TDDE15-Lab 2

frera064, oscho091, hjaoh082, olosw720

Contribution

All four members solved the problems on their own. We then discussed the answers and compiled the group report by combining our solutions. We used frera064 solution for environment A, oscho091 solution for Environment B, hjaoh082s solution for environment C and olosw720 solution for the second part, REINFORCE.

2.1 Q-Learning

You are asked to complete the implementation. We will work with a gridworld environment consisting of H $\times$ W tiles laid out in a 2-dimensional grid. An agent acts by moving up, down, left or right in the grid-world. This corresponds to the following Markov decision process:

The state space is defined as:

$$S = \{(x, y) \mid x \in \{1, \ldots, H\}, y \in \{1, \ldots, W\}\}$$

The action space is defined as:

$$A = \{\text{up}, \text{down}, \text{left}, \text{right}\}$$

Additionally, we assume state space to be fully observable. The reward function is a deterministic function of the state and does not depend on the actions taken by the agent. We assume the agent gets the reward as soon as it moves to a state. The transition model is defined by the agent moving in the direction chosen with probability $(1 - \beta)$. The agent might also slip and end up moving in the direction to the left or right of its chosen action, each with probability $\beta/2$. The transition model is unknown to the agent, forcing us to resort to model-free solutions. The environment is episodic and all states with a non-zero reward are terminal. Throughout this lab we use integer representations of the different actions: Up=1, right=2, down=3 and left=4.

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

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

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

  # 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)))

}
```

Environment A

Implement the greedy and ϵ-greedy policies in the functions GreedyPolicy and EpsilonGreedyPolicy of the file RL Lab1.R. The functions should break ties at random, i.e. they should sample uniformly from the set of actions with maximal Qvalue

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

  return (max)

  # max <- which.max(q_table[x,y,])#which(q_table[x,y,] == max(q_table[x,y,]))
  #
  # return (max)#sample(max, size = 1))

}

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.

  max <- GreedyPolicy(x,y)

  epsilon_prob <- runif(1)
  if (epsilon_prob <= epsilon){
    max <-sample(1:4,1)
  }

  return(max)
  # If i want to put in a validity constraint
  # repeat{
  #   new_state <- c(x,y) + unlist(action_deltas[max])
  #   xn <- new_state[1]
  #   yn <- new_state[2]
  #   if(!(yn < 1 || xn < 1 || yn > W || xn > H)){
  #     return(max)
```

```
  #   }
  # }

}
```

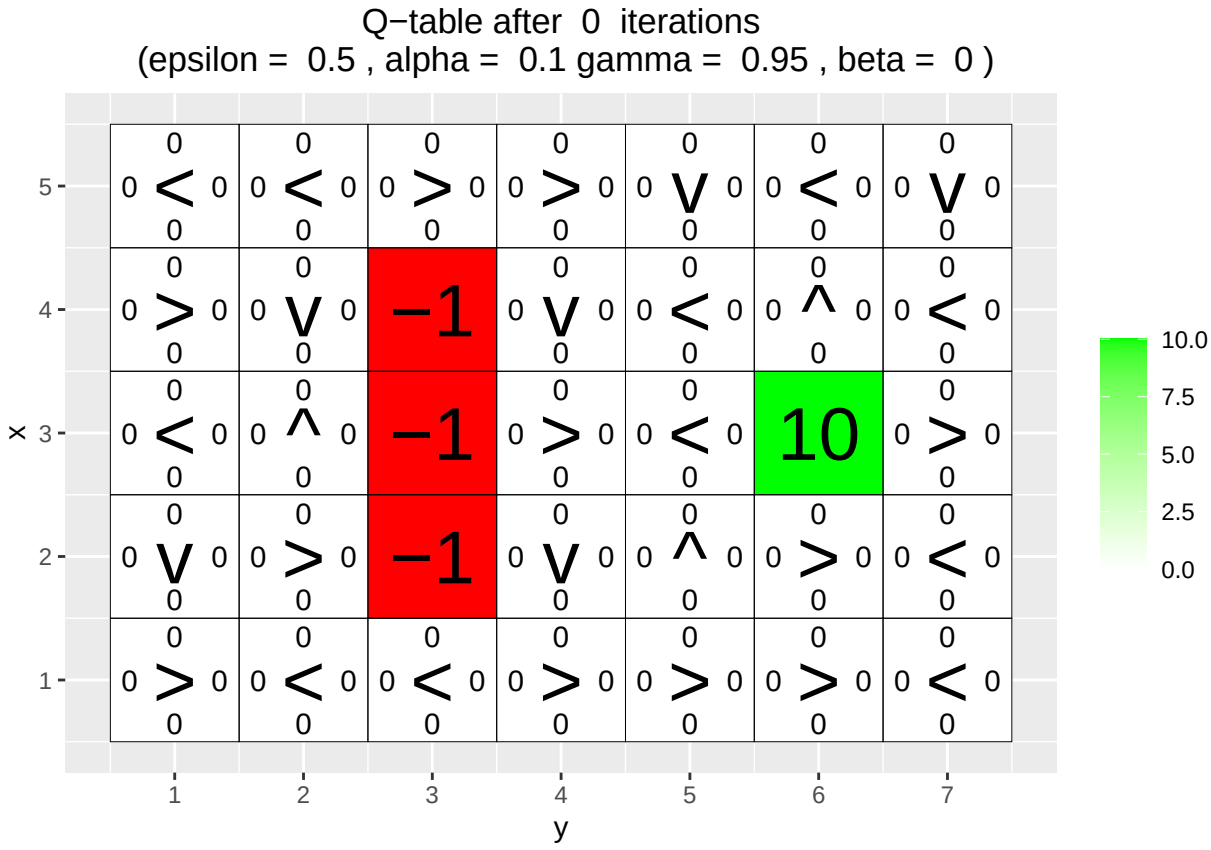
Implement the Q-learning algorithm in the function q learning of the file RL Lab1.R. The function should run one episode of the agent acting in the environment and update the Q-table accordingly. The function should return the episode reward and the sum of the temporal-difference correction terms $R+\gamma \max_a Q(S',a)-Q(S,A)$ for all steps in the episode. Note that a transition model taking β as input is already implemented for you in the function transition model.

```
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 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 <<-.
```

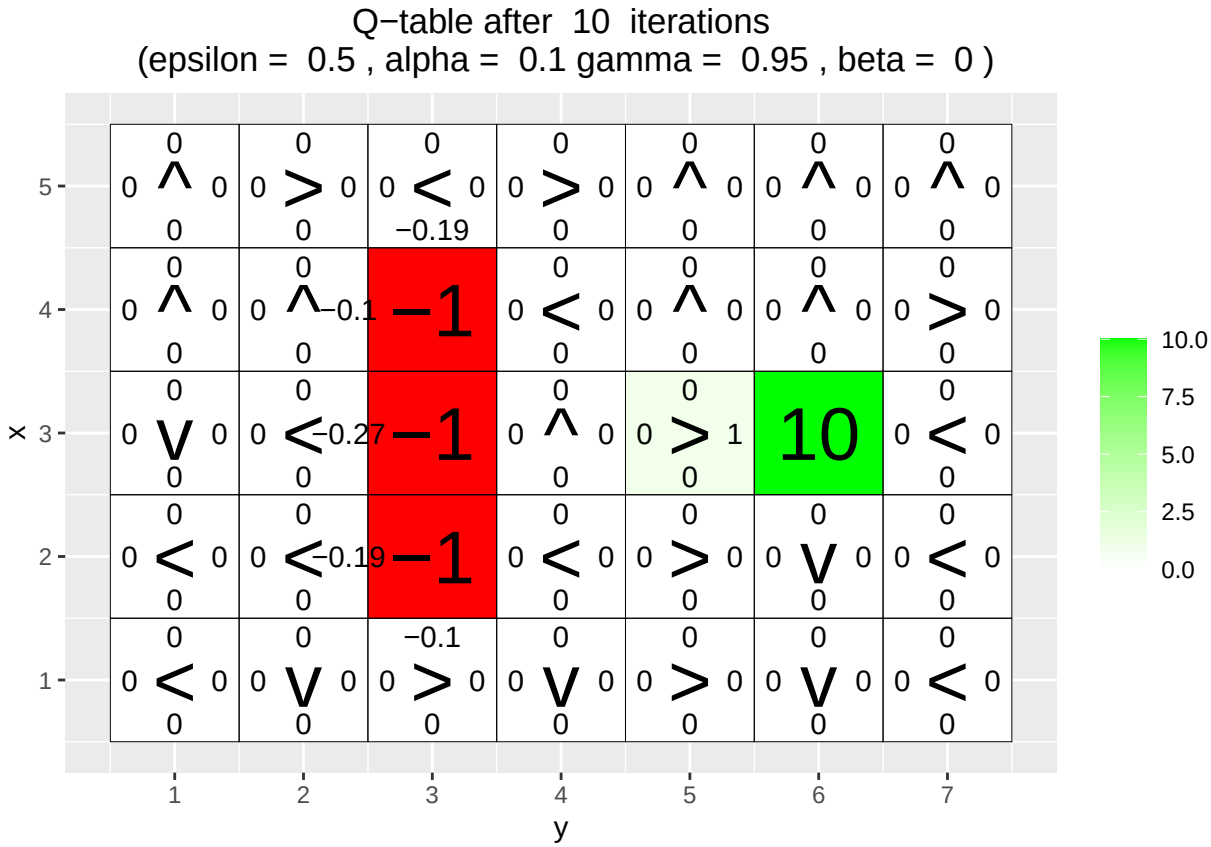
```
  # Your code here.

  # h <- sample(1:nrow(reward_map), size=1)
  # w <- sample(1:ncol(reward_map), size=1)
  # state <- c(h,w)
  state <- start_state
  episode_correction <- 0
  repeat{
    # Follow policy, execute action, get reward.
    action <- EpsilonGreedyPolicy(x = state[1], y = state[2], epsilon = epsilon)
    next_state <- transition_model(x = state[1], y = state[2], action, beta)
    reward <- reward_map[next_state[1], next_state[2]]

    # Q-table update.
    temp_diff <- alpha*(reward + gamma * max(q_table[next_state[1], next_state[2],]) - q_table[state[1]
    q_table[state[1], state[2], action] <<- q_table[state[1], state[2], action] + temp_diff

    episode_correction <- episode_correction + temp_diff

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

}
```

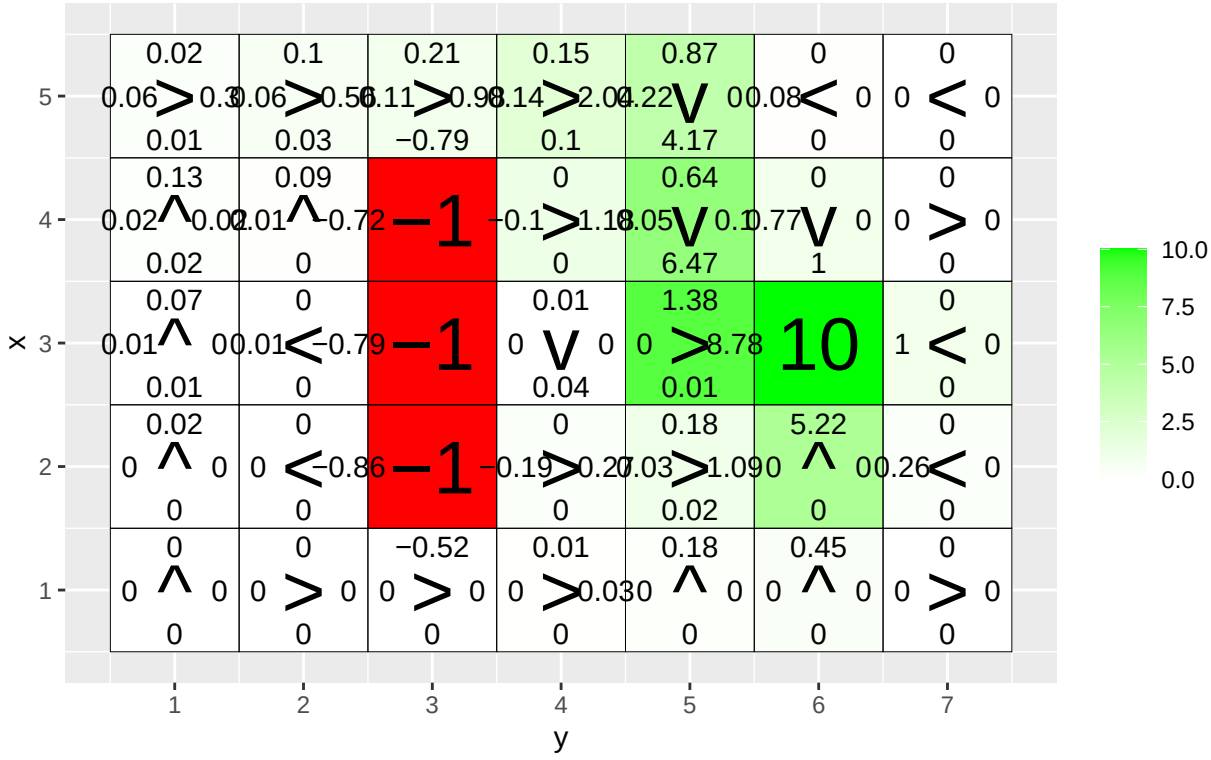
Run 10000 episodes of Q-learning with $\epsilon = 0.5$, $\beta = 0$, $\alpha = 0.1$ and $\gamma = 0.95$. To do so, simply run the code provided in the file RL Lab1.R. The code visualizes the Q-table and a greedy policy derived from it after episodes 10, 100, 1000 and 10000.

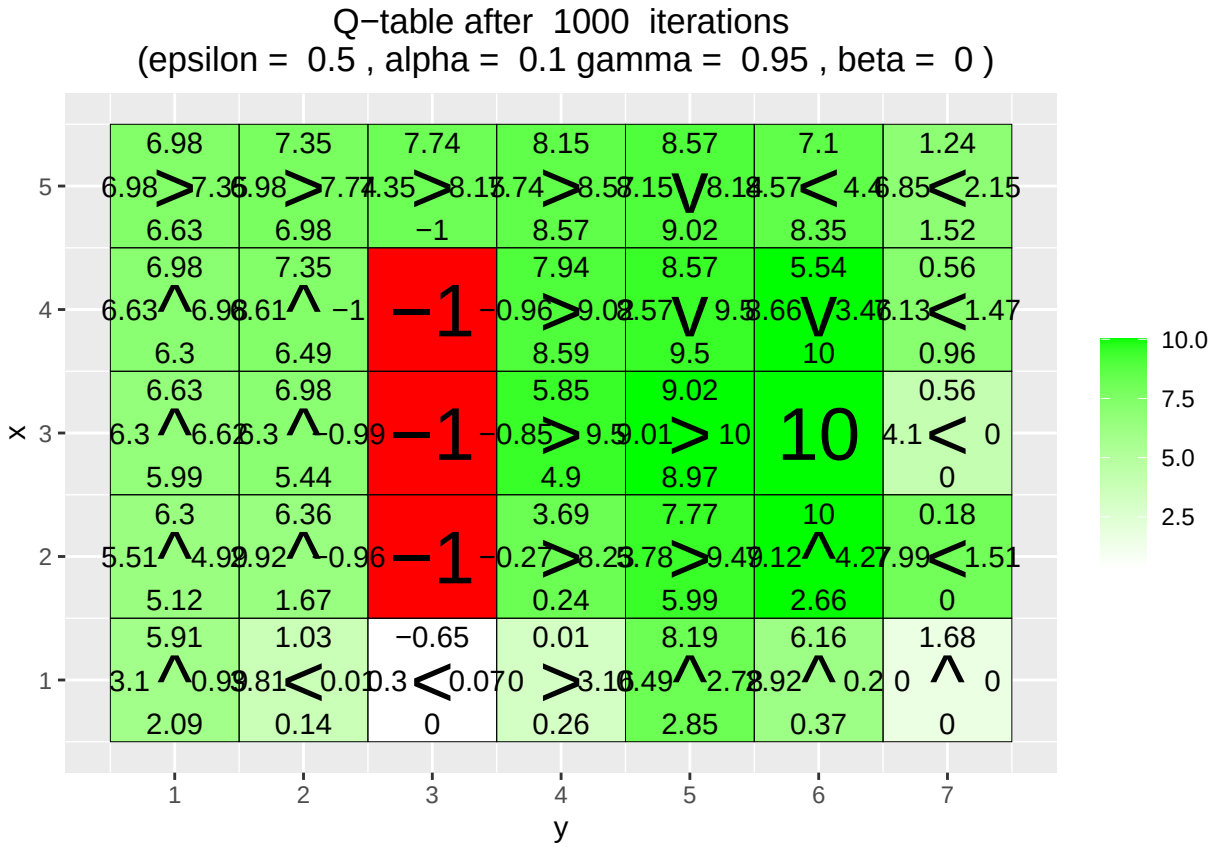
```
#############################################################################################################
# Q-Learning Environments
#############################################################################################################
set.seed(12345)
# Environment A (learning)
H <- 5
W <- 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()
```

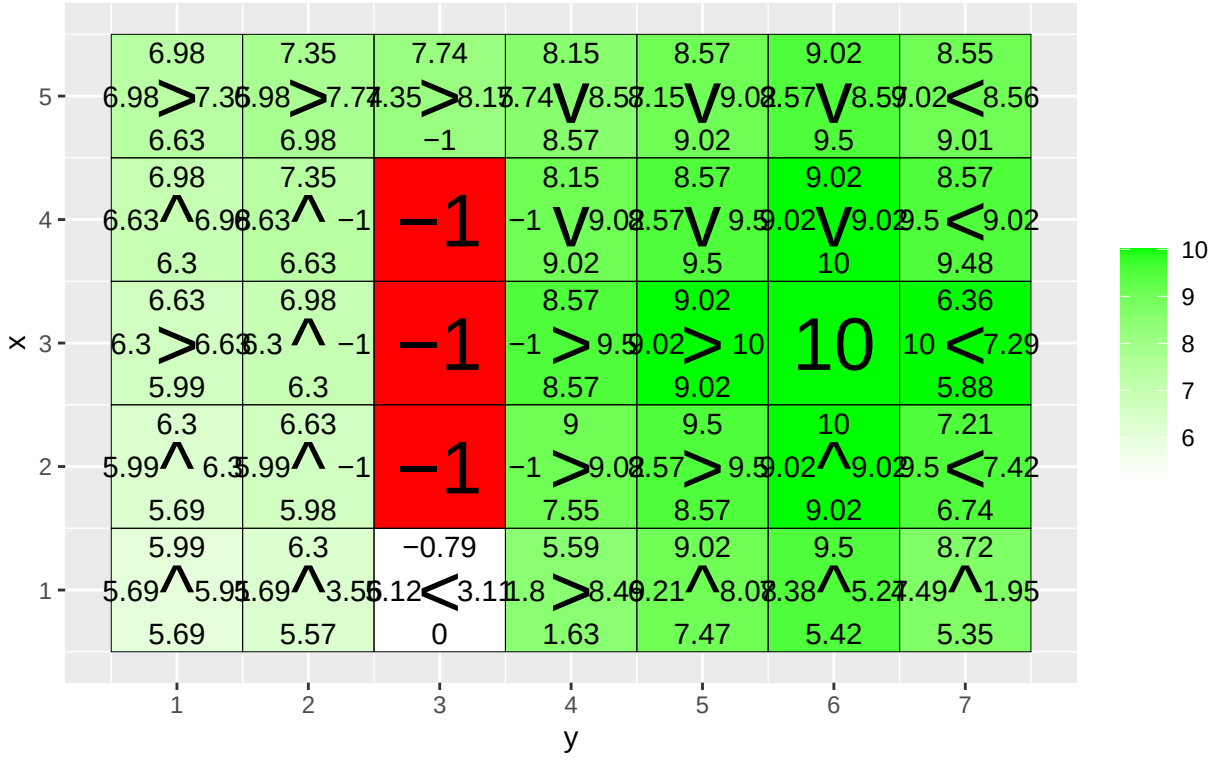



```
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))

  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)
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
10.0
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


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10
9
8
7
6

Answers:

– *What has the agent learned after the first 10 episodes ?*

It has learnt that from some states adjacent to the -1 rewards it is bad to take a step into them.

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

Yes, the greedy policy seems to be optimal for some states but some others such as the bottom left corner having the q-values for downwards movement. This could be to due to the low epsilon leading to the model not exploring enough.

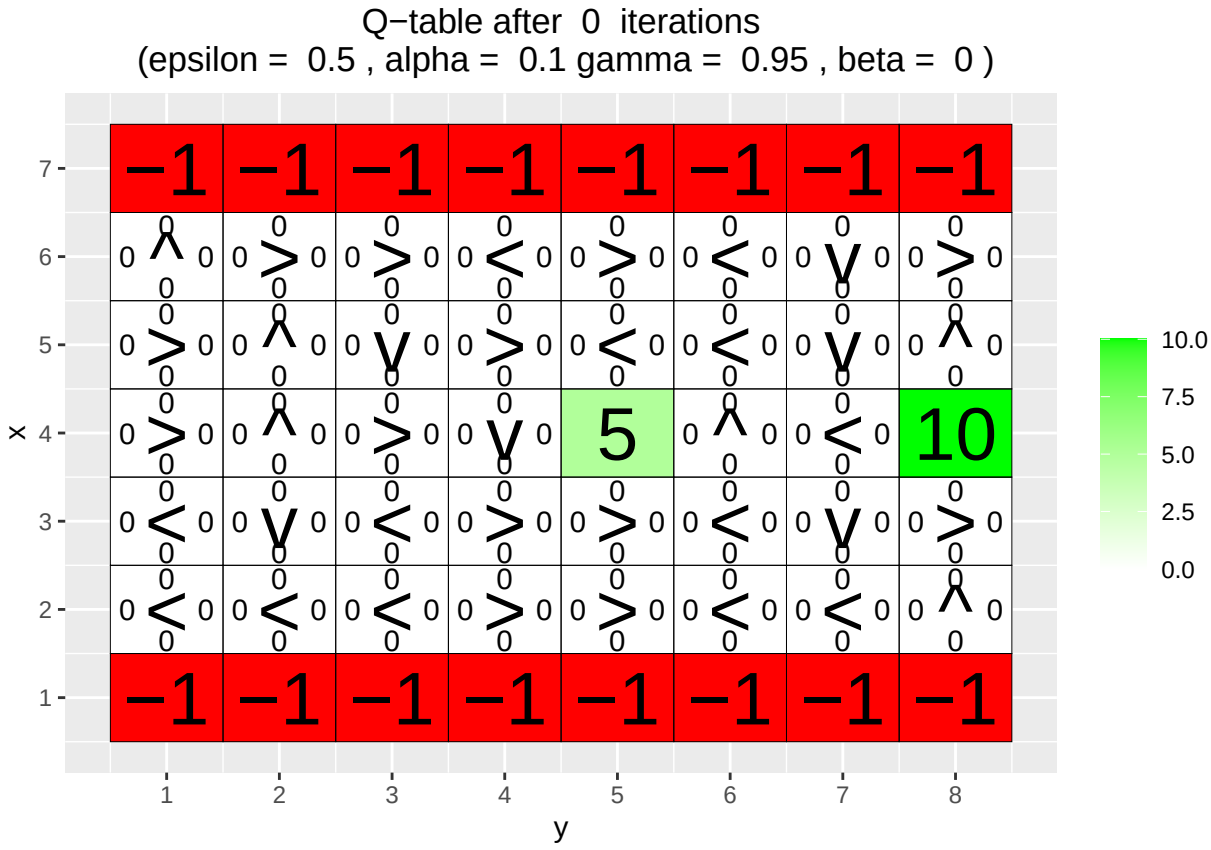
– *Do the learned values in the Q-table reflect the fact that there are multiple paths (above and below the negative rewards) to get to the positive reward ? If not, what could be done to make it happen ?*

No, since the model has learned the lower path only. This is probably also due to the low epsilon not exploring new paths when one is found.

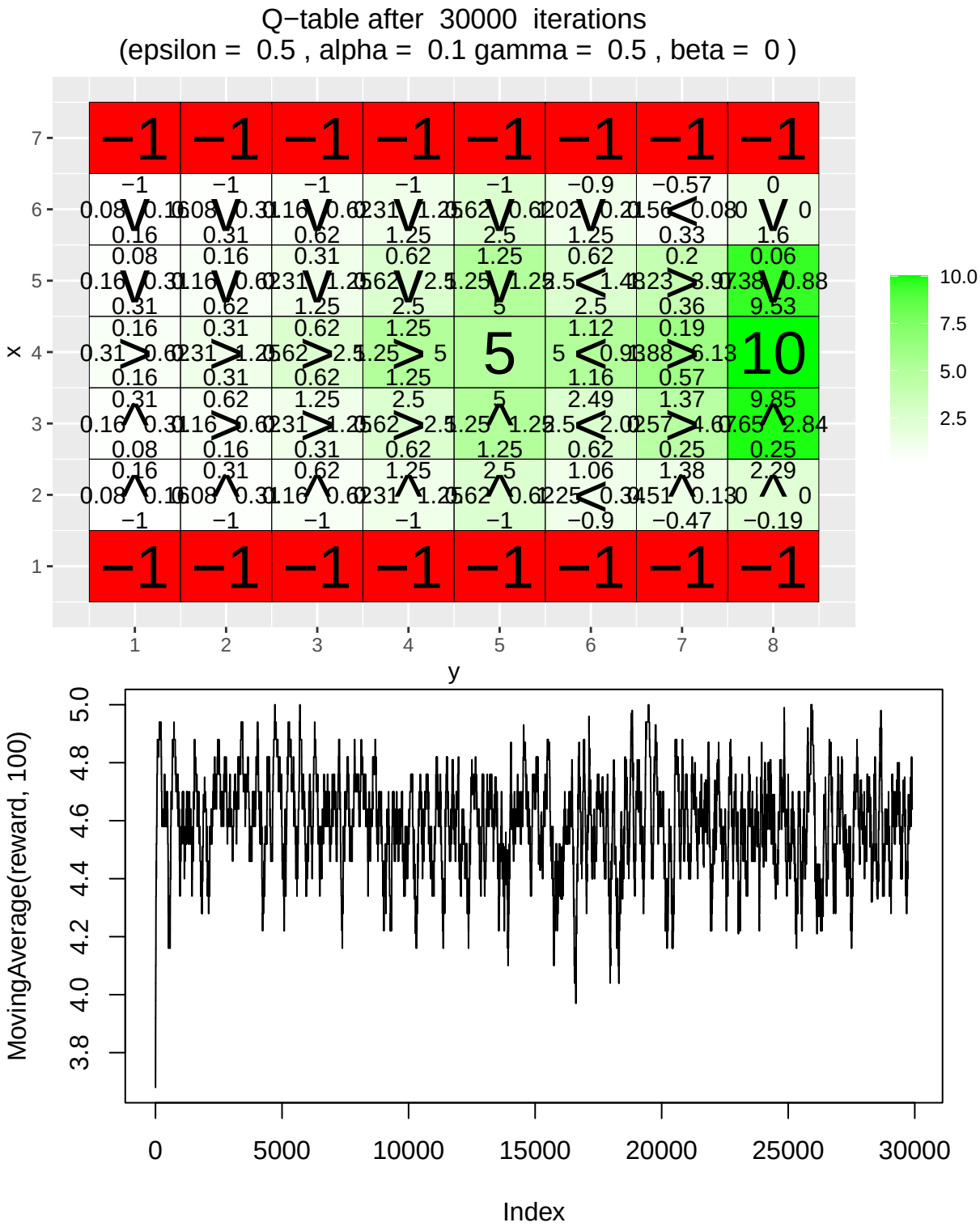
Environment B.

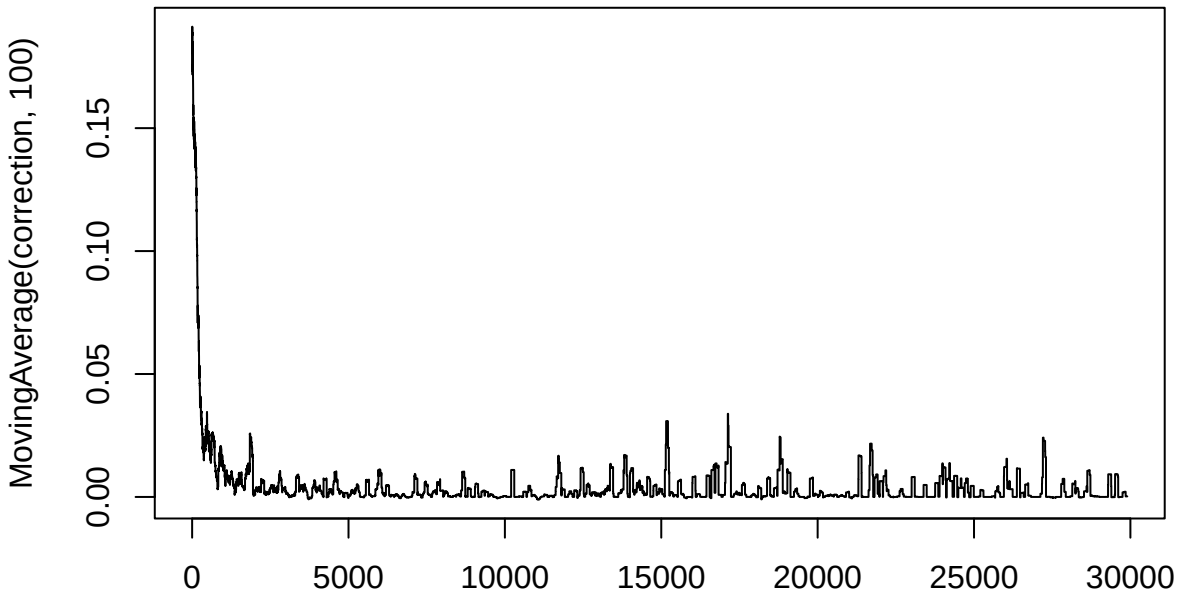
This is a 7Ö8 environment where the top and bottom rows have negative rewards. In this environment, the agent starts each episode in the state $(4, 1)$. There are two positive rewards, of 5 and 10. The reward of 5 is easily reachable, but the agent has to navigate around the first reward in order to find the reward worth 10. Your task is to investigate how the ϵ and γ parameters affect the learned policy by running 30000 episodes of Q-learning with $\epsilon = 0.1, 0.5$, $\gamma = 0.5, 0.75, 0.95$, $\beta = 0$ and $\alpha = 0.1$. To do so, simply run the code provided in the file RL Lab1.R and explain your observations.

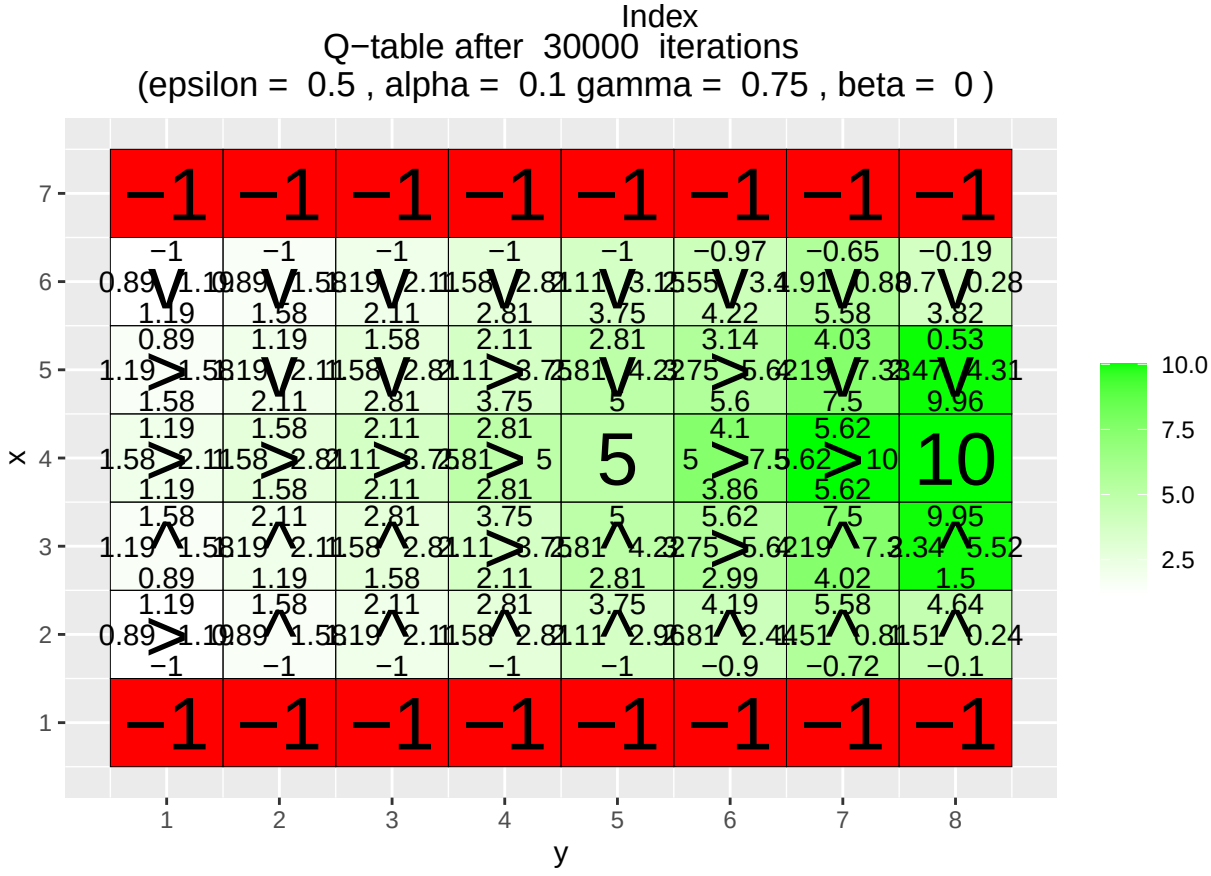
```
# Environment B (the effect of epsilon and gamma)
set.seed(1234)

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()
```

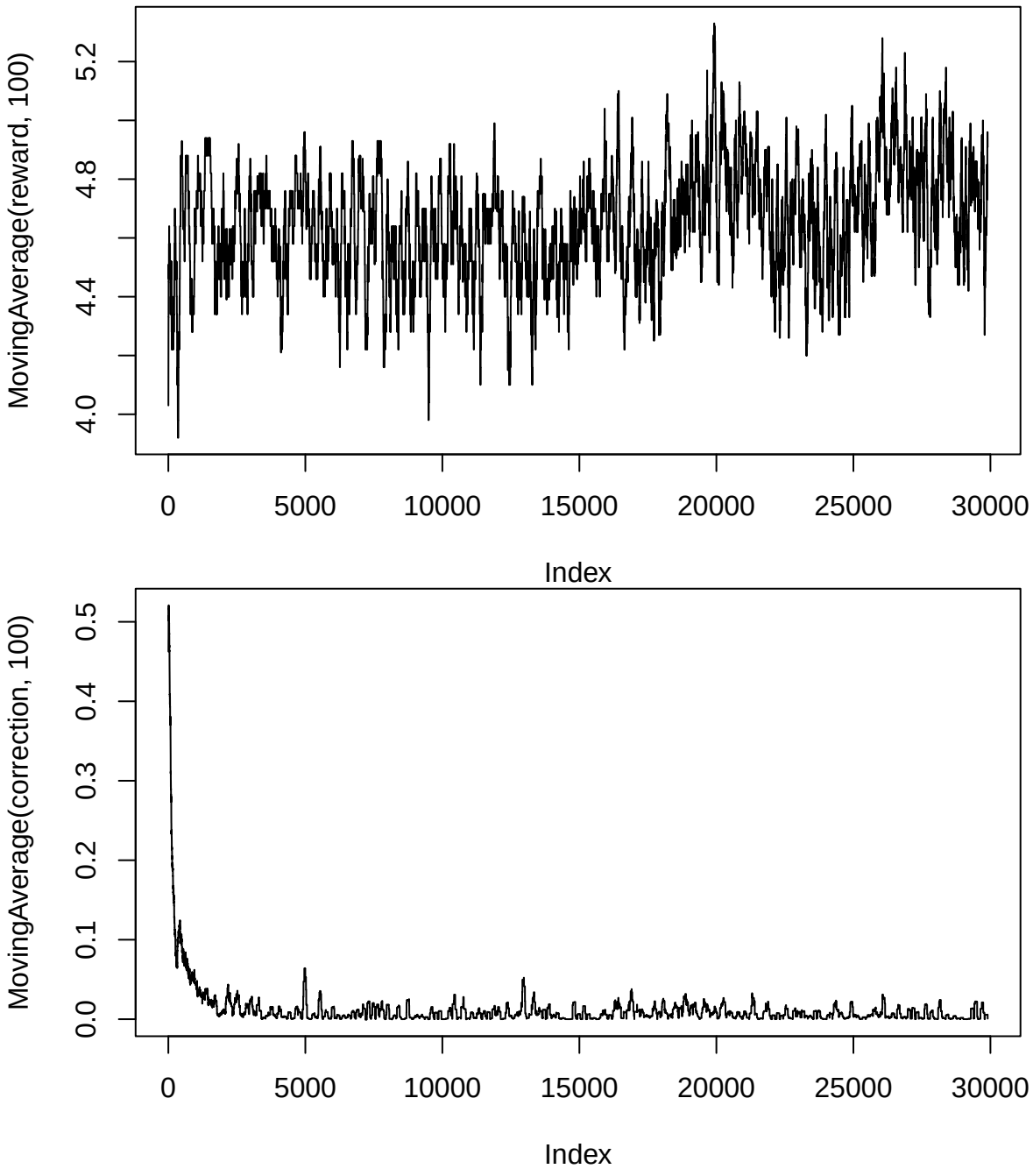


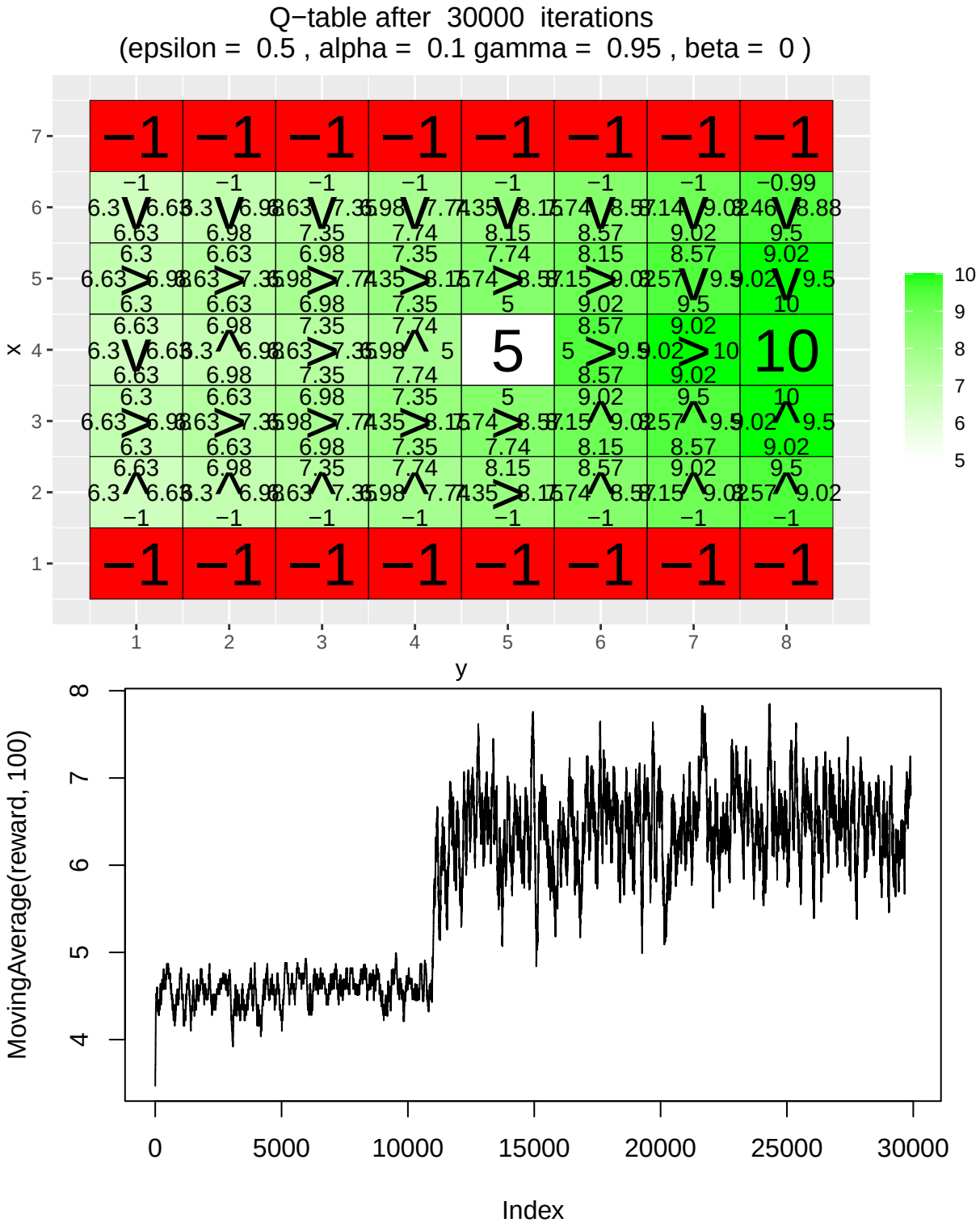

```
MovingAverage <- function(x, n){

  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")
}
```

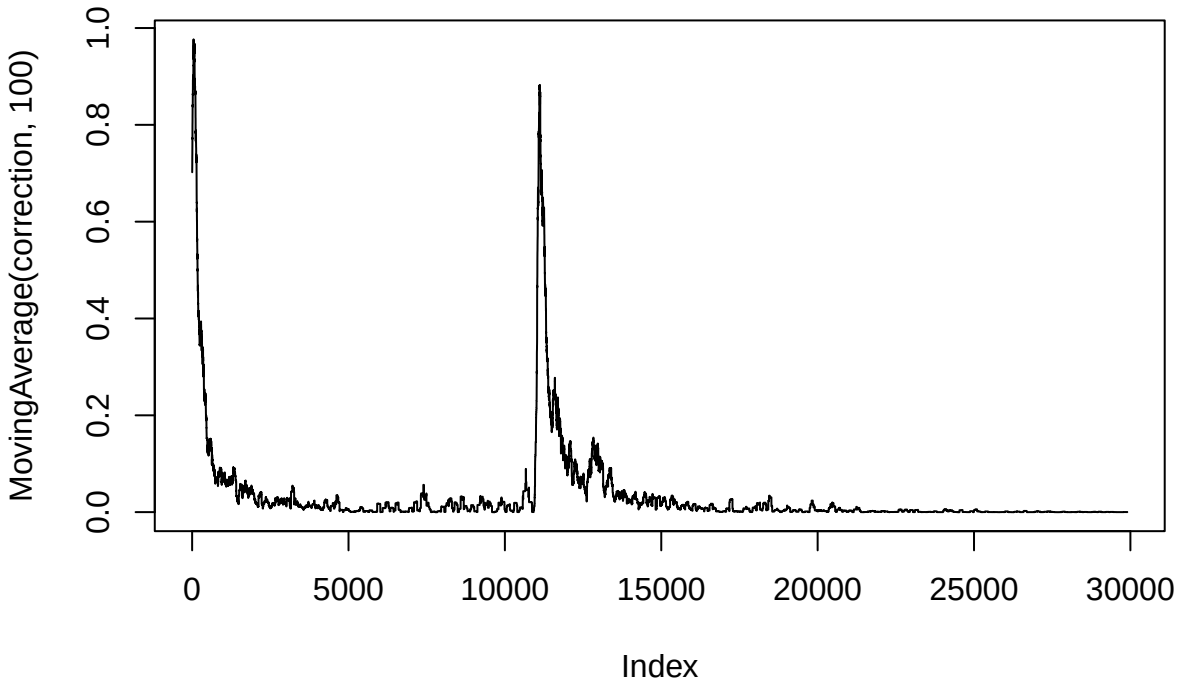

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10.0
7.5
5.0
2.5
MovingAverage(reward, 100)
Index


MovingAverage(correction, 100)
0.00
0.05
0.10
0.15
0
5000
10000
15000
20000
25000
30000
Index

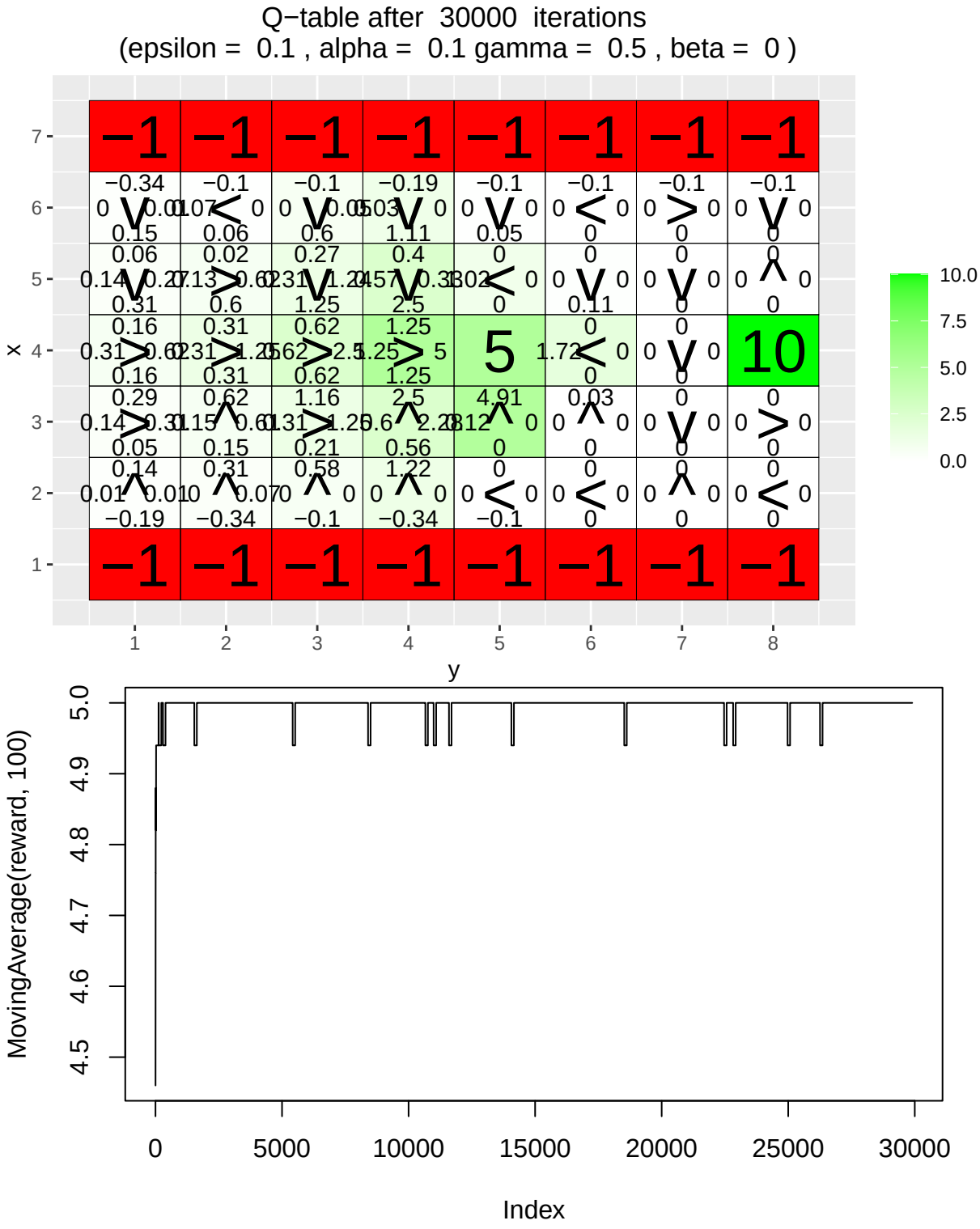

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5

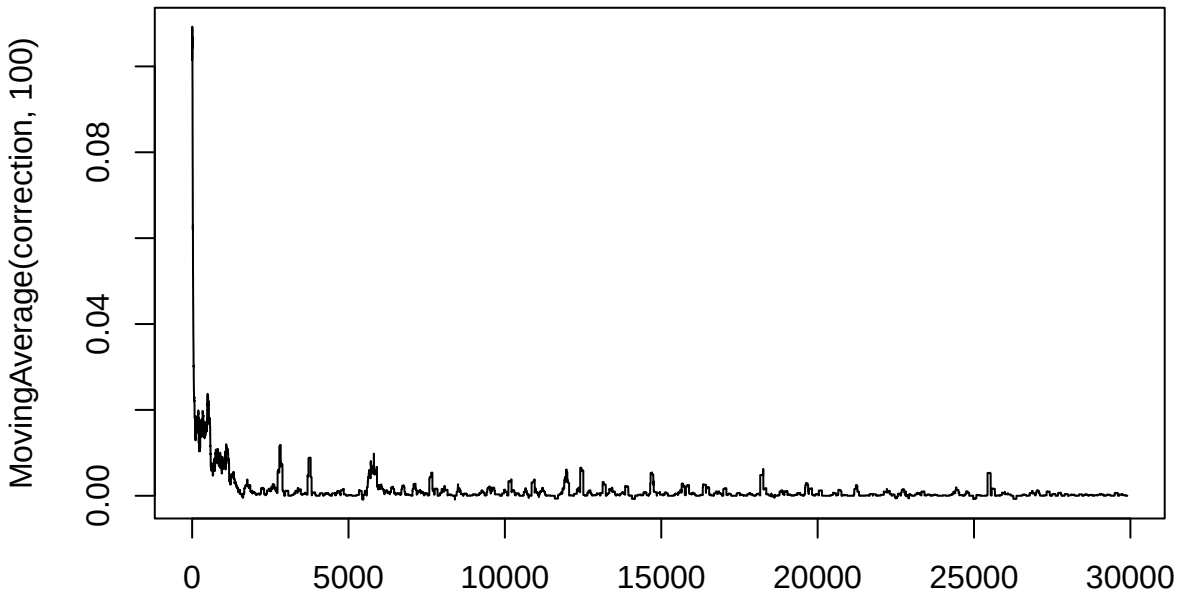

MovingAverage(reward, 100)
4.0
4.4
4.8
5.2
0
5000
10000
15000
20000
25000
30000
Index
MovingAverage(correction, 100)
0.0
0.1
0.2
0.3
0.4
0.5
0
5000
10000
15000
20000
25000
30000
Index


Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10
9
8
7
6
5
MovingAverage(reward, 100)
Index
0
5000
10000
15000
20000
25000
30000

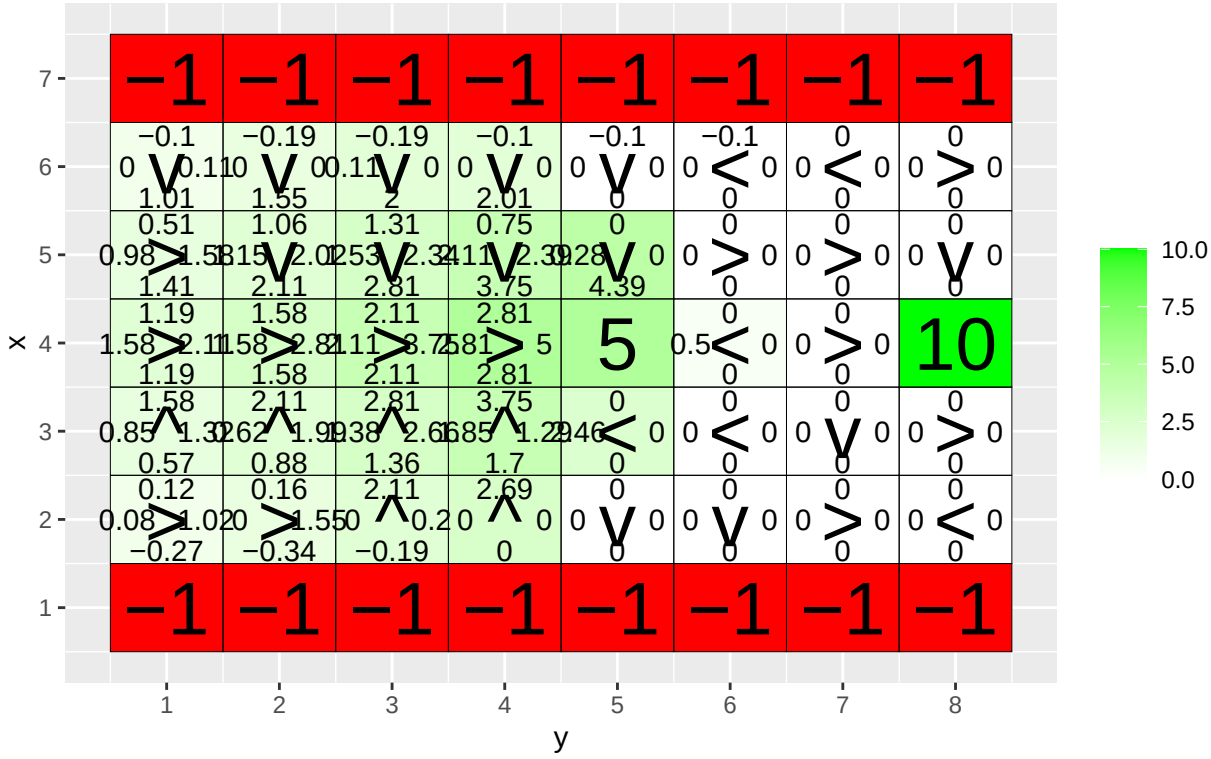


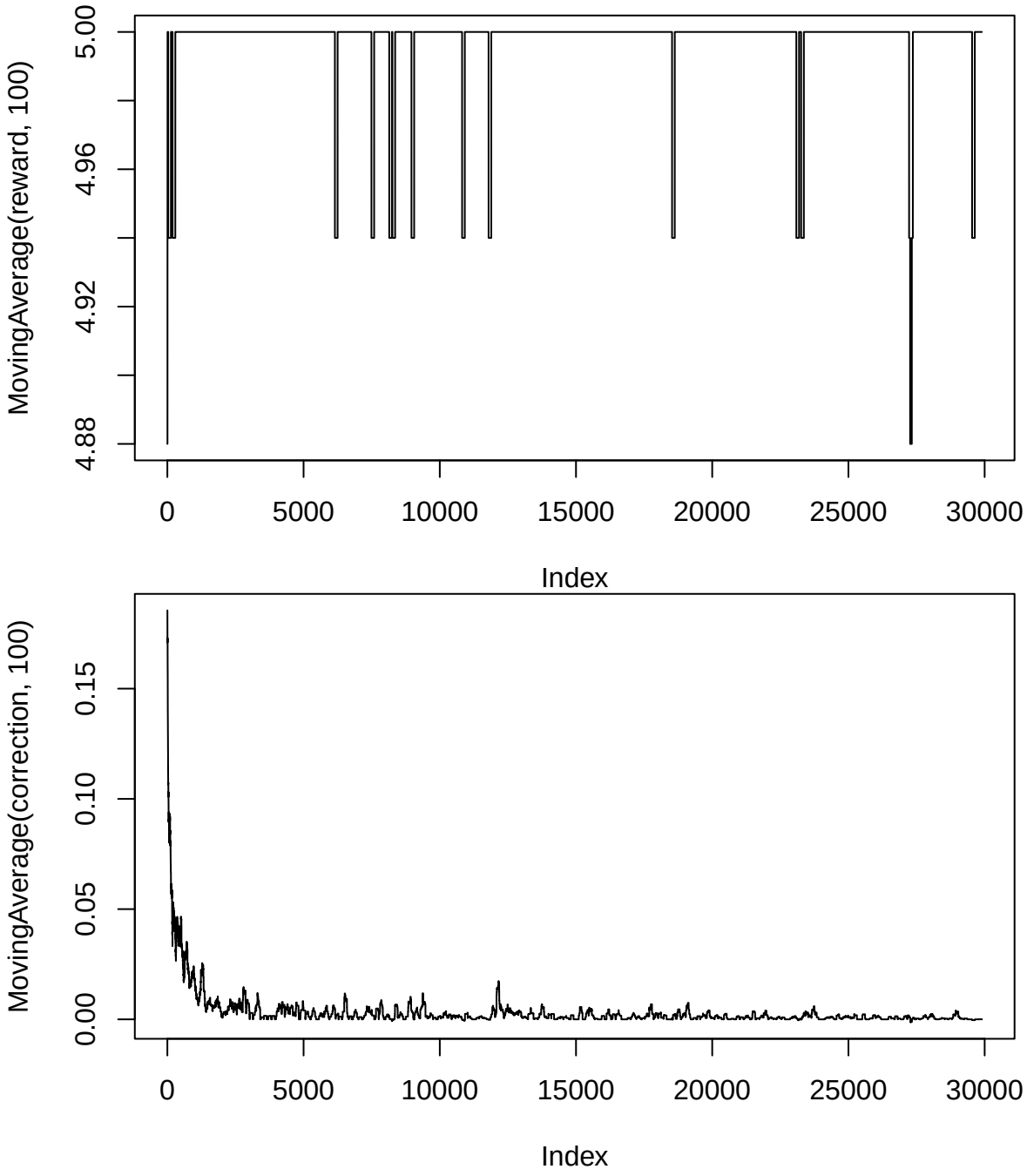

```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  for(i in 1:30000){
    foo <- q_learning(epsilon = 0.1, gamma = j, start_state = c(4,1))
    reward <- c(reward,foo[1])
    correction <- c(correction,foo[2])
  }

  vis_environment(i, epsilon = 0.1, gamma = j)
  plot(MovingAverage(reward,100),type = "l")
  plot(MovingAverage(correction,100),type = "l")
}
```

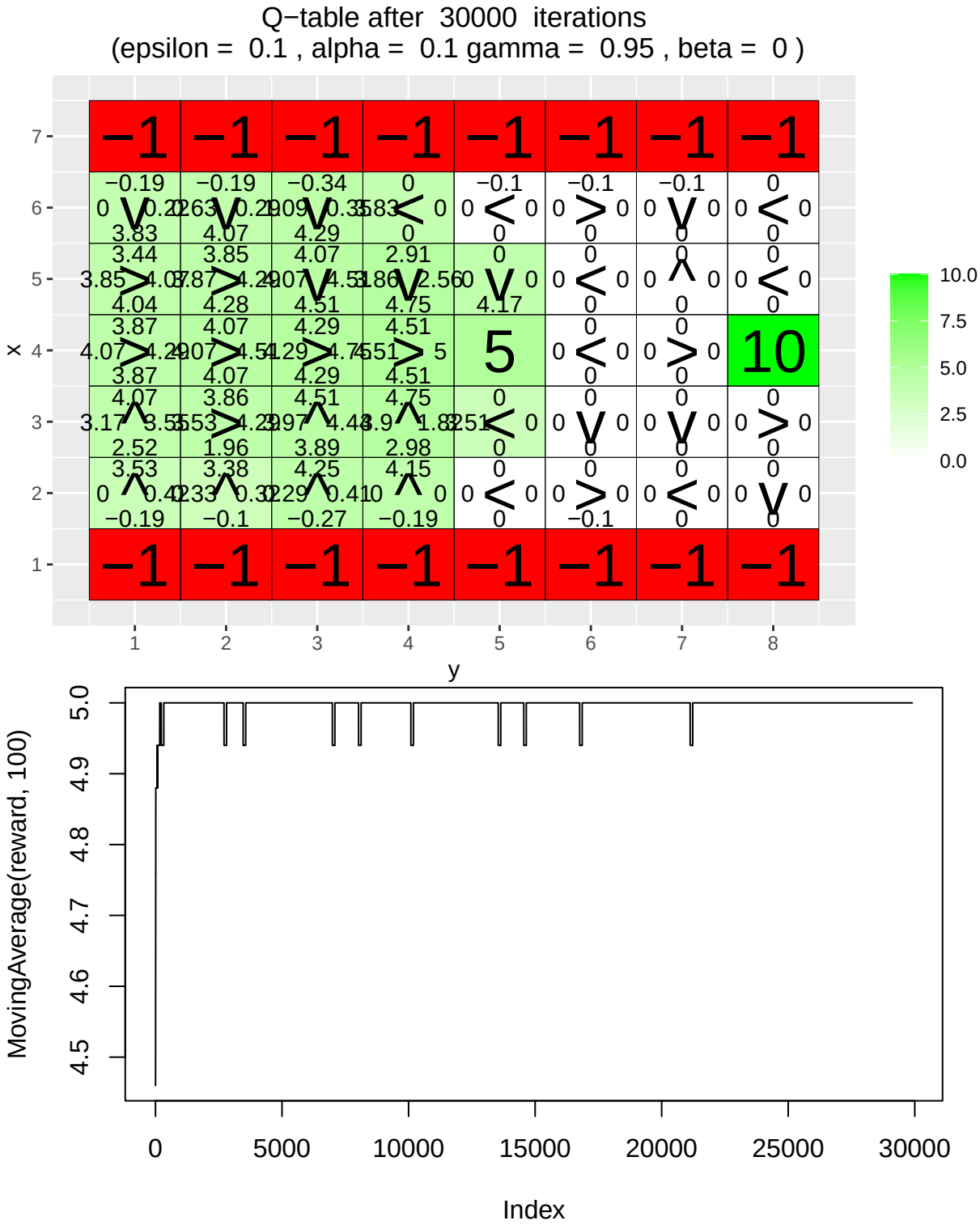

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0
MovingAverage(reward, 100)
Index

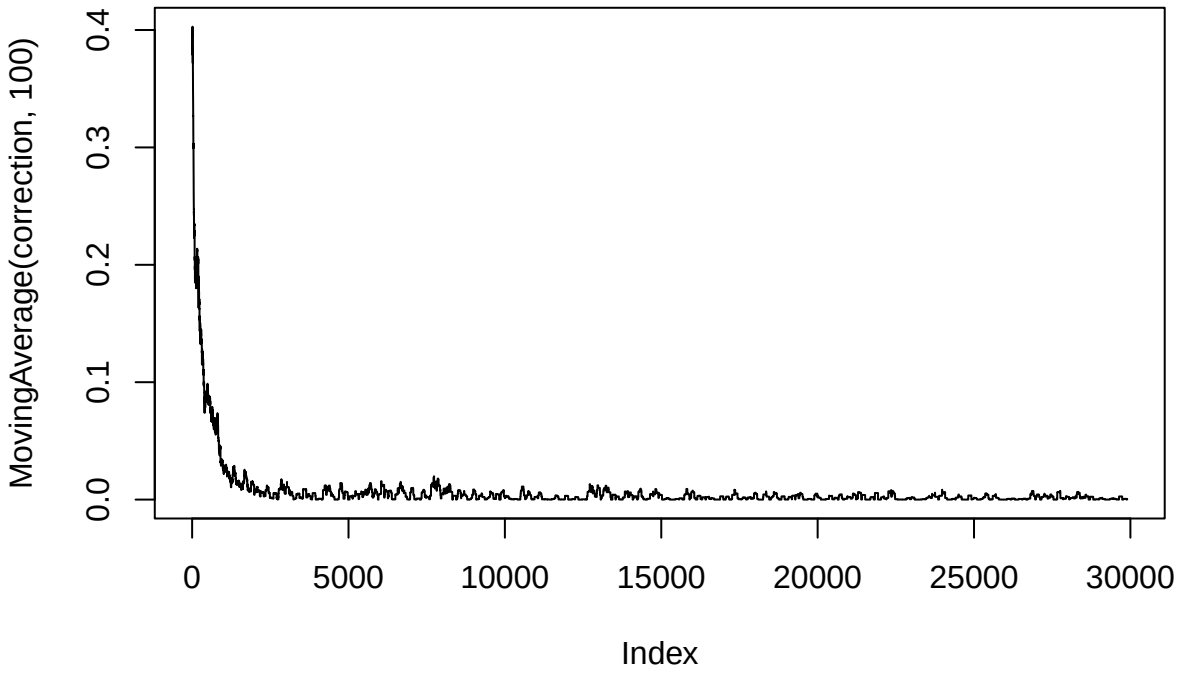


Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)




MovingAverage(reward, 100)
4.88
4.92
4.96
5.00
0
5000
10000
15000
20000
25000
30000
Index
MovingAverage(correction, 100)
0.00
0.05
0.10
0.15
0
5000
10000
15000
20000
25000
30000
Index


Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0
MovingAverage(reward, 100)
Index



Our observations are:

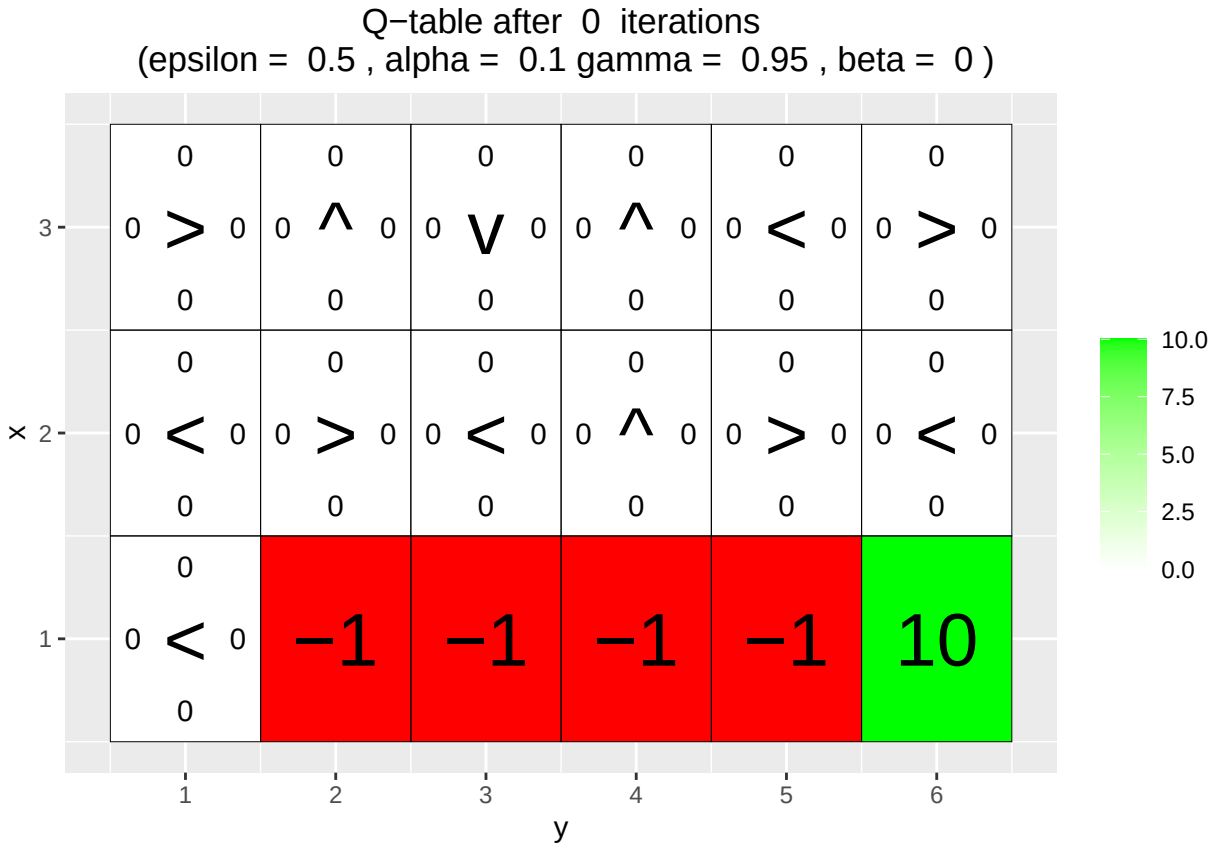
$\epsilon = 0.5$ $\gamma = 0.5$: It chooses 5 often since the discount factor gamma is low meaning that it prioritizes future rewards less then current $\gamma = 0.75$: Most often chooses 10 but sometimes 5 $\gamma = 0.95$: results in that it learns to avoid 5 reward and go for 10 reward, the reward graph is also delayed and correction spikes up later

With $\epsilon = 0.1$ the agent doesn't explore as much rather exploits the found policy and therefore only finds upper paths around 5, doesn't learn to completely ignore 5

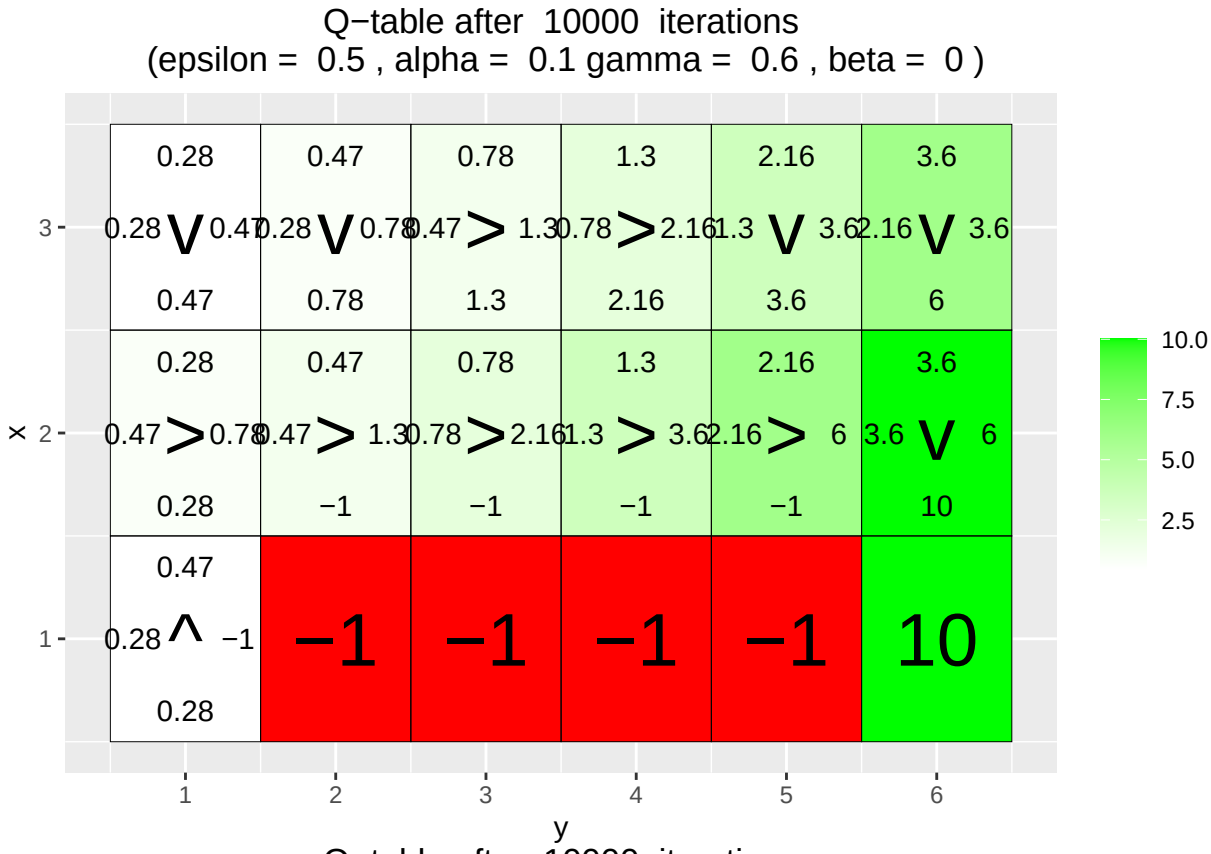
Environment C

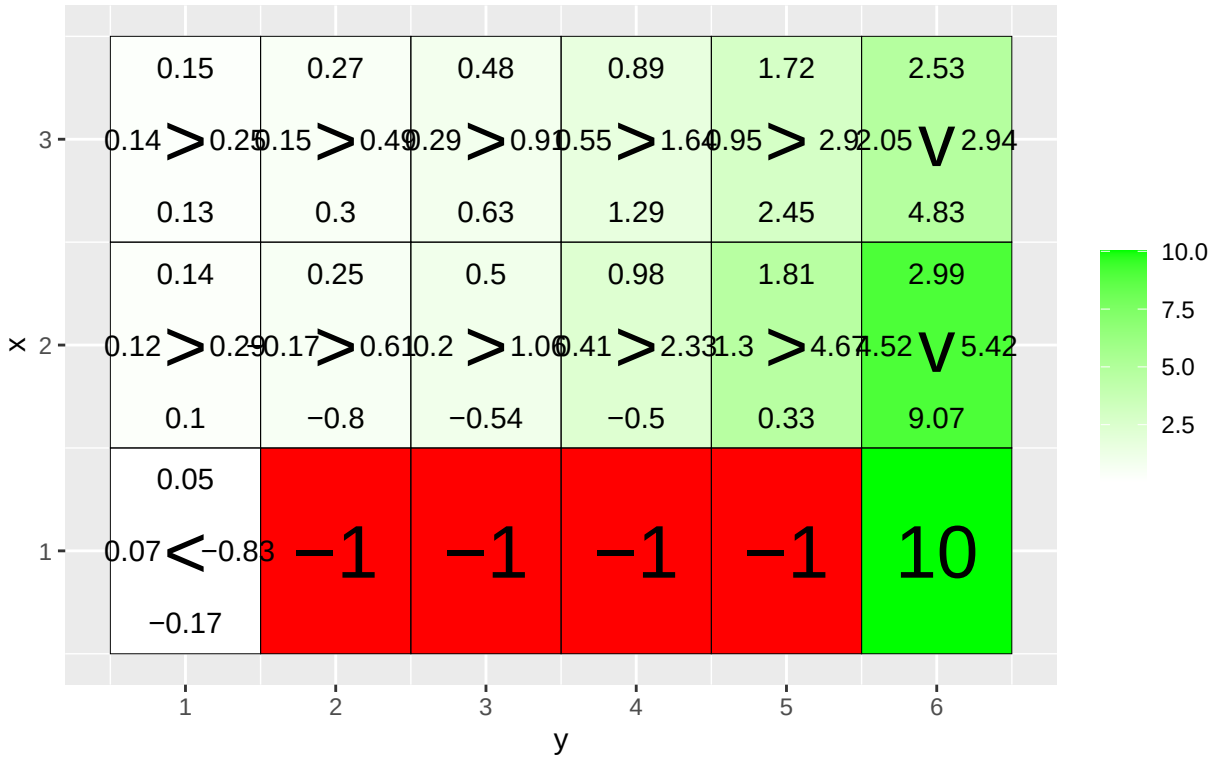
This is a smaller 3Ö6 environment. Here the agent starts each episode in the state $(1, 1)$. Your task is to investigate how the β parameter affects the learned policy by running 10000 episodes of Q-learning with $\beta = 0, 0.2, 0.4, 0.66$, $\epsilon = 0.5$, $\gamma = 0.6$ and $\alpha = 0.1$. To do so, simply run the code provided in the file RL Lab1.R and explain your observations.

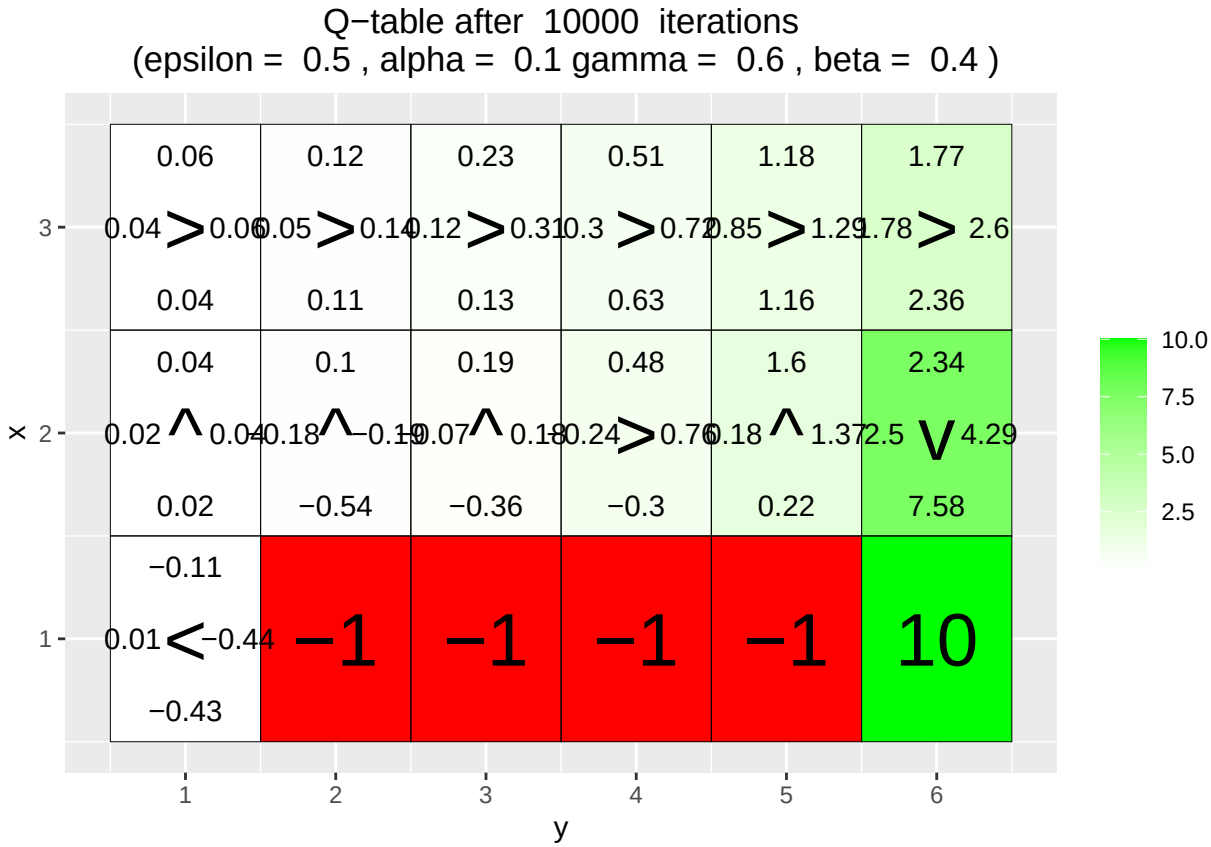
```
# Environment C (the effect of beta).
set.seed(1234)

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()
```

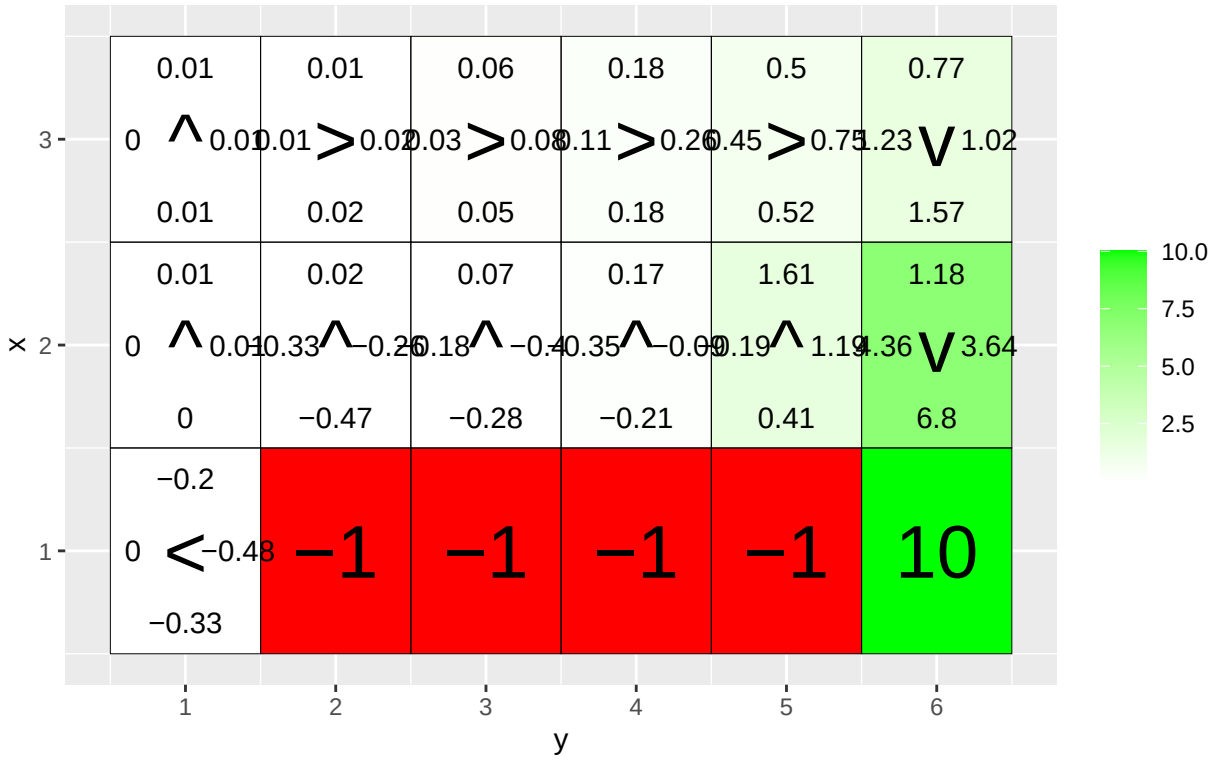



```
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

  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 0.47 0.78 1.3 2.16 3.6
0.28 V 0.47 0.28 V 0.78 0.47 > 1.3 0.78 > 2.16 1.3 V 3.6 2.16 V 3.6
0.47 0.78 1.3 2.16 3.6 6
0.28 0.47 0.78 1.3 2.16 3.6
0.47 > 0.78 0.47 > 1.3 0.78 > 2.16 1.3 > 3.6 2.16 > 6 3.6 V 6
0.28 −1 −1 −1 −1 10
0.47
0.28 ^ −1 −1 −1 −1 −1 10
0.28
3
2
1
x
1 2 3 4 5 6
y
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)
0.15 0.27 0.48 0.89 1.72 2.53
0.14 > 0.25 0.15 > 0.49 0.29 > 0.91 0.55 > 1.64 0.95 > 2.92 2.05 V 2.94
0.13 0.3 0.63 1.29 2.45 4.83
0.14 0.25 0.5 0.98 1.81 2.99
0.12 > 0.29 0.17 > 0.61 0.2 > 1.06 0.41 > 2.33 1.3 > 4.67 4.52 V 5.42
0.1 −0.8 −0.54 −0.5 0.33 9.07
0.05
0.07 < −0.83 −1 −1 −1 −1 10
−0.17
3
2
1
x
1 2 3 4 5 6
y
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
0.12
0.23
0.51
1.18
1.77
0.04 > 0.06
0.05 > 0.14
0.12 > 0.31
0.3 > 0.72
0.85 > 1.29
1.78 > 2.6
0.04
0.11
0.13
0.63
1.16
2.36
0.04
0.1
0.19
0.48
1.6
2.34
0.02 ^ 0.04
0.18 ^ 0.19
0.07 ^ 0.18
0.24 > 0.76
0.18 ^ 1.37
2.5 V 4.29
0.02
−0.54
−0.36
−0.3
0.22
7.58
−0.11
0.01 < −0.44
−1
−1
−1
−1
10
−0.43
3
2
1
x
1
2
3
4
5
6
y
10.0
7.5
5.0
2.5


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.66)
0.01
0.01
0.06
0.18
0.5
0.77
0 ^ 0.01
0.01 > 0.02
0.03 > 0.08
0.11 > 0.26
0.45 > 0.75
1.23 V 1.02
0.01
0.02
0.05
0.18
0.52
1.57
0.01
0.02
0.07
0.17
1.61
1.18
0 ^ 0.01
0.33 ^ 0.26
0.18 ^ 0.4
0.35 ^ 0.09
0.19 ^ 1.19
4.36 V 3.64
0
−0.47
−0.28
−0.21
0.41
6.8
−0.2
0 < −0.48
−1
−1
−1
−1
10
−0.33
3
2
1
x
1
2
3
4
5
6
y
10.0
7.5
5.0
2.5

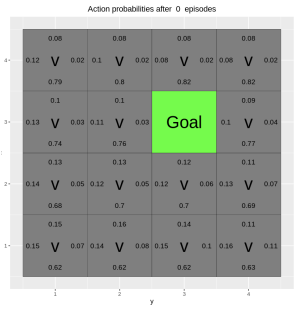
Investigate how the β parameter affects the learned policy.

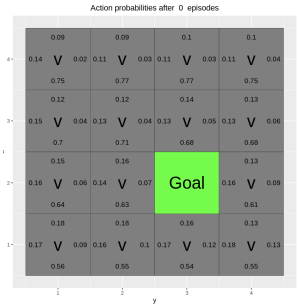
Here, β determines the probability that the agent will end perform another action than planned, making the agent risk averse and moving away from the negative states, since unexpected behavior might occur. Having a higher β worsen the model. This is due to it becoming more difficult for the agent to understand the task. Choosing an optimal action can lead to a negative reward.

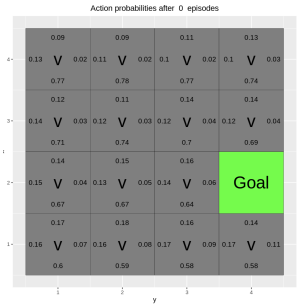
Reinforce

Environment D

In this task, we will use eight goal positions for training and, then, validate the learned policy on the remaining eight possible goal positions. The training and validation goal positions are stored in the lists train goals and val goals in the code in the file RL Lab2 Colab.ipynb. The results provided in the file correspond to running the REINFORCE algorithm for 5000 episodes with $\beta = 0$ and $\gamma = 0.95$. Each training episode uses a random goal position from train goals. The initial position for the episode is also chosen at random. When training is completed, the code validates the learned policy for the goal positions in val goals. This is done by with the help of the function vis prob, which shows the grid, goal position and learned policy. Note that each non-terminal tile has four values. These represent the action probabilities associated to the tile (state). Note also that each nonterminal tile has an arrow. This indicates the action with the largest probability for the tile (ties are broken at random). Finally, answer the following questions:

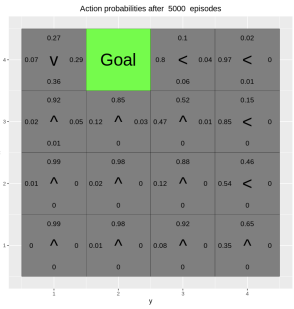

Action probabilities after 0 episodes
0.08 0.08 0.08 0.08
0.12 V 0.02 0.1 V 0.02 0.08 V 0.02 0.08 V 0.02
0.79 0.8 0.82 0.82
0.1 0.1 0.09
0.13 V 0.03 0.11 V 0.03 Goal 0.1 V 0.04
0.74 0.76 0.77
0.13 0.13 0.12 0.11
0.14 V 0.05 0.12 V 0.05 0.12 V 0.06 0.13 V 0.07
0.68 0.7 0.7 0.69
0.15 0.16 0.14 0.11
0.15 V 0.07 0.14 V 0.08 0.15 V 0.1 0.16 V 0.11
0.62 0.62 0.62 0.63
4 3 2 1
1 2 3 4
y

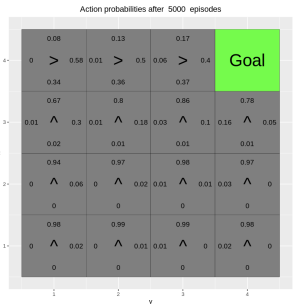

Action probabilities after 0 episodes
0.09 0.09 0.1 0.1
0.14 V 0.02 0.11 V 0.03 0.11 V 0.03 0.11 V 0.04
0.75 0.77 0.77 0.75
0.12 0.12 0.14 0.13
0.15 V 0.04 0.13 V 0.04 0.13 V 0.05 0.13 V 0.06
0.7 0.71 0.68 0.68
0.15 0.16 0.13
0.16 V 0.06 0.14 V 0.07 Goal 0.16 V 0.09
0.64 0.63 0.61
0.18 0.18 0.16 0.13
0.17 V 0.09 0.16 V 0.1 0.17 V 0.12 0.18 V 0.13
0.56 0.55 0.54 0.55
4 3 2 1
1 2 3 4
y

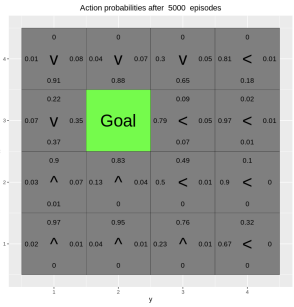

Action probabilities after 0 episodes
0.09 0.09 0.11 0.13
0.13 V 0.02 0.11 V 0.02 0.1 V 0.02 0.1 V 0.03
0.77 0.78 0.77 0.74
0.12 0.11 0.14 0.14
0.14 V 0.03 0.12 V 0.03 0.12 V 0.04 0.12 V 0.04
0.71 0.74 0.7 0.69
0.14 0.15 0.16
0.15 V 0.04 0.13 V 0.05 0.14 V 0.06 Goal
0.67 0.67 0.64
0.17 0.18 0.16 0.14
0.16 V 0.07 0.16 V 0.08 0.17 V 0.09 0.17 V 0.11
0.6 0.59 0.58 0.58
4 3 2 1
1 2 3 4
y

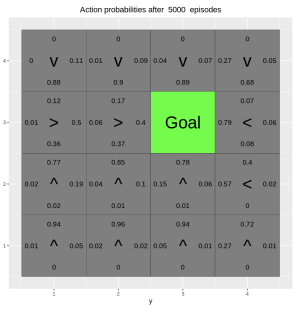

Action probabilities after 0 episodes
0.08 0.08 0.08 0.08
0.19 V 0.07 0.16 V 0.06 0.15 V 0.08 0.16 V 0.1
0.65 0.69 0.69 0.66
0.12 0.12 0.12 0.1
0.2 V 0.08 0.18 V 0.09 0.18 V 0.12 0.19 V 0.14
0.6 0.61 0.59 0.57
0.17 0.17 0.15 0.12
0.2 V 0.11 0.2 V 0.13 0.21 V 0.16 0.22 V 0.16
0.52 0.51 0.49 0.49
0.19 0.17 0.14
Goal 0.22 V 0.18 0.23 V 0.17 0.22 V 0.15
0.41 0.43 0.49
4 3 2 1
1 2 3 4
y

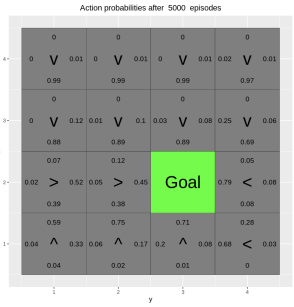

Action probabilities after 0 episodes
0.09 0.13 0.13 0.14
0.15 V 0.02 0.16 V 0.03 0.13 V 0.03 0.13 V 0.04
0.73 0.69 0.7 0.69
0.14 0.16 0.18 0.17
0.18 V 0.04 0.16 V 0.04 0.15 V 0.05 0.16 V 0.05
0.65 0.63 0.63 0.62
0.18 0.19 0.19 0.17
0.19 V 0.06 0.17 V 0.06 0.18 V 0.07 0.19 V 0.08
0.57 0.58 0.56 0.56
0.21 0.21 0.19
0.2 V 0.08 0.2 V 0.09 0.2 V 0.1 Goal
0.5 0.5 0.51
4 3 2 1
1 2 3 4
y

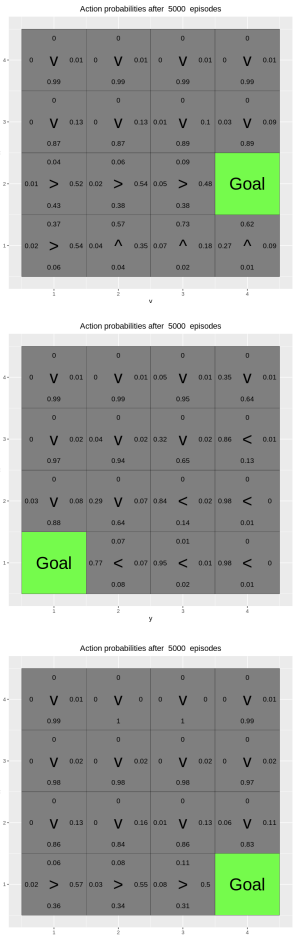

Action probabilities after 5000 episodes
Goal
y


Action probabilities after 5000 episodes
Goal
y


Action probabilities after 5000 episodes
Goal
y


Action probabilities after 5000 episodes
Goal
y


Action probabilities after 5000 episodes
Goal
y

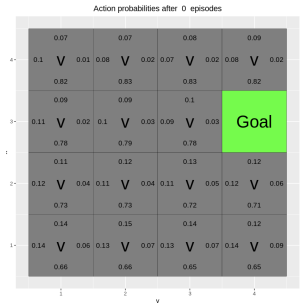


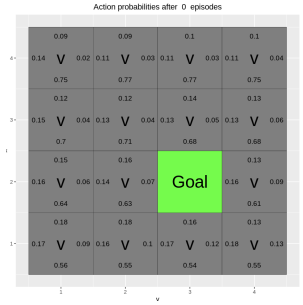
Has the agent learned a good policy? Why / Why not ? Yes, by reading from the graphs were 5000 episodes have been executed, he arrows in each state point towards the goal position, indicating the agent has learned to find the goal.

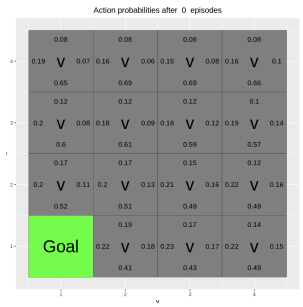
Could you have used the Q-learning algorithm to solve this task ? Yes, for Q-learning to work in an environment with a random goal, the state needs to include information about the goal, which it does in this case. Therefore, Q-learning can learn different state-action values depending on the goal's position. However, the state space becomes significantly larger and the conversion is slower compared to REINFORCE.

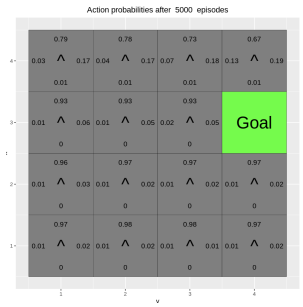
Environment E

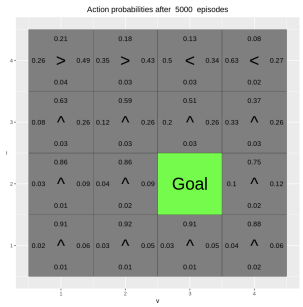
In this task, the goals for training are all from the top row of the grid. The validation goals are three positions from the rows below. To solve this task, simply study the code and results provided in the file RL Lab2 Colab.ipynb and answer the following questions:

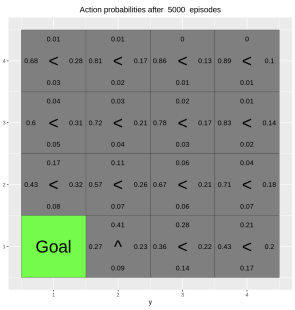

Action probabilities after 0 episodes
0.07 0.07 0.08 0.09
0.1 V 0.01 0.08 V 0.02 0.07 V 0.02 0.08 V 0.02
0.82 0.83 0.83 0.82
0.09 0.09 0.1
0.11 V 0.02 0.1 V 0.03 0.09 V 0.03 Goal
0.78 0.79 0.78
0.11 0.12 0.13 0.12
0.12 V 0.04 0.11 V 0.04 0.11 V 0.05 0.12 V 0.06
0.73 0.73 0.72 0.71
0.14 0.15 0.14 0.12
0.14 V 0.06 0.13 V 0.07 0.13 V 0.07 0.14 V 0.09
0.66 0.66 0.65 0.65
v


Action probabilities after 0 episodes
0.09 0.09 0.1 0.1
0.14 V 0.02 0.11 V 0.03 0.11 V 0.03 0.11 V 0.04
0.75 0.77 0.77 0.75
0.12 0.12 0.14 0.13
0.15 V 0.04 0.13 V 0.04 0.13 V 0.05 0.13 V 0.06
0.7 0.71 0.68 0.68
0.15 0.16 0.13
0.16 V 0.06 0.14 V 0.07 Goal 0.16 V 0.09
0.64 0.63 0.61
0.18 0.18 0.16 0.13
0.17 V 0.09 0.16 V 0.1 0.17 V 0.12 0.18 V 0.13
0.56 0.55 0.54 0.55
v


Action probabilities after 0 episodes
0.08 0.08 0.08 0.08
0.19 V 0.07 0.16 V 0.06 0.15 V 0.08 0.16 V 0.1
0.65 0.69 0.69 0.66
0.12 0.12 0.12 0.1
0.2 V 0.08 0.18 V 0.09 0.18 V 0.12 0.19 V 0.14
0.6 0.61 0.59 0.57
0.17 0.17 0.15 0.12
0.2 V 0.11 0.2 V 0.13 0.21 V 0.16 0.22 V 0.16
0.52 0.51 0.49 0.49
0.19 0.17 0.14
Goal 0.22 V 0.18 0.23 V 0.17 0.22 V 0.15
0.41 0.43 0.49
v


Action probabilities after 5000 episodes
0.79 0.78 0.73 0.67
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
0.01 ^ 0.06 0.01 ^ 0.05 0.02 ^ 0.05 Goal
0 0 0
0.96 0.97 0.97 0.97
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
0.01 ^ 0.02 0.01 ^ 0.02 0.01 ^ 0.01 0.01 ^ 0.02
0 0 0 0
v


Action probabilities after 5000 episodes
0.21 0.18 0.13 0.08
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
0.08 ^ 0.26 0.12 ^ 0.26 0.2 ^ 0.26 0.33 ^ 0.26
0.03 0.03 0.03 0.03
0.86 0.86 0.75
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
0.02 ^ 0.06 0.03 ^ 0.05 0.03 ^ 0.05 0.04 ^ 0.06
0.01 0.01 0.01 0.02
v



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

No, since the training data learned the model to find the goal at the top of the grid, it does not understand where to find the goal. Training data needs to be more generalized in order for the model to understand what the true goal is.

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

They differ in terms of how accurate the model is on finding the goal, the reason is becuase they have been trained differently, as described in the previous question.