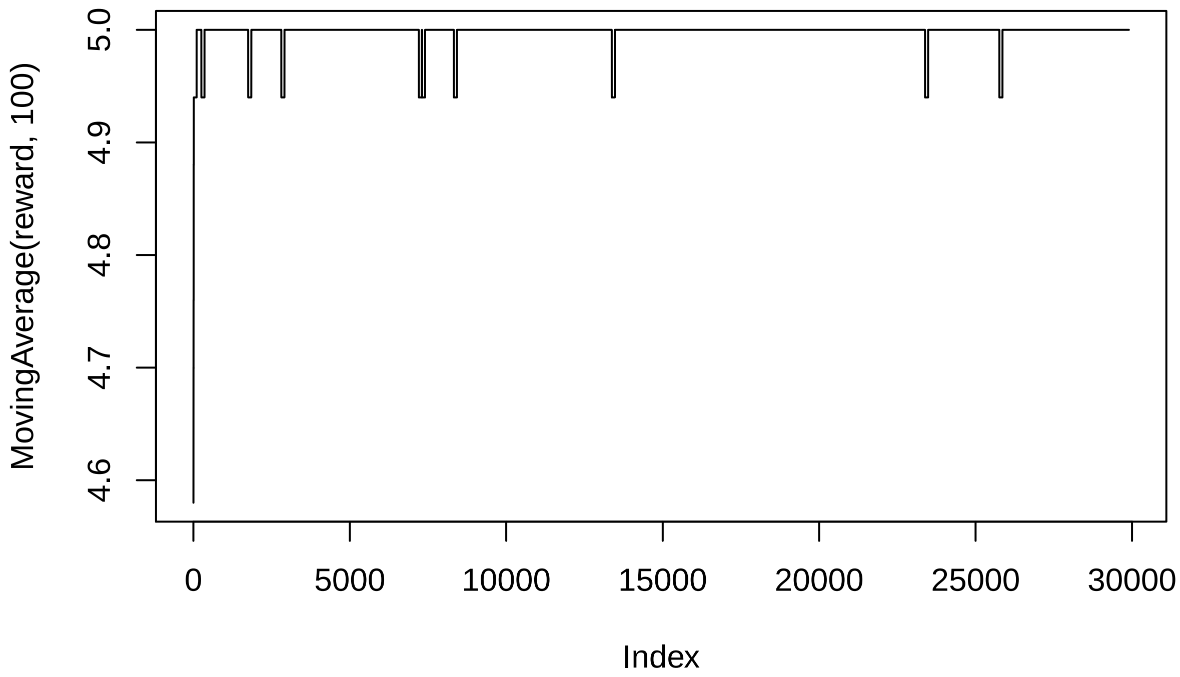
When $\gamma = 0.5$,we notice that there are a few states close to 10 which point to 10, and many other states seem to point towards 5. We also notice that a lot of these states are not colored, which means that they don't have significant values of q. This is probably because states far away from the reward states get a low cumulative reward as the weighting factor is less (0.5).

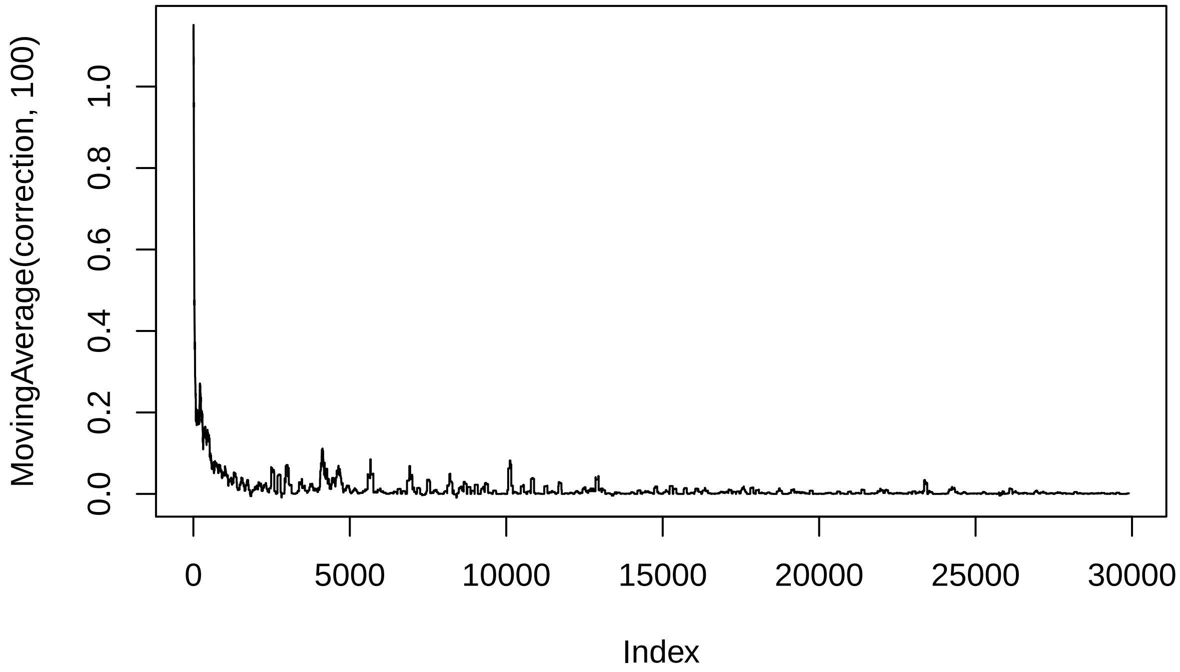
When $\gamma = 0.75$, we notice that a lot more of these states are colored green, because they get a better cumulative reward. We also notice that there is more tendency to move to the +10 reward.

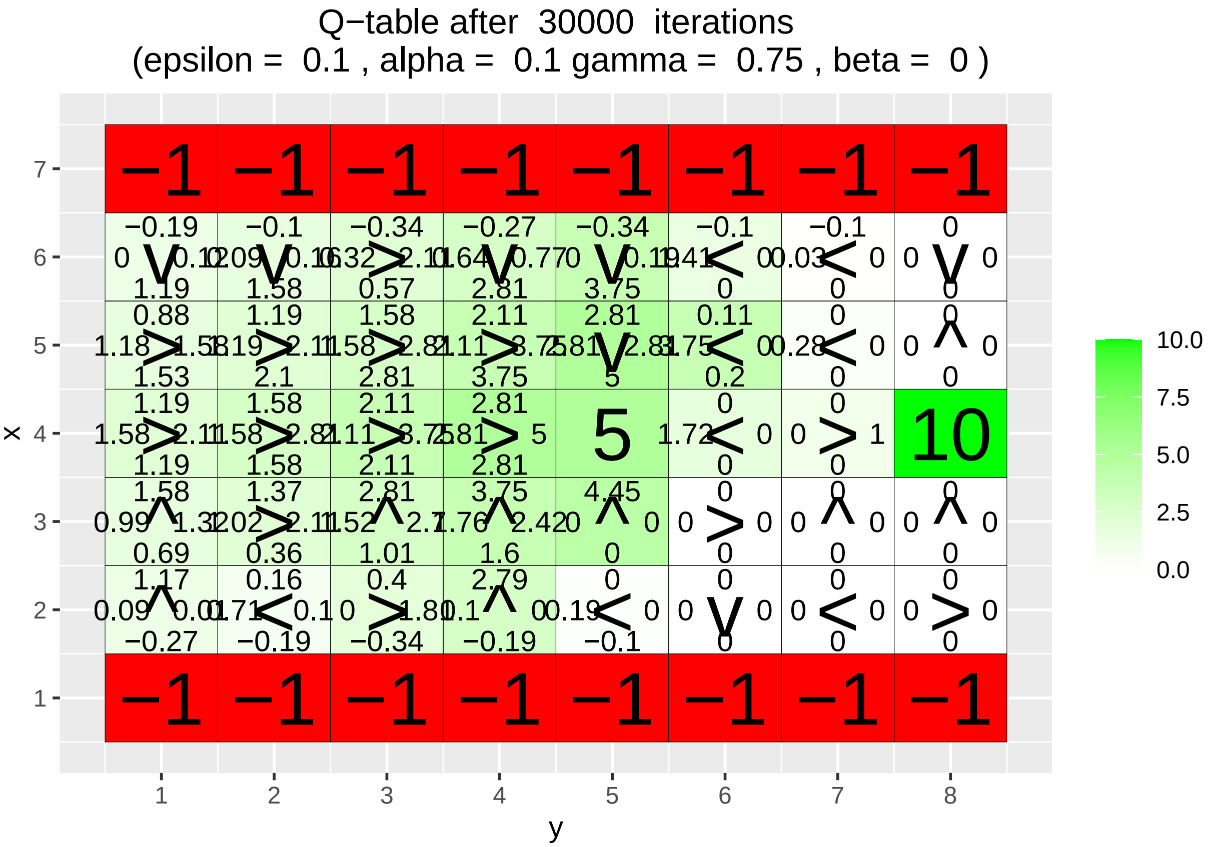
When $\gamma = 0.95$ most of the states are colored green implying that they get the best cumulative rewards because of the best discounting. We also notice that there is most tendency to go to the +10 reward. We also see a jump in the cumulative reward showing that the agent now is tending to go to the +10 reward states,

```r
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")
}
```

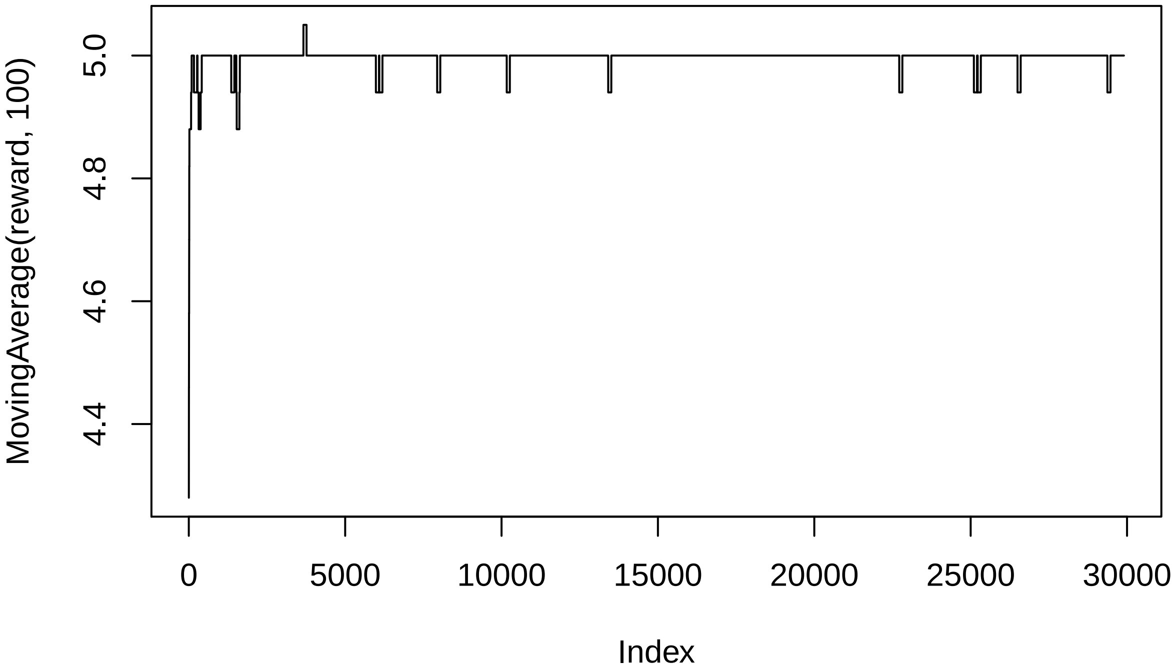

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
7
6
5
4
3
2
1
y
1
2
3
4
5
6
7
8
10.0
7.5
5.0
2.5
0.0
−1
−1
−1
−1
−1
−1
−1
−1
−0.1 0 > 0.12 0
−0.19 0 V 0.01 0.31
0 0 V 0 0.59
−0.1 0.03 > 0.1 0
−0.1 0 V 0 0.4
−0.1 0 > 0 0
−0.1 0 < 0 0
0 0 < 0 0
0.04 0.13 V 0.31
0.12 V 0.62
0.23 V 1.25
0.02 V 2.5
0 < 0 0.5
0 0 V 0 0
0 0 > 0 0
0 0 ^ 0 0
0.16 0.31 > 0.16
0.31 > 0.31
0.62 > 2.5 0.62
1.25 1.25 > 5 1.25
5
0 0 < 0 0
0 0 < 0 0
10
0.28 0.14 > 0.07
0.62 ^ 0.03
1.21 > 0.26
2.5 ^ 0.59
4.89 ^ 0 0.03
0 0 > 0 0
0 0 < 0 0
0 0 ^ 0 0
0.16 0.01 ^ 0 −0.19
0.03 0.07 < −0.1
0.6 0 ^ −0.1
1.24 ^ 0 −0.27
0.52 0 ^ 0 −0.1
0 0.02 < 0 −0.1
0 0 < 0 −0.1
0 0 V 0 0
−1
−1
−1
−1
−1
−1
−1
−1

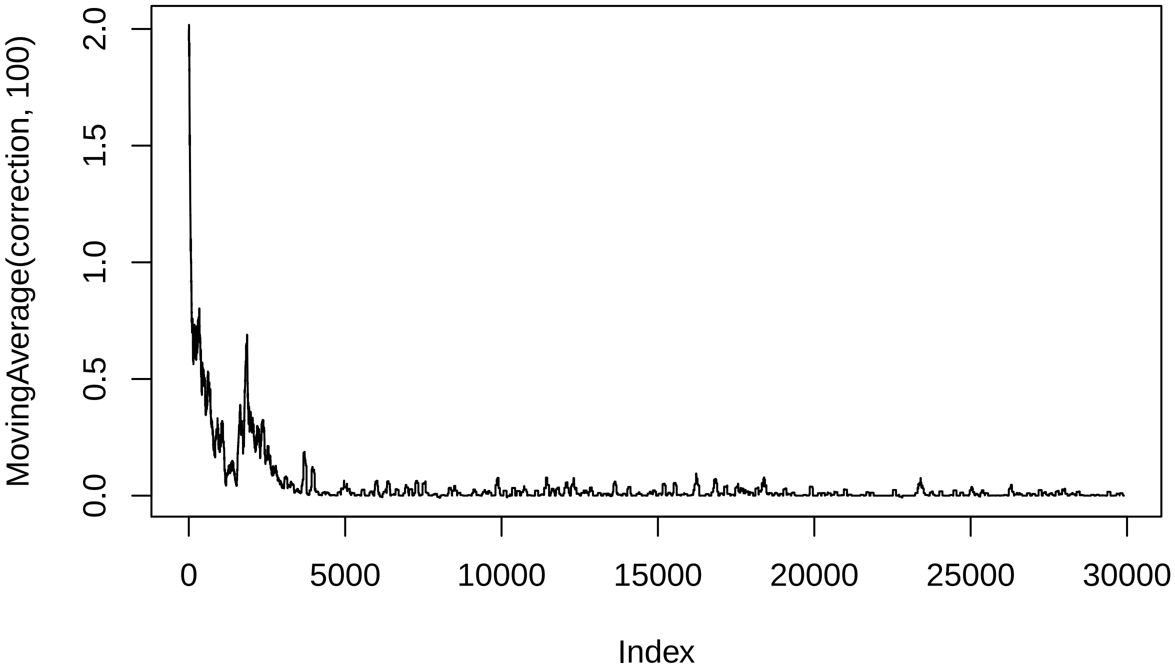

MovingAverage(reward, 100)
4.6
4.7
4.8
4.9
5.0
0
5000
10000
15000
20000
25000
30000
Index


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

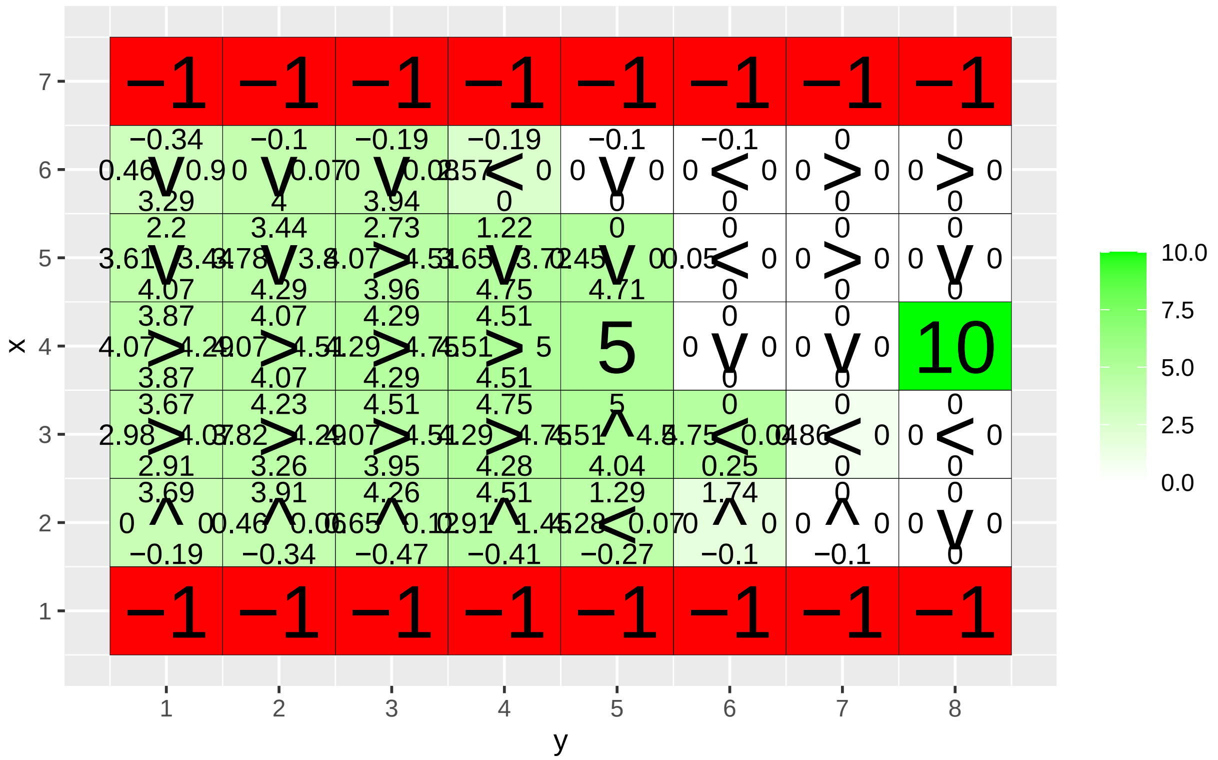


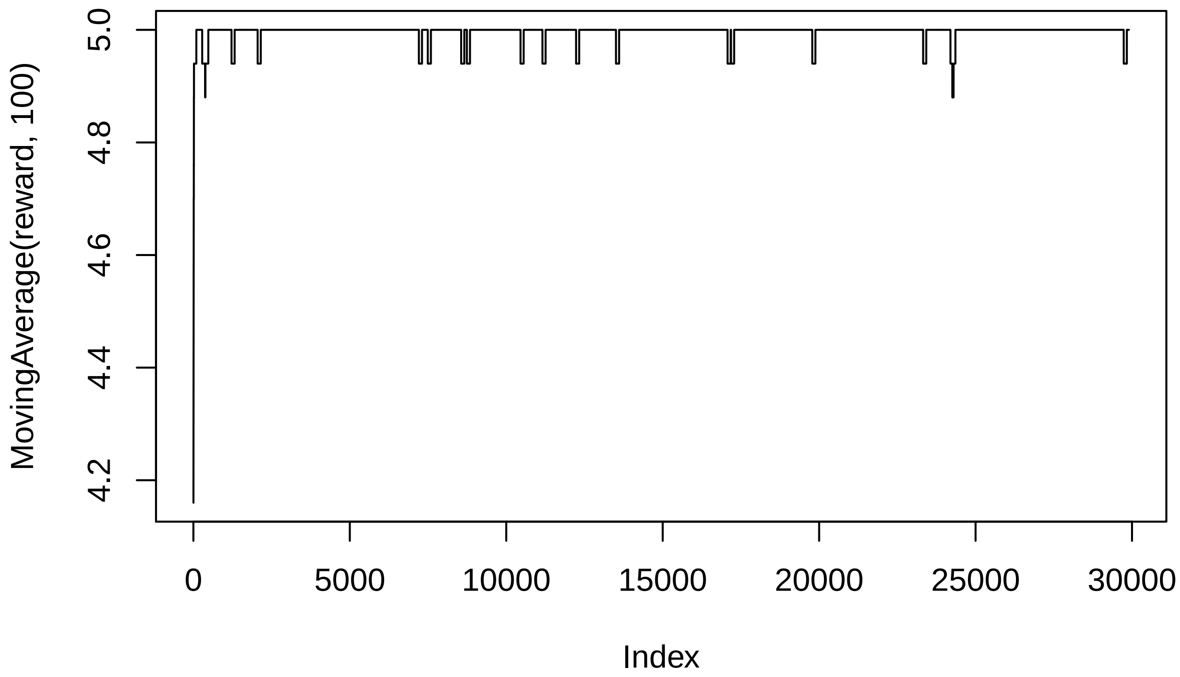
Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)

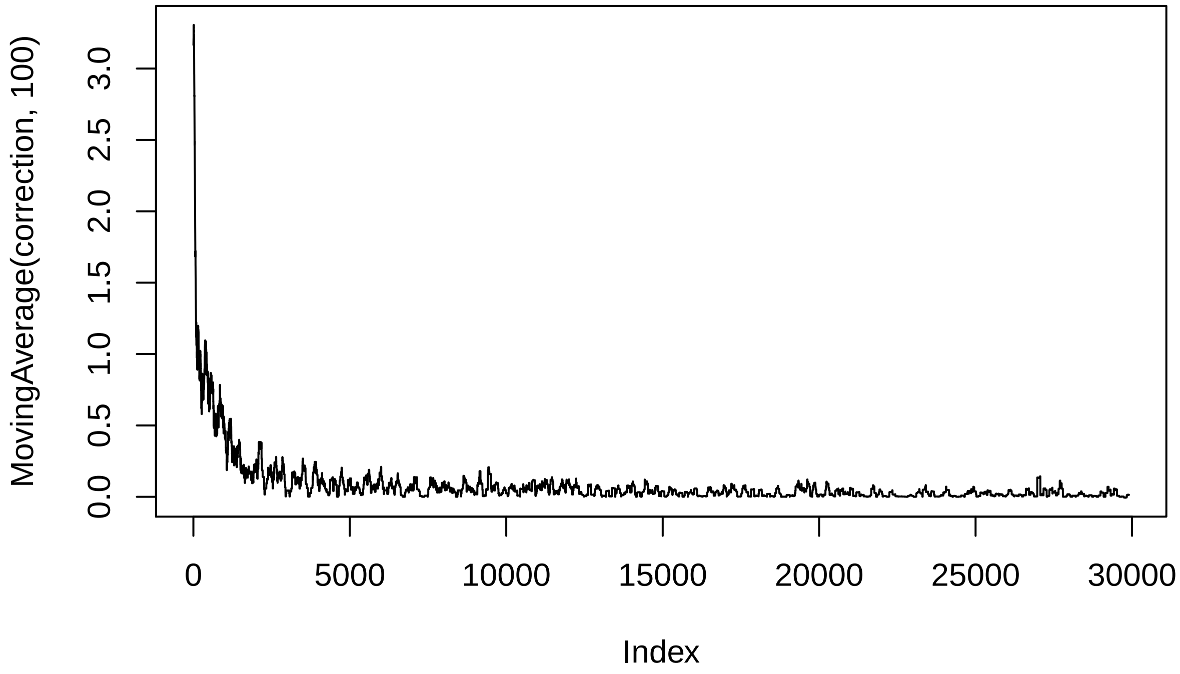

MovingAverage(reward, 100)
5.0
4.8
4.6
4.4
0
5000
10000
15000
20000
25000
30000
Index


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)



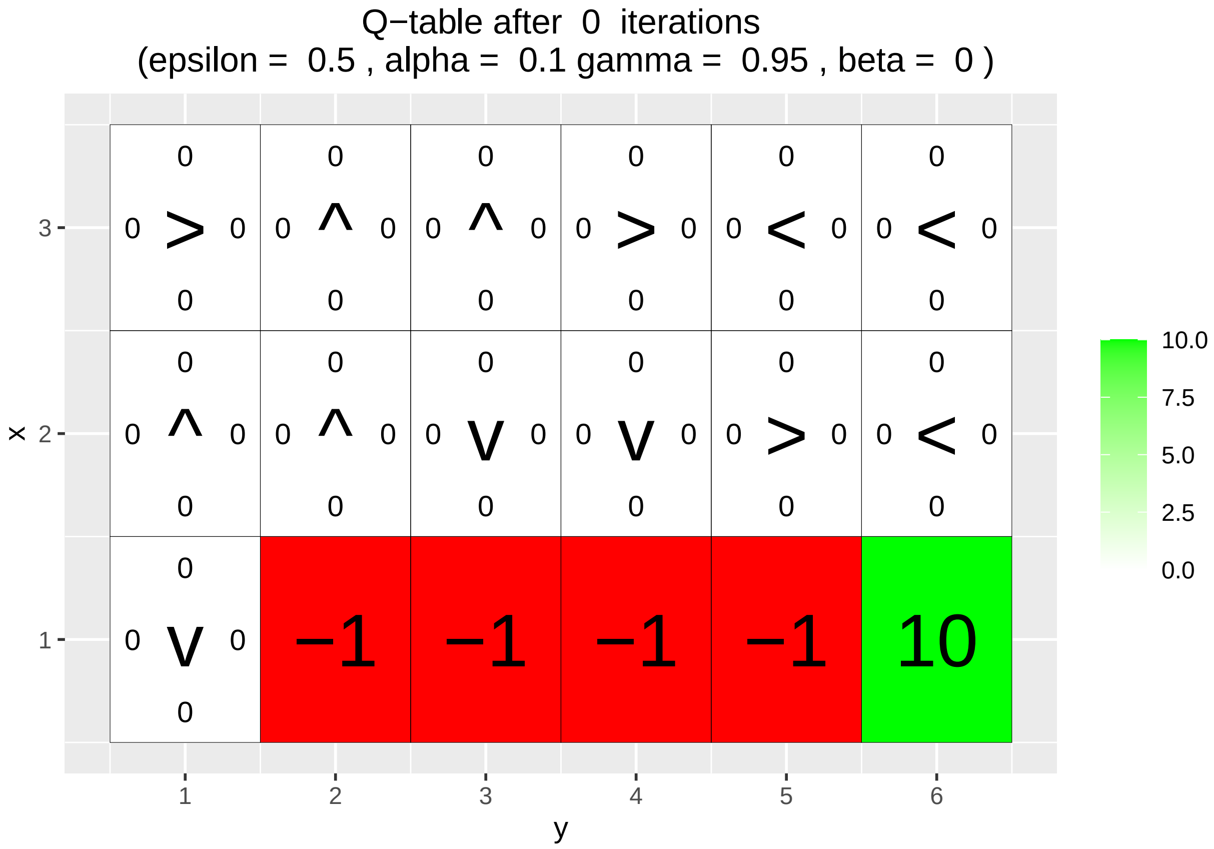

MovingAverage(reward, 100)
4.2
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index



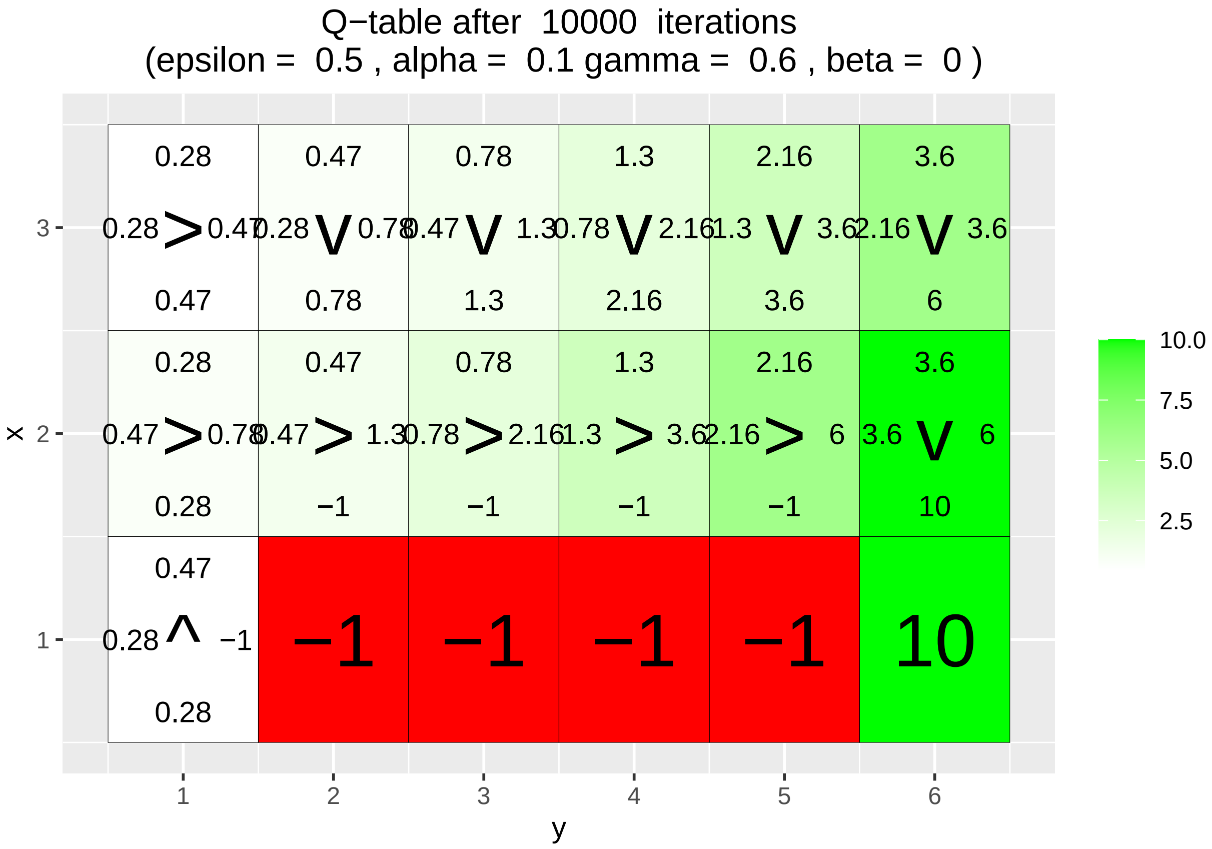
The first thing we notice is that now the agent seems to find the +5 reward in each episode. This is because of a low value of $\epsilon = 0.1$ leading to less exploration and more trust in believing the existing policy. Therefore the episode correction converges faster as the solution is found faster. However, we notice that now generally there is a less chance of visiting +10 again due to lesser exploration. The gamma values don't really seem to influence the rewards much.

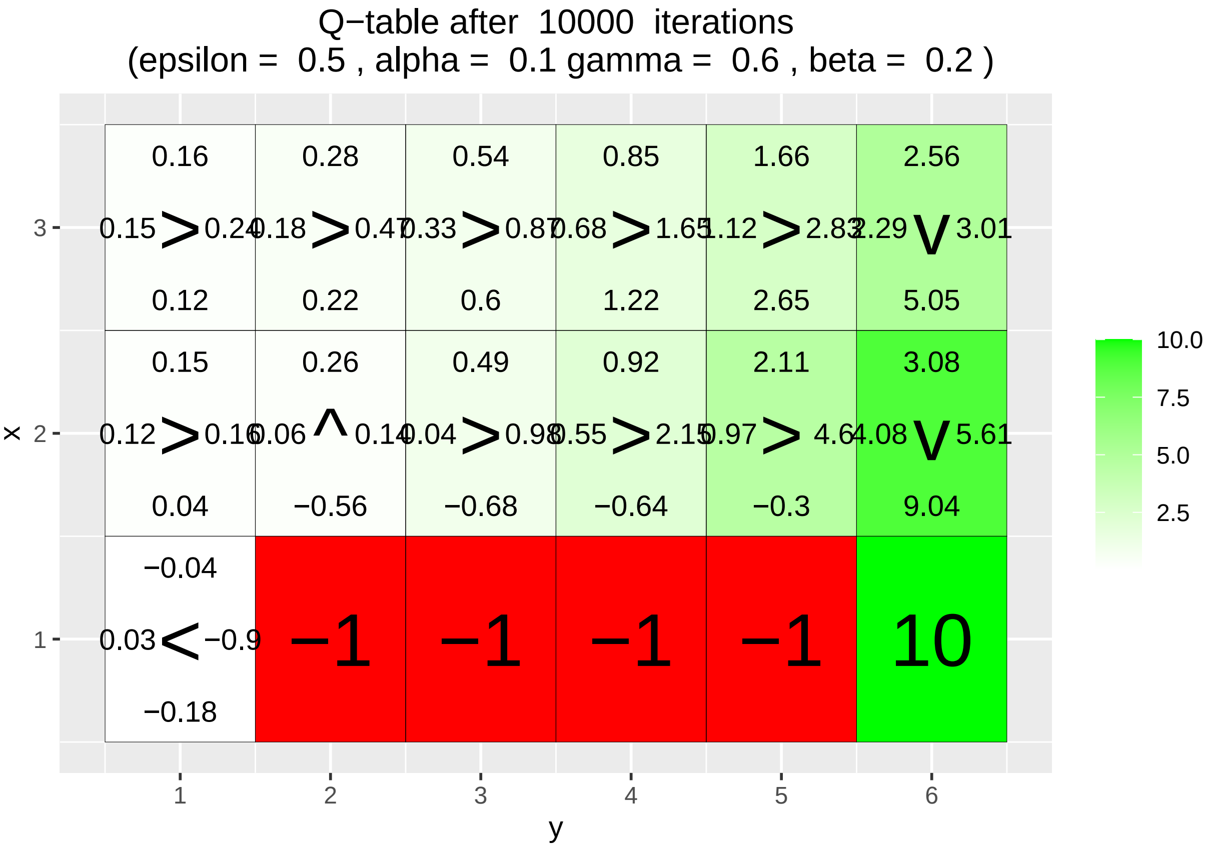
2.4 Environment C

```
# 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()
```

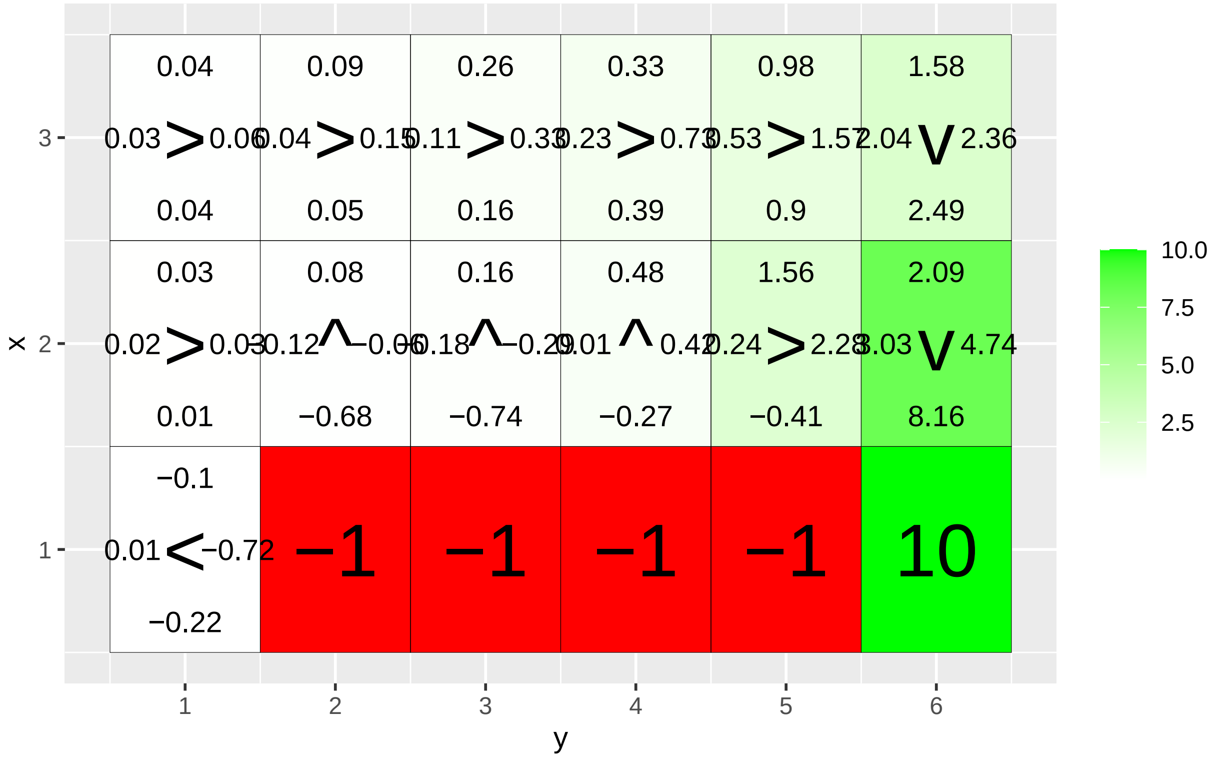


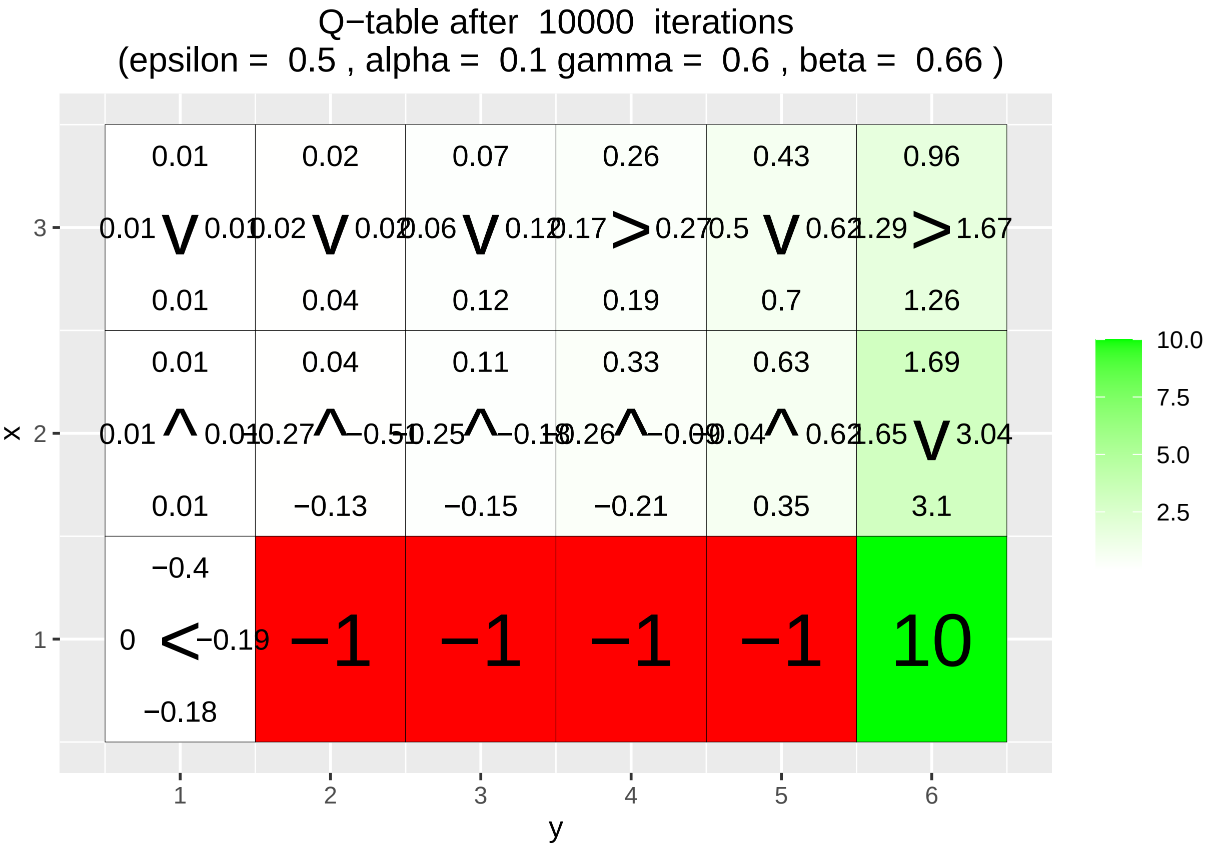
```
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

  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)
0.28
0.28 > 0.47
0.47
0.47
0.28 V 0.78
0.78
0.78
0.47 V 1.3
1.3
1.3
0.78 V 2.16
2.16
2.16
1.3 V 3.6
3.6
3.6
2.16 V 3.6
6
0.28
0.47 > 0.78
0.28
0.47
0.47 > 1.3
−1
0.78
0.78 > 2.16
−1
1.3
1.3 > 3.6
−1
2.16
2.16 > 6
−1
3.6
3.6 V 6
10
0.47
0.28 ^ −1
0.28
−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 = 0.2)
0.16
0.15 > 0.24
0.12
0.28
0.18 > 0.47
0.22
0.54
0.33 > 0.87
0.6
0.85
0.68 > 1.65
1.22
1.66
1.12 > 2.83
2.65
2.56
2.29 V 3.01
5.05
0.15
0.12 > 0.16
0.04
0.26
0.06 ^ 0.14
−0.56
0.49
0.04 > 0.98
−0.68
0.92
0.55 > 2.15
−0.64
2.11
0.97 > 4.6
−0.3
3.08
4.08 V 5.61
9.04
−0.04
0.03 < −0.9
−0.18
−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)





We see that as the slipping factor increases, the policy changes from finding the +10 reward to avoiding the -1 reward.

2.5 Environment D

1. Yes, we believe the agent has learned a good policy that is stochastic. Depending on where the goal position is, the policy is changed accordingly. It has done so because the policy parameters have updated to maximize the expected reward.
2. No Q-learning cannot be very easily applied here as there are a lot of state action pairs. There are n*(n-1) states (where n is the number of grids) and 4 actions. This would require updating a lot of Q values. Instead it would be easier to update the policy itself.

2.6 Environment E

1. No the agent has not learned a good policy because it was trained only on some goals at the side of the grid. These goals were not randomly distributed for training and made the agent learn a different policy (like going to the sides of the grid rather than trying to go closer to a goal)
2. Yes the results are different because both these environments had different kinds of training -where one was biased and one was not.