Lab3

Oscar Hoffmann

2024-10-05

2.1 Q-Learning

Complete the Q-learning algorithm for a grid-world environment where the agent navigates using four actions (up, down, left, right) and faces a probability of slipping. The agent must learn the optimal policy using a model-free approach, as the transition model is unknown.

```r
rm(list =ls())
set.seed(1234)
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){

  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 the Q-values for all actions at state (x, y)
  q_values <- q_table[x, y, ]

  # Find the max Q-value
  max_q <- max(q_values)

  # Identify all actions with maximum Q-value
  max_actions <- which(q_values == max_q)

  # Check and resolve ties
  if (length(max_actions) > 1) {
    action <- sample(max_actions, 1)
  } else {
    action <- max_actions
  }
  return(action)
}

EpsilonGreedyPolicy <- function(x, y, epsilon){
  # Generate a random numb
  rand_num <- runif(1)
  if (rand_num < epsilon){
   #select a random action
    action <- sample(1:4, 1)
  } else {
  # use the greedy policy
    action <- GreedyPolicy(x, y)
  }
  return(action)
}

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

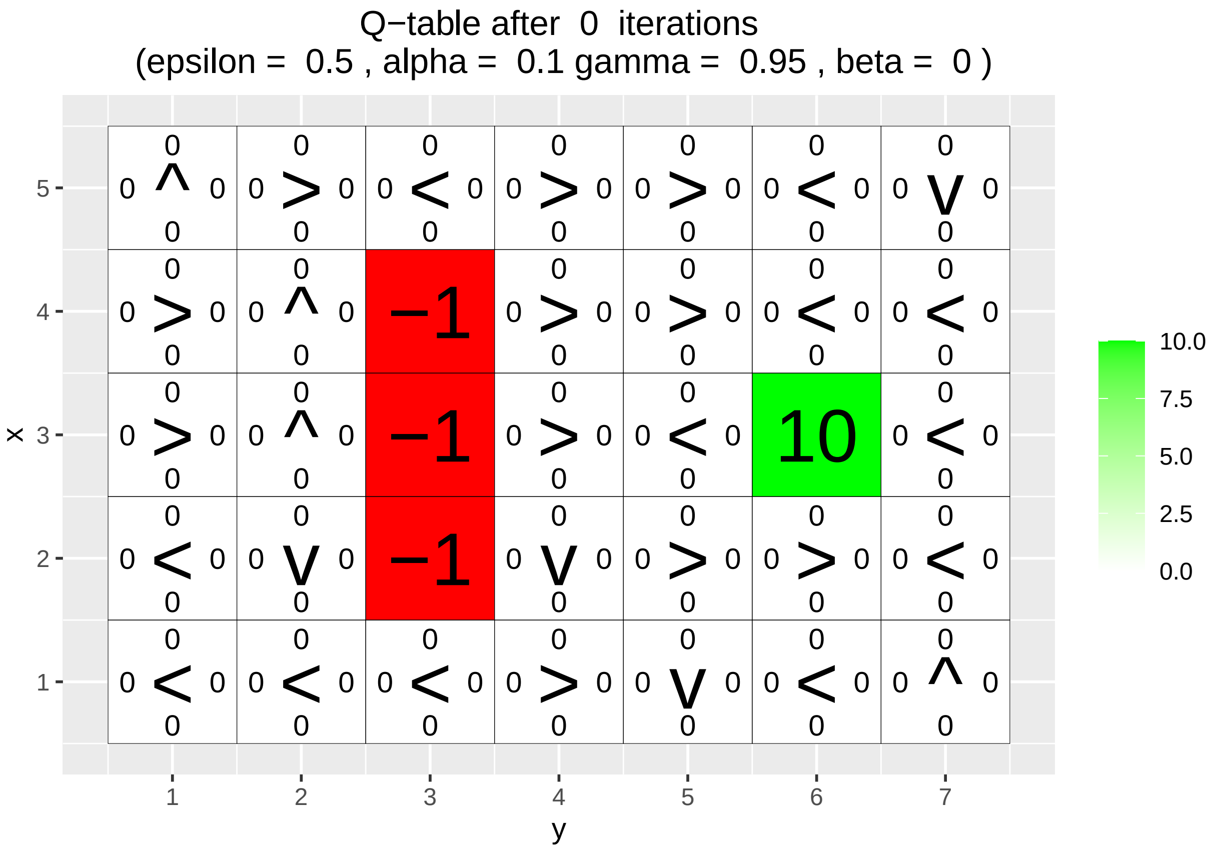
```r
#
# Args:
#   start_state: array with two entries, describing the starting position of the agent.
#   epsilon (optional): probability of acting randomly.
#   alpha (optional): learning rate.
#   gamma (optional): discount factor.
#   beta (optional): slipping factor.
#   reward_map (global variable): 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 <<-.

# Initialize state
x <- start_state[1]
y <- start_state[2]

# Initialize variables to track episode reward and TD corrections
episode_reward <- 0
episode_correction <- 0

repeat{
  # Choose an action A with epsilon-greedy policy
  action <- EpsilonGreedyPolicy(x, y, epsilon)

  # Observe next state S' & reward R after taking action A
  next_state <- transition_model(x, y, action, beta)
  x_new <- next_state[1]
  y_new <- next_state[2]
  R <- reward_map[x_new, y_new]

  # Get current Q-value Q(S, A)
  Q_SA <- q_table[x, y, action]

  # if R != 0 it means next state is terminal, this check if for when the terminal state
  # is the first action and the end of the loop hasn't been reached
  if (R != 0){
    max_QSAprime <- 0
  } else {
    # Compute max Q(S', a') over all possible actions a'
    max_QSAprime <- max(q_table[x_new, y_new, ])
  }

  # Calc TD correction term
  TD_correction <- R + gamma * max_QSAprime - Q_SA
  episode_correction <- episode_correction + TD_correction

  # Update Q(S, A)
  q_table[x, y, action] <<- Q_SA + alpha * TD_correction
```

```
    # Ep reward accumulated
    episode_reward <- episode_reward + R

    # Next state
    x <- x_new
    y <- y_new

    # Check if the episode has ended (terminal state)
    if (R != 0){
      return (c(episode_reward, episode_correction))
    }
  }
}
```

2.2 Environment A

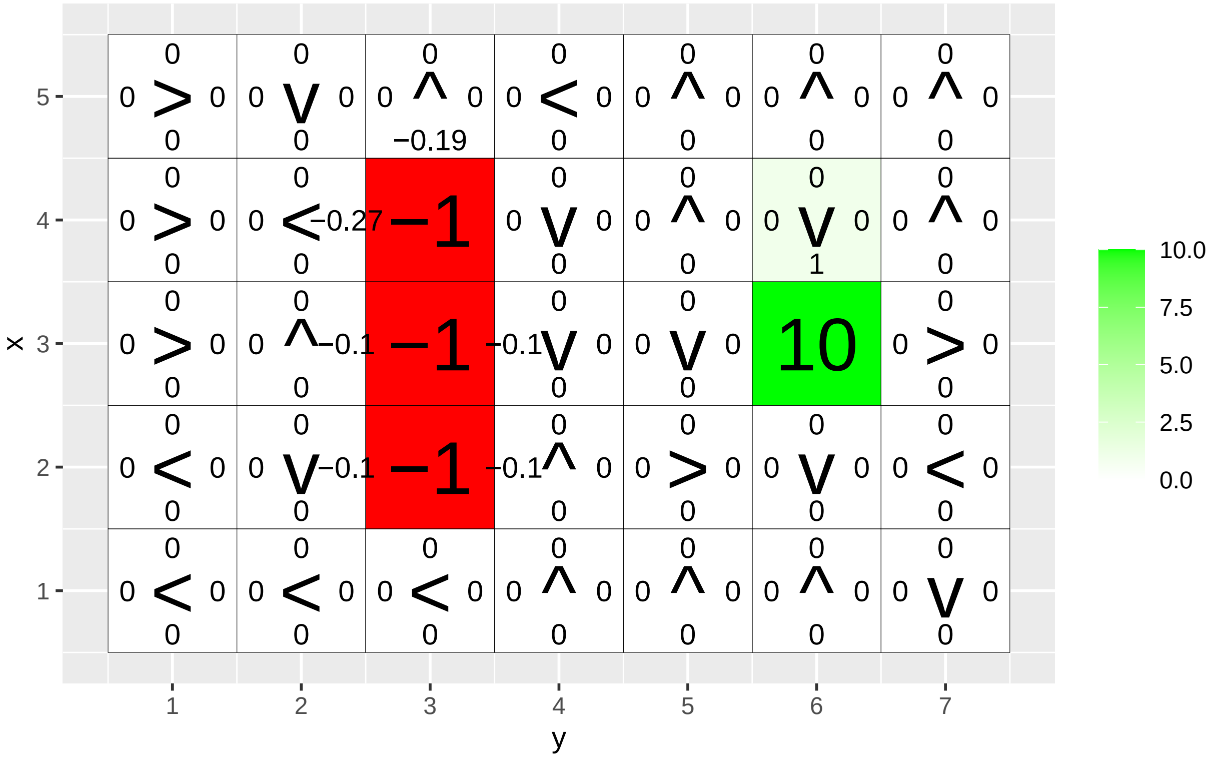
Implement and visualize Q-learning in a 5×7 grid-world environment with specific rewards, and complete tasks including implementing greedy and epsilon-greedy policies, and running 10,000 episodes of Q-learning to analyze the learned policy and Q-table.

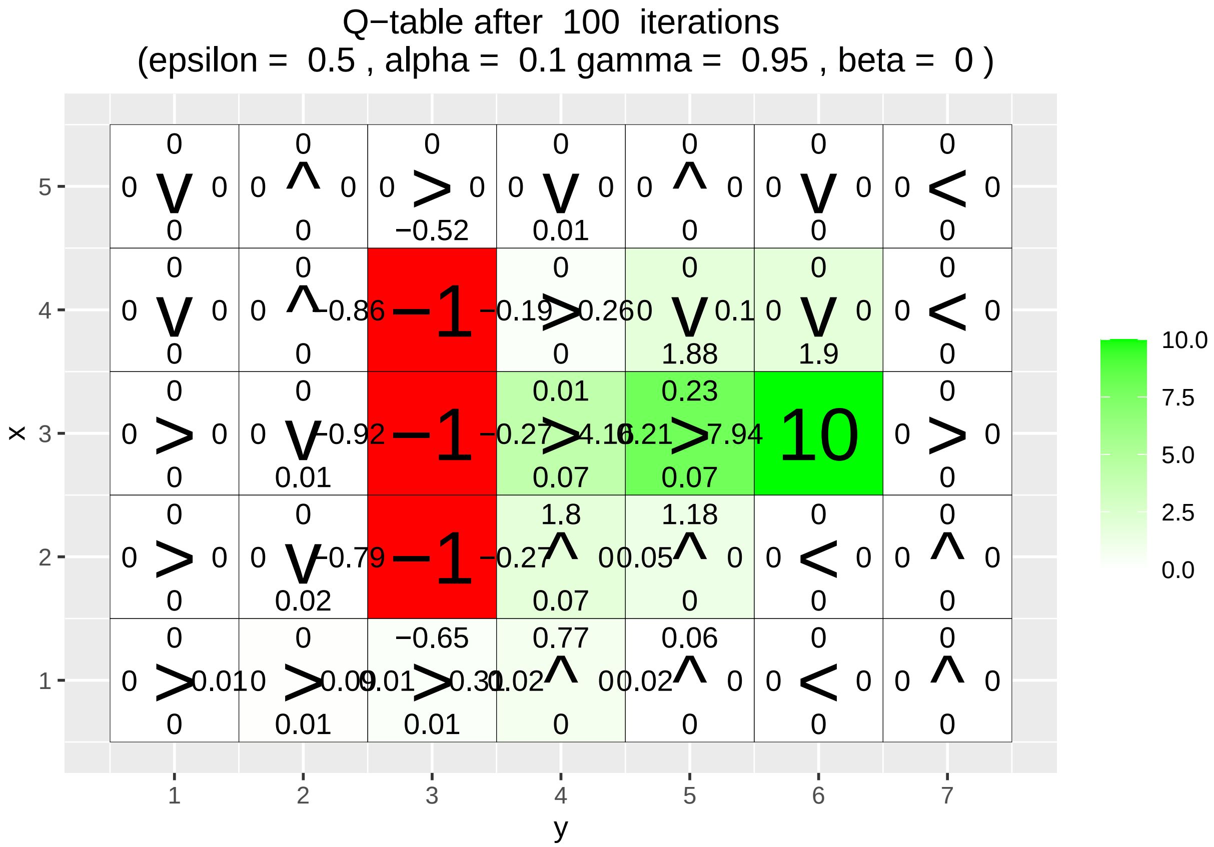
```
H <- 5
W <- 7

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()
```

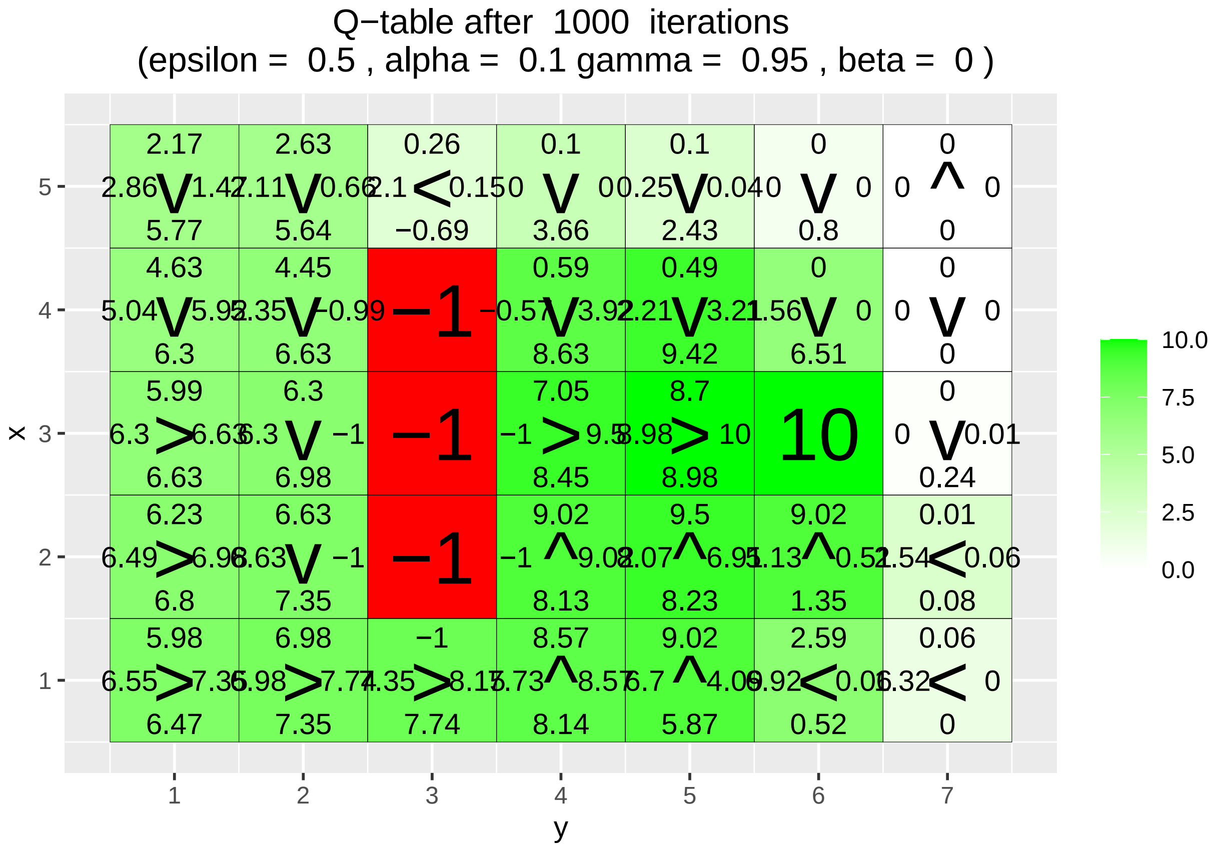


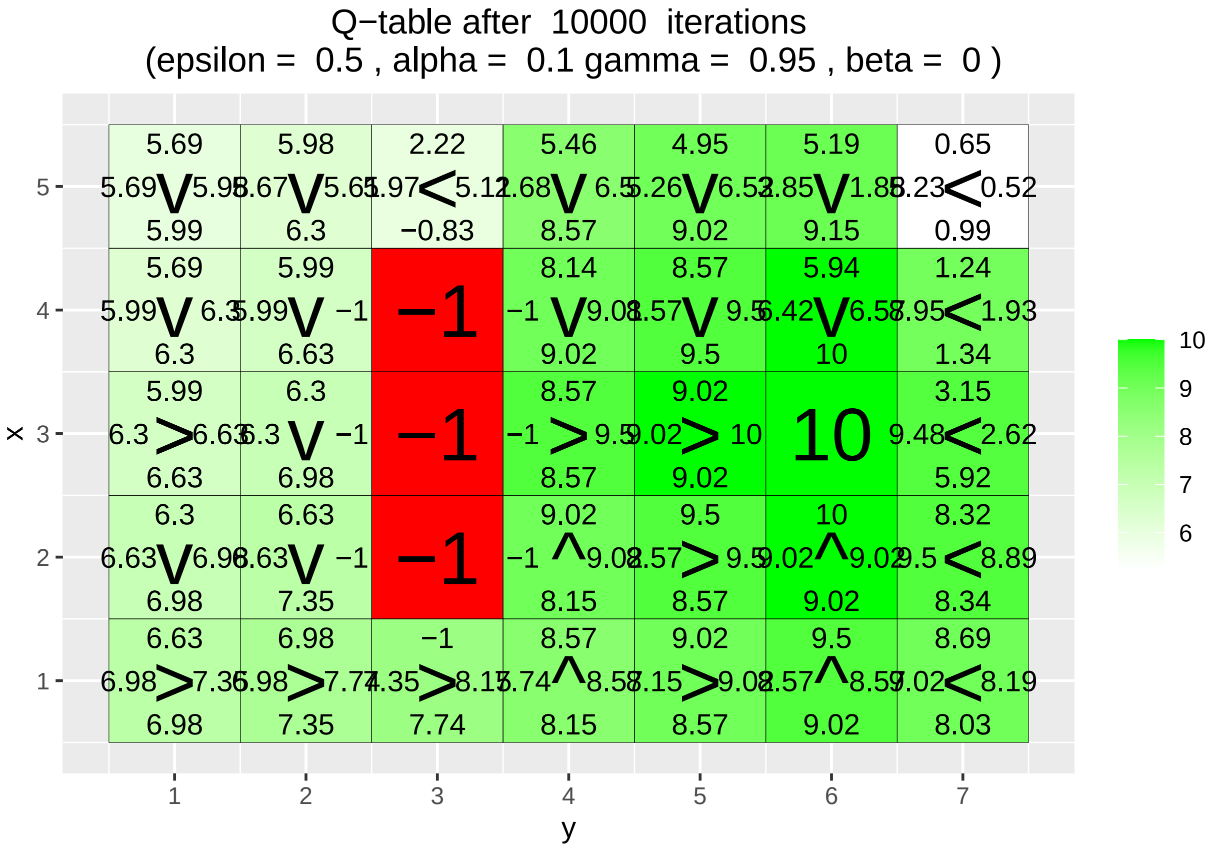
```r
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))

  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)




Q−table after 100 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
0 0 0 0 0 0 0
0 V 0 0 ^ 0 0 > 0 0 V 0 0 ^ 0 0 V 0 0 < 0
0 0 −0.52 0.01 0 0 0
0 0 0 0 0 0
0 V 0 0 ^−0.86 −1 −0.19>0.260 V 0.1 0 V 0 0 < 0
0 0 0 1.88 1.9 0
0 0 0.01 0.23 0
0 > 0 0 V−0.92 −1 −0.27>4.16.21>7.94 10 0 > 0
0 0.01 0.07 0.07 0
0 0 1.8 1.18 0 0
0 > 0 0 V−0.79 −1 −0.27^ 00.05^ 0 0 < 0 0 ^ 0
0 0.02 0.07 0 0 0
0 0 −0.65 0.77 0.06 0 0
0 >0.010 >0.09.01>0.31.02^ 00.02^ 0 0 < 0 0 ^ 0
0 0.01 0.01 0 0 0 0
5
4
3
2
1
x
1
2
3
4
5
6
7
y
10.0
7.5
5.0
2.5
0.0


Q−table after 1000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
5
4
3
2
1
x
1
2
3
4
5
6
7
y
2.17 2.86 V 1.42 5.77
2.63 2.11 V 0.66 5.64
0.26 2.1 < 0.15 −0.69
0.1 0 V 0 3.66
0.1 0.25 V 0.04 2.43
0 0 V 0 0.8
0 0 ^ 0 0
4.63 5.04 V 5.95 6.3
4.45 5.35 V −0.99 6.63
−1
0.59 −0.57 V 3.92 8.63
0.49 2.21 V 3.21 9.42
0 1.56 V 0 6.51
0 0 V 0 0
5.99 6.3 > 6.63 6.63
6.3 6.3 V −1 6.98
−1
7.05 −1 > 9.5 8.45
8.7 8.98 > 10 8.98
10
0 0 V 0.01 0.24
6.23 6.49 > 6.98 6.8
6.63 6.63 V −1 7.35
−1
9.02 −1 ^ 9.02 8.13
9.5 8.07 ^ 6.95 8.23
9.02 1.13 ^ 0.52 1.35
0.01 2.54 < 0.06 0.08
5.98 6.55 > 7.36 6.47
6.98 6.98 > 7.74 7.35
−1 7.35 > 8.15 7.74
8.57 7.73 ^ 8.57 8.14
9.02 6.7 ^ 4.09 5.87
2.59 0.92 < 0.06 0.52
0.06 6.32 < 0 0
10.0
7.5
5.0
2.5
0.0



Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)

What has the agent learned after the first 10 episodes?

The Q-table values are mostly zeros, with slight updates in some states around -1. The greedy policy (arrows on the grid) point randomly due to insufficient learning.

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

The agent has clearly learned to navigate towards the goal at position, as seen by the arrows pointing rightwards and downwards towards this cell. The policy is optimal for the most states that the agent has visited often during the episodes, however some areas are under explored leading to sub-optimal policies in those areas. An example is the states above the -1 penalty wall where it would be better to go above and not down again.

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 ?

Mostly no, the learned values in the Q-table do not reflect the fact that there are multiple path that lead to the positive reward. The agent clearly shows a strong preference for the lower path over the upper path. This is likely due to the agent's early discovery of the upper path and subsequent exploitation of it.

To counteract this one could:

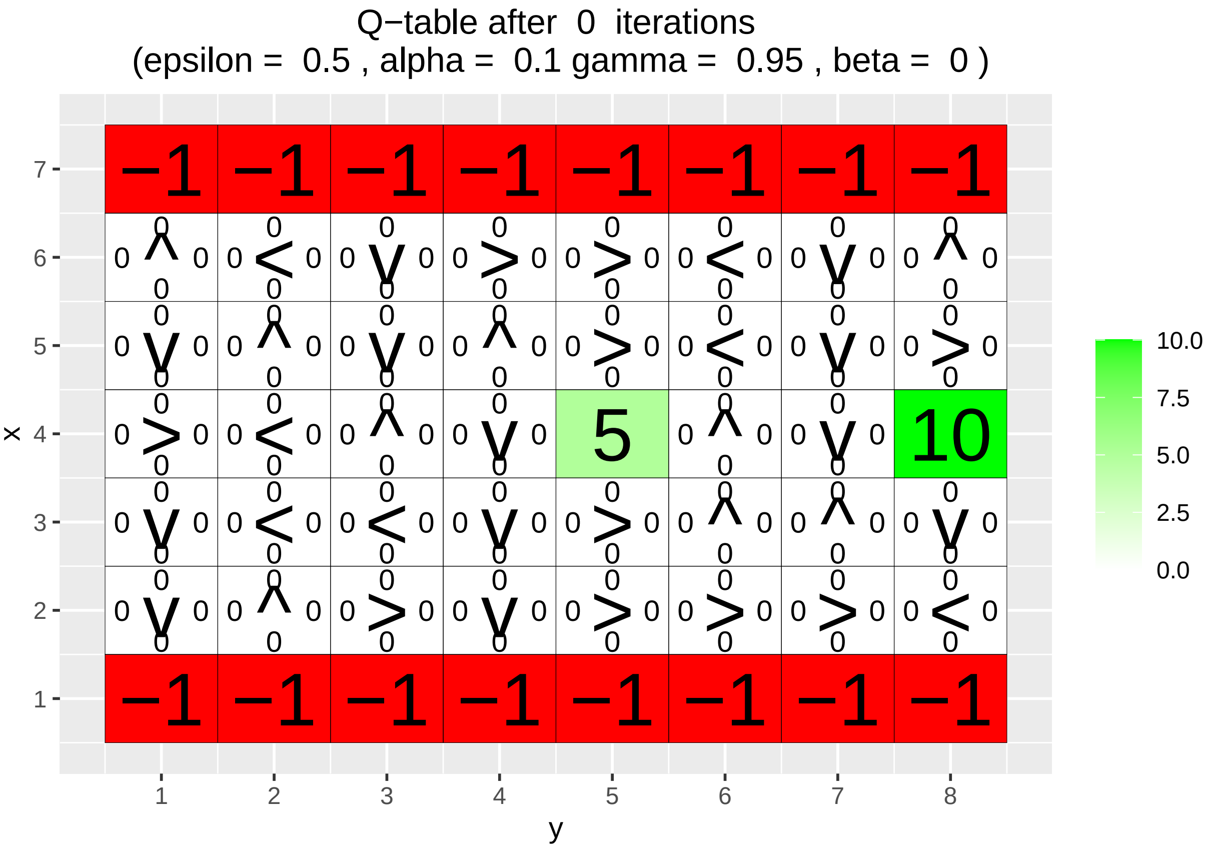
- increase exploration

- Reducing the learning rate

2.3 Environment B

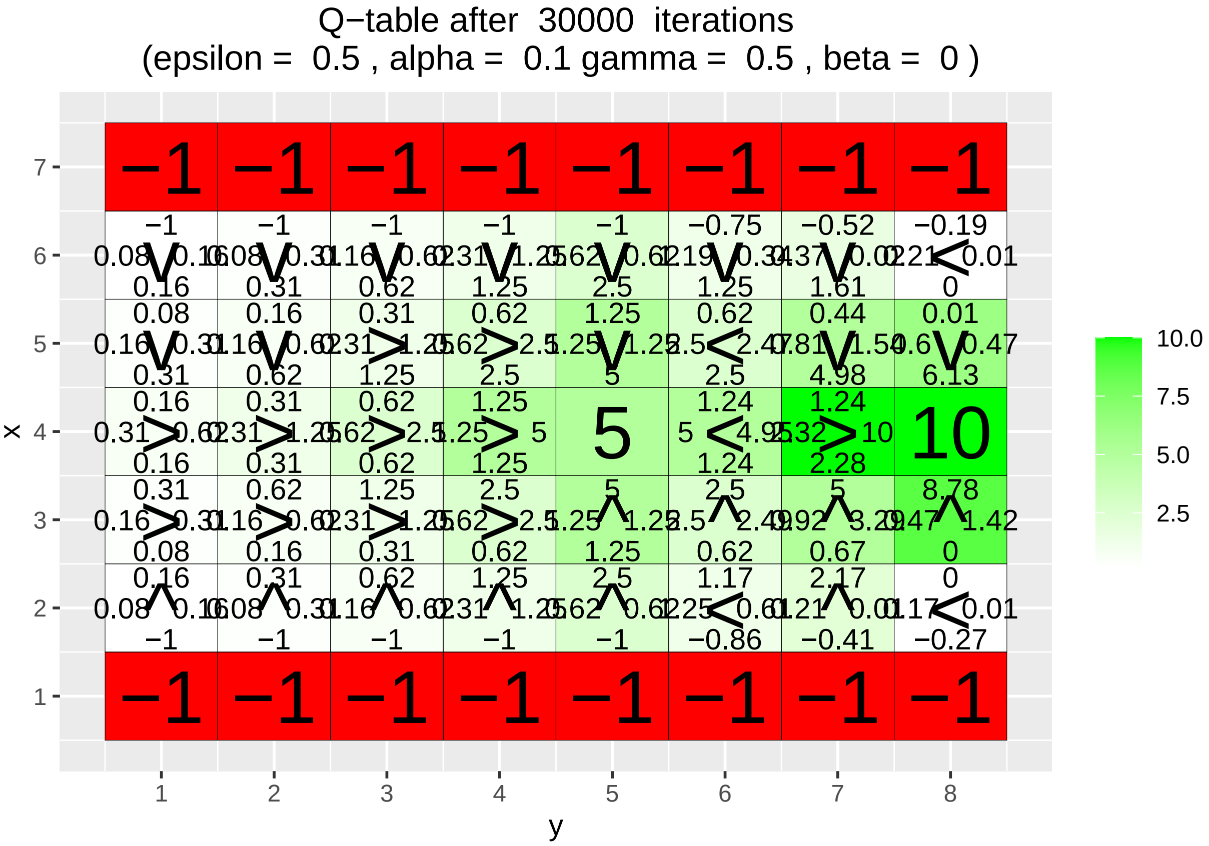
Investigate how the parameters epsilon and gamma affect the learned policy in a 7×8 environment with negative and positive rewards by running 30,000 episodes of Q-learning with different parameter settings.

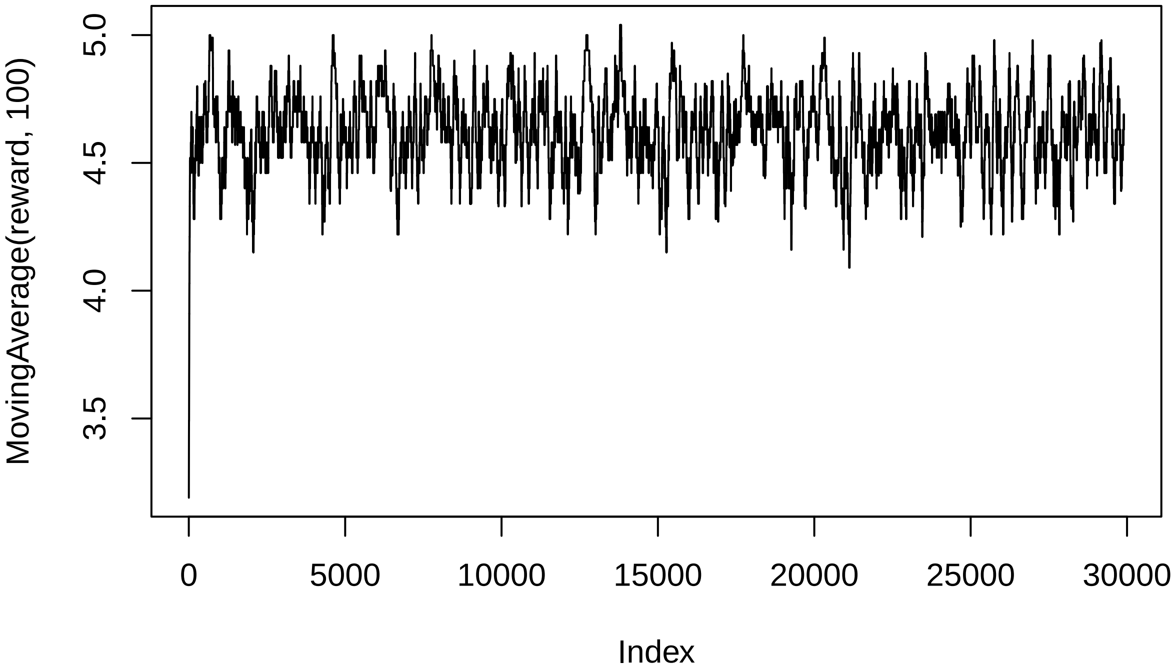
```
H <- 7
W <- 8

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()
```

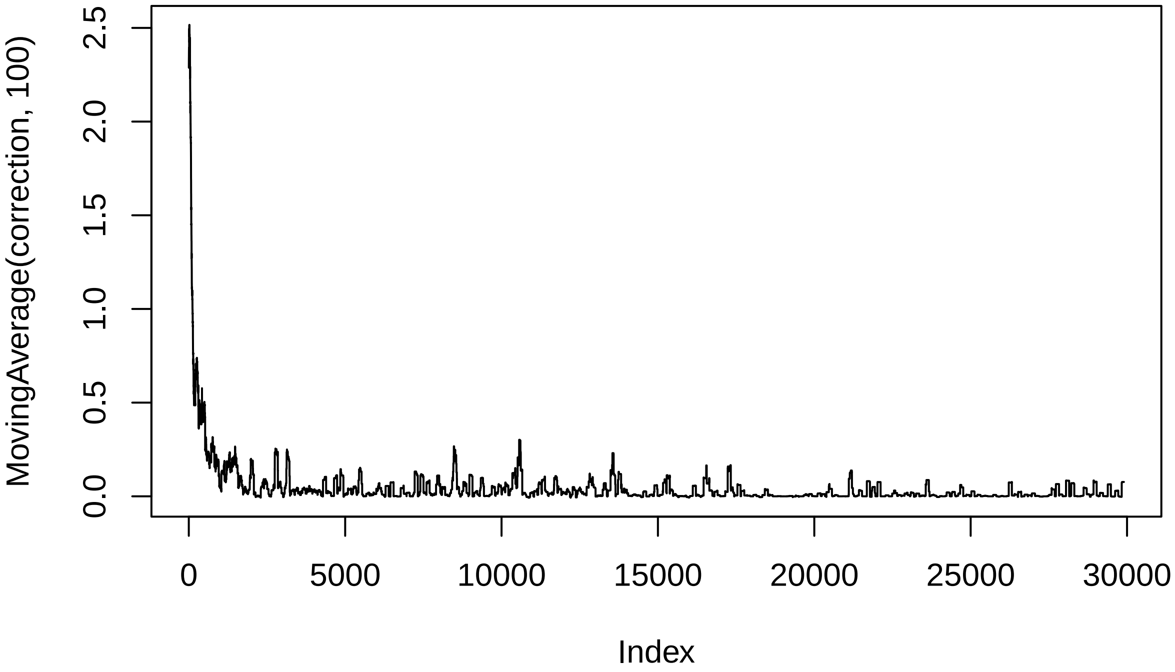


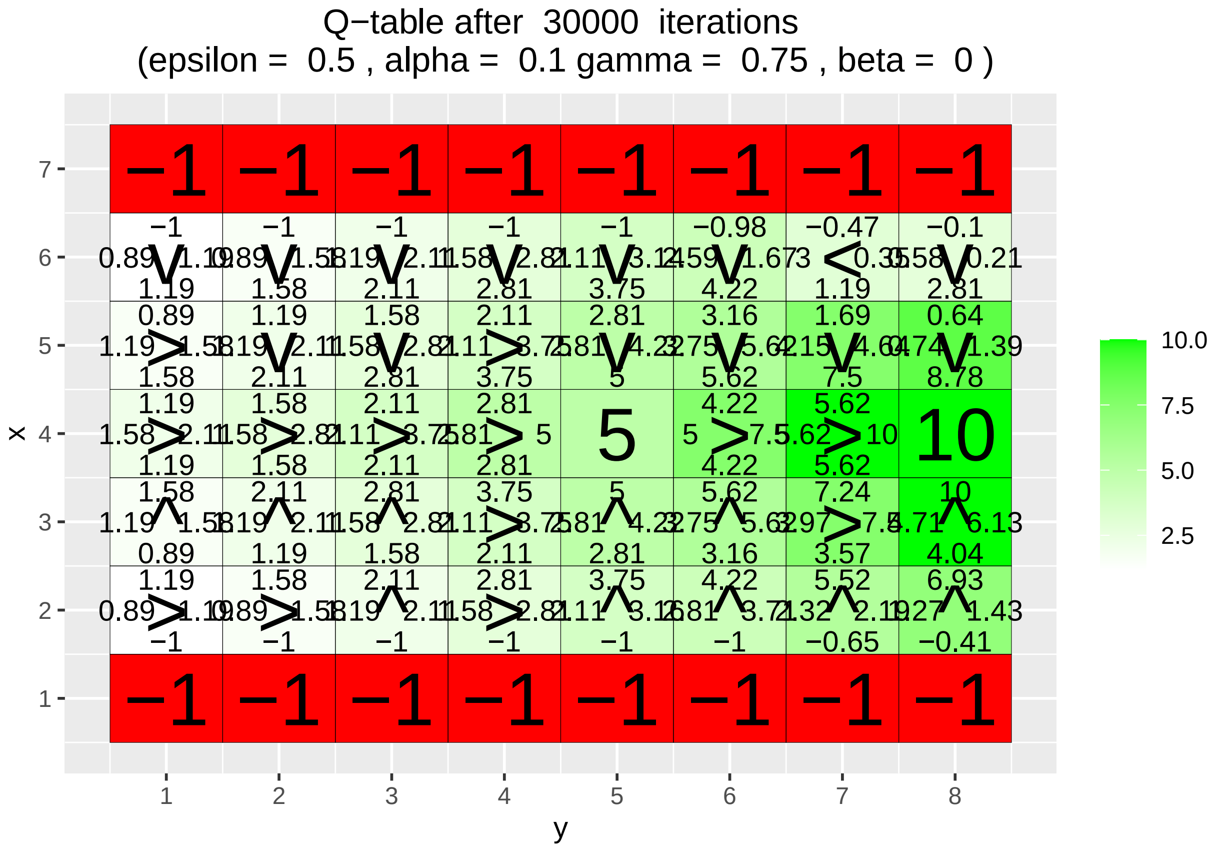
```
MovingAverage <- function(x, n){

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

  return (rsum)
}
```

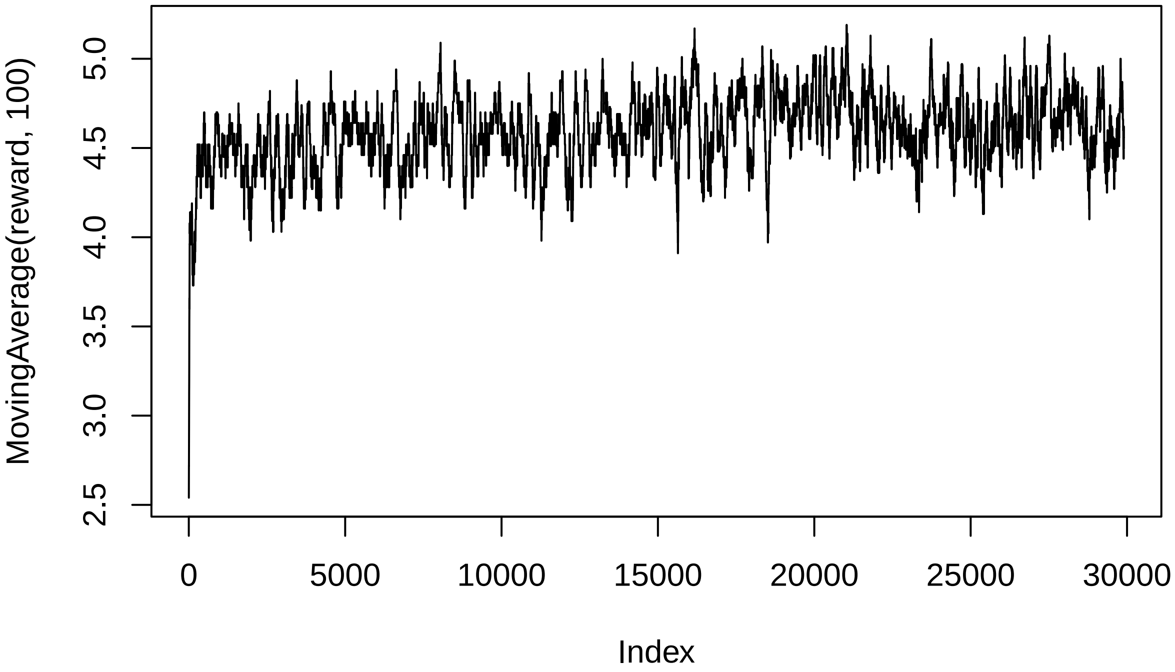
```r
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  for(i in 1:30000){
    # This is for epsilon = 0.5 standard
    foo <- q_learning(gamma = j, start_state = c(4,1))
    reward <- c(reward,foo[1])
    correction <- c(correction,foo[2])
  }

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

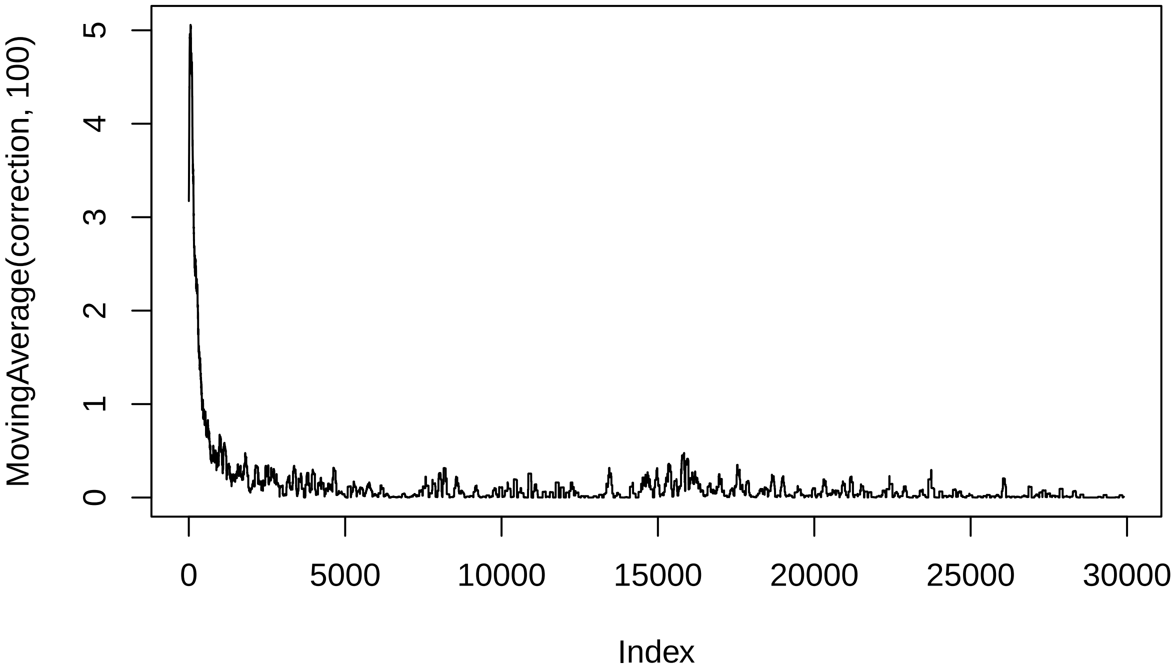


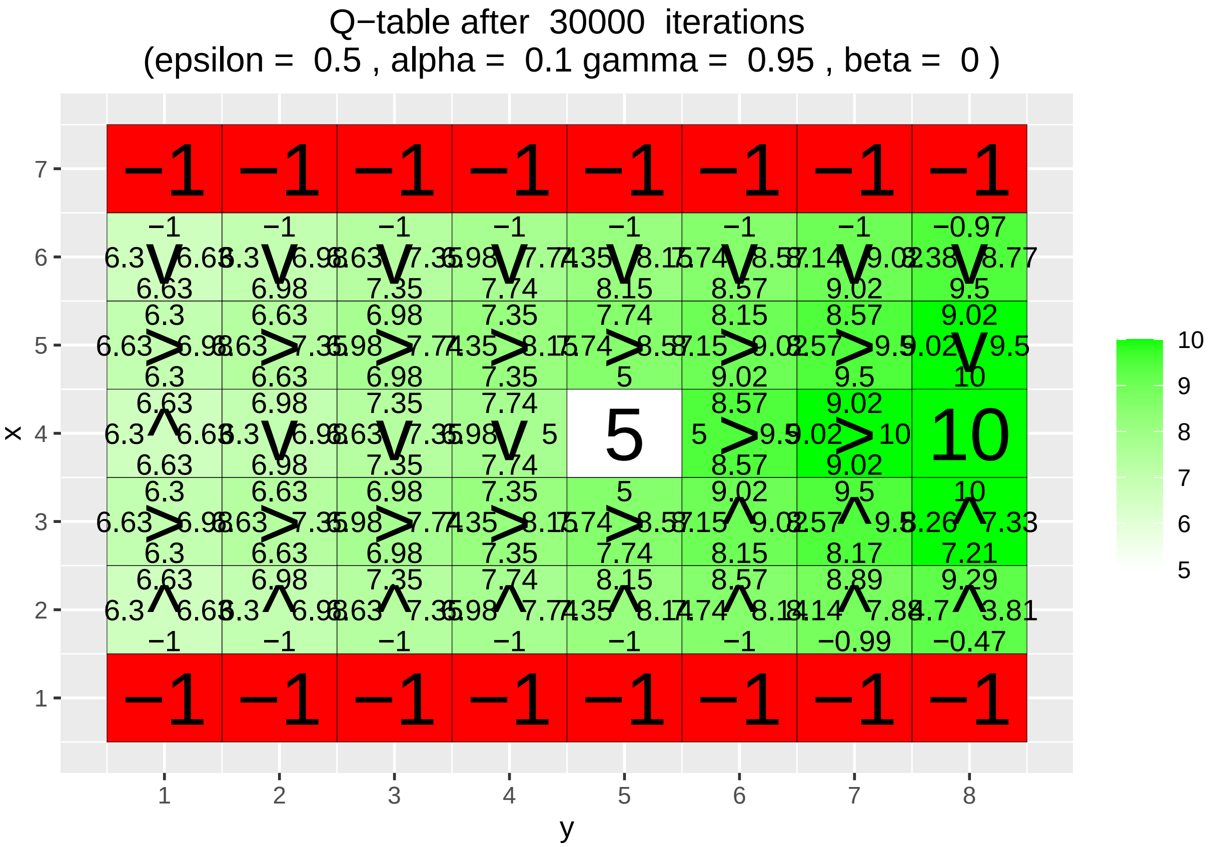

MovingAverage(reward, 100)
5.0
4.5
4.0
3.5
0
5000
10000
15000
20000
25000
30000
Index

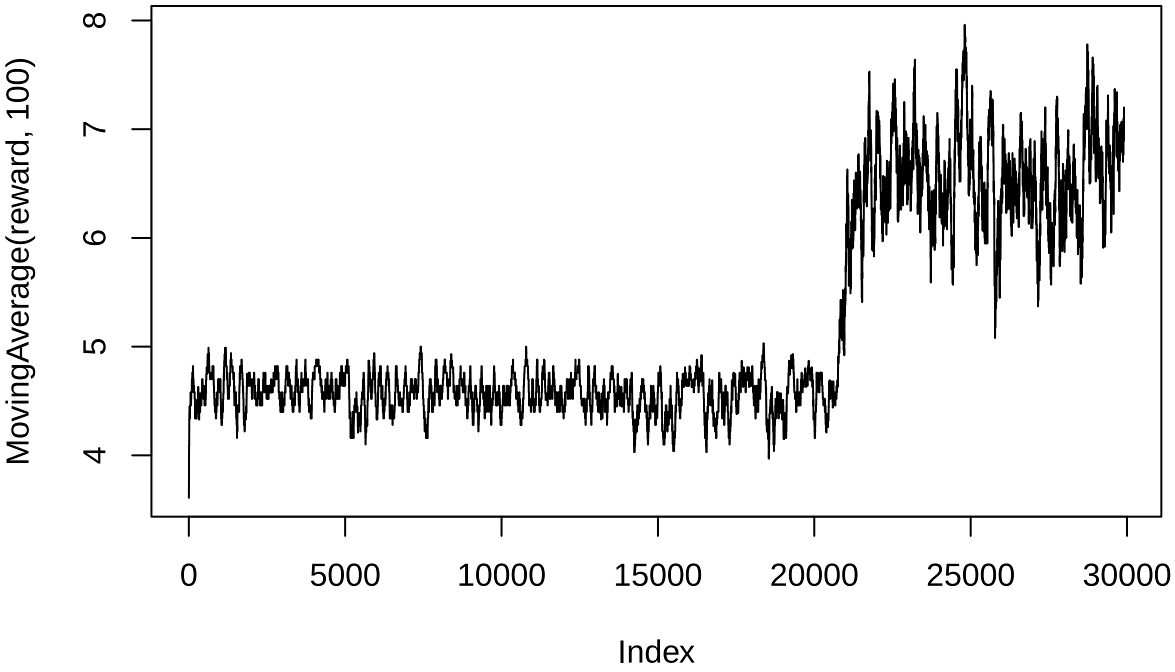

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
2.5
0
5000
10000
15000
20000
25000
30000
Index

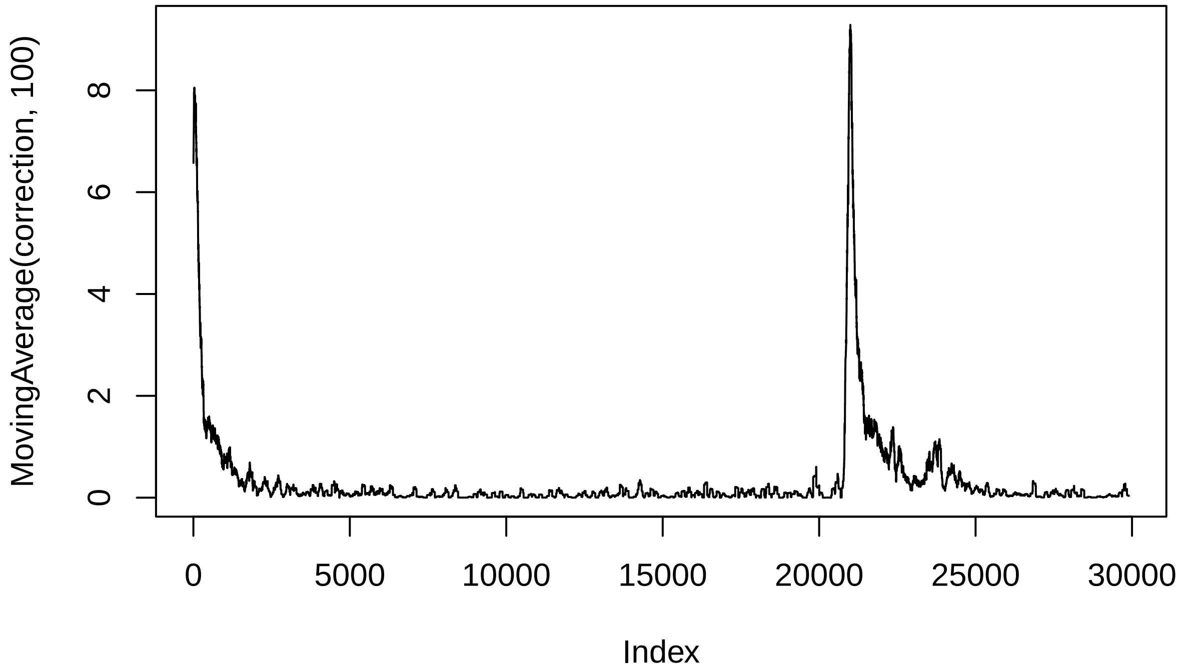

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
7
−1 −1 −1 −1 −1 −1 −1 −1
−1 −1 −1 −1 −1 −0.98 −0.47 −0.1
6
0.89 V 1.10.89 V 1.51.19 V 2.11.58 V 2.82.11 V 3.14.59 V 1.673 < 0.30.58 V 0.21
1.19 1.58 2.11 2.81 3.75 4.22 1.19 2.81
0.89 1.19 1.58 2.11 2.81 3.16 1.69 0.64
5
1.19 > 1.51.19 V 2.11.58 V 2.82.11 > 3.72.81 V 4.23.75 V 5.64.15 V 4.60.74 V 1.39
1.58 2.11 2.81 3.75 5 5.62 7.5 8.78
1.19 1.58 2.11 2.81 4.22 5.62
4
x
1.58 > 2.11.58 > 2.82.11 > 3.72.81 > 5 5 5 > 7.5.62 > 10 10
1.19 1.58 2.11 2.81 4.22 5.62
1.58 2.11 2.81 3.75 5 5.62 7.24 10
3
1.19 ^ 1.51.19 ^ 2.11.58 ^ 2.82.11 > 3.72.81 ^ 4.23.75 ^ 5.63.97 > 7.54.71 ^ 6.13
0.89 1.19 1.58 2.11 2.81 3.16 3.57 4.04
1.19 1.58 2.11 2.81 3.75 4.22 5.52 6.93
2
0.89 > 1.10.89 > 1.51.19 ^ 2.11.58 > 2.82.11 ^ 3.12.81 ^ 3.72.32 ^ 2.19.27 ^ 1.43
−1 −1 −1 −1 −1 −1 −0.65 −0.41
1
−1 −1 −1 −1 −1 −1 −1 −1
1 2 3 4 5 6 7 8
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

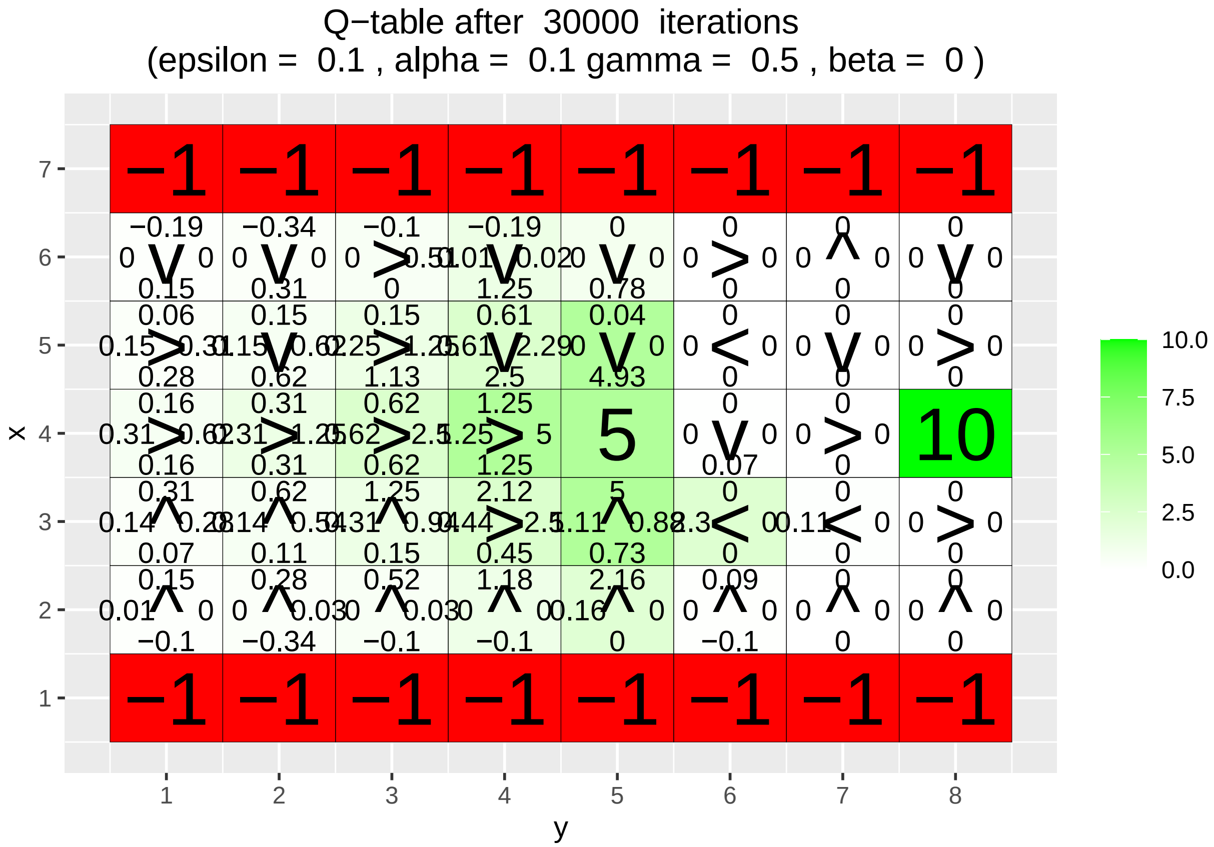

MovingAverage(correction, 100)
0
1
2
3
4
5
0
5000
10000
15000
20000
25000
30000
Index

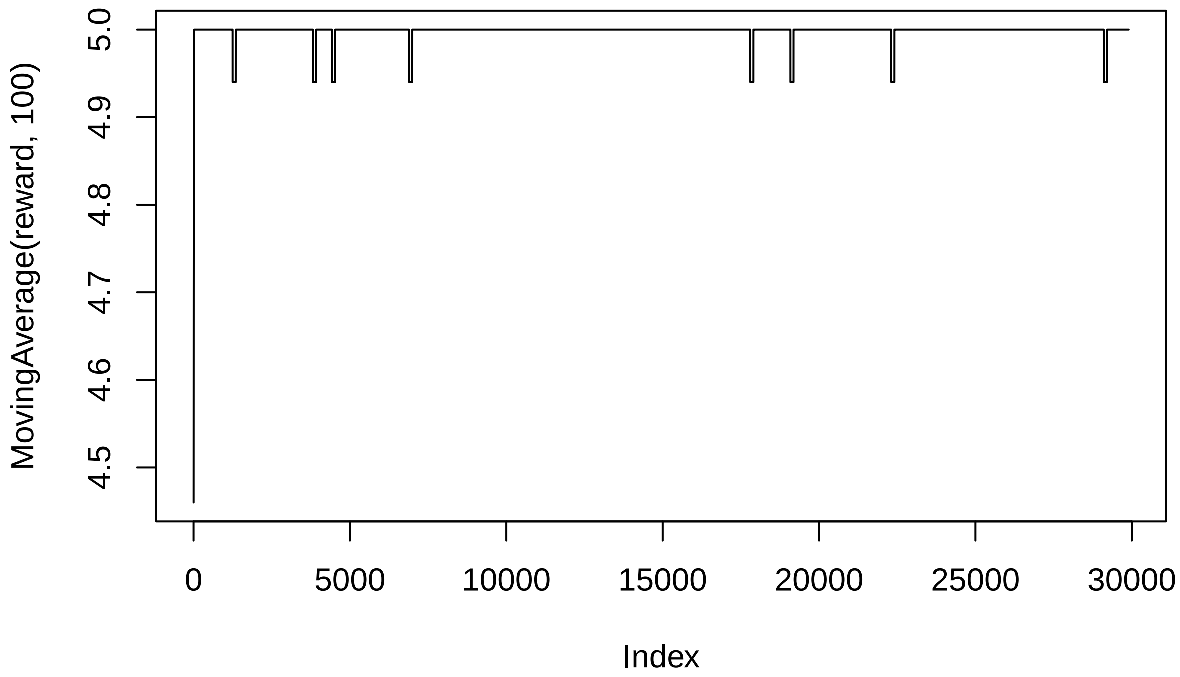

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
1 2 3 4 5 6 7 8
7 6 5 4 3 2 1
−1 −1 −1 −1 −1 −1 −1 −1
−1 6.3 V 6.63 6.63 | −1 6.3 V 6.98 6.98 | −1 6.63 V 7.35 7.35 | −1 6.98 V 7.74 7.74 | −1 7.35 V 8.15 8.15 | −1 7.74 V 8.57 8.57 | −1 8.14 V 9.02 9.02 | −0.97 8.38 V 8.77 9.5
6.3 6.63 > 6.98 6.3 | 6.63 6.63 > 7.35 6.63 | 6.98 6.98 > 7.74 6.98 | 7.35 7.35 > 8.15 7.35 | 7.74 7.74 > 8.57 5 | 8.15 8.15 > 9.02 9.02 | 8.57 8.57 > 9.5 9.5 | 9.02 9.02 V 9.5 10
6.63 6.3 ^ 6.63 6.63 | 6.98 6.3 V 6.98 6.98 | 7.35 6.63 V 7.35 7.35 | 7.74 6.98 V 5 7.74 | 5 | 8.57 5 > 9.5 8.57 | 9.02 9.02 > 10 9.02 | 10
6.3 6.63 > 6.98 6.3 | 6.63 6.63 > 7.35 6.63 | 6.98 6.98 > 7.74 6.98 | 7.35 7.35 > 8.15 7.35 | 5 7.74 > 8.57 7.74 | 9.02 8.15 ^ 9.02 8.15 | 9.5 8.57 ^ 9.5 8.17 | 10 8.26 ^ 7.33 7.21
6.63 6.3 ^ 6.63 −1 | 6.98 6.3 ^ 6.98 −1 | 7.35 6.63 ^ 7.35 −1 | 7.74 6.98 ^ 7.74 −1 | 8.15 7.35 ^ 8.17 −1 | 8.57 7.74 ^ 8.18 −1 | 8.89 8.14 ^ 7.84 −0.99 | 9.29 4.7 ^ 3.81 −0.47
−1 −1 −1 −1 −1 −1 −1 −1
10 9 8 7 6 5

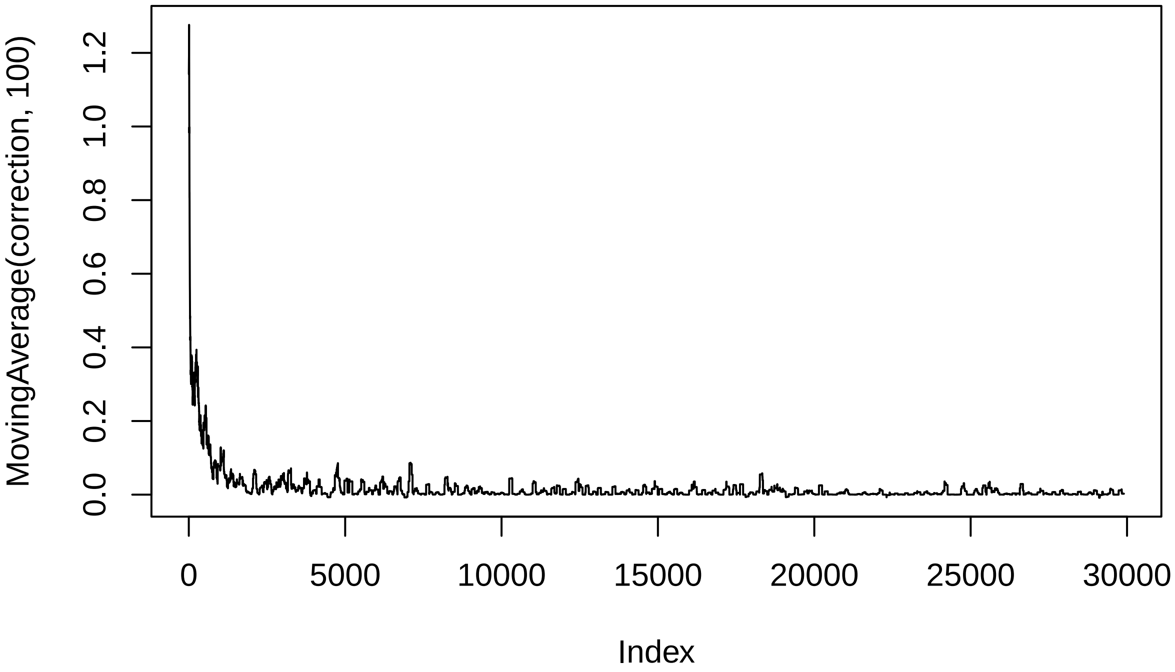

MovingAverage(reward, 100)
4
5
6
7
8
0
5000
10000
15000
20000
25000
30000
Index



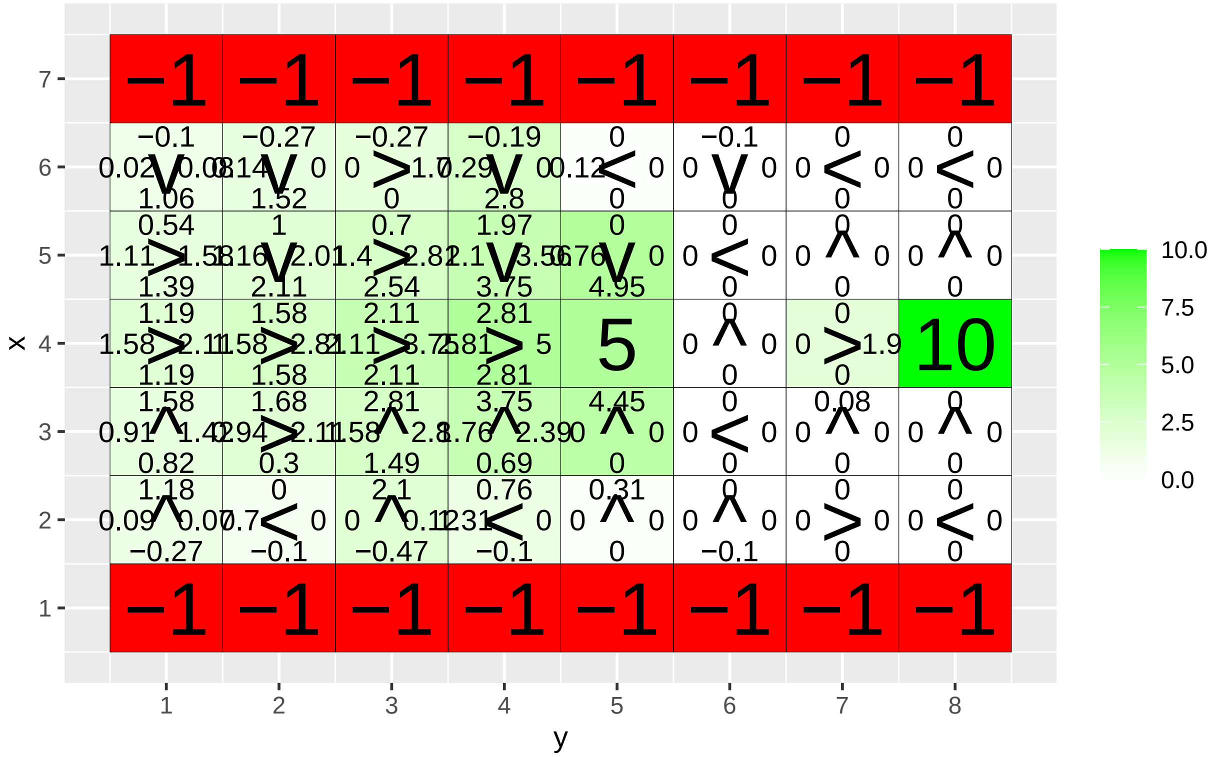
```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  for(i in 1:30000){
    # This is epsilon 0.1
    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")
}
```

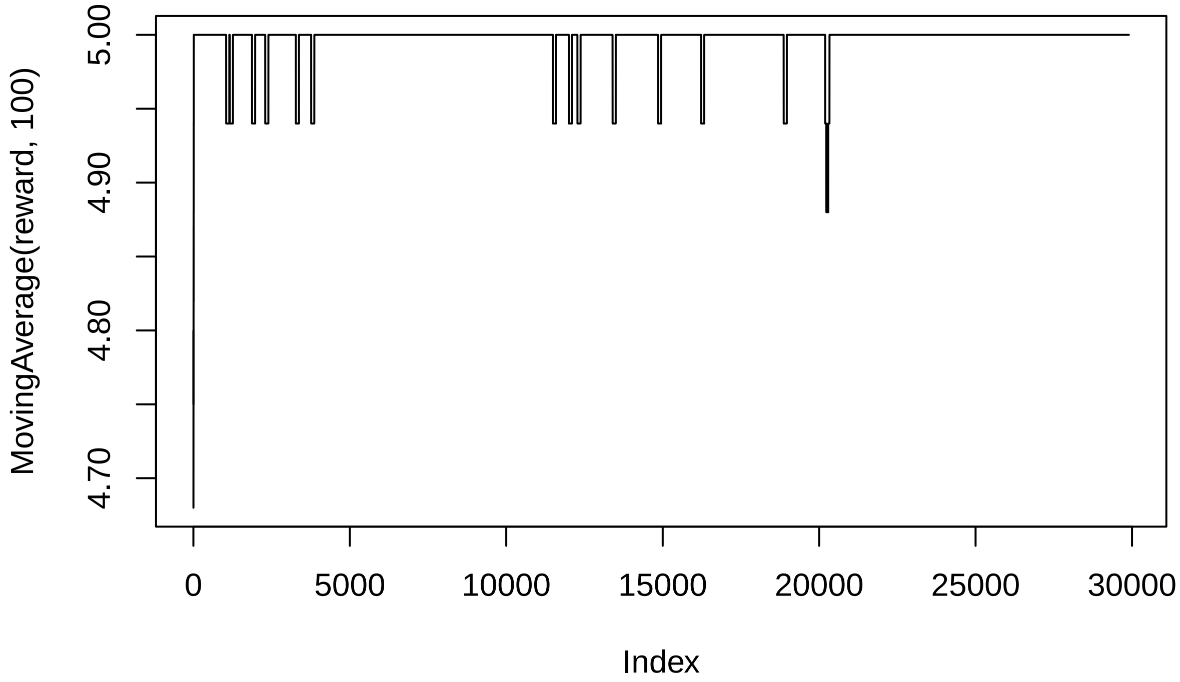

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
−1 −1 −1 −1 −1 −1 −1 −1
−0.19 −0.34 −0.1 −0.19 0 0 0 0
0 V 0 0 V 0 0 > 0.5 0.01 V 0.02 0 V 0 0 > 0 0 ^ 0 0 V 0
0.15 0.31 0 1.25 0.78 0 0 0
0.06 0.15 0.15 0.61 0.04 0 0 0
0.15 > 0.31 0.15 V 0.62 0.25 > 1.25 0.61 V 2.29 0 V 0 0 < 0 0 V 0 0 > 0
0.28 0.62 1.13 2.5 4.93 0 0 0
0.16 0.31 0.62 1.25 0 0
0.31 > 0.62 0.31 > 1.25 0.62 > 2.5 1.25 > 5 5 0 V 0 0 > 0 10
0.16 0.31 0.62 1.25 0.07 0
0.31 0.62 1.25 2.12 5 0 0 0
0.14 ^ 0.28 0.14 ^ 0.54 0.31 ^ 0.94 0.44 > 2.5 1.11 ^ 0.88 2.3 < 0 0.11 < 0 0 > 0
0.07 0.11 0.15 0.45 0.73 0 0 0
0.15 0.28 0.52 1.18 2.16 0.09 0 0
0.01 ^ 0 0 ^ 0.03 0 ^ 0.03 0 ^ 0 0.16 ^ 0 0 ^ 0 0 ^ 0 0 ^ 0
−0.1 −0.34 −0.1 −0.1 0 −0.1 0 0
−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
0.0

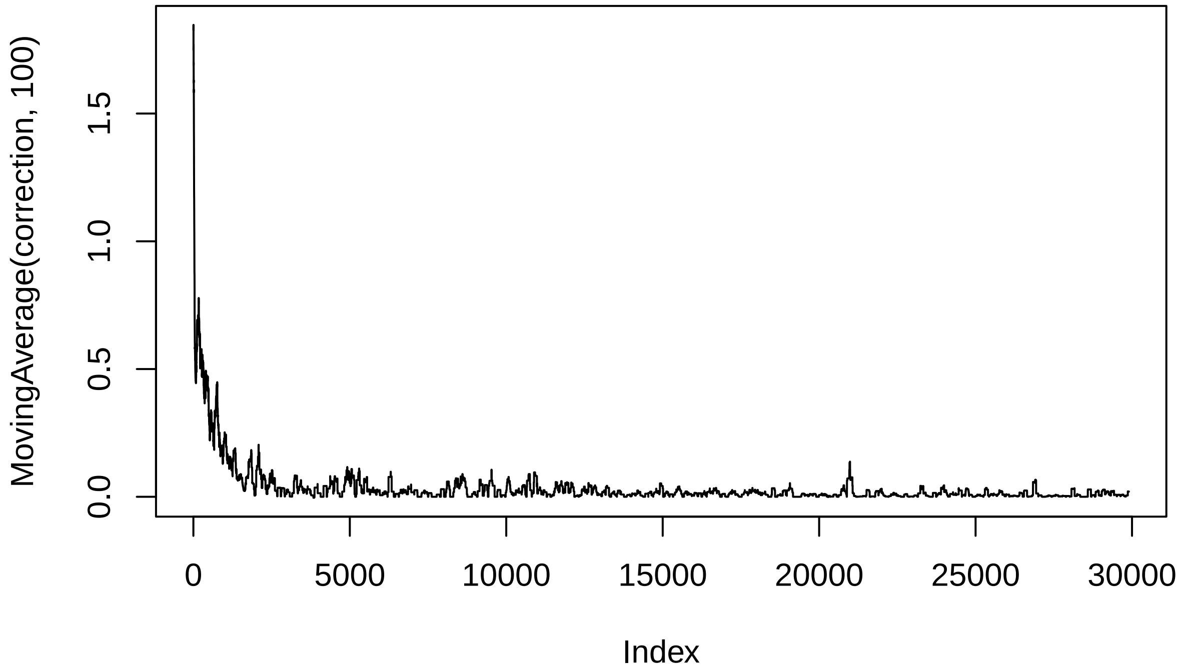

MovingAverage(reward, 100)
4.5
4.6
4.7
4.8
4.9
5.0
0
5000
10000
15000
20000
25000
30000
Index

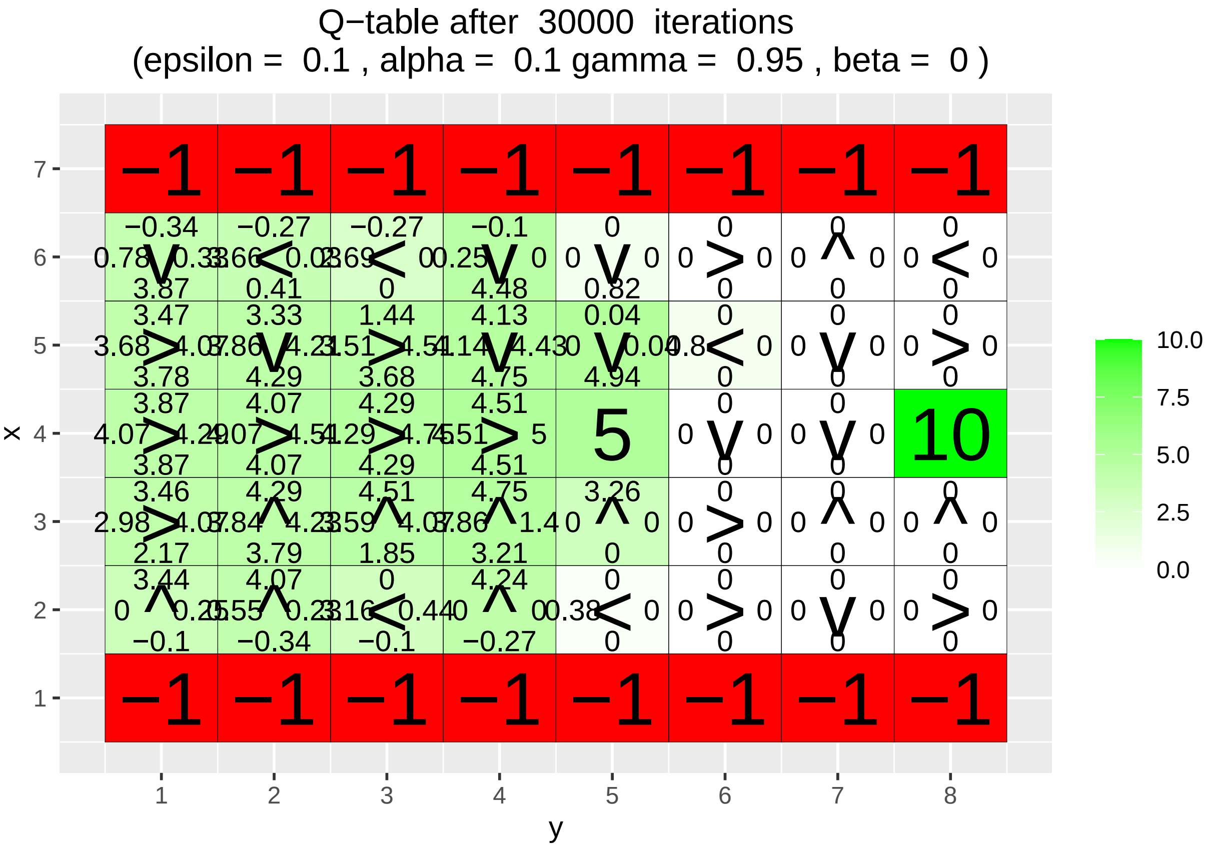

MovingAverage(correction, 100)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
5000
10000
15000
20000
25000
30000
Index

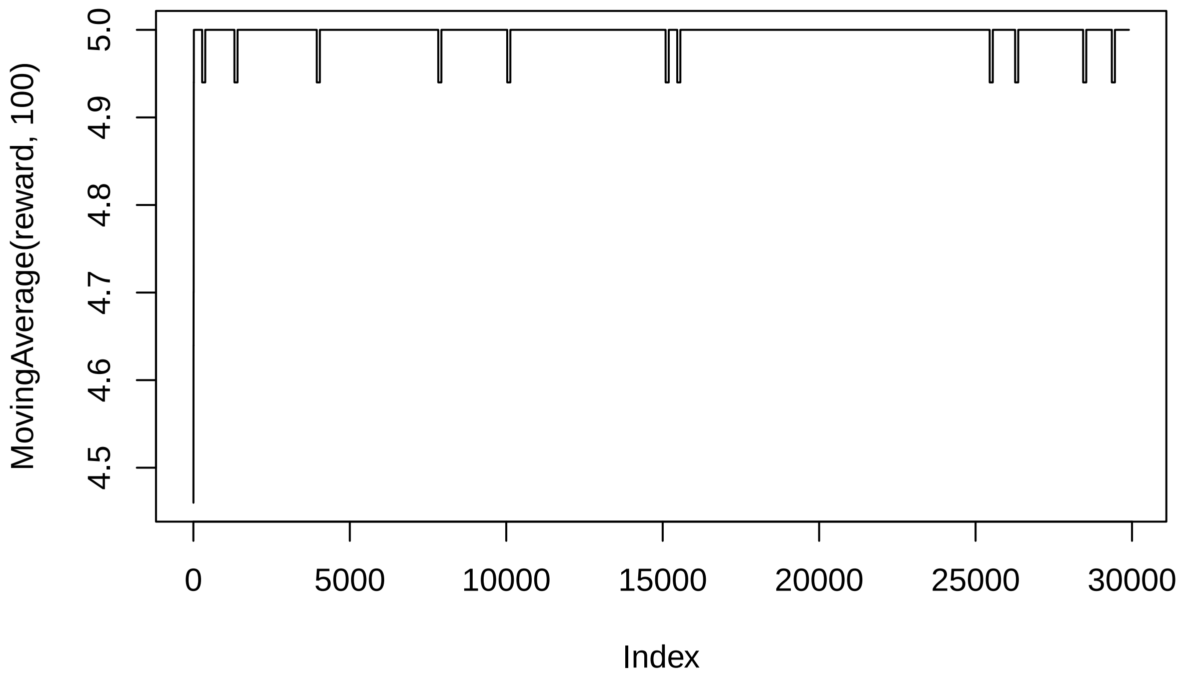
Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)

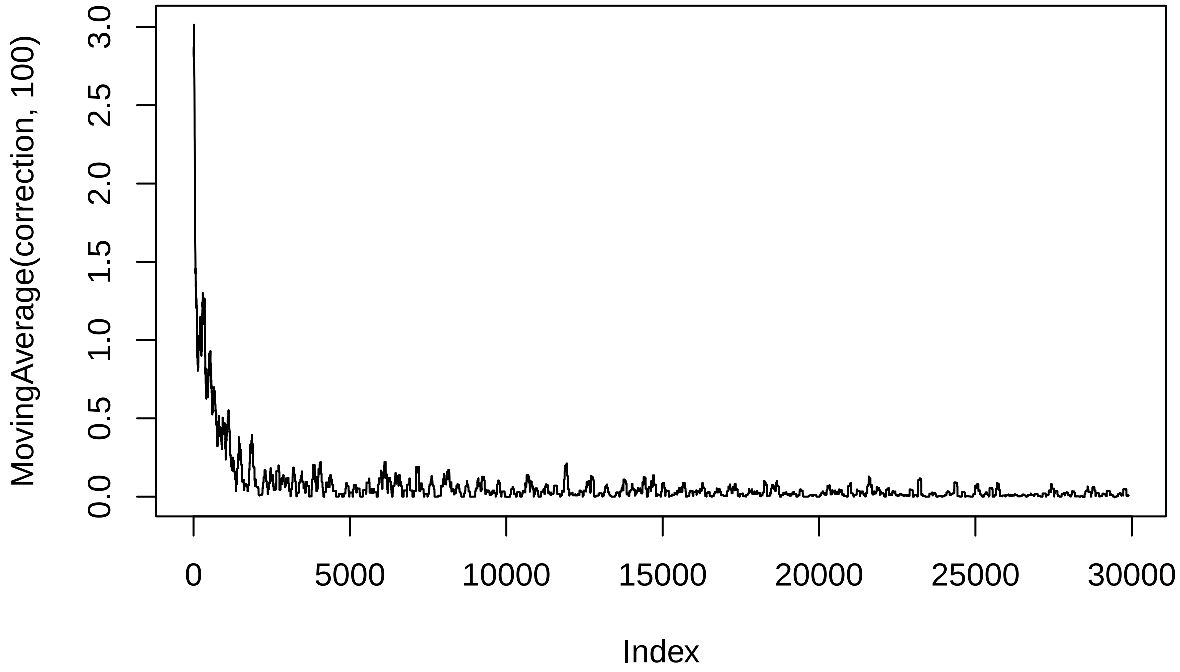



MovingAverage(reward, 100)
5.00
4.90
4.80
4.70
0
5000
10000
15000
20000
25000
30000
Index


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 = 0.1 , alpha = 0.1 gamma = 0.95 , 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
−0.34
−0.27
−0.1
3.87
0.41
4.48
0.82
3.47
3.33
1.44
4.13
0.04
3.68
3.78
4.29
3.68
4.75
4.94
3.87
4.07
4.29
4.51
3.46
2.98
2.17
3.79
1.85
3.21
3.26
1.4
3.44
4.24
−0.27
0.38
0.25
0.8


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



Your task is to investigate how the epsilon and gamma parameters affect the learned policy by running

My observations are:

epsilon = 0.5

gamma = 0.5: It chooses 5 quite often since the discount factor gamma is low meaning that it prioritizes future rewards low

gamma = 0.75: Most often chooses 10 but sometimes 5

gamma = 0.95: results in that it learns to avoid 5 reward and go for 10 reward

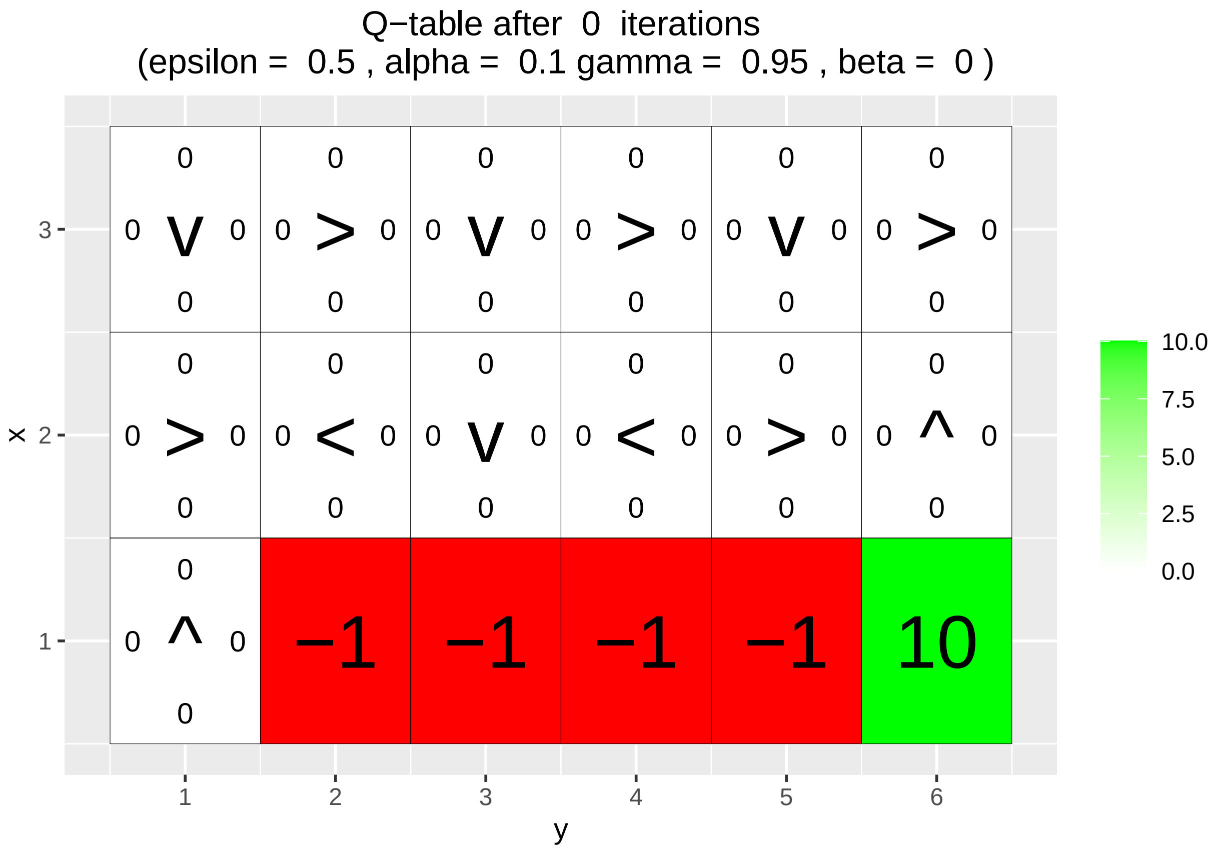
With epsilon = 0.1 the agent doesn't explore as much and subsequently doesn't find the 10 reward. It only finds 5 reward.

2.4 Environment C

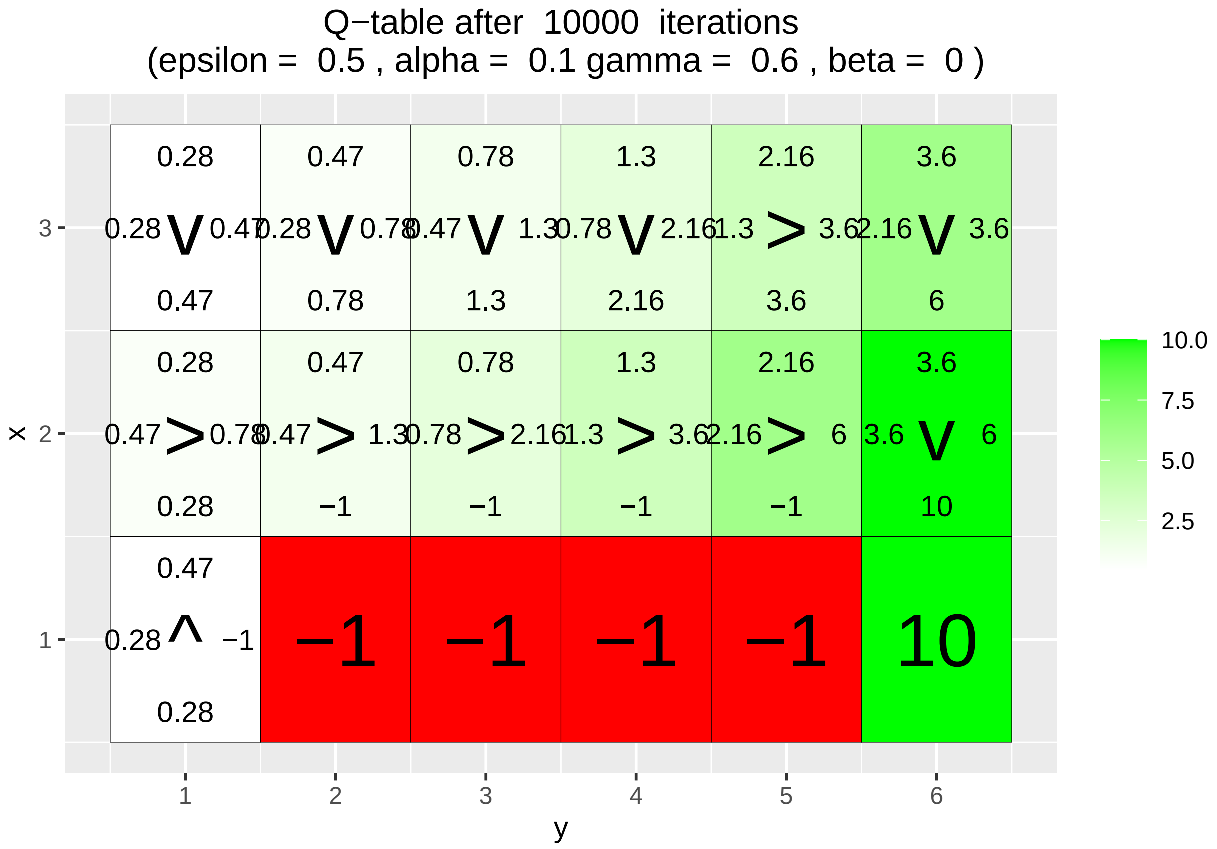
Explore how the beta parameter affects the learned policy in a 3×6 environment by running 10,000 episodes of Q-learning with different beta values and analyzing the outcomes.

```
H <- 3
W <- 6

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

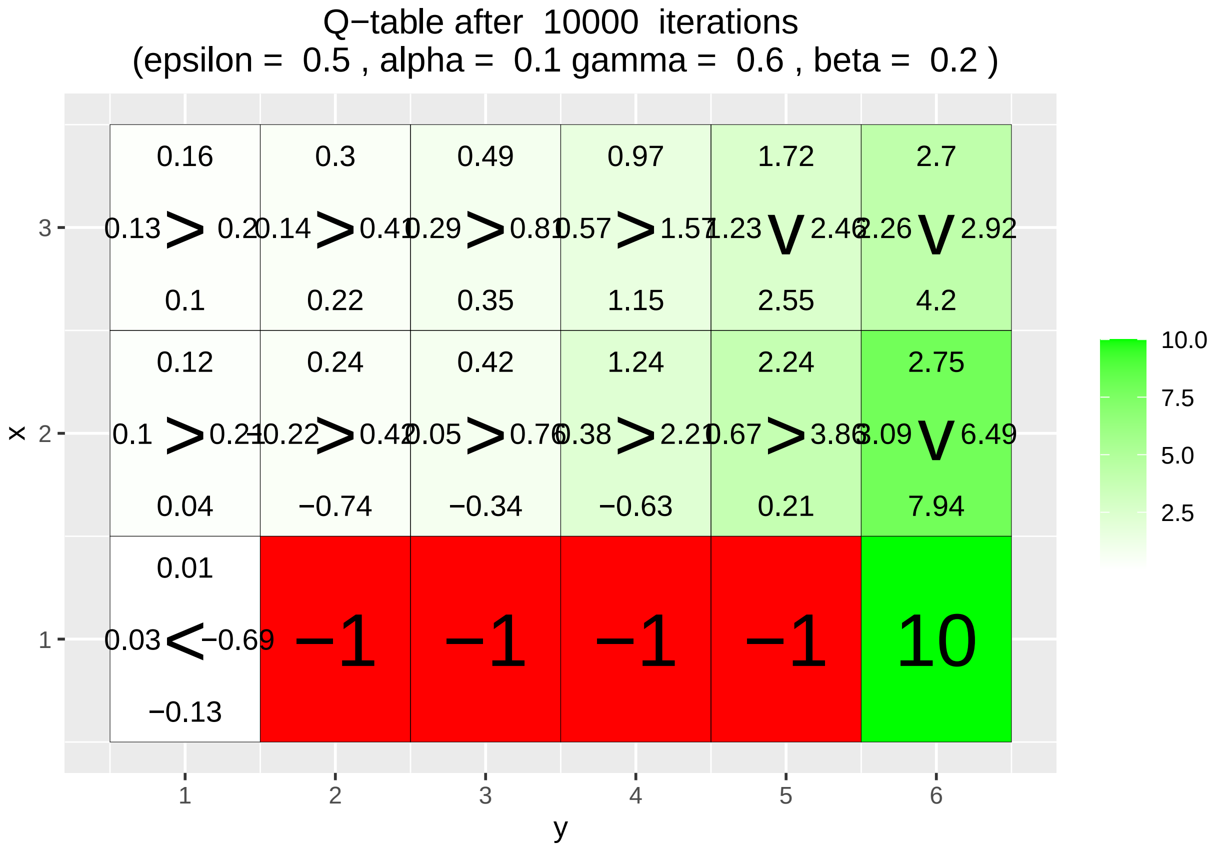
```r
q_table <- array(0,dim = c(H,W,4))

vis_environment()
```



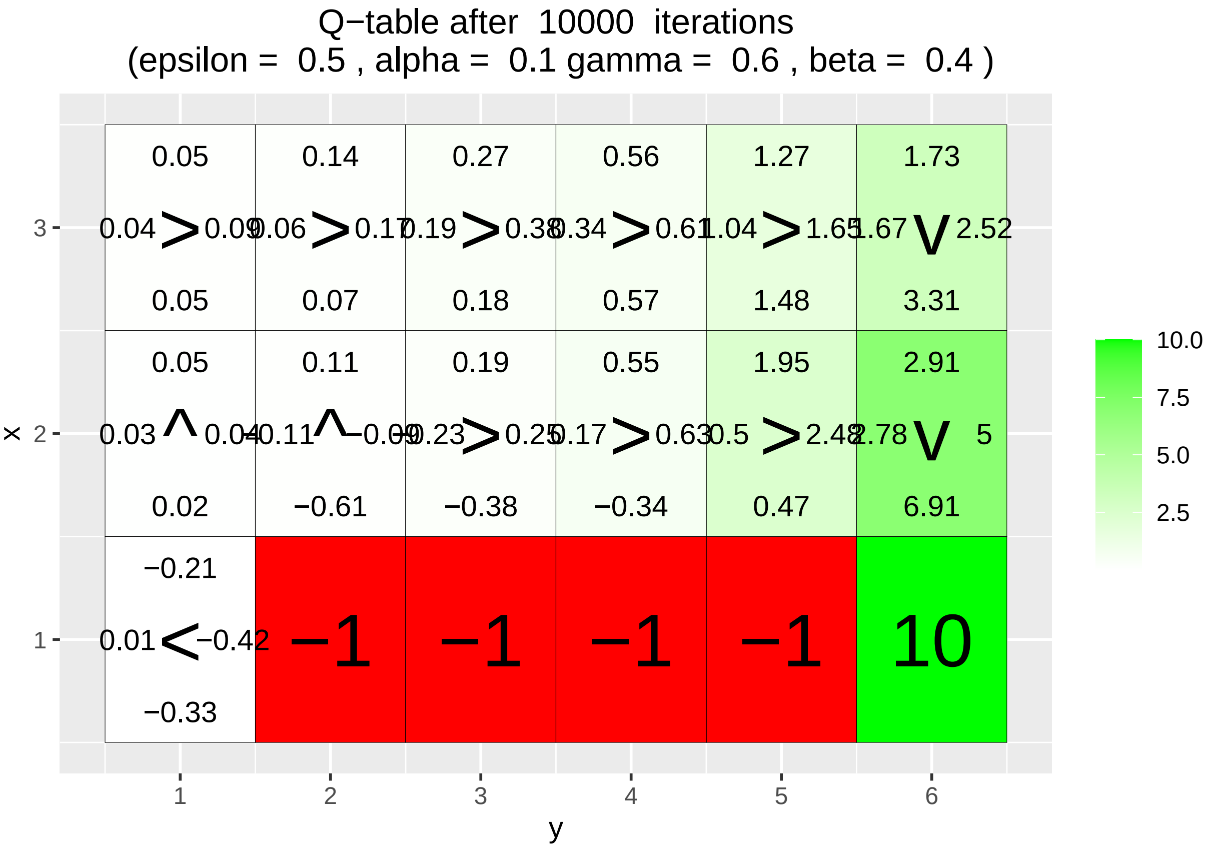
```r
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

  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)
}
```

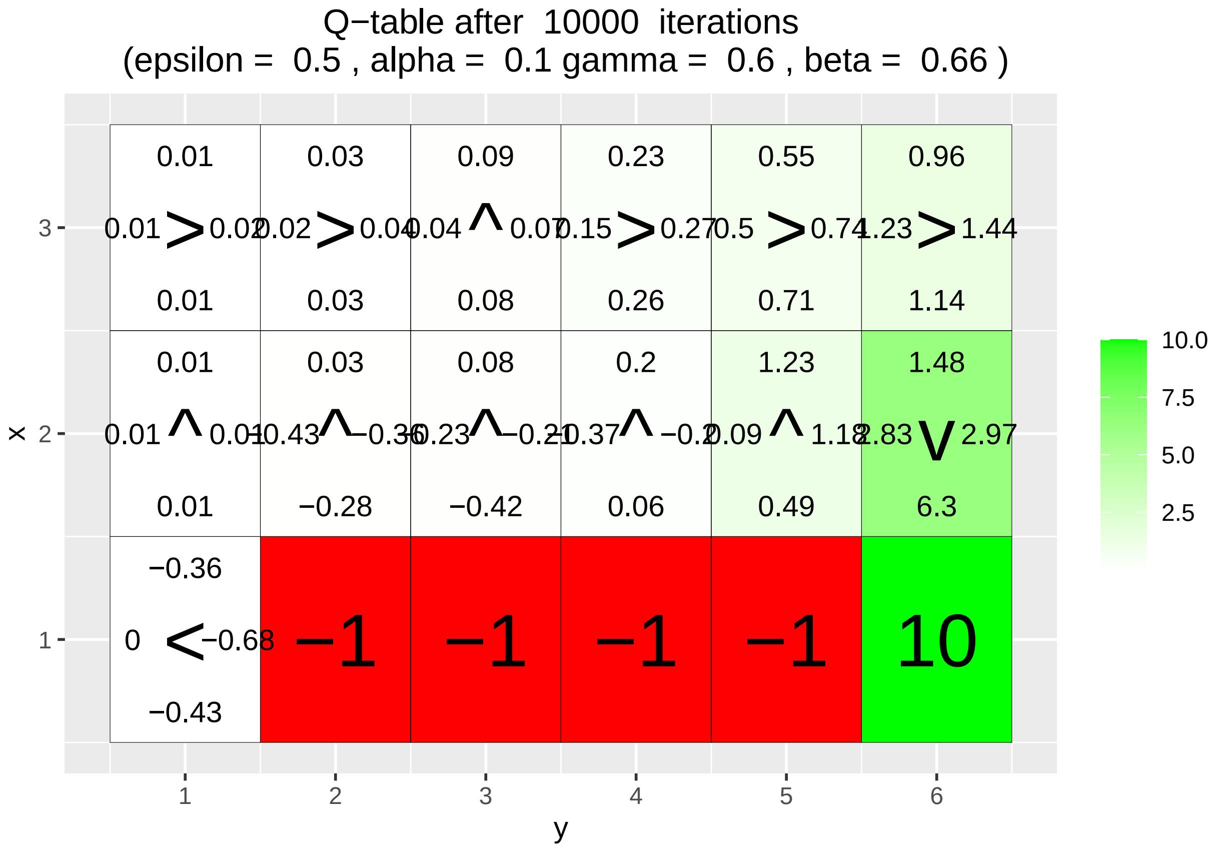


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0)



Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.4)
0.05
0.14
0.27
0.56
1.27
1.73
0.04 > 0.09
0.06 > 0.17
0.19 > 0.38
0.34 > 0.61
1.04 > 1.65
1.67 V 2.52
0.05
0.07
0.18
0.57
1.48
3.31
0.05
0.11
0.19
0.55
1.95
2.91
0.03 ^ 0.04
−0.11 ^ −0.09
−0.23 > 0.25
0.17 > 0.63
0.5 > 2.48
2.78 V 5
0.02
−0.61
−0.38
−0.34
0.47
6.91
−0.21
0.01 < −0.42
−1
−1
−1
−1
10
−0.33
x
3
2
1
y
1
2
3
4
5
6
10.0
7.5
5.0
2.5

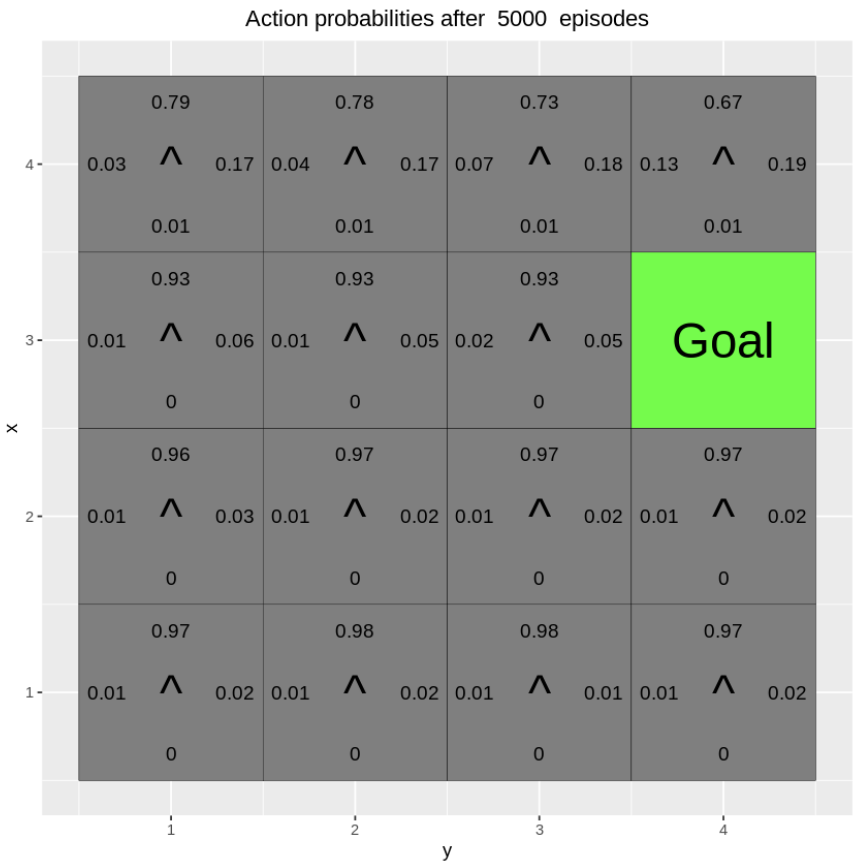


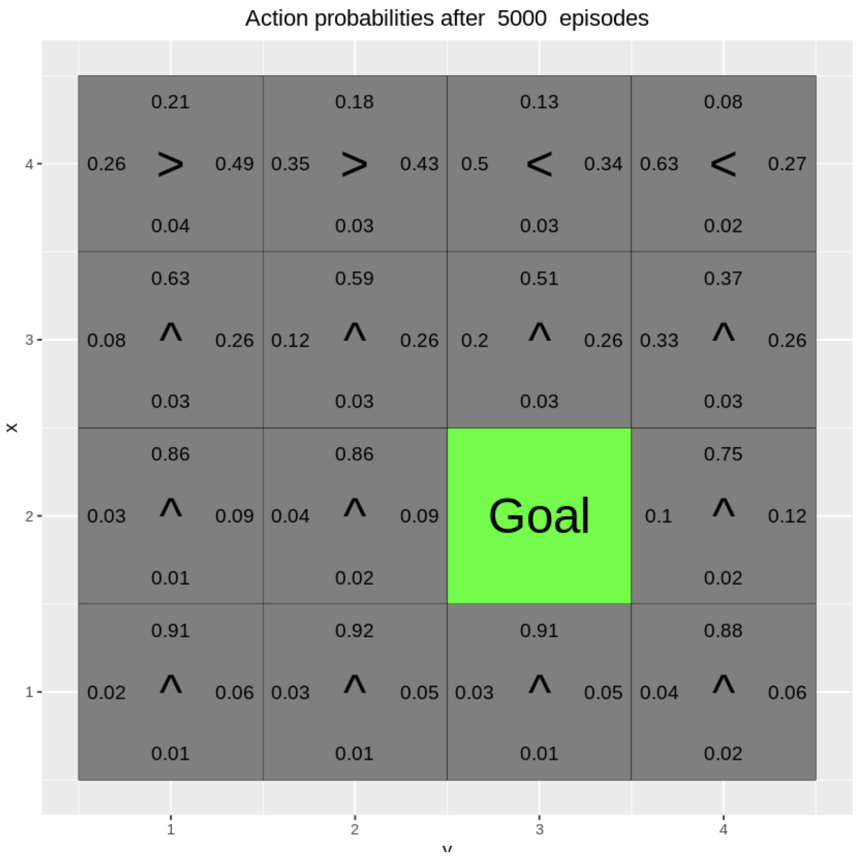
Here, Beta determines the probability that the agent will slip. A higher beta results in the agent avoiding the -1 wall with greater margin since it might slip.

2.5 REINFORCE

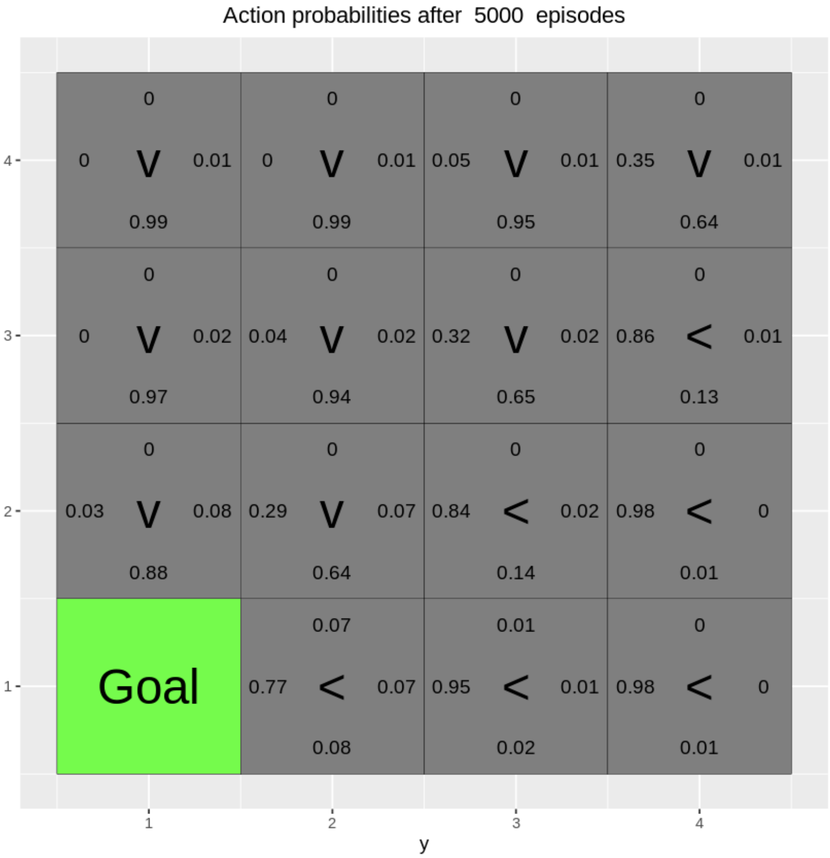
Study the provided REINFORCE algorithm for a 4×4 grid-world, where the agent must learn to navigate to a random goal position, and analyze the code and results without needing to run it.

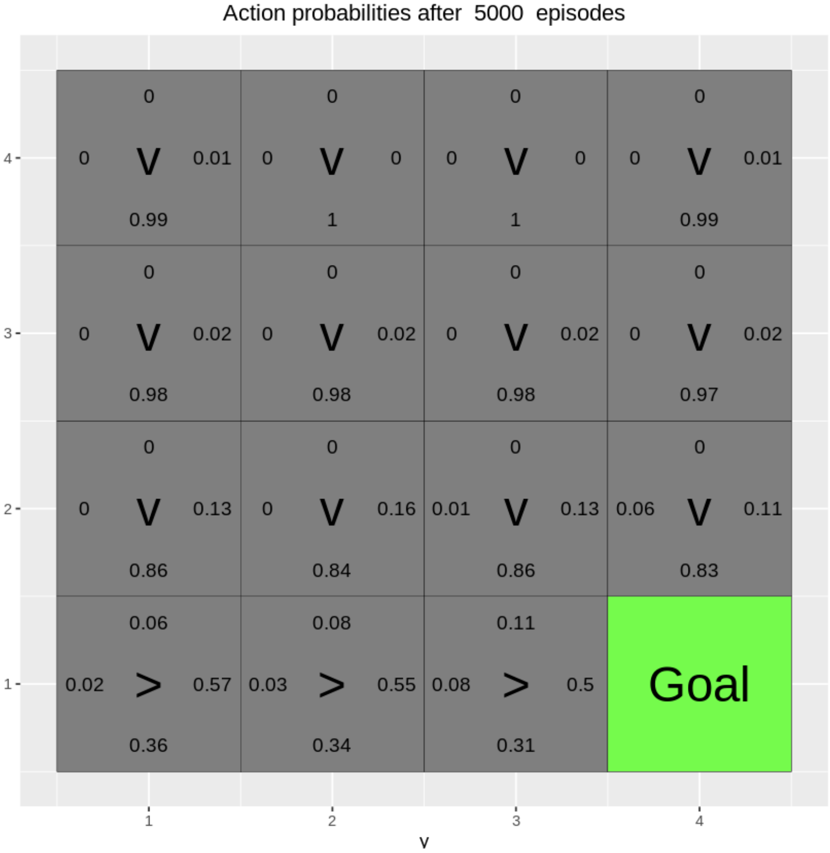
Plot 1 and 2


Action probabilities after 5000 episodes
0.79
0.78
0.73
0.67
4
0.03 ^ 0.17
0.04 ^ 0.17
0.07 ^ 0.18
0.13 ^ 0.19
0.01
0.01
0.01
0.01
0.93
0.93
0.93
3
0.01 ^ 0.06
0.01 ^ 0.05
0.02 ^ 0.05
Goal
0
0
0
x
0.96
0.97
0.97
0.97
2
0.01 ^ 0.03
0.01 ^ 0.02
0.01 ^ 0.02
0.01 ^ 0.02
0
0
0
0
0.97
0.98
0.98
0.97
1
0.01 ^ 0.02
0.01 ^ 0.02
0.01 ^ 0.01
0.01 ^ 0.02
0
0
0
0
1
2
3
4
y


Action probabilities after 5000 episodes
0.21
0.18
0.13
0.08
4
0.26 > 0.49
0.35 > 0.43
0.5 < 0.34
0.63 < 0.27
0.04
0.03
0.03
0.02
0.63
0.59
0.51
0.37
3
0.08 ^ 0.26
0.12 ^ 0.26
0.2 ^ 0.26
0.33 ^ 0.26
0.03
0.03
0.03
0.03
x
0.86
0.86
0.75
2
0.03 ^ 0.09
0.04 ^ 0.09
Goal
0.1 ^ 0.12
0.01
0.02
0.02
0.91
0.92
0.91
0.88
1
0.02 ^ 0.06
0.03 ^ 0.05
0.03 ^ 0.05
0.04 ^ 0.06
0.01
0.01
0.01
0.02
1
2
3
4
y

Plot 3 and 4


Action probabilities after 5000 episodes
0 0 0 0
0 V 0.01 0 V 0.01 0.05 V 0.01 0.35 V 0.01
0.99 0.99 0.95 0.64
0 0 0 0
0 V 0.02 0.04 V 0.02 0.32 V 0.02 0.86 < 0.01
0.97 0.94 0.65 0.13
0 0 0 0
0.03 V 0.08 0.29 V 0.07 0.84 < 0.02 0.98 < 0
0.88 0.64 0.14 0.01
0.07 0.01 0
Goal 0.77 < 0.07 0.95 < 0.01 0.98 < 0
0.08 0.02 0.01
4
3
2
1
1 2 3 4
y


Action probabilities after 5000 episodes
0 0 0 0
0 V 0.01 0 V 0 0 V 0 0 V 0.01
0.99 1 1 0.99
0 0 0 0
0 V 0.02 0 V 0.02 0 V 0.02 0 V 0.02
0.98 0.98 0.98 0.97
0 0 0 0
0 V 0.13 0 V 0.16 0.01 V 0.13 0.06 V 0.11
0.86 0.84 0.86 0.83
0.06 0.08 0.11
0.02 > 0.57 0.03 > 0.55 0.08 > 0.5 Goal
0.36 0.34 0.31
4
3
2
1
1 2 3 4
y

2.6 Environment D

Train a REINFORCE agent on eight goal positions and validate it on the remaining eight positions, then answer questions on whether the agent learned a good policy and whether Q-learning could solve the task.

Has the agent learned a good policy? Why / Why not ?

Yes, the agent has learned relatively good policy (plot 3 and 4). The policy generalizes well during validation since the training goals were spread across the grid, allowing the agent to learn a flexible policy.

Could you have used the Q-learning algorithm to solve this task?

Yes, although REINFORCE is much more suitable. Q-learning could solve this task by incorporating the goal coordinates into the state representation. However, Q-learning might struggle with generalizing as effectively as REINFORCE.

2.7 Environment E

Analyze the agent's performance when trained with goals from the top row and validated with goals from lower rows, and compare the results with environment D, explaining any differences.

Has the agent learned a good policy? Why / Why not ?

No, the agent has not learned a fully generalized policy (plot 1 and 2). It has overfitted to the training where the goals were located in the top row making the policy arrows pointing upward.

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

The difference arises from the diversity of the training goals. In Environment D, the training goals were spread across the grid, resulting in better generalization. In Environment E, the training goals were restricted to the top row, leading to overfitting and poor generalization to new goal positions during validation.