# Advanced Mahcine Learning: Reinforcement Learning Lab - Individual report

Nicolas Taba (nicta839)

03/10/2021

## Q learning

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

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

```r
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)))

}
```

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


}
```

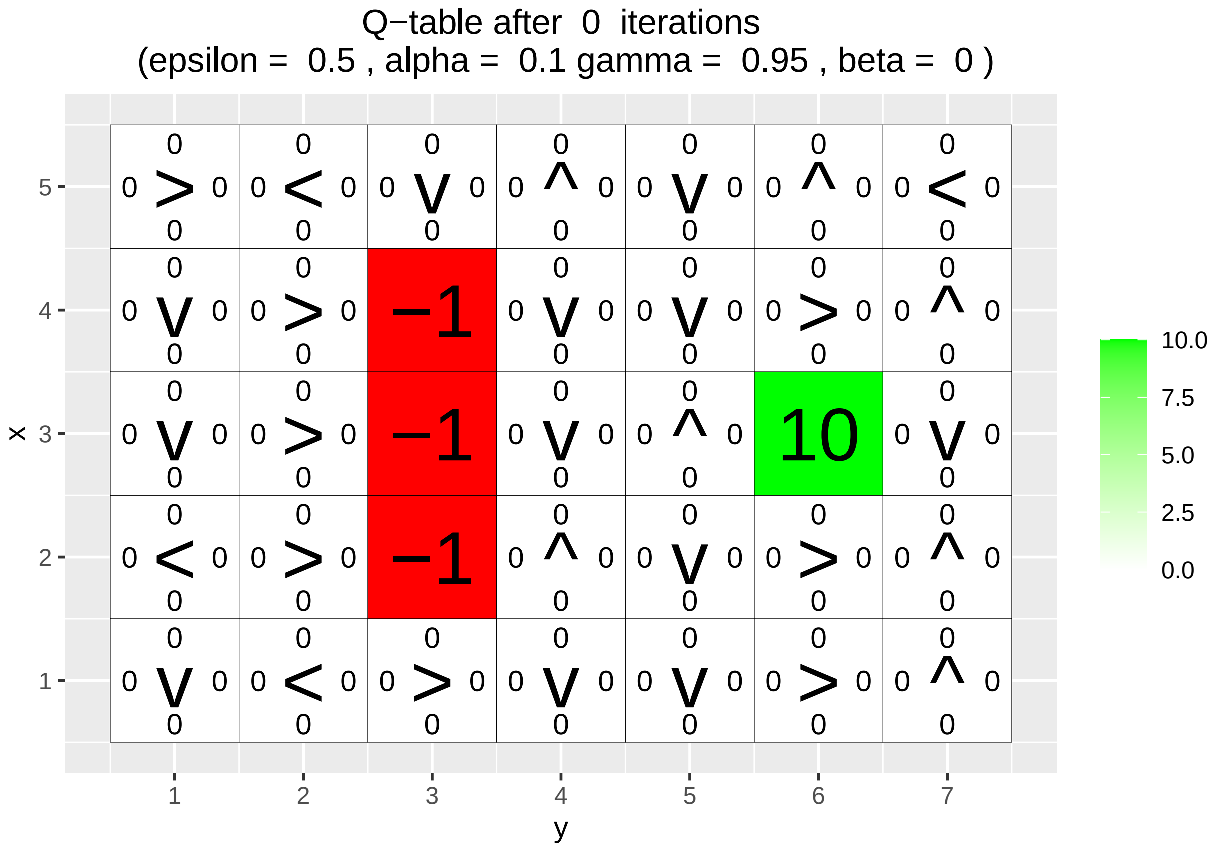
```r
EpsilonGreedyPolicy <- function(x, y, epsilon){

  # 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}.

  if(runif(1) < epsilon){
    eps_action <- sample(c(1:4), 1)
  }else{
    eps_action <- GreedyPolicy(x, y)
  }
  return(eps_action)
}
```

```r
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)
}
```

```r
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 <<-.

  sx <- start_state[1]
  sy <- start_state[2]
  episode_correction <- 0

  repeat{
    # Follow policy, execute action, get reward.
    action <- EpsilonGreedyPolicy(sx, sy, epsilon)
    State <- transition_model(sx, sy, action, beta)
    reward <- reward_map[State[1], State[2]]
    correction <- reward + (gamma * max(q_table[State[1], State[2], ])) - q_table[sx, sy, action]
    episode_correction <- episode_correction + correction
    # Q-table update.
    q_table[sx, sy, action] <<- q_table[sx, sy, action] + (alpha * correction)
    sx <- State[1]
    sy <- State[2]

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

}
```

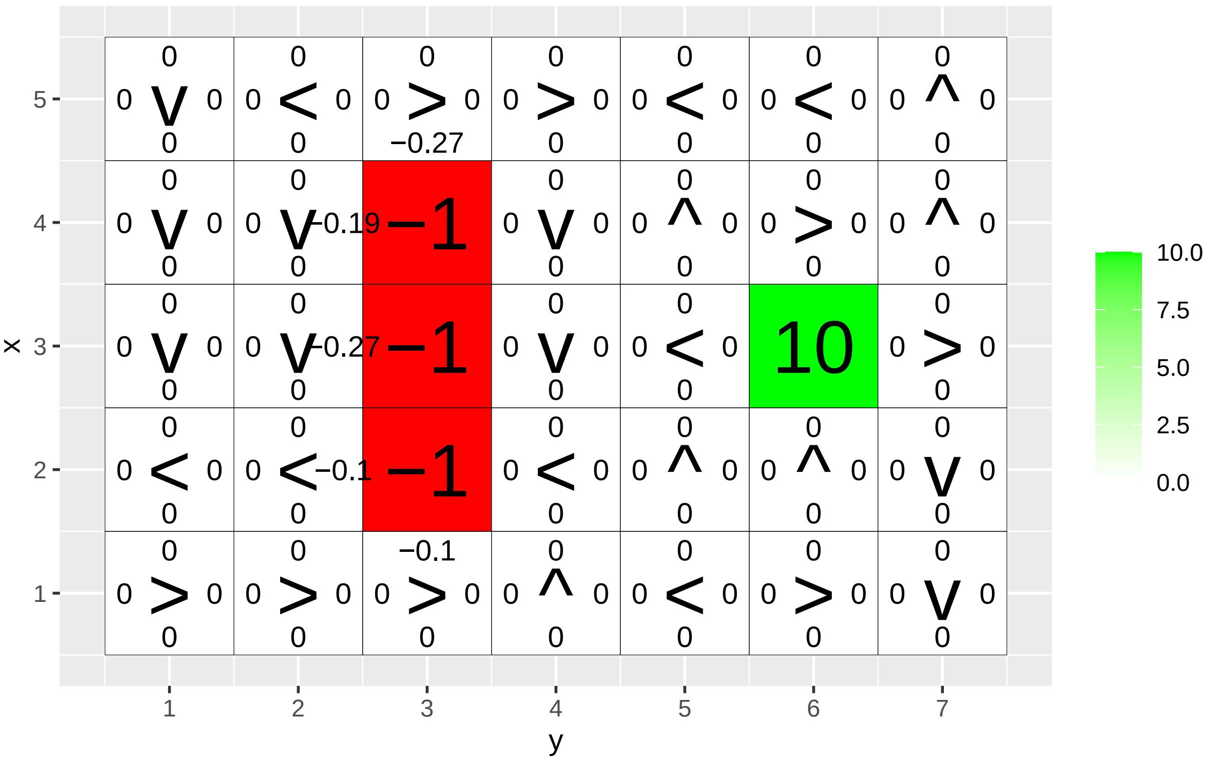
#### Environment A

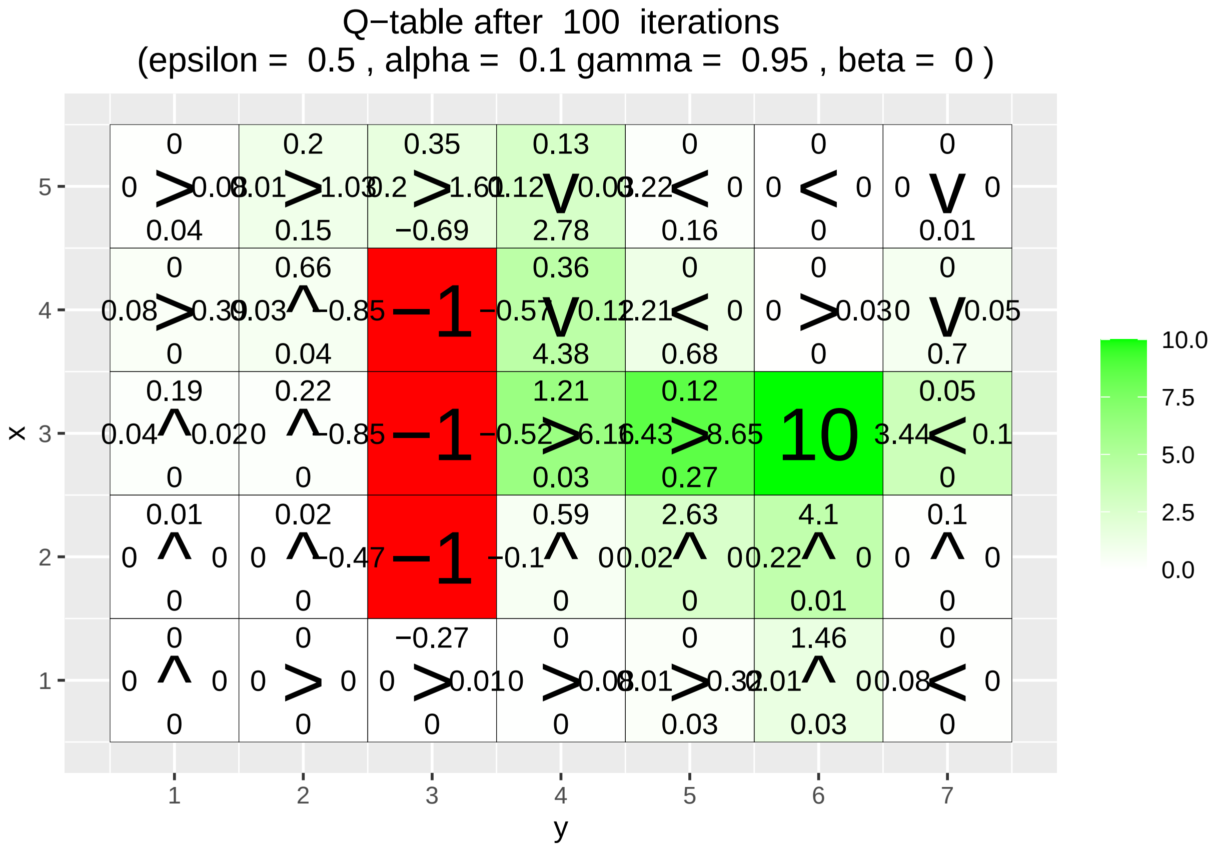
```
# Environment A (learning)

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()
```

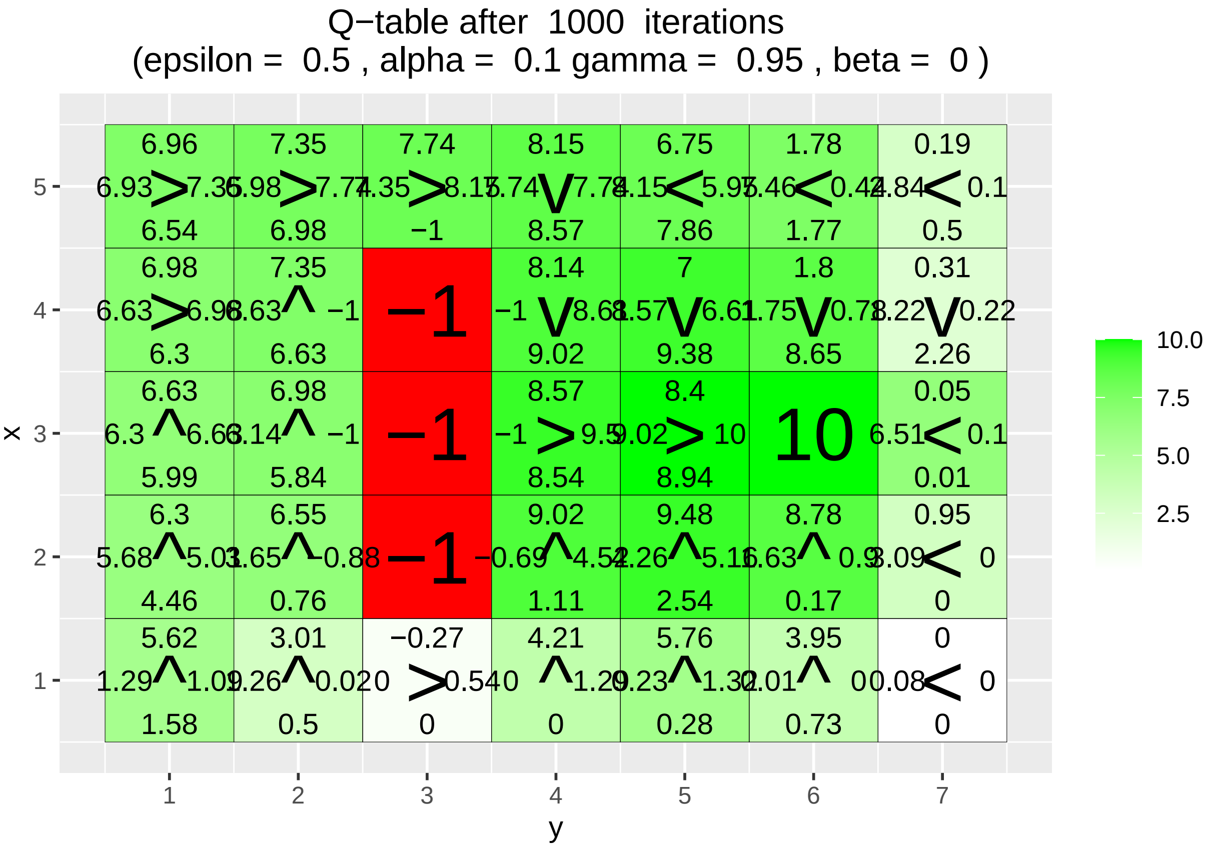


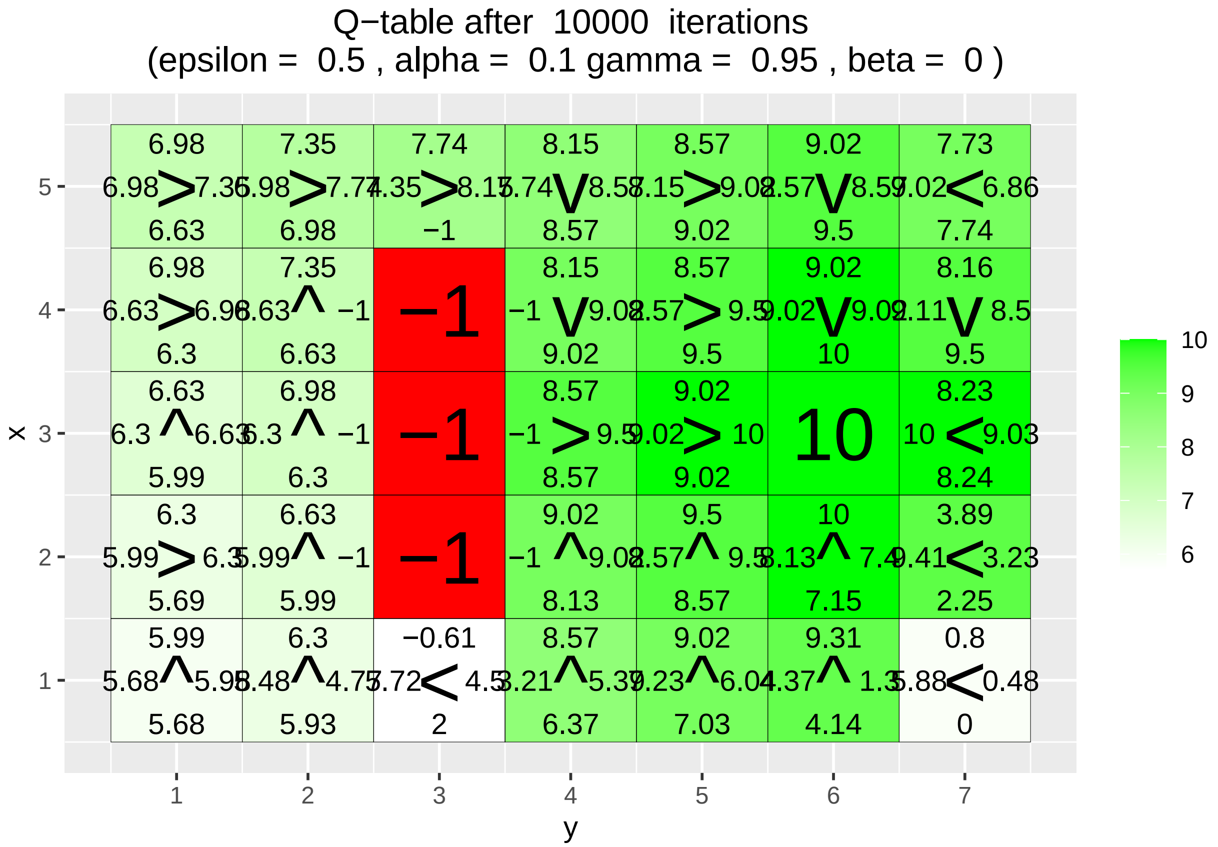
```r
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))

  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 )




Q−table after 100 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0 )
0
0 > 0.08
0.04
0.2
0.01 > 1.03
0.15
0.35
0.2 > 1.6
−0.69
0.13
0.12 V 0.03
2.78
0
0.22 < 0
0.16
0
0 < 0
0
0
0 V 0
0.01
0
0.08 > 0.39
0
0.66
0.03 ^ −0.85
0.04
−1
0.36
−0.57 V 0.12
4.38
0
0.21 < 0
0.68
0
0 > 0.03
0
0
0 V 0.05
0.7
0.19
0.04 ^ 0.02
0
0.22
0 ^ −0.85
0
−1
1.21
−0.52 > 6.16
0.03
0.12
6.43 > 8.65
0.27
10
0.05
3.44 < 0.1
0
0.01
0 ^ 0
0
0.02
0 ^ −0.47
0
−1
0.59
−0.1 ^ 0
0
2.63
0.02 ^ 0
0
4.1
0.22 ^ 0
0.01
0.1
0 ^ 0
0
0
0 ^ 0
0
0
0 > 0
0
−0.27
0 > 0.01
0
0
0 > 0.08
0
0
0.01 > 0.32
0.03
1.46
0.01 ^ 0
0.03
0
0.08 < 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 )
6.96 7.35 7.74 8.15 6.75 1.78 0.19
6.93 > 7.36.98 > 7.74.35 > 8.15.74 V 7.74.15 < 5.95.46 < 0.44.84 < 0.1
6.54 6.98 −1 8.57 7.86 1.77 0.5
6.98 7.35 8.14 7 1.8 0.31
6.63 > 6.96.63 ^ −1 −1 −1 V 8.68.57 V 6.61.75 V 0.78.22 V 0.22
6.3 6.63 9.02 9.38 8.65 2.26
6.63 6.98 8.57 8.4 0.05
6.3 ^ 6.66.14 ^ −1 −1 −1 > 9.59.02 > 10 10 6.51 < 0.1
5.99 5.84 8.54 8.94 0.01
6.3 6.55 9.02 9.48 8.78 0.95
5.68 ^ 5.03.65 ^ −0.88 −1 −0.69 ^ 4.54.26 ^ 5.16.63 ^ 0.93.09 < 0
4.46 0.76 1.11 2.54 0.17 0
5.62 3.01 −0.27 4.21 5.76 3.95 0
1.29 ^ 1.09.26 ^ 0.020 > 0.540 ^ 1.29.23 ^ 1.32.01 ^ 00.08 < 0
1.58 0.5 0 0 0.28 0.73 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:** The agent has learned mostly to go around the reward penalty zone.

**Is the final greedy policy (after 10000 episodes) optimal for all states, i.e. not only for the initial state? Why/ Why not?** The final greedy policy is optimal because we do not observe any loops and all paths lead to the reward state and all paths avoid the penalty zone. However, we will note that alternate paths are not found. This in turn means that if our optimality condition also encompassed the shortest path, this condition would not be fulfilled.

**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?** As stated above, there are other routes that are also valid solutions. In order to find alternate routes, we could allow the exploration rate to be higher in the beginning and have it decrease as the number of episodes increases.

One such approach uses:

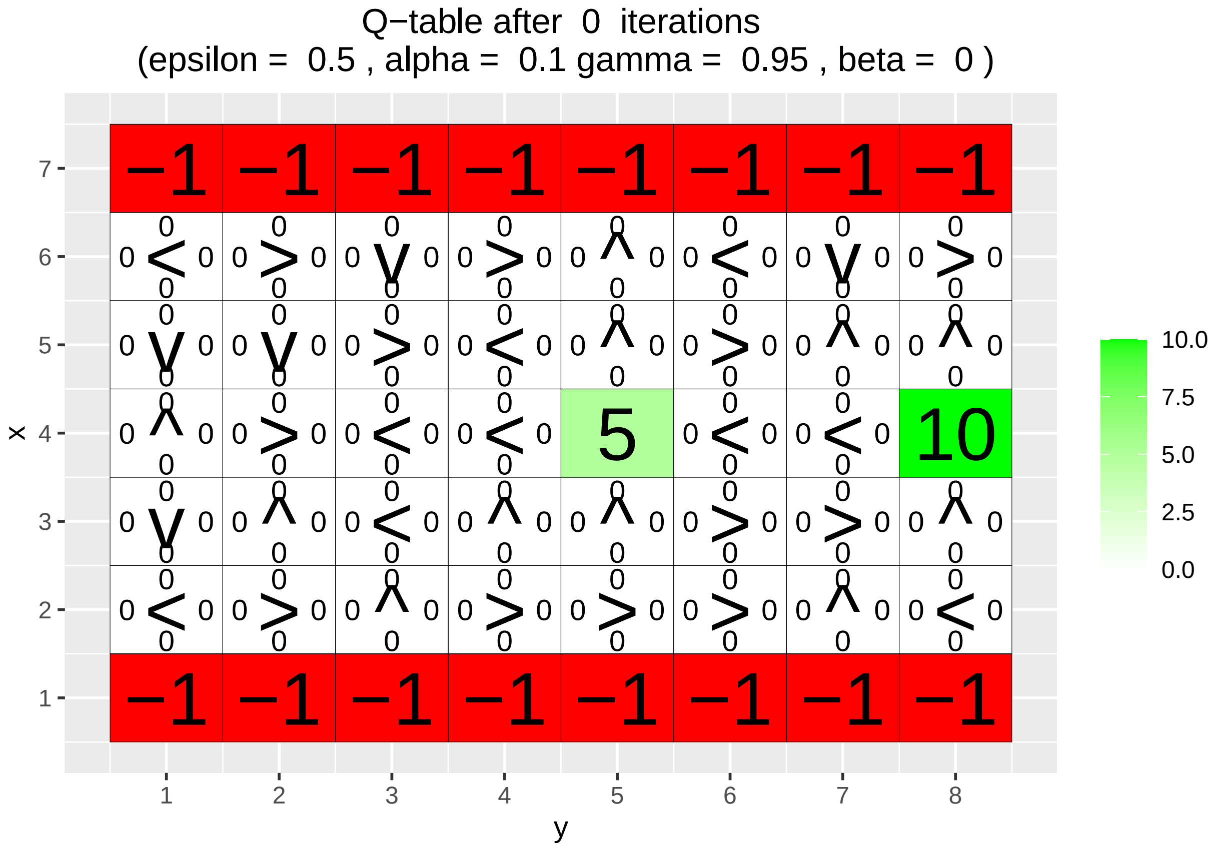
$$\epsilon = 1 - \frac{episode.number}{Episodes}$$

Which sees a decrease in the value of epsilon as we train our agent in successive episodes. This dynamic approach to the value of epsilon allows the agent to trade-off between exploration and exploitation during learning. For sufficiently high values of epsilon, we should find those alternate routes. Given our starting position, we should expect symmetric paths along the x=3 line.

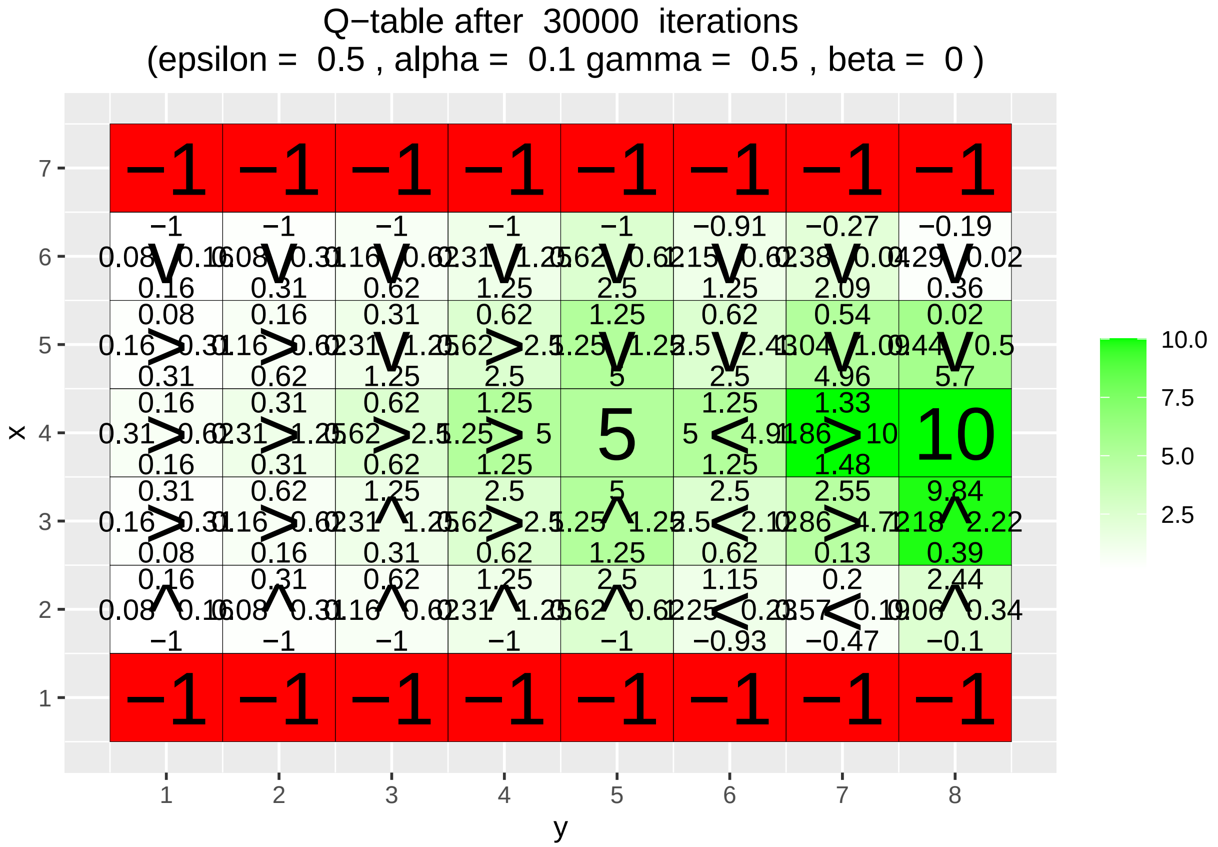
Another approach to finding other possible paths would be to randomize the starting state such that the visitation frequency of each state is uniformly distributed. This would in turn allow the Q-table to be updated more often for states that would otherwise not be visited often. Finally, we could penalize the agent for not completing the task and would thus encourage the agent to find the shortest path from the starting position.

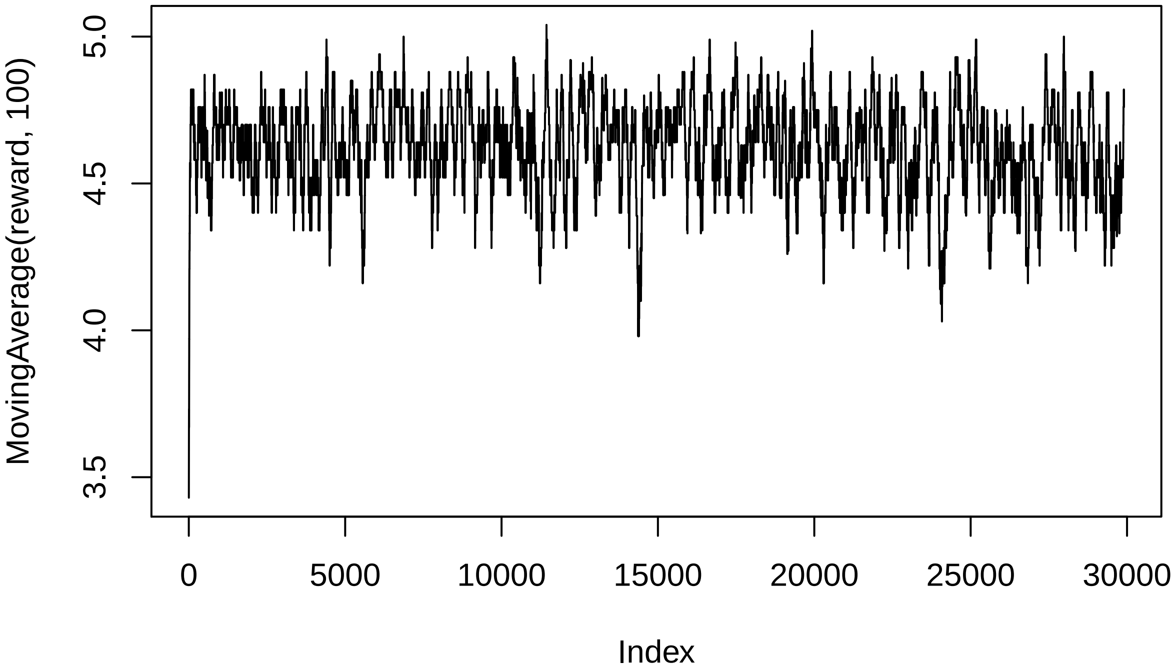
### Environment B

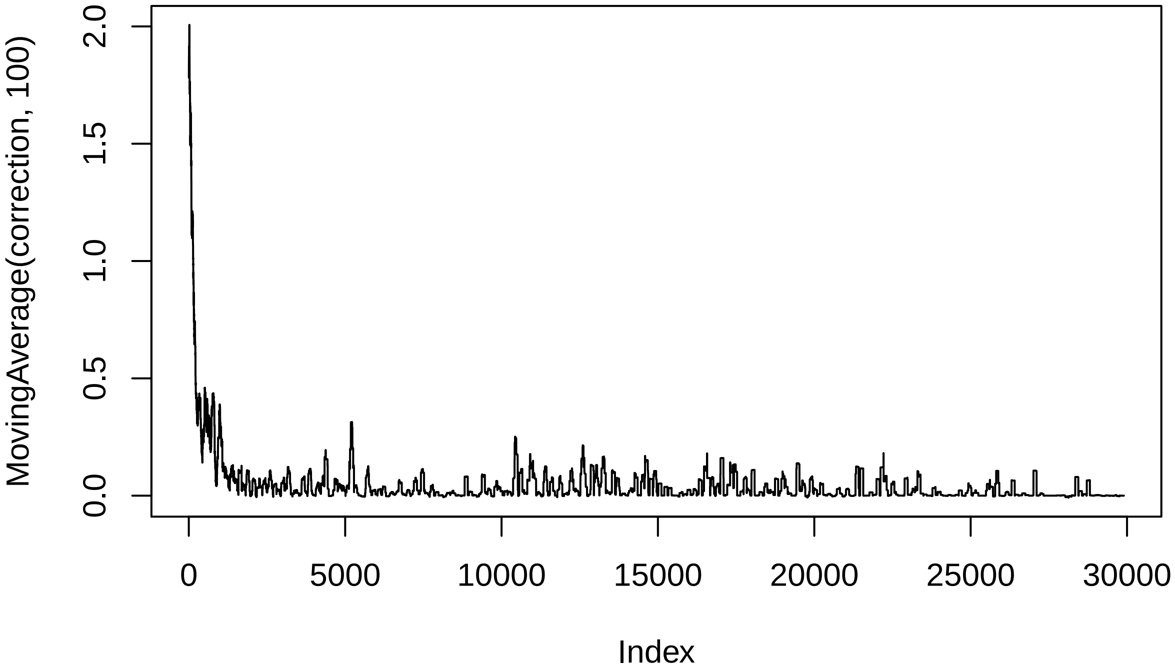
```r
# Environment B (the effect of epsilon and gamma)

H <- 7
W <- 8

reward_map <- matrix(0, nrow = H, ncol = W)
reward_map[1,] <- -1
reward_map[7,] <- -1
reward_map[4,5] <- 5
reward_map[4,8] <- 10

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

vis_environment()
```

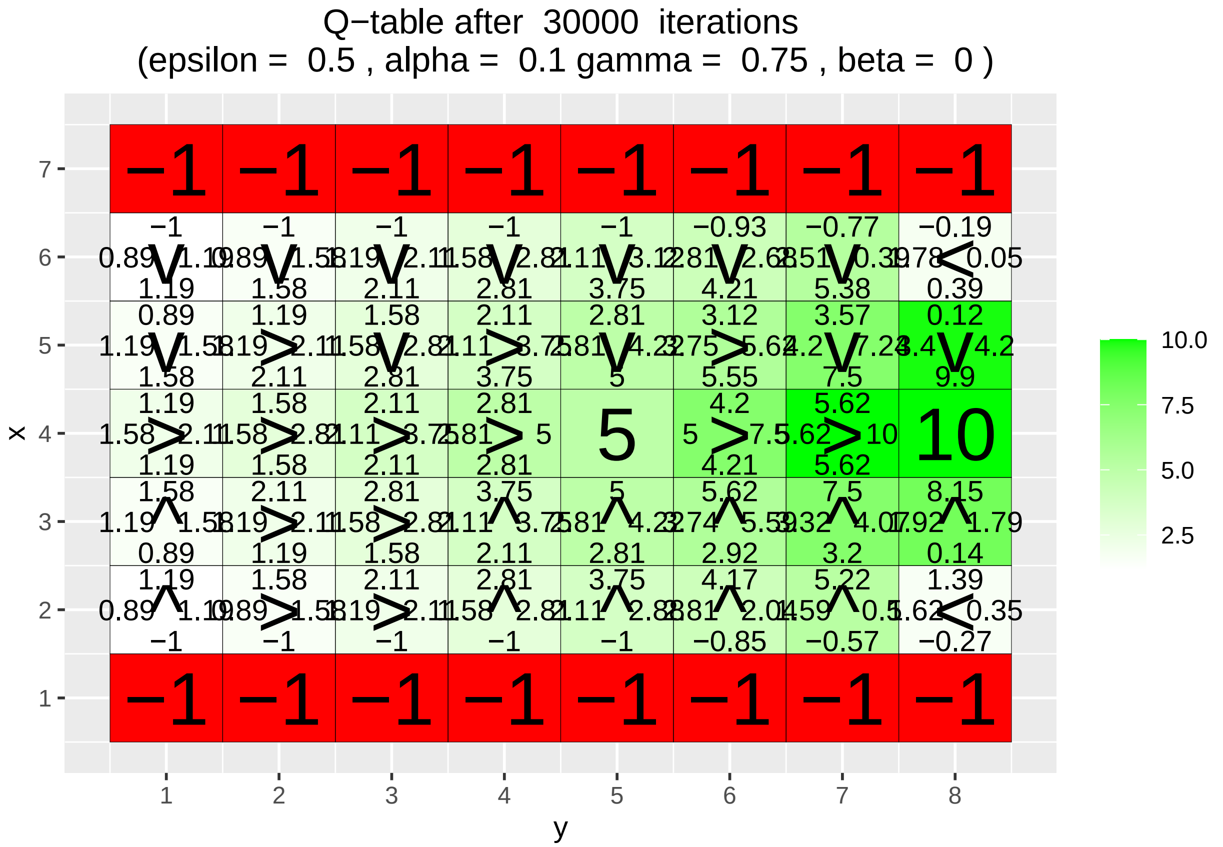


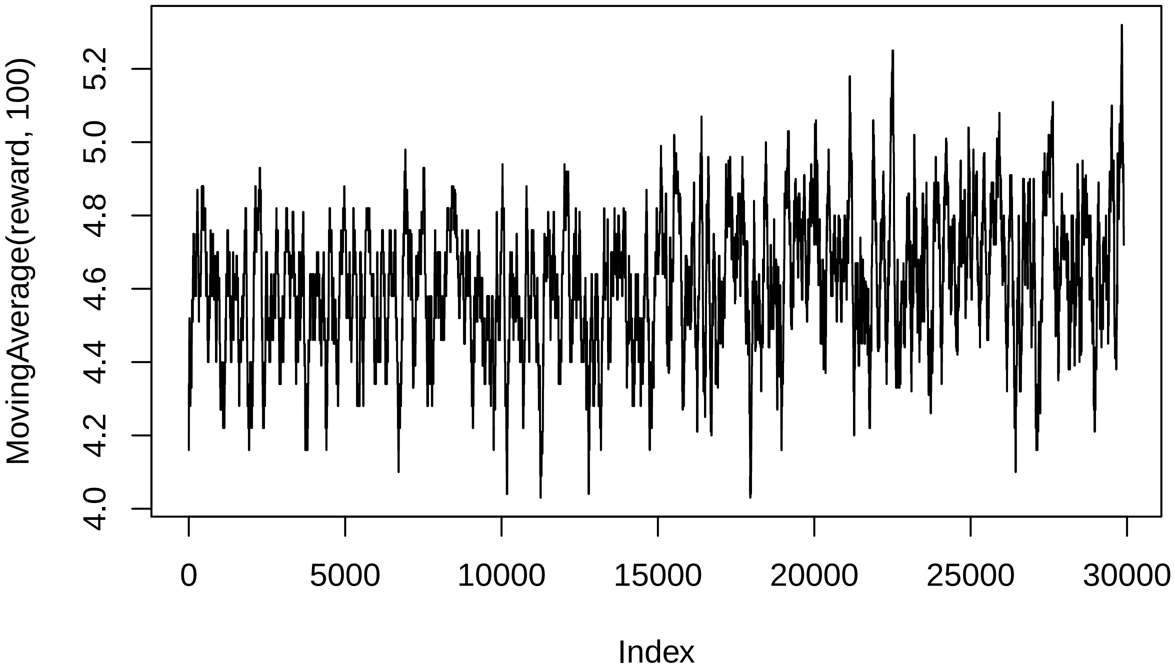
```r
MovingAverage <- function(x, n){

  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")
}
```

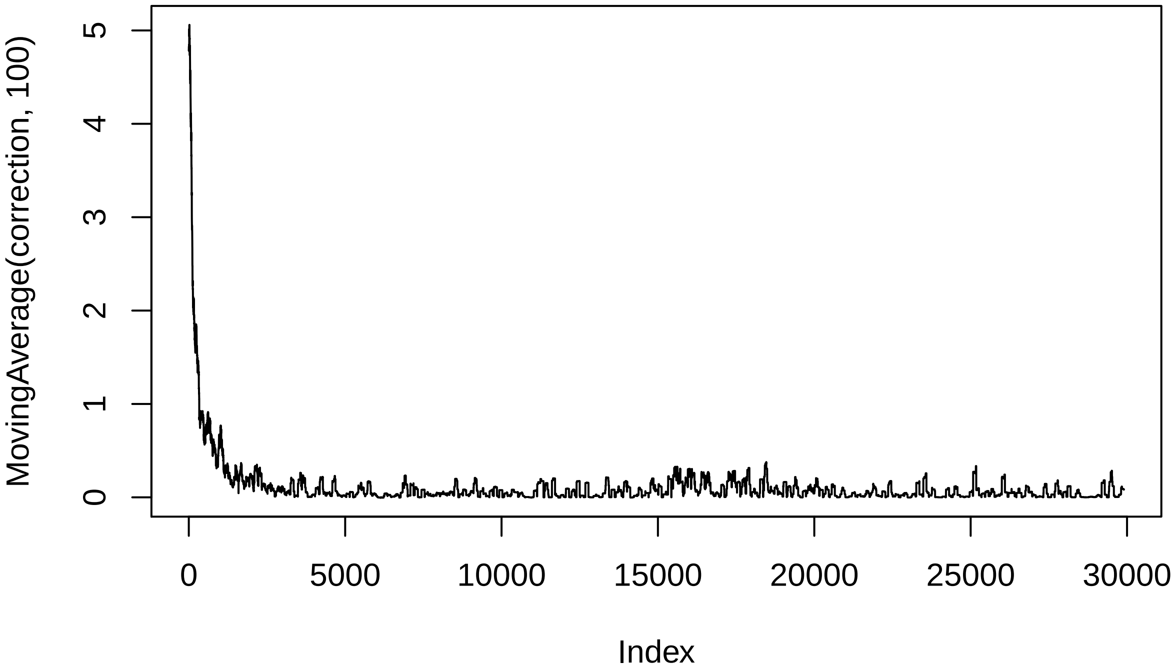



MovingAverage(reward, 100)
3.5
4.0
4.5
5.0
0
5000
10000
15000
20000
25000
30000
Index

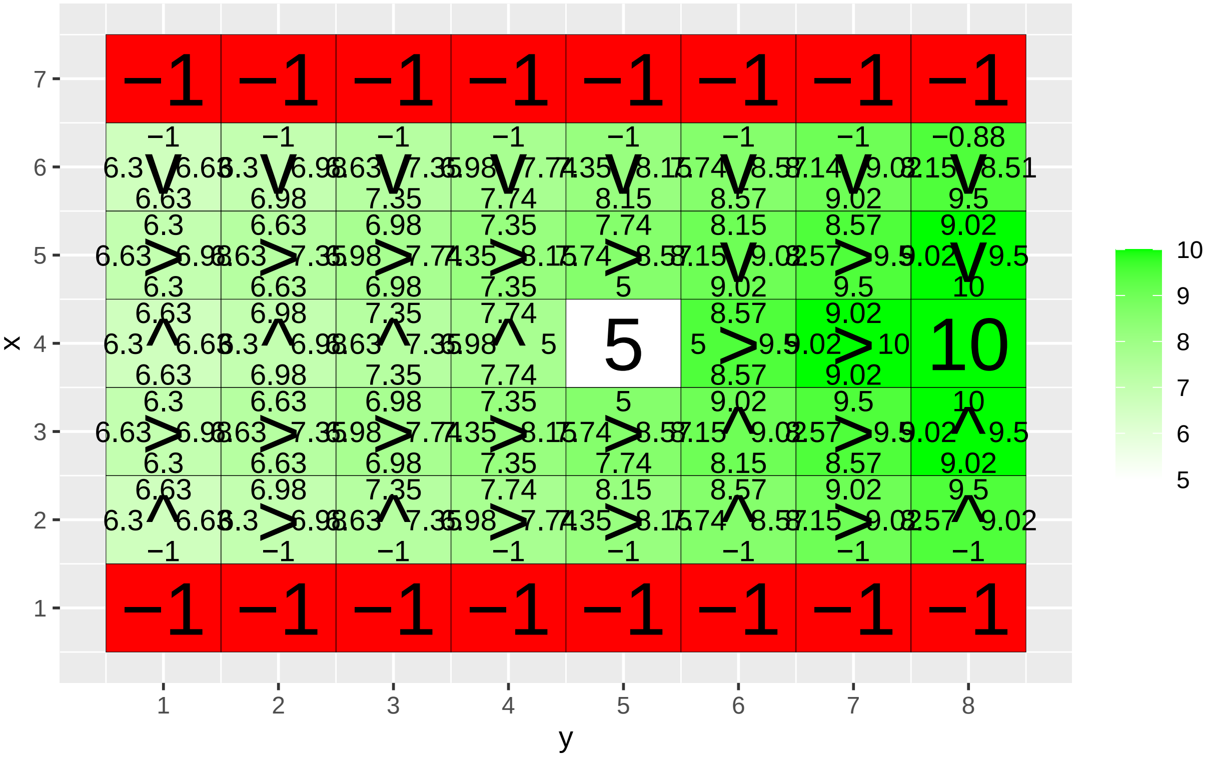

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index

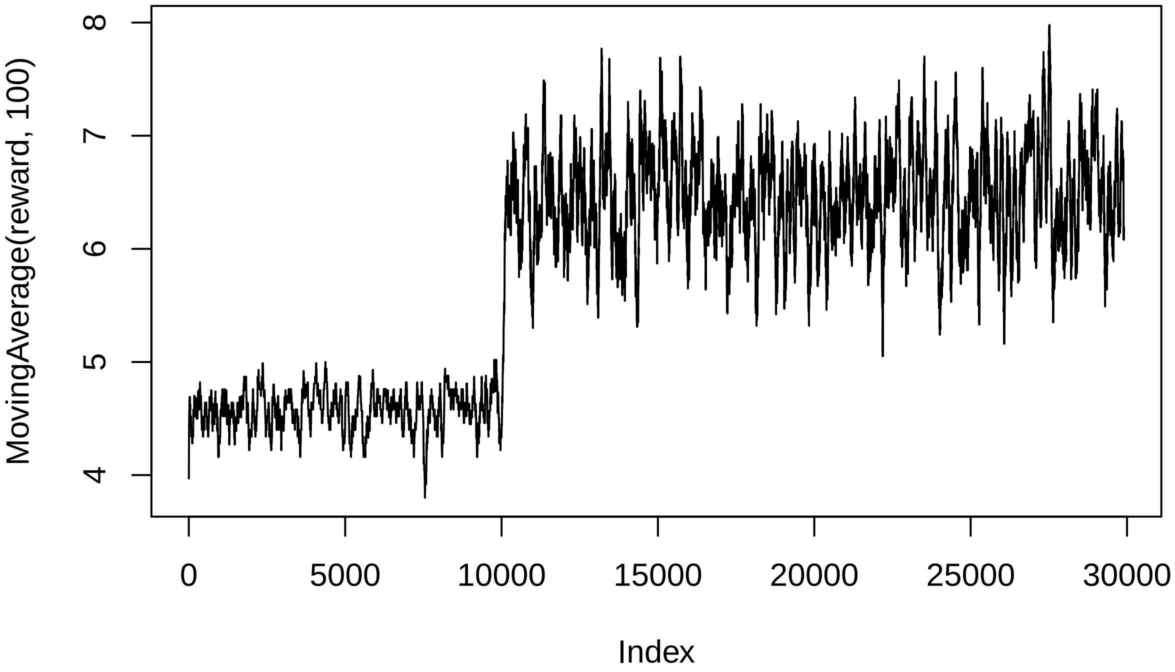

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0 )
−1 −1 −1 −1 −1 −1 −1 −1
−1 −1 −1 −1 −1 −0.93 −0.77 −0.19
0.89 V 1.19 0.89 V 1.58 1.19 V 2.11 1.58 V 2.81 2.11 V 3.12 2.81 V 2.68 3.51 V 0.39 0.78 < 0.05
1.19 1.58 2.11 2.81 3.75 4.21 5.38 0.39
0.89 1.19 1.58 2.11 2.81 3.12 3.57 0.12
1.19 V 1.58 1.19 > 2.11 1.58 V 2.81 2.11 > 3.75 2.81 V 4.23 2.75 > 5.64 4.2 V 7.24 3.4 V 4.2
1.58 2.11 2.81 3.75 5 5.55 7.5 9.9
1.19 1.58 2.11 2.81 4.2 5.62
1.58 > 2.11 1.58 > 2.81 2.11 > 3.75 2.81 > 5 5 5 > 7.5 5.62 > 10 10
1.19 1.58 2.11 2.81 4.21 5.62
1.58 2.11 2.81 3.75 5 5.62 7.5 8.15
1.19 ^ 1.58 1.19 > 2.11 1.58 > 2.81 2.11 ^ 3.75 2.81 ^ 4.23 2.74 ^ 5.59 3.32 ^ 4.07 7.92 ^ 1.79
0.89 1.19 1.58 2.11 2.81 2.92 3.2 0.14
1.19 1.58 2.11 2.81 3.75 4.17 5.22 1.39
0.89 ^ 1.19 0.89 > 1.58 1.19 > 2.11 1.58 ^ 2.81 2.11 ^ 2.83 2.81 ^ 2.04 1.59 ^ 0.5 1.62 < 0.35
−1 −1 −1 −1 −1 −0.85 −0.57 −0.27
−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

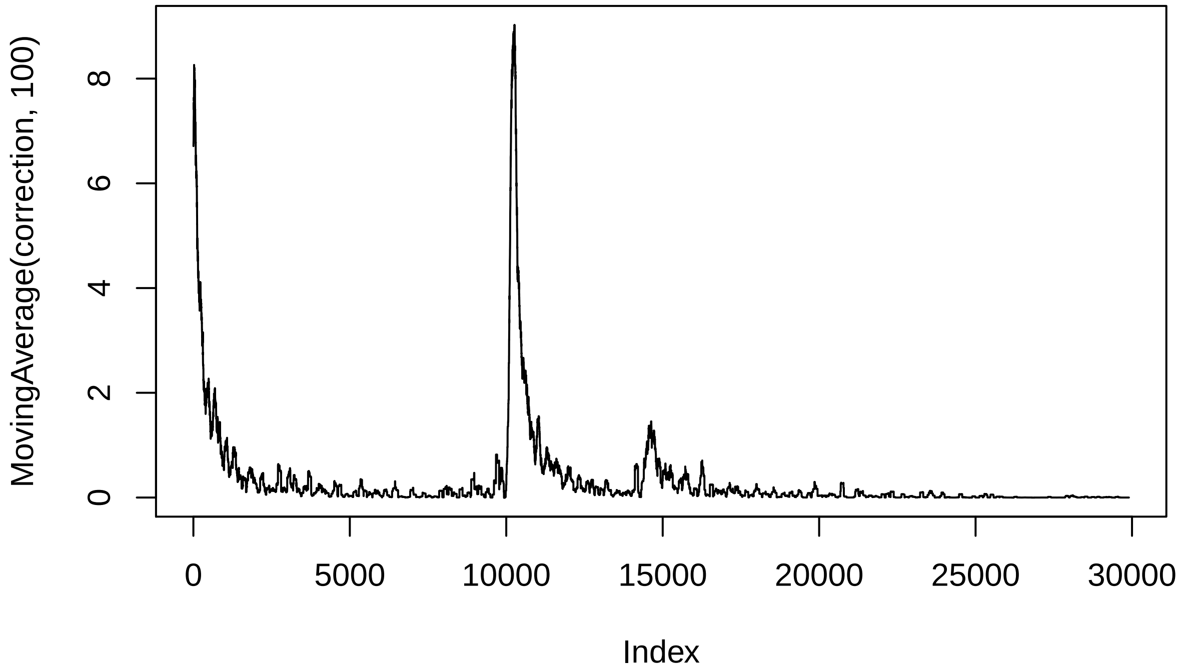

MovingAverage(reward, 100)
4.0
4.2
4.4
4.6
4.8
5.0
5.2
0
5000
10000
15000
20000
25000
30000
Index


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

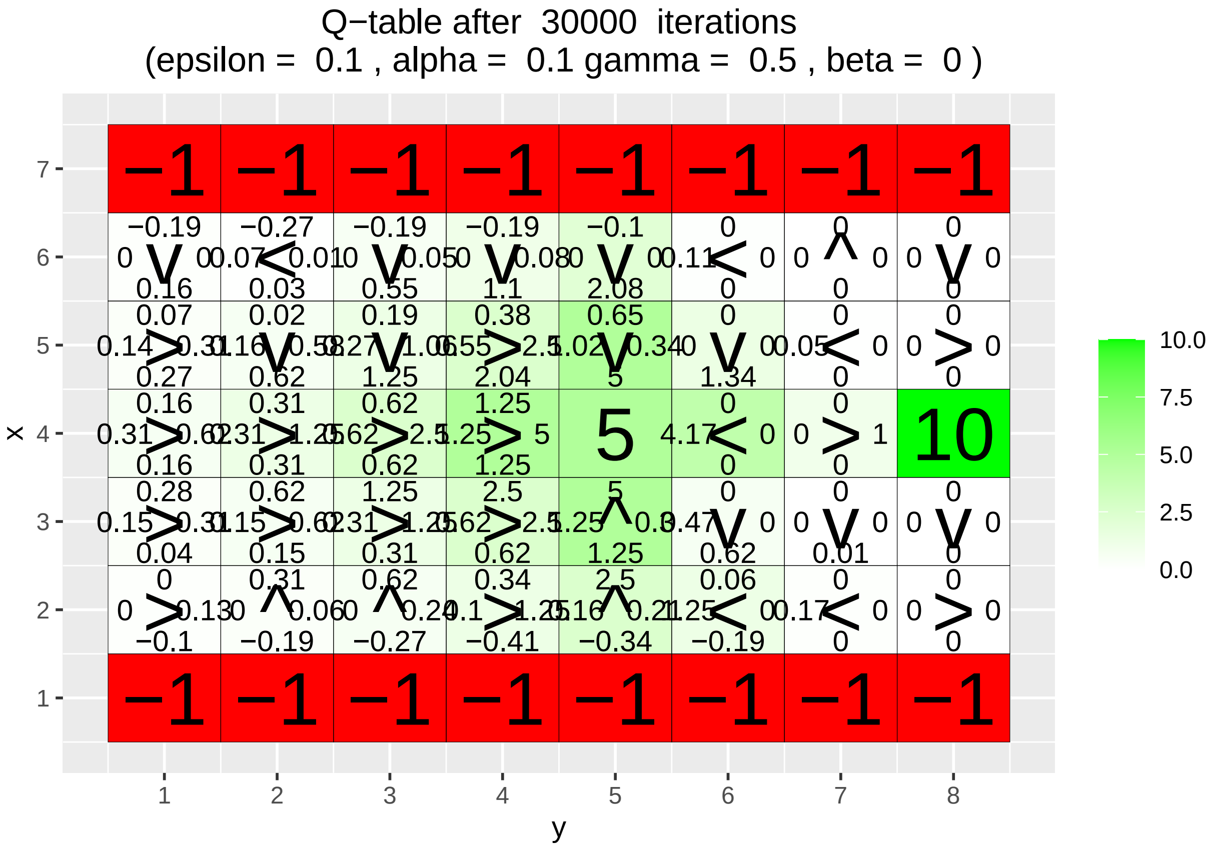
Q−table after  30000  iterations
(epsilon =  0.5 , alpha =  0.1 gamma =  0.95 , beta =  0 )



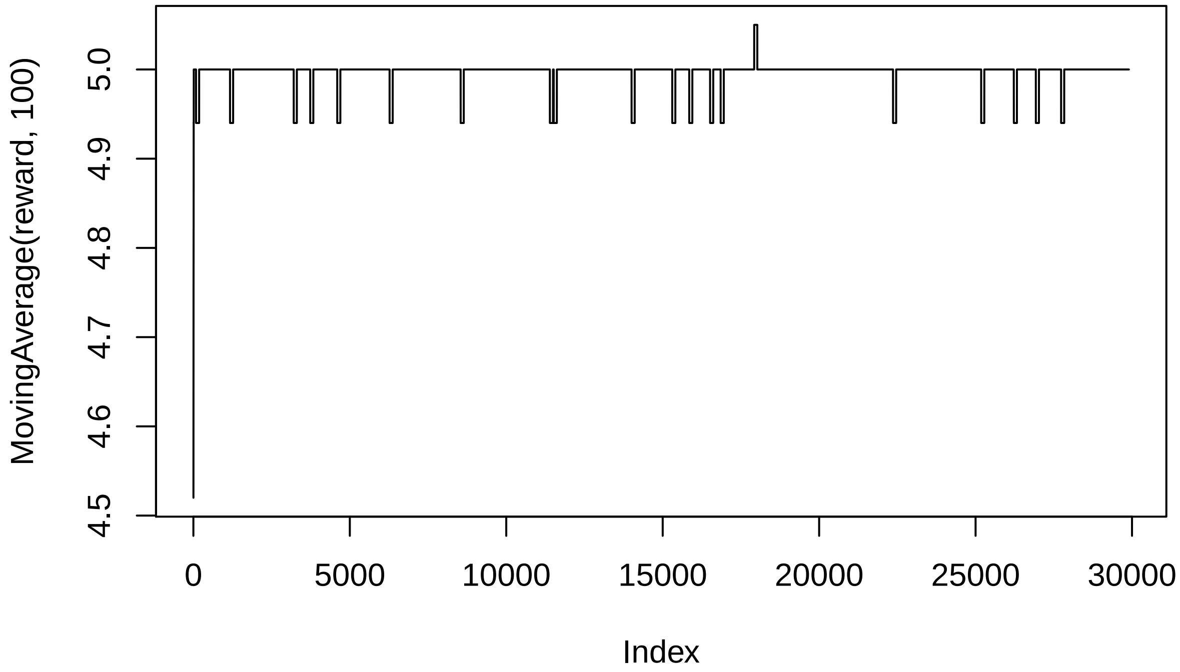

MovingAverage(reward, 100)
4
5
6
7
8
0
5000
10000
15000
20000
25000
30000
Index

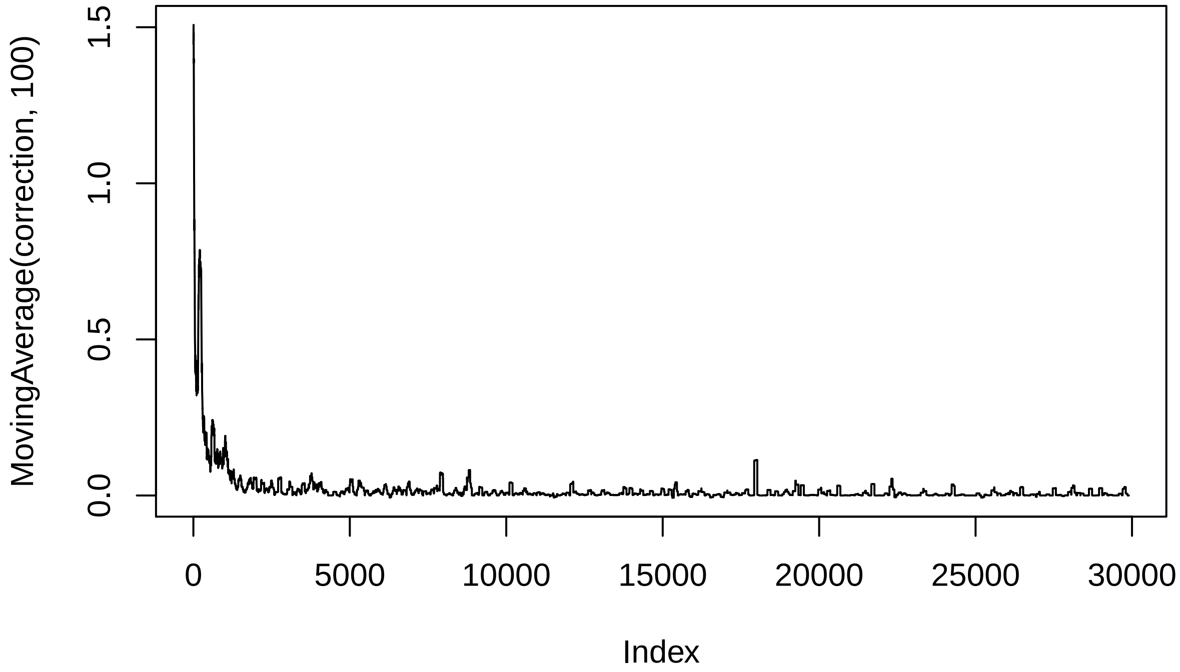


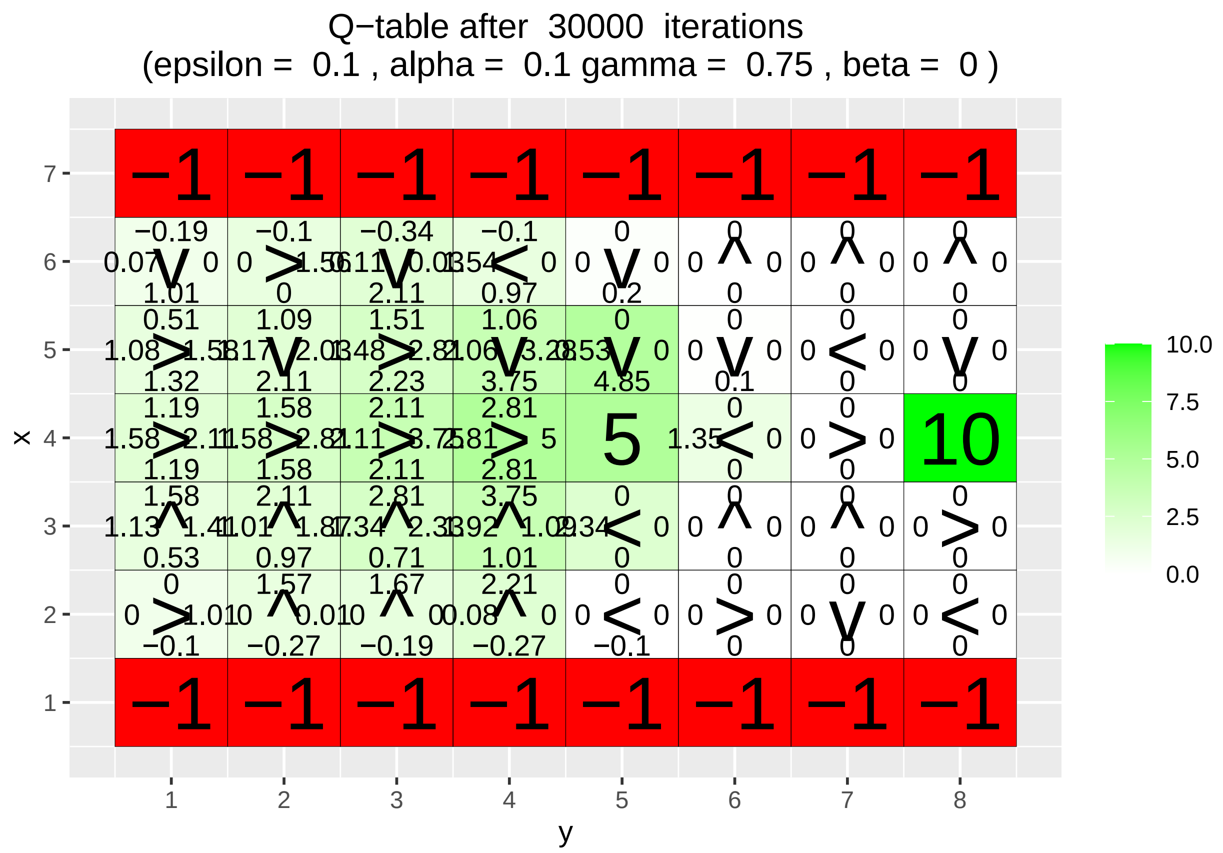
```r
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  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")
}
```

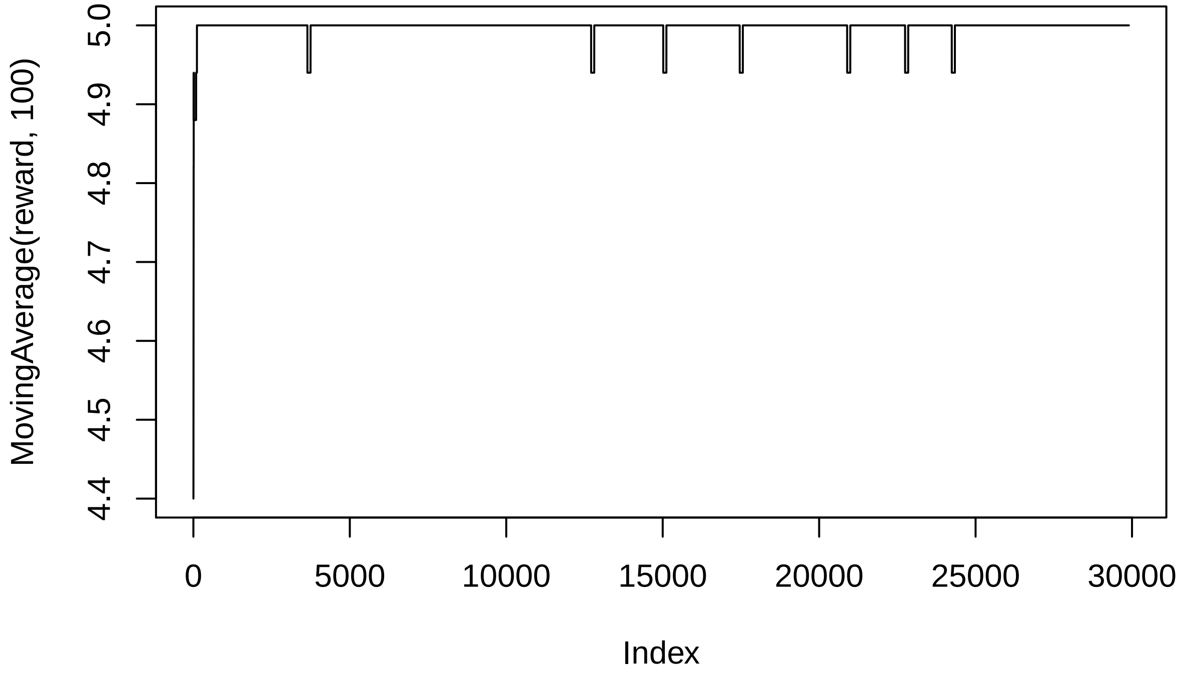


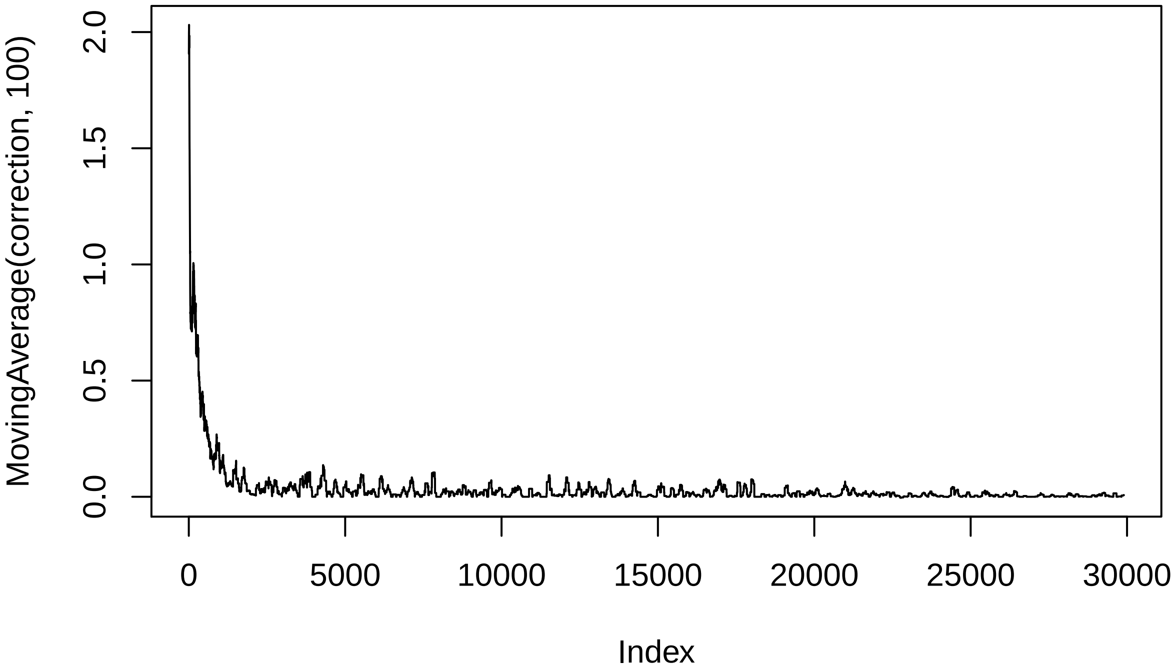
Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0 )

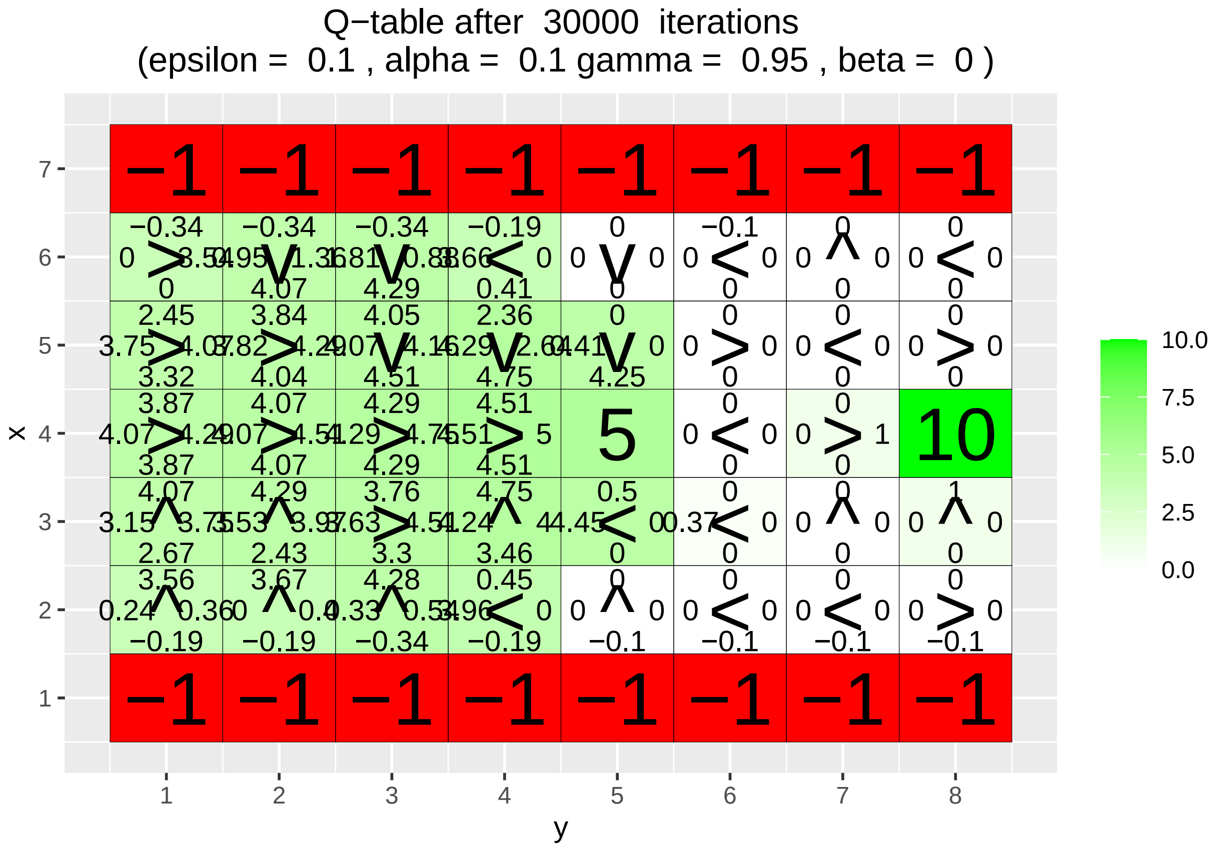

MovingAverage(reward, 100)
4.5
4.6
4.7
4.8
4.9
5.0
0
5000
10000
15000
20000
25000
30000
Index

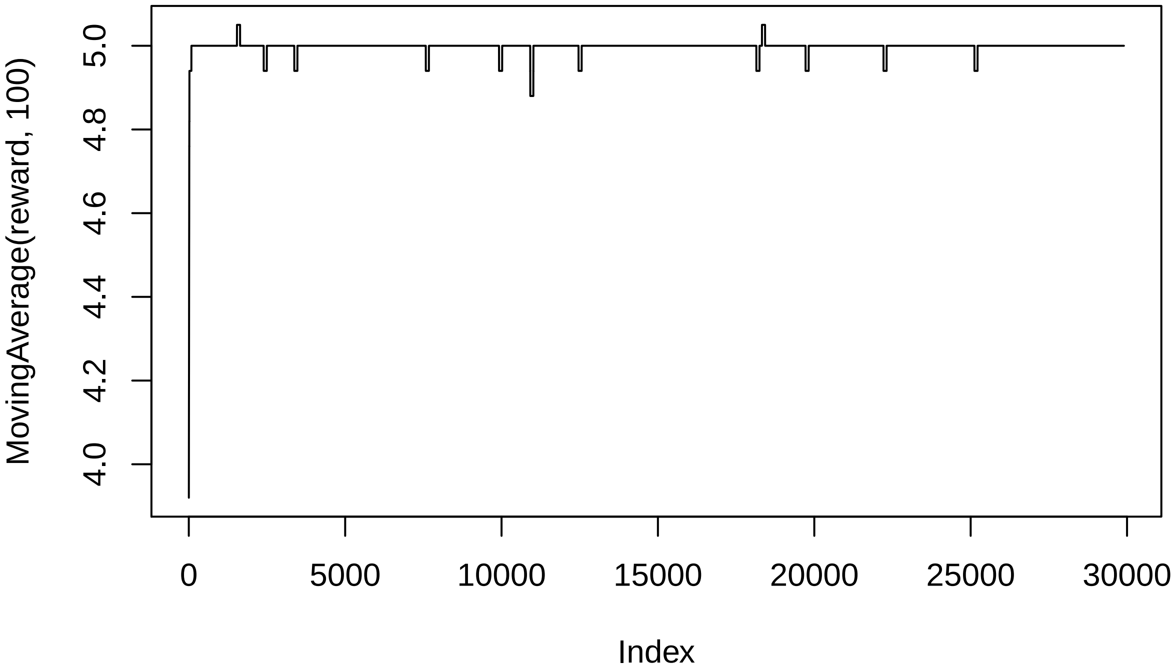

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
0
5000
10000
15000
20000
25000
30000
Index

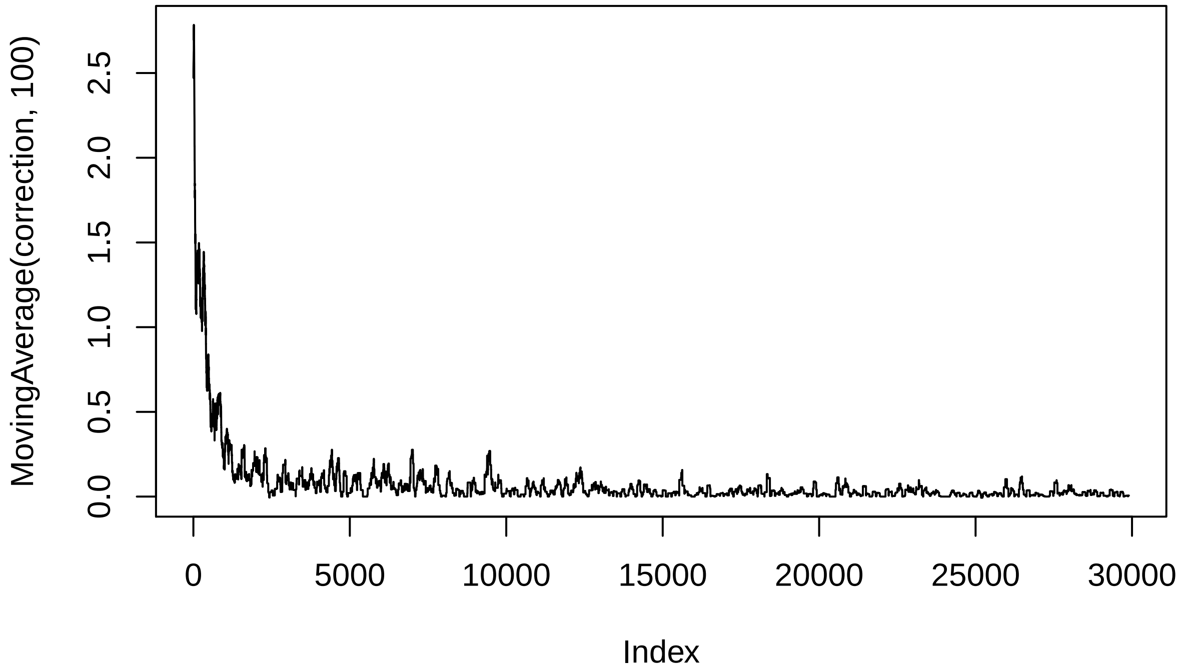

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0 )
7
−1 −1 −1 −1 −1 −1 −1 −1
6
−0.19 −0.1 −0.34 −0.1 0 0 0 0
0.07 V 0 0 > 1.56.11 V 0.01.354 < 0 0 V 0 0 ^ 0 0 ^ 0 0 ^ 0
1.01 0 2.11 0.97 0.2 0 0 0
5
0.51 1.09 1.51 1.06 0 0 0 0
1.08 > 1.58.17 V 2.01.348 > 2.82.106 V 3.28.853 V 0 0 V 0 0 < 0 0 V 0
1.32 2.11 2.23 3.75 4.85 0.1 0 0
4
1.19 1.58 2.11 2.81 0 0
1.58 > 2.11.58 > 2.82.11 > 3.72.81 > 5 5 1.35 < 0 0 > 0 10
1.19 1.58 2.11 2.81 0 0
3
1.58 2.11 2.81 3.75 0 0 0 0
1.13 ^ 1.41.01 ^ 1.87.34 ^ 2.31.392 ^ 1.02.934 < 0 0 ^ 0 0 ^ 0 0 > 0
0.53 0.97 0.71 1.01 0 0 0 0
2
0 1.57 1.67 2.21 0 0 0 0
0 > 1.010 ^ 0.010 ^ 00.08 ^ 0 0 < 0 0 > 0 0 V 0 0 < 0
−0.1 −0.27 −0.19 −0.27 −0.1 0 0 0
1
−1 −1 −1 −1 −1 −1 −1 −1
x
1 2 3 4 5 6 7 8
y
10.0
7.5
5.0
2.5
0.0


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


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 )
−1 −1 −1 −1 −1 −1 −1 −1
−0.34 0 > 3.54 0
−0.34 4.95 V 1.36 4.07
−0.34 1.68 V 0.83 4.29
−0.19 3.66 < 0 0.41
0 0 V 0 0
−0.1 0 < 0 0
0 0 ^ 0 0
0 0 < 0 0
2.45 3.75 > 4.07 3.32
3.84 3.82 > 4.29 4.04
4.05 4.07 V 4.16 4.51
2.36 4.29 V 2.64 4.75
0 0.41 V 0 4.25
0 0 > 0 0
0 0 < 0 0
0 0 > 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 0 < 0 0
0 0 > 1 0
10
4.07 3.15 ^ 3.75 2.67
4.29 3.53 ^ 3.98 2.43
3.76 3.63 > 4.51 3.3
4.75 4.24 ^ 4 3.46
0.5 4.45 < 0 0
0 0.37 < 0 0
0 0 ^ 0 0
1 0 ^ 0 0
3.56 0.24 ^ 0.36 −0.19
3.67 0 ^ 0.4 −0.19
4.28 0.33 ^ 0.53 −0.34
0.45 3.96 < 0 −0.19
0 0 ^ 0 −0.1
0 0 < 0 −0.1
0 0 < 0 −0.1
0 0 > 0 −0.1
−1 −1 −1 −1 −1 −1 −1 −1
x
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8
y
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



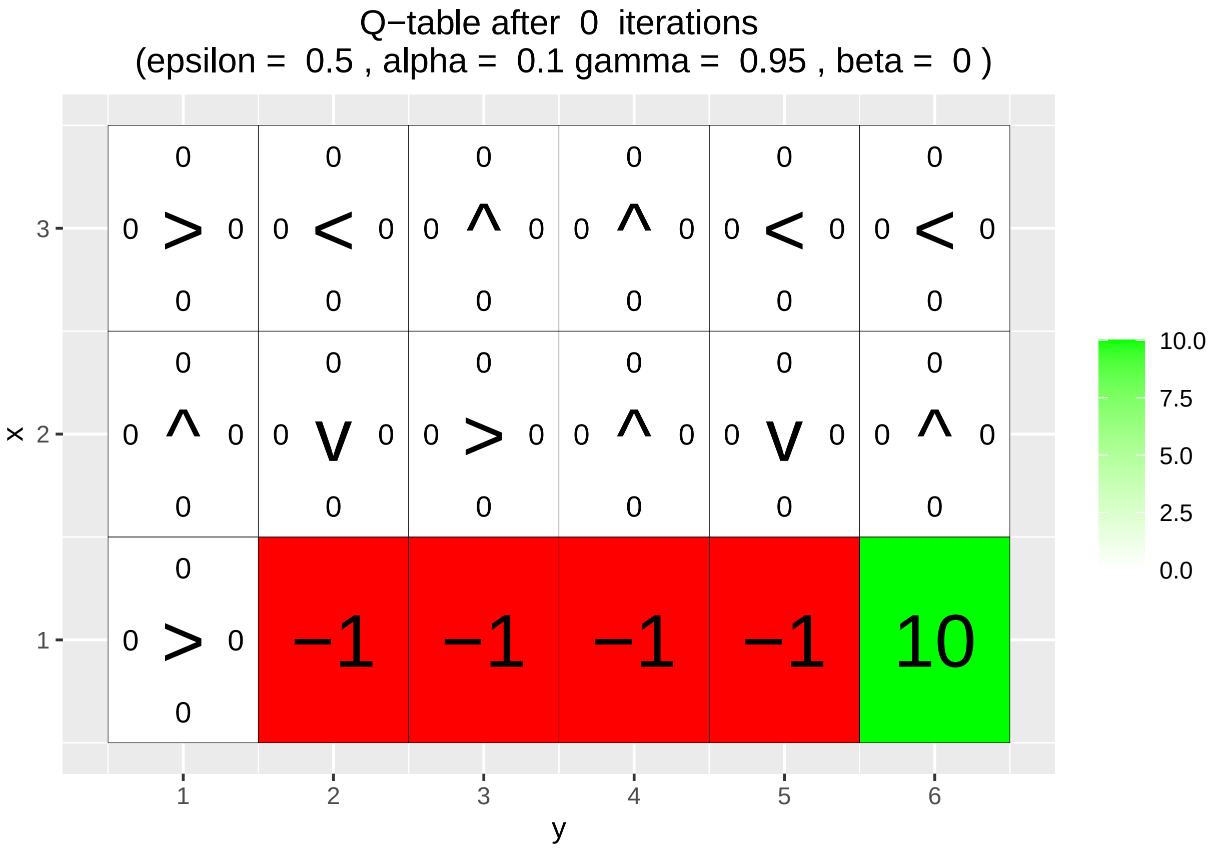
With a higher exploration rate (epsilon), the agent can find the high reward state and explores the space more. For low values of epsilon, the agent does not reach the high reward state as shown in the policy graphs. THis is also illustrated in the moving average graphs for rewards that converge and stabilize around the value 5 (low reward state) for small values of epsilon. For high value of gamma and epsilon, we see that the low reward state is avoided completely and we observe the episode where the agent "finds" the high reward state to be around 10000 episodes. At this point, we see the moving average reward function change its regime with an average value increasing from 4.8 to 6.5. This is also an indication that the value of gamma affects how long term rewards are taken into consideration.

The discount factor (gamma) affects the results in that higher values of gamma allow for long term rewards to be taken into account. We observe this for the first three sets of graphs produced. When we allow epsilon to be large enough, the increasing value of gamma shows the agent establishing policies that lead it to the high reward state as stated previously.

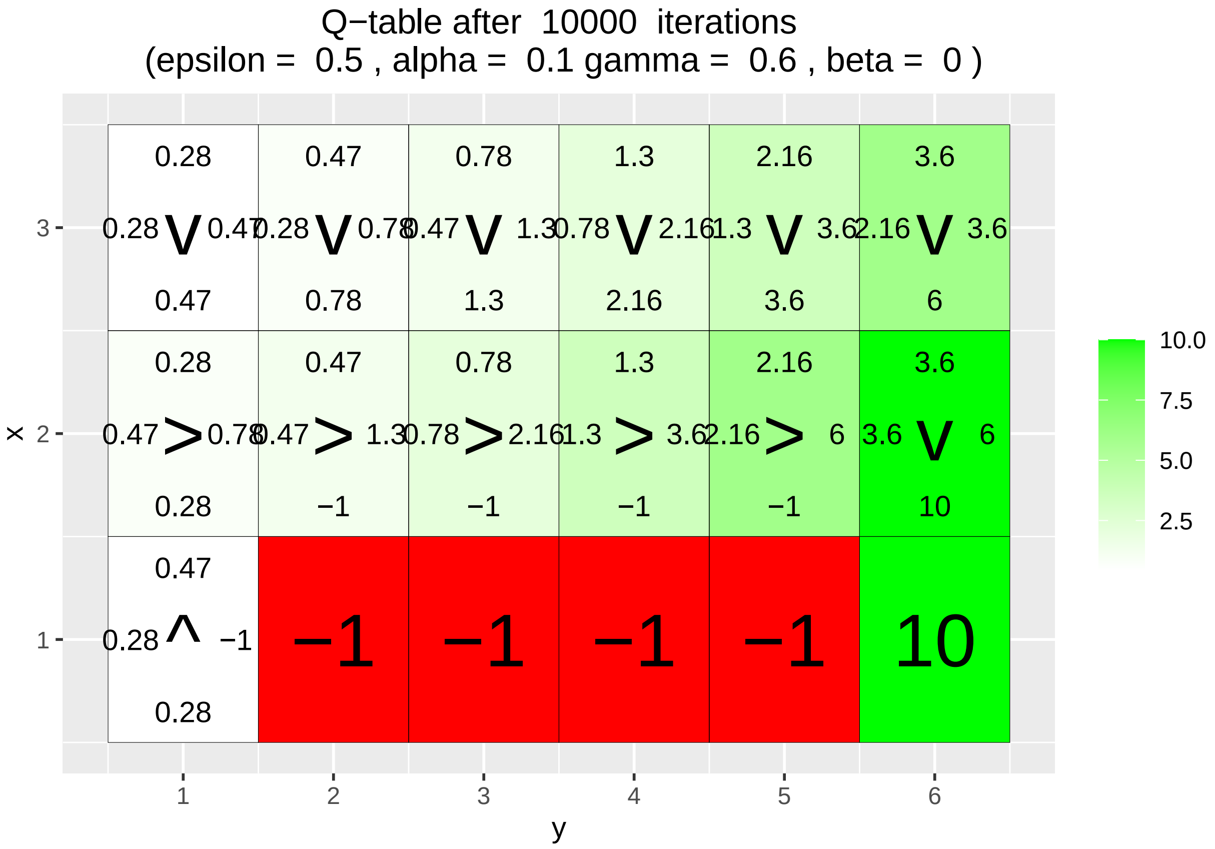
**Environment C**

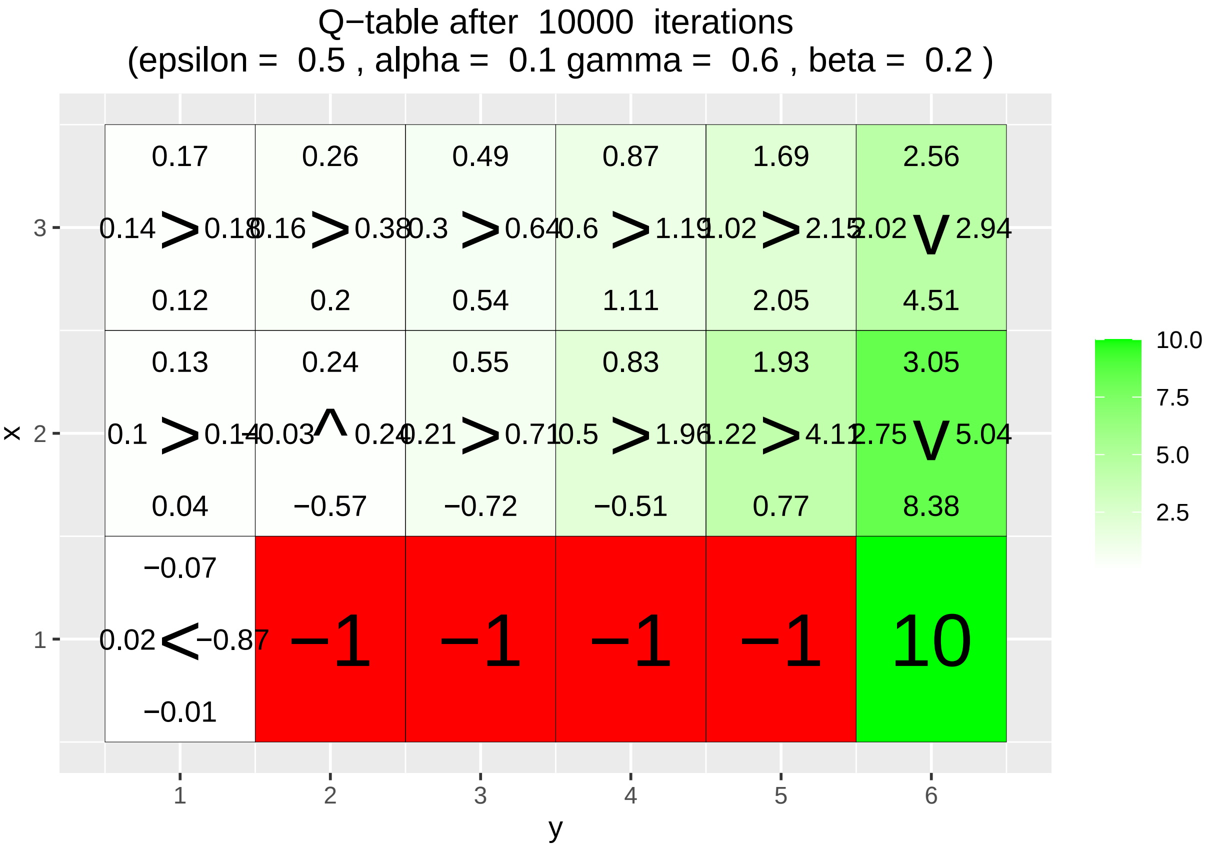
```r
# Environment C (the effect of beta).

H <- 3
W <- 6

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

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

```r
vis_environment()
```

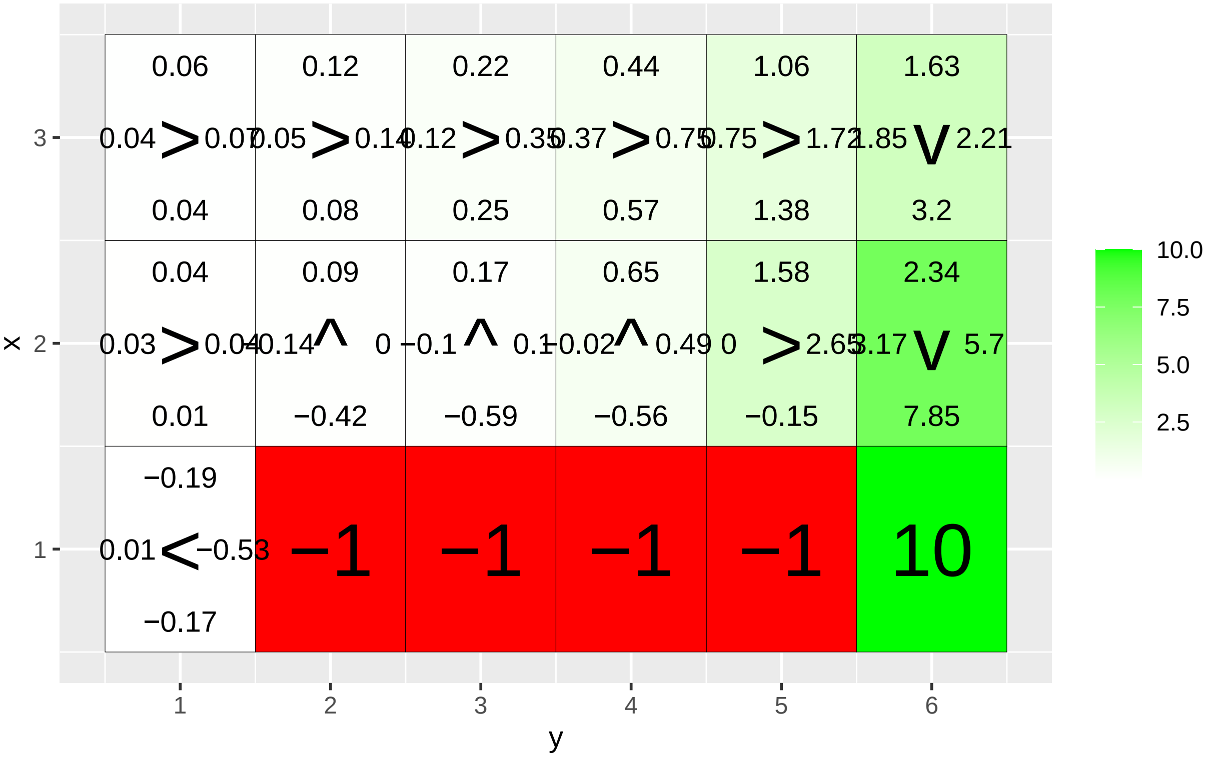


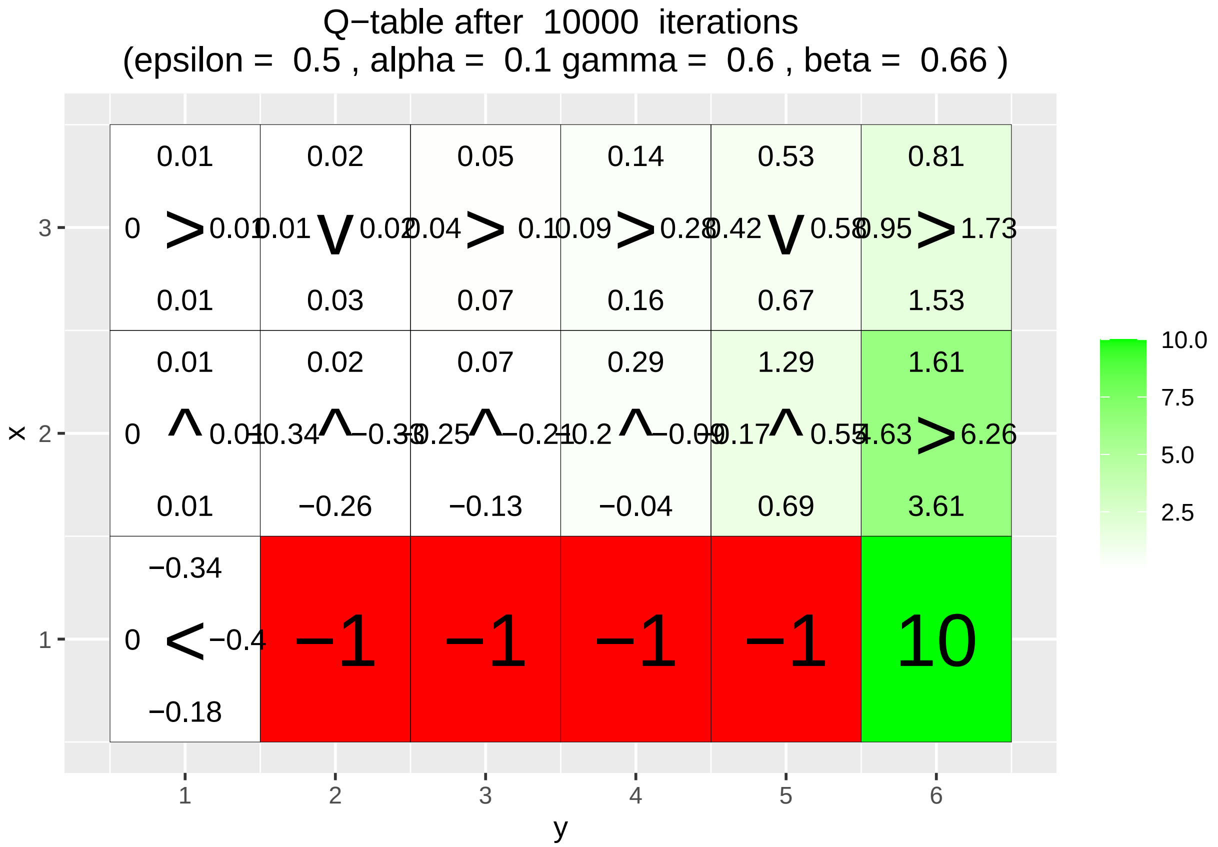
```r
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

  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 )
0.28 0.47 0.78 1.3 2.16 3.6
0.28 V 0.47 0.28 V 0.78 0.47 V 1.3 0.78 V 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
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 )
0.17
0.14 > 0.18
0.12
0.26
0.16 > 0.38
0.2
0.49
0.3 > 0.64
0.54
0.87
0.6 > 1.19
1.11
1.69
1.02 > 2.15
2.05
2.56
2.02 V 2.94
4.51
0.13
0.1 > 0.14
0.04
0.24
−0.03 ^ 0.24
−0.57
0.55
0.21 > 0.71
−0.72
0.83
0.5 > 1.96
−0.51
1.93
1.22 > 4.11
0.77
3.05
2.75 V 5.04
8.38
−0.07
0.02 < −0.87
−0.01
−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 )





The value of beta influences the “sliding” of the agents at random. With higher values of beta, the agent learns that there is more uncertainty in the movements and learns to avoid negative reward areas from a longer distance.

We will note that depending on the value of beta, even if there is a risk to fall into the penalty area, it is sometimes beenfitial to move near the penalty zone. The risk is low enough of slipping into the penalty area.

## REINFORCE

### Environment D

**Has the agent learned a good policy? Why/ Why not?** On the one hand, we could say that the agent has learned a good policy. Most paths lead to the goals on the validation set. There are however some states that lead to loops. This may be caused by a too small number of training episodes, too few training set examples or a too great number of parameters in the neural network.

**Could you have used the Q-learning algorithm to solve this task?** This could have been solved using Q-learning, but it would be very demanding computationally as the Q-table needed would be a very large tensor to account for all possible combinations of state, actions, goals and starting positions. If we restrict ourselves to a HxWx4 tensor, this problem cannot be solved using Q-learning.

### Environment E

**Has the agent learned a good policy? Why/ Why not?** The policy learned is not good because the agent only moves in a single direction and did not learn to find the goal. The agent did not learn to move in the “down” direction.

**If the results obtained for environments D and E differ, explain why?** The training set for environment E has all goals set in the x=4 position. In the validation set, all goals are placed in positions that are not x=4. It is thus understood that the training set is not representative of all possible goal positions. This bias in our training data leads to a training model that doesn't give generalizable policies.