isak_berntsson

Isak Berntsson

9/30/2022

This file contains alot of output. This has been kept in according to feedback on previous labs.

Q-learning

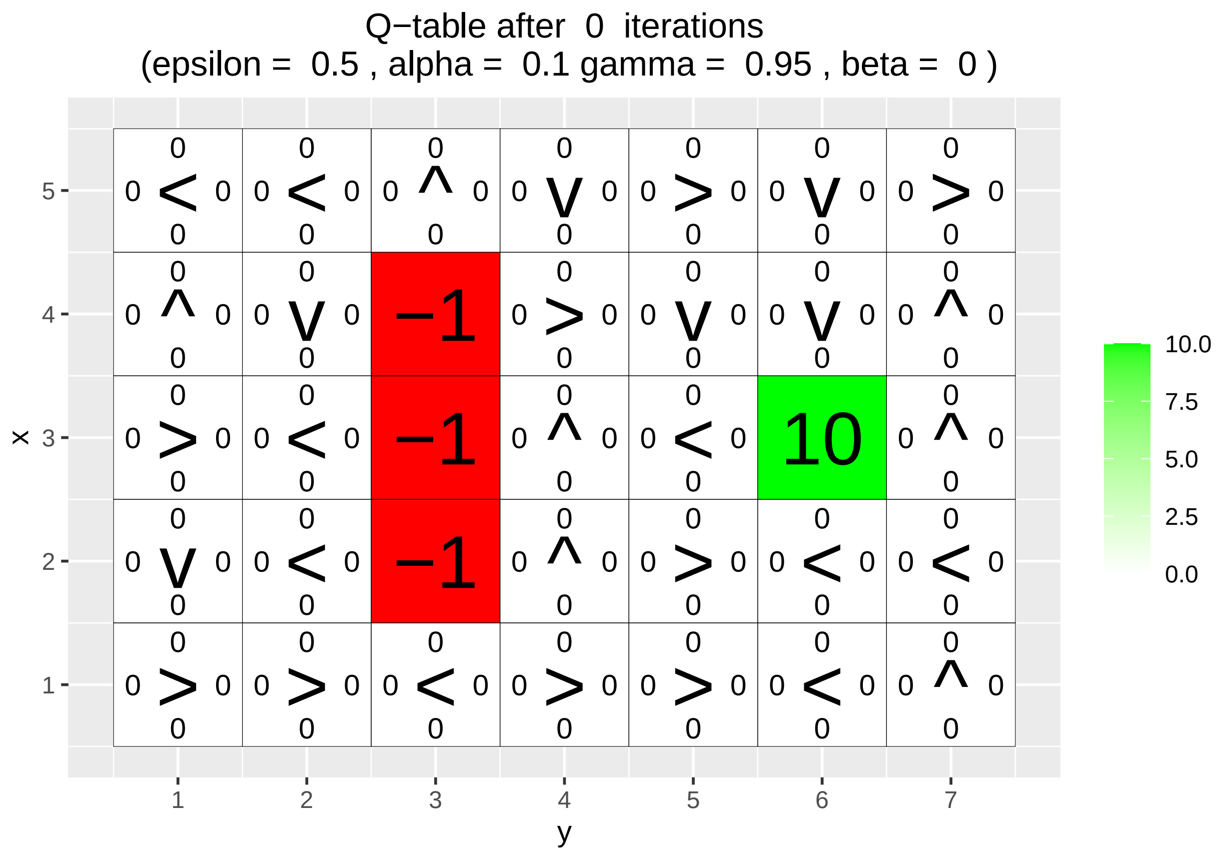
```r
q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95,
                       beta = 0){

  # 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 <<-.

  # Your code here.


  state = start_state

  correction = 0

  episode_correction = 0
  repeat{

    action = EpsilonGreedyPolicy(state[1],state[2],epsilon = epsilon )
    #print(action)
    new_state = transition_model(state[1],state[2],action = action, beta = beta)

    #print(new_state)

    reward = reward_map[new_state[1], new_state[2]]
    #print(reward)

    Q_SA = q_table[state[1],state[2],action]

    correction = alpha*(reward + gamma*max(q_table[new_state[1], new_state[2],]) - Q_SA)

    episode_correction = episode_correction + correction/alpha


    q_table[state[1],state[2],action] <<- Q_SA + correction
    #q_table[state[1],state[2],action] <<- Q_SA + alpha*(reward + gamma*max(q_table[new_state[1], new_s

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

    state = new_state
  }

}
```

Policies

```
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}.

  return(sample(which(q_table[x,y,]==max(q_table[x,y,])),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.
  if (runif(1) > epsilon){ # P(random) = 1-epsilon
    return(sample(1:4,1))
  }else{

    return(sample(which(q_table[x,y,]==max(q_table[x,y,])),1))

  }


}
```

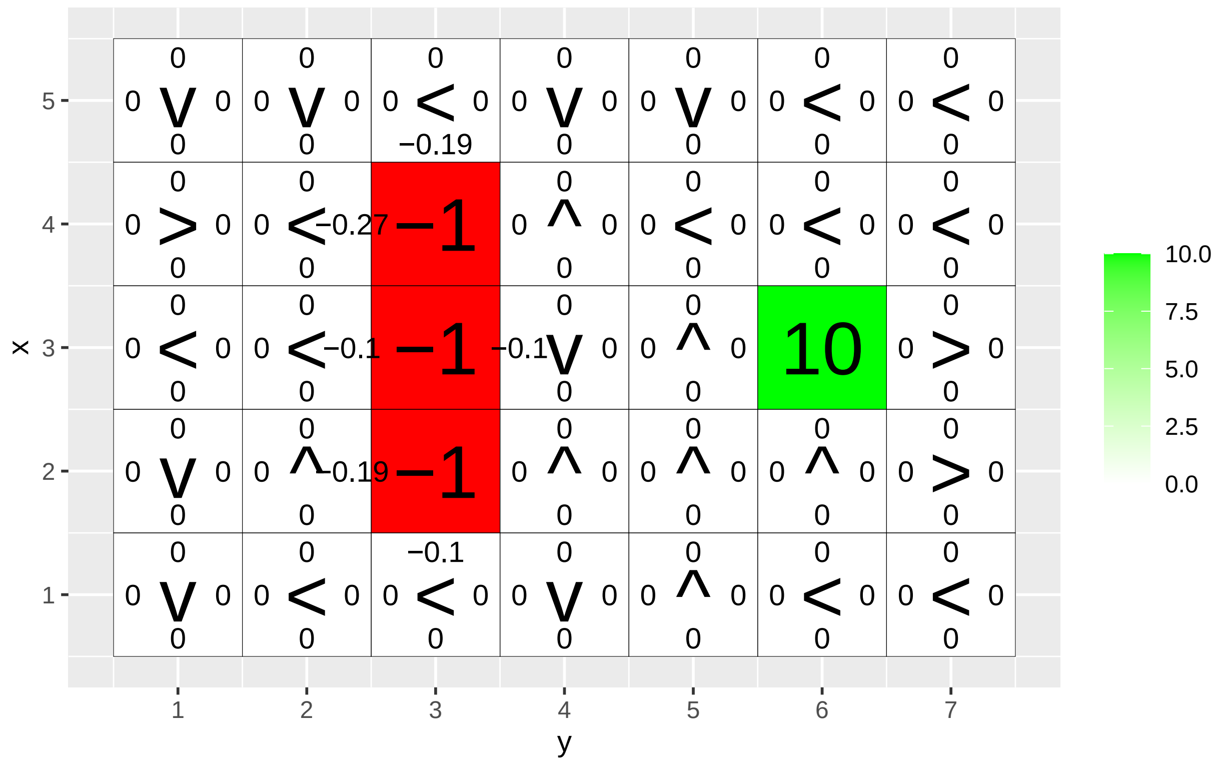
Envrionment A

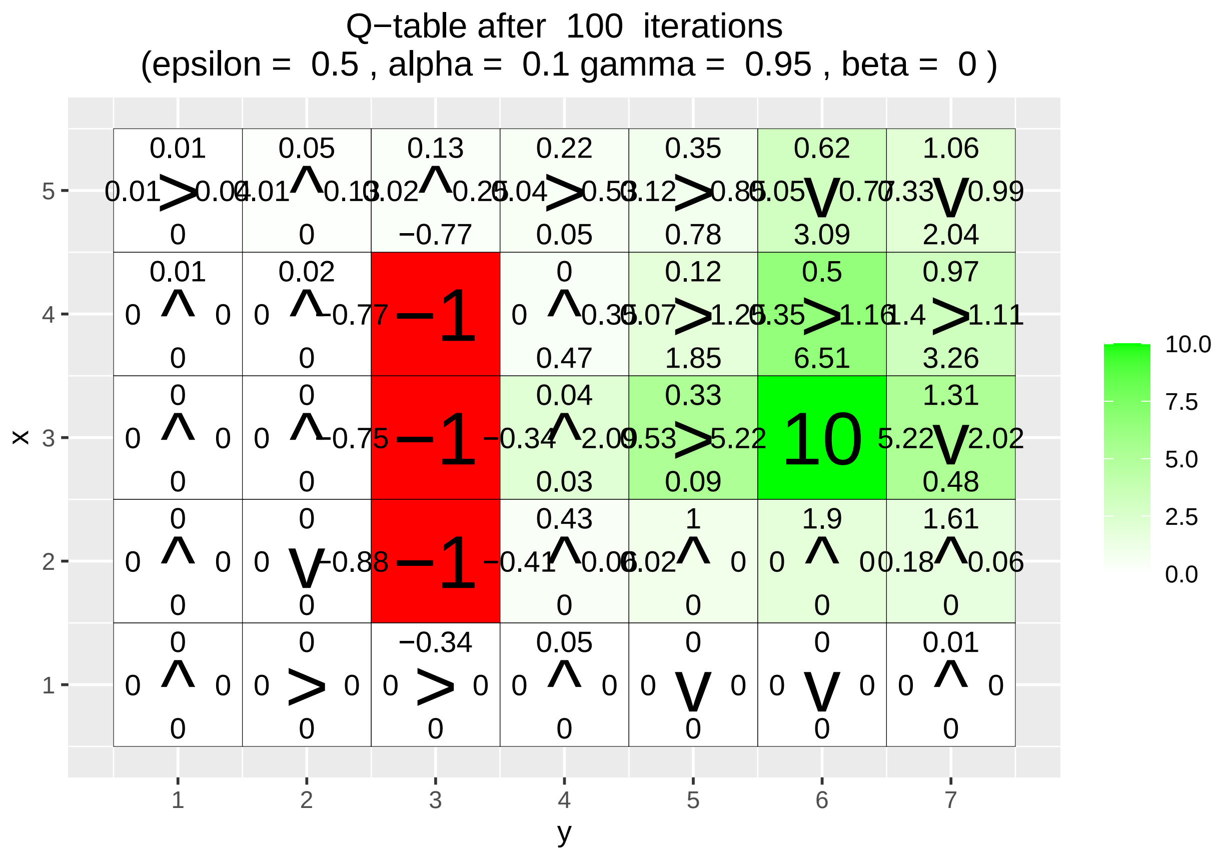
```
# Environment A (learning)
set.seed(1337)
H <- 5
W <- 7

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()
```

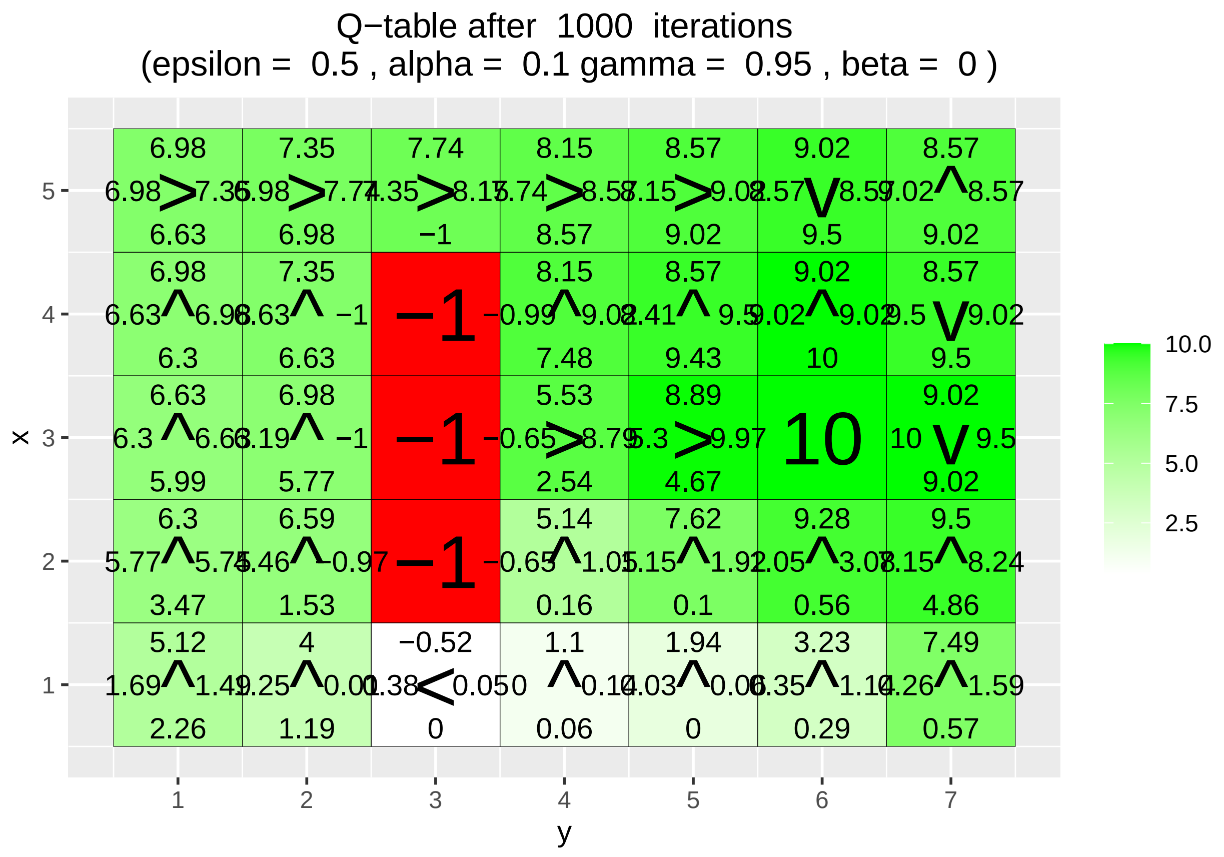


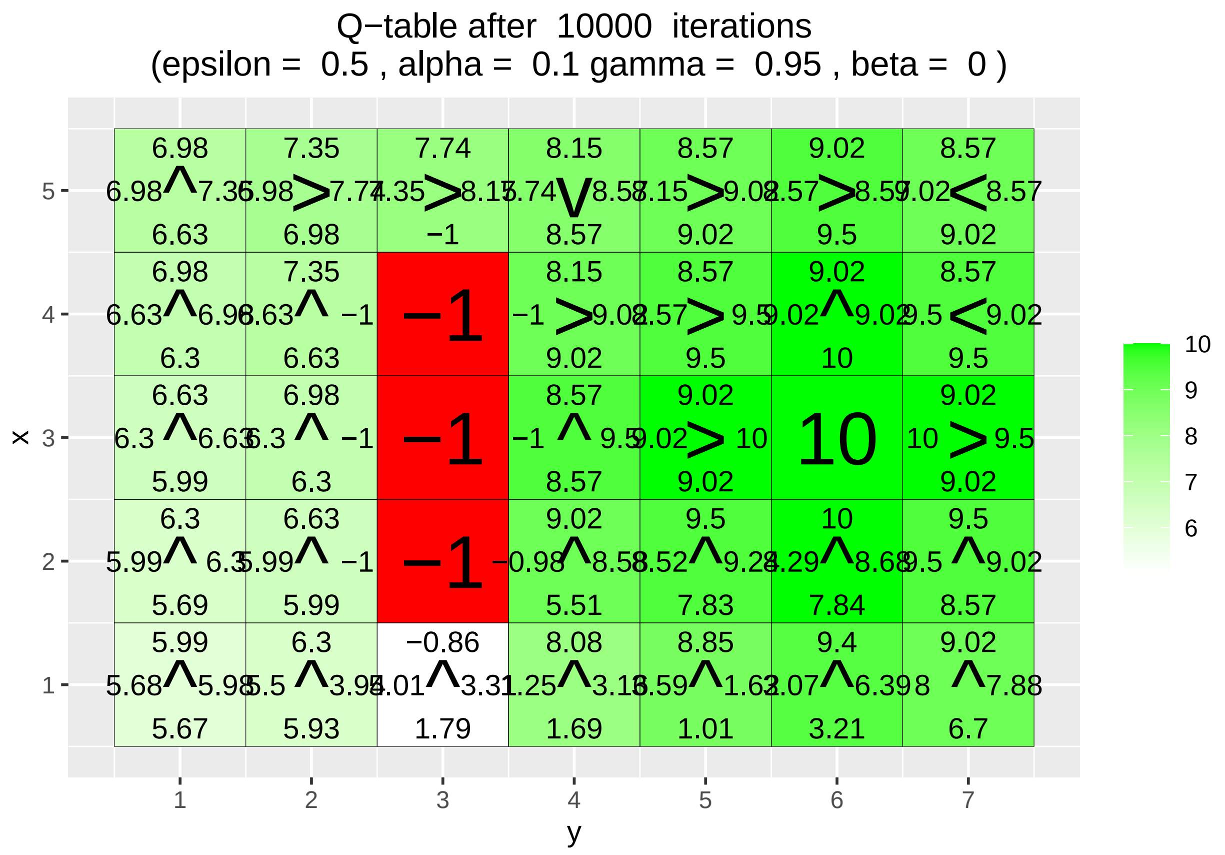
```r
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))

  if(any(i==c(10,100,1000,10000,100000)))
    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)
x
y
1
2
3
4
5
6
7
−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)
x
y
10.0
7.5
5.0
2.5



– What has the agent learned after the first 10 episodes? After 10 episodes the agent has learned that some of the squares close to the 'wall' are likely to lead to a negative result.

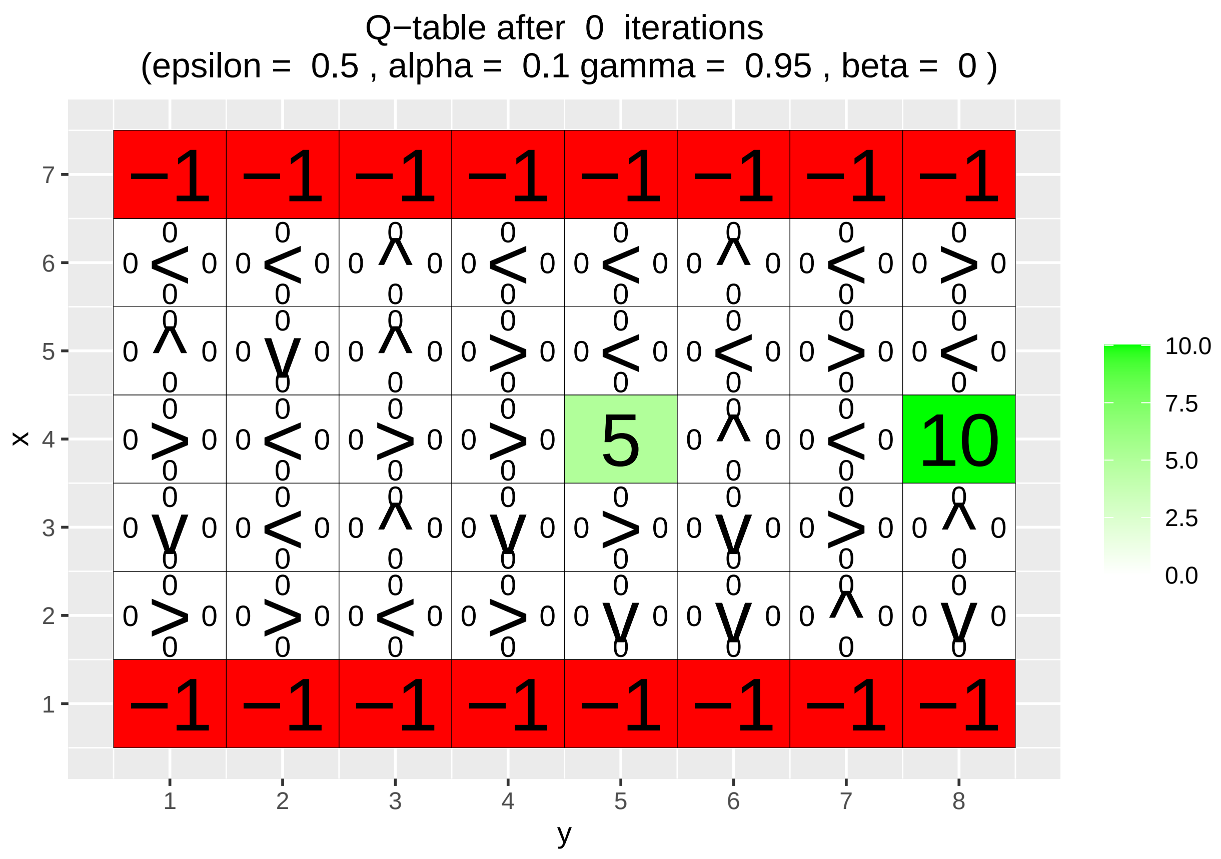
- Is the final greedy policy (after 10000 episodes) optimal for all states, i.e. not only for the initial state ? Why / Why not? No, state[1,3] has greedy policy to go straight into the 'wall'.

– 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? Yes somewhat, the lower path around the wall has lower values. We could implement epsiloan annealing, thus forcing the agent to expolre he space in the earlier iterations. Before finding the optimal paths

Environment B (the effect of epsilon and gamma)

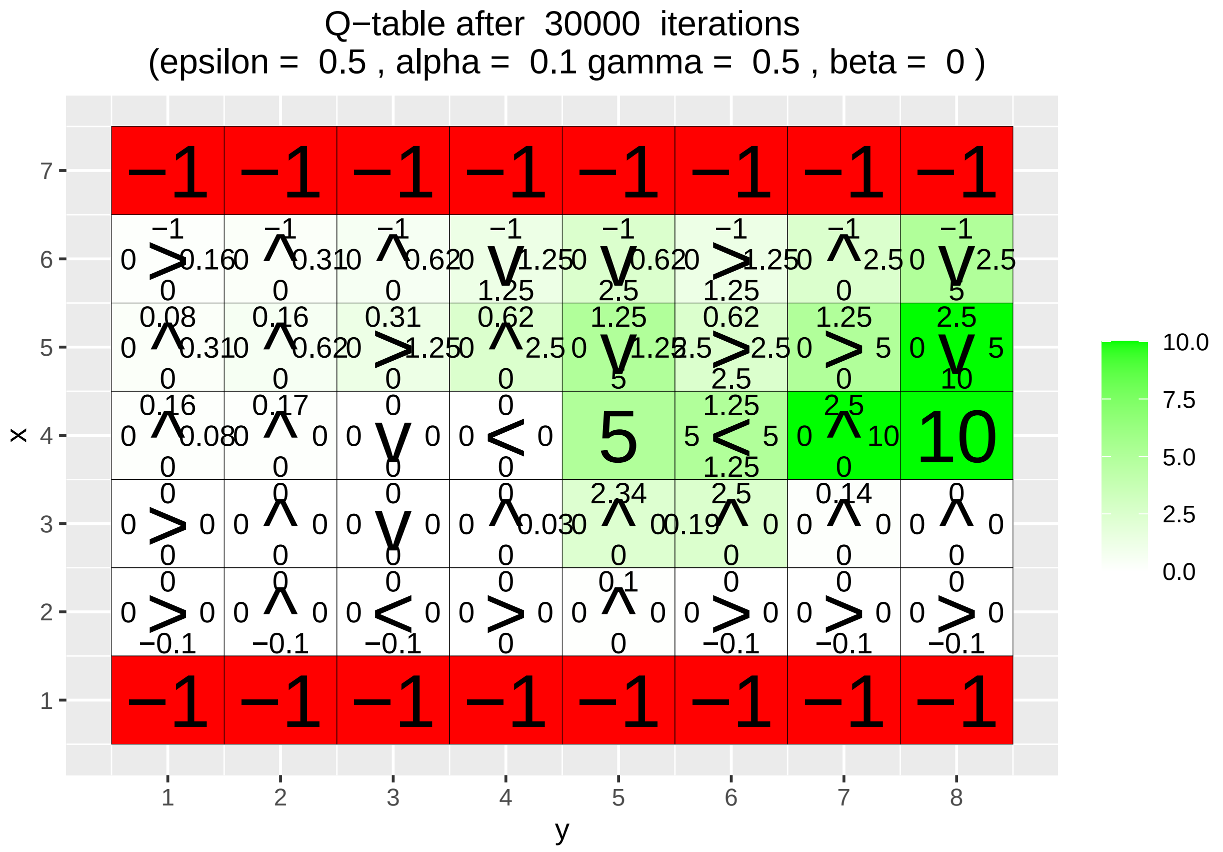
```
set.seed(1337)


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
```

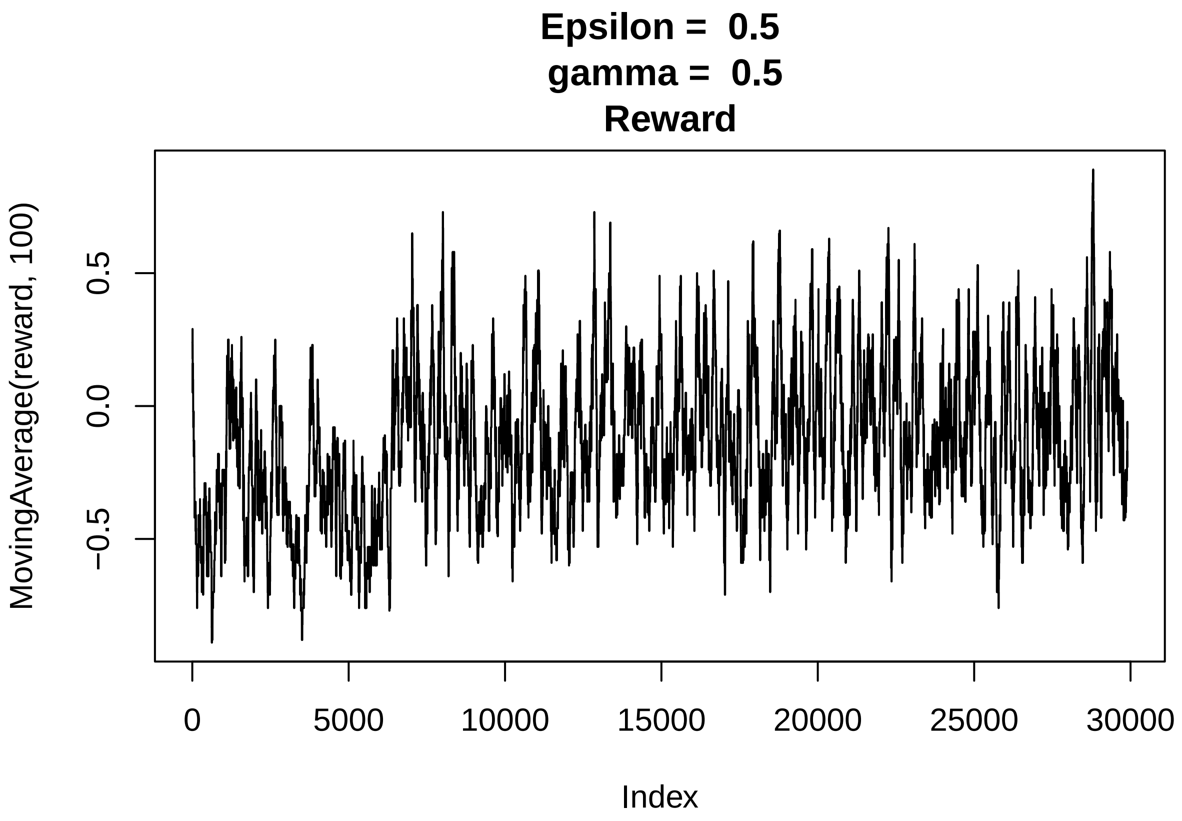
```r
q_table <- array(0,dim = c(H,W,4))

vis_environment()
```

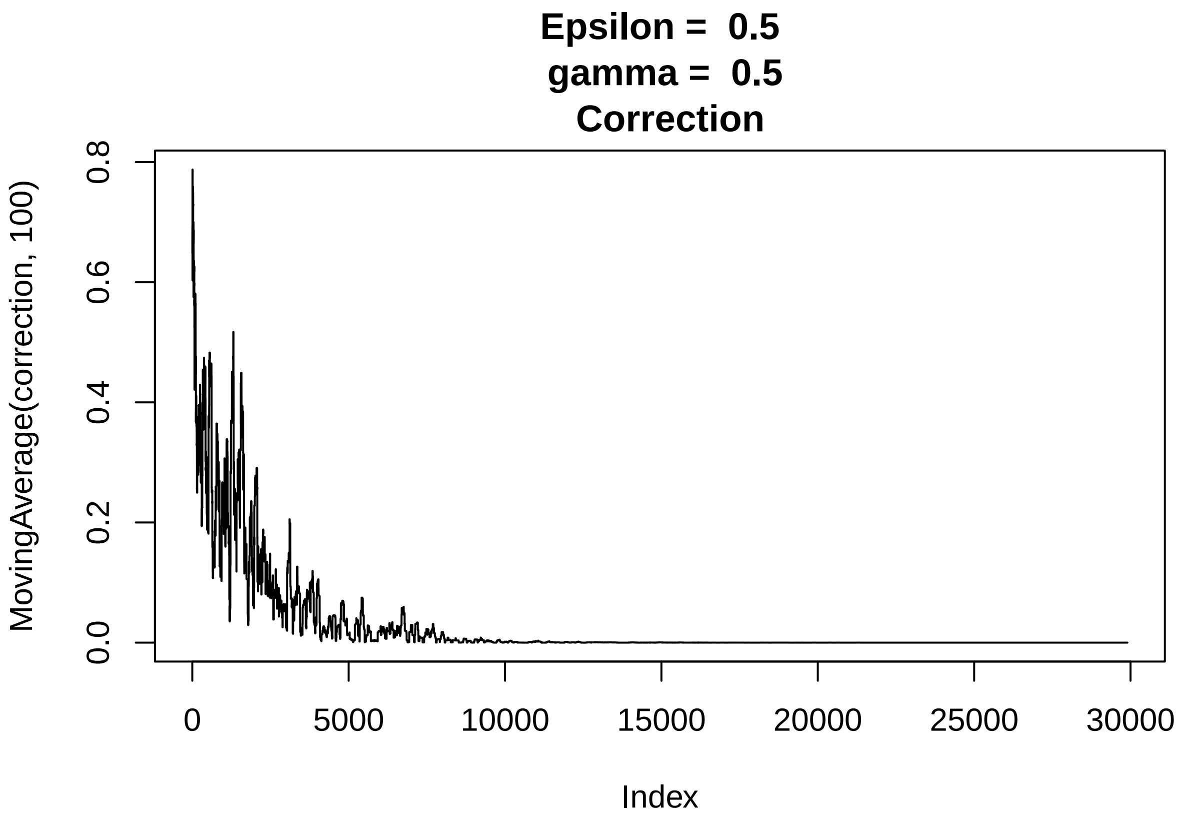


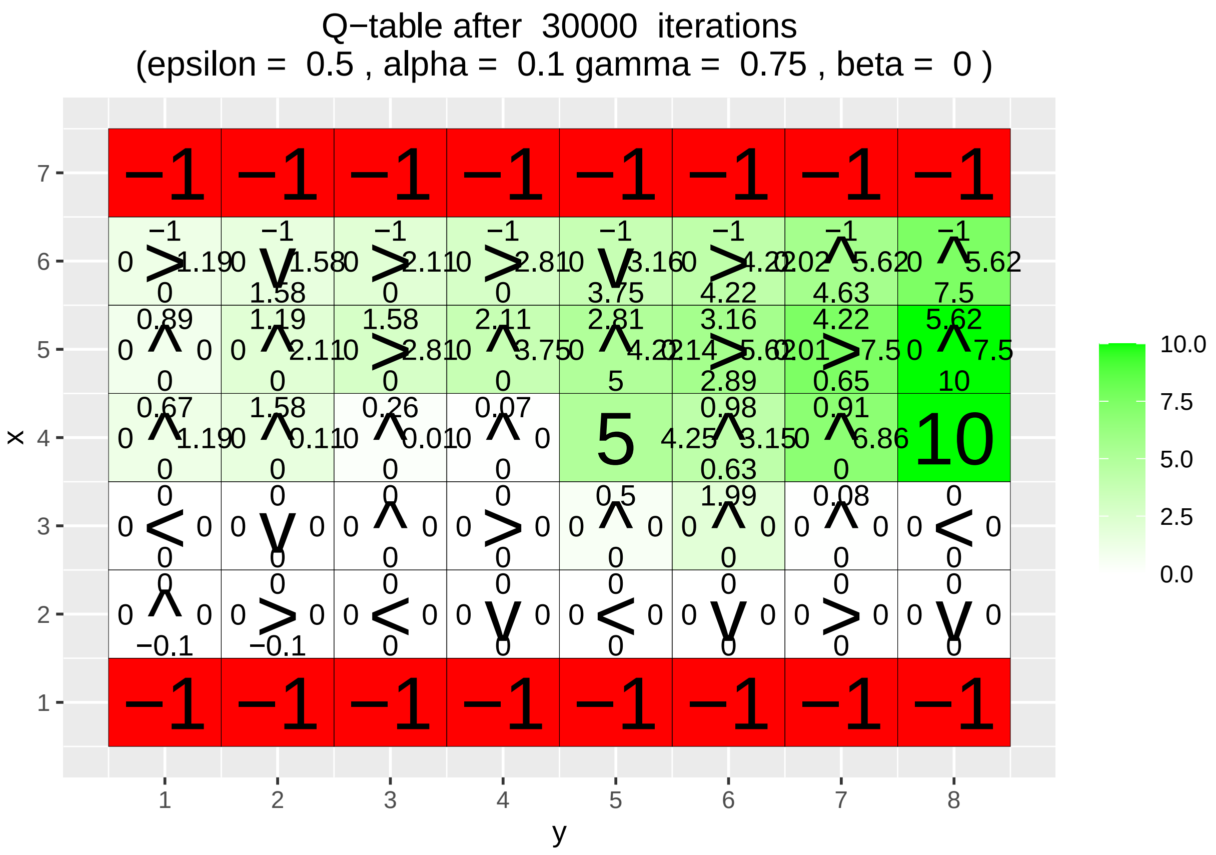
```r
MovingAverage <- function(x, n){

  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(epsilon=5, gamma = j, start_state = c(4,1))
    reward <- c(reward,foo[1])
    correction <- c(correction,foo[2])
  }

  vis_environment(i, gamma = j)
```

```
 title = paste(" Epsilon = ", .5, "\n",
        " gamma = ", j , "\n",
        " Reward"
        )
 plot(MovingAverage(reward,100),type = "l", main=title)

 title = paste(" Epsilon = ", .5, "\n",
        " gamma = ", j , "\n",
        " Correction"
 )
 plot(MovingAverage(correction,100),type = "l",main=title)
}
```

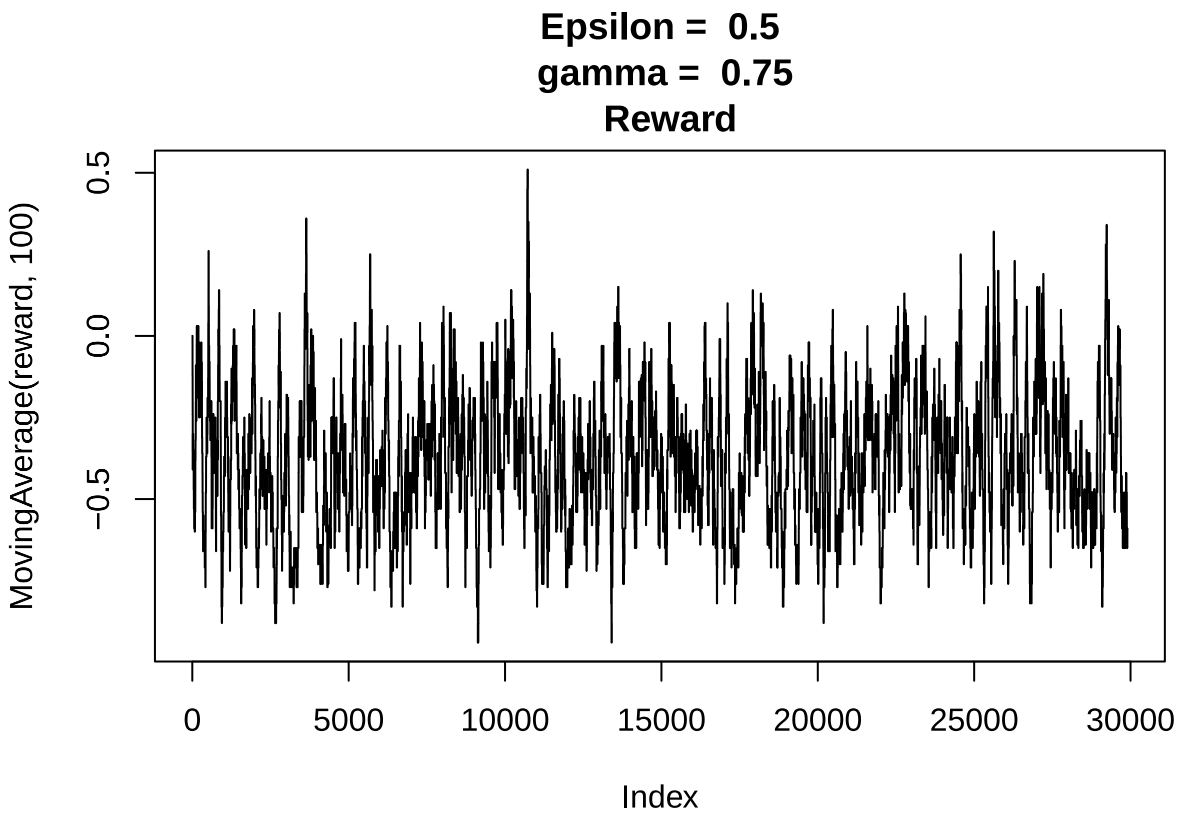


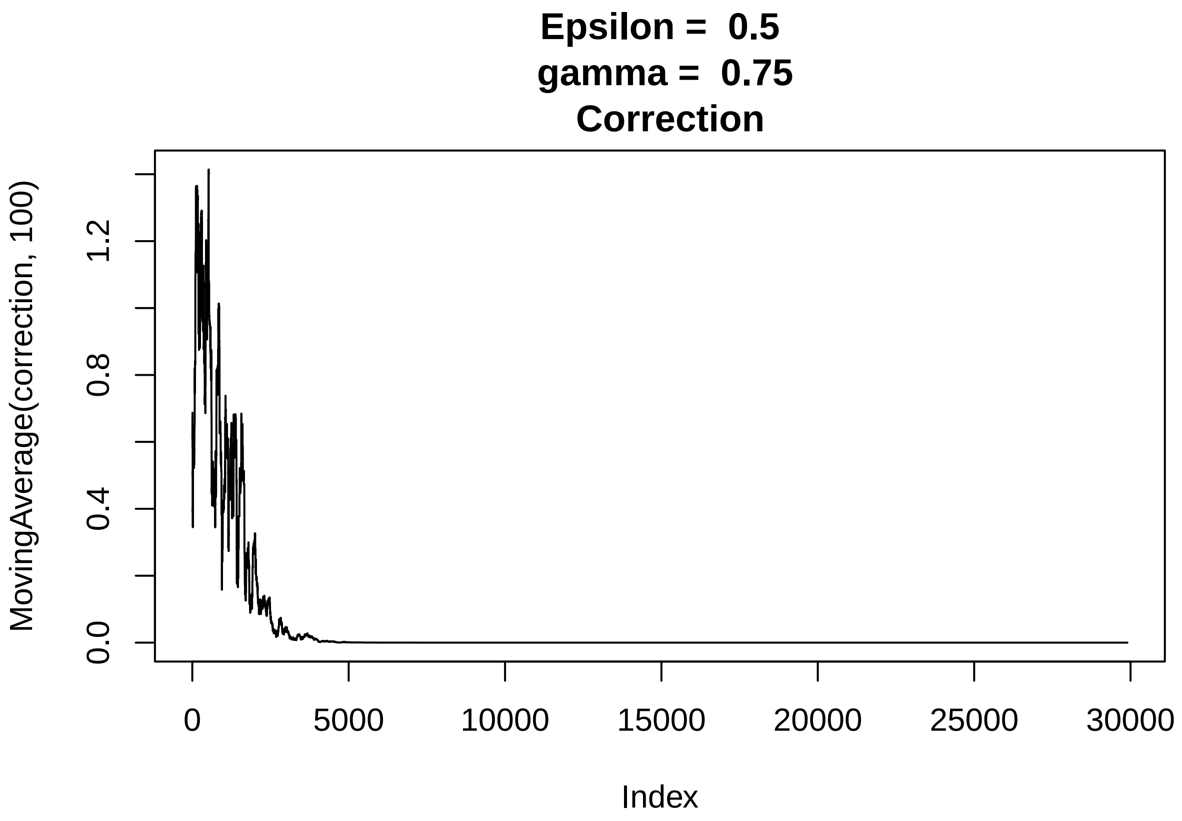

Epsilon = 0.5
gamma = 0.5
Reward
MovingAverage(reward, 100)
−0.5
0.0
0.5
0
5000
10000
15000
20000
25000
30000
Index

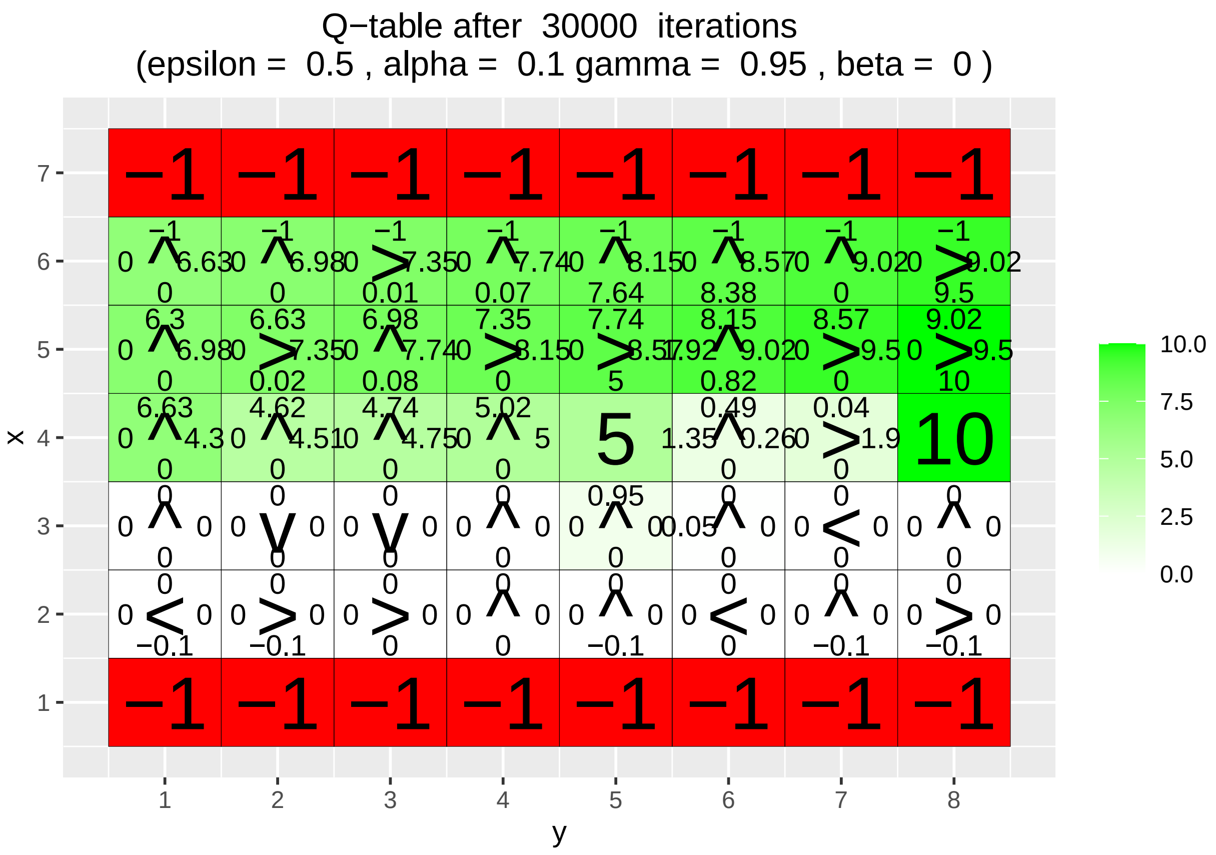

Epsilon = 0.5
gamma = 0.5
Correction
MovingAverage(correction, 100)
0.0
0.2
0.4
0.6
0.8
0
5000
10000
15000
20000
25000
30000
Index

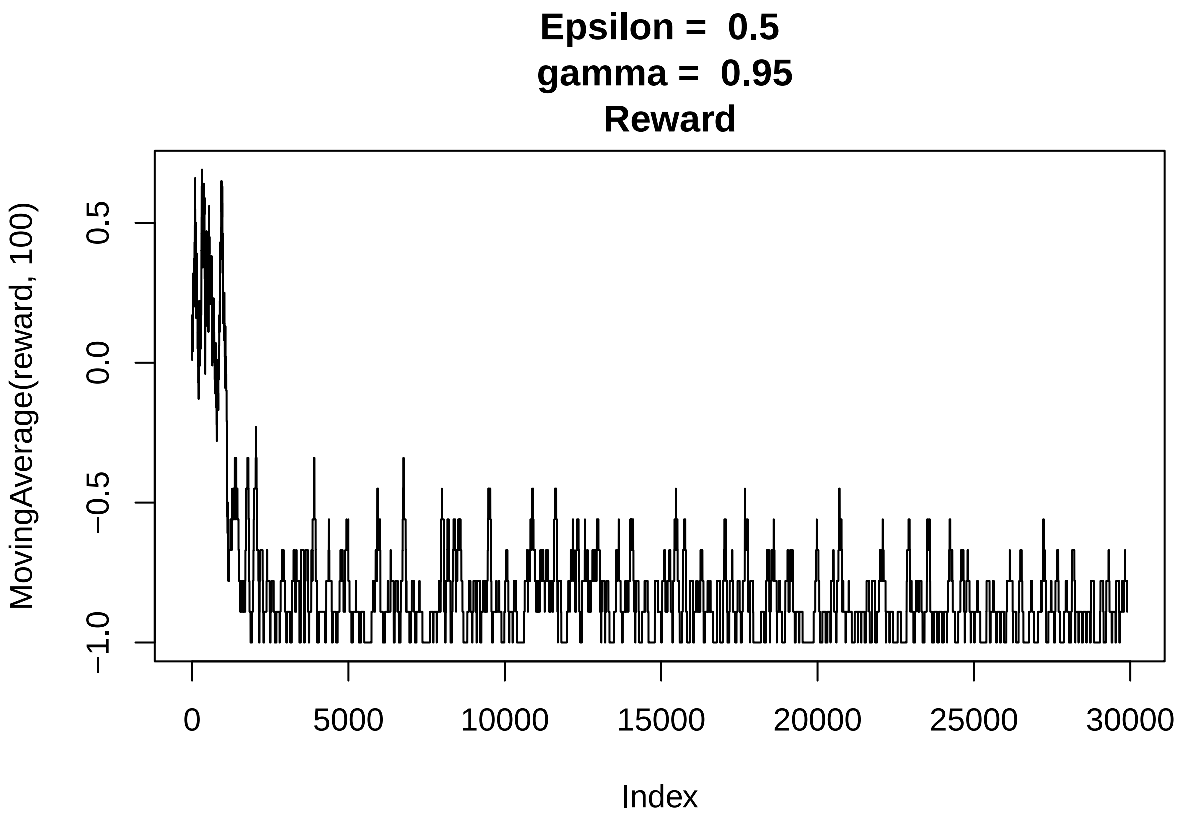


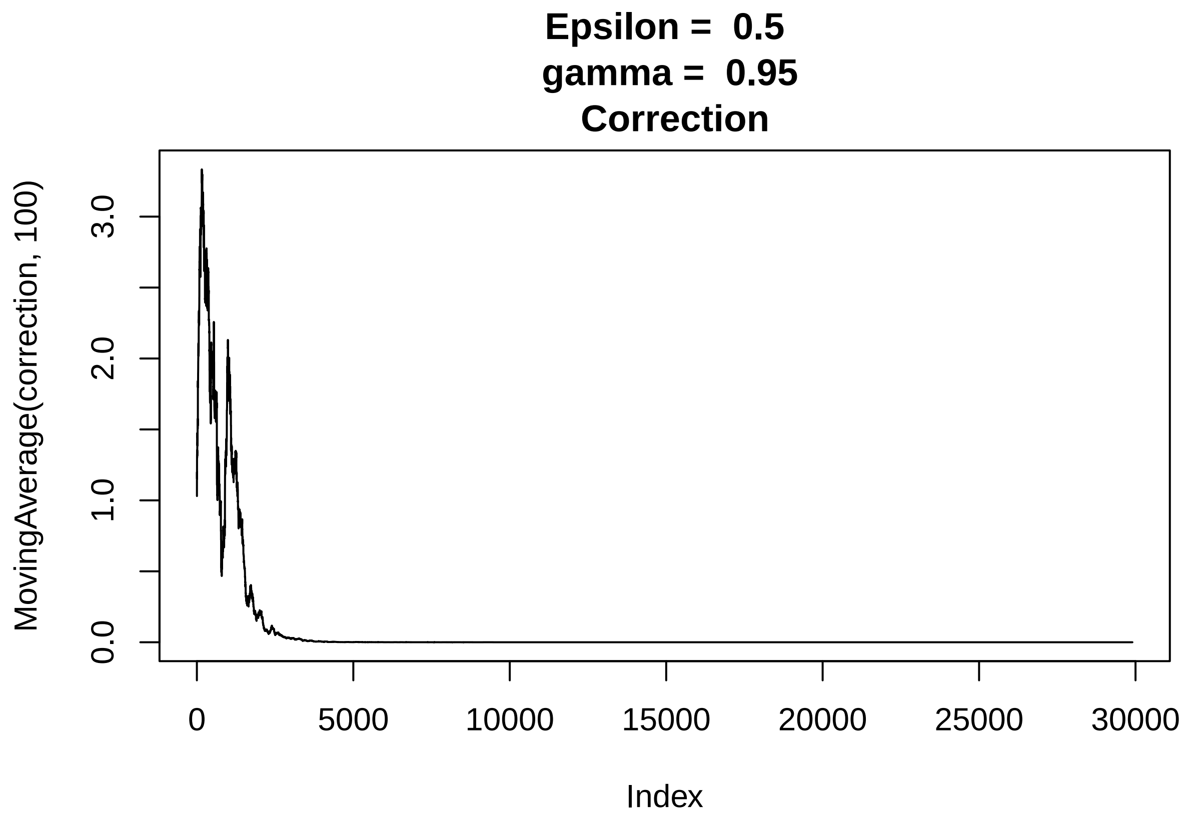
Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0


Epsilon = 0.5
gamma = 0.75
Reward
MovingAverage(reward, 100)
0.5
0.0
−0.5
0
5000
10000
15000
20000
25000
30000
Index

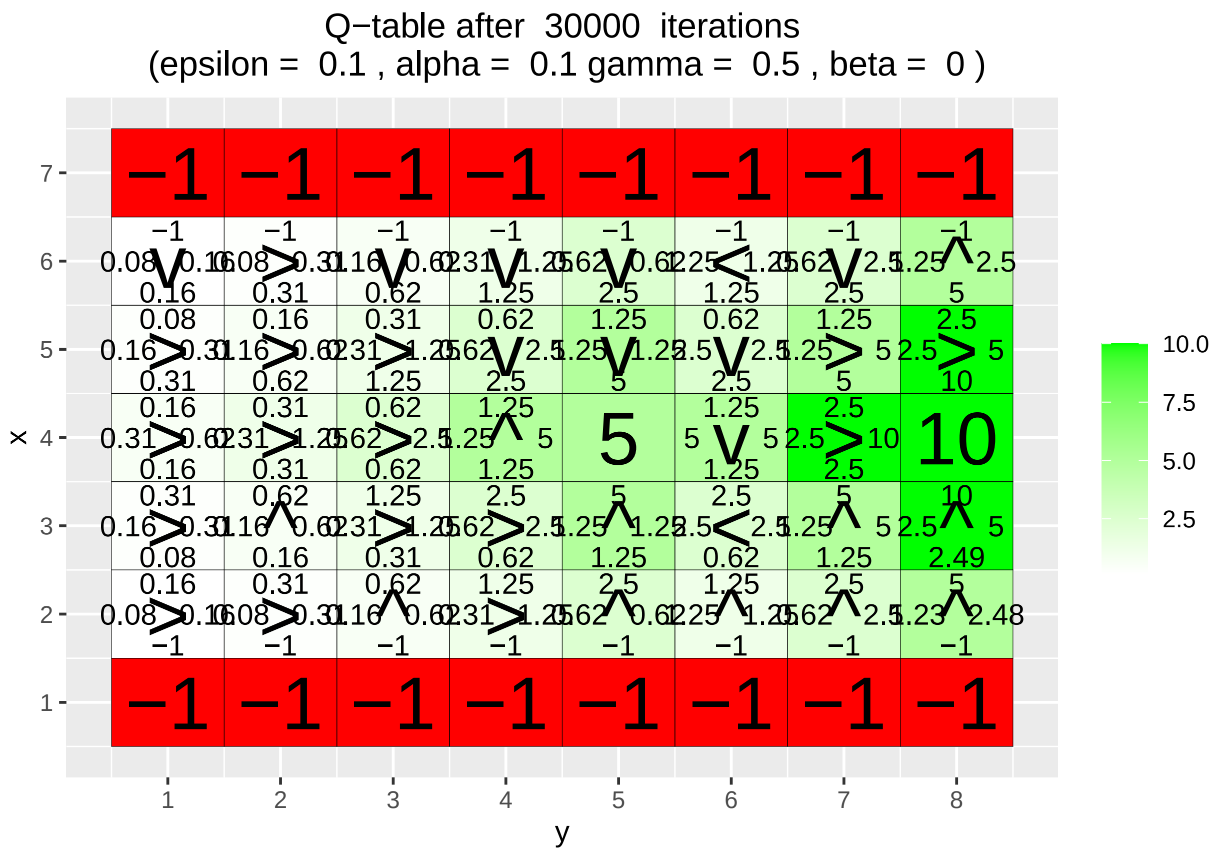

Epsilon = 0.5
gamma = 0.75
Correction
MovingAverage(correction, 100)
0.0
0.4
0.8
1.2
0
5000
10000
15000
20000
25000
30000
Index

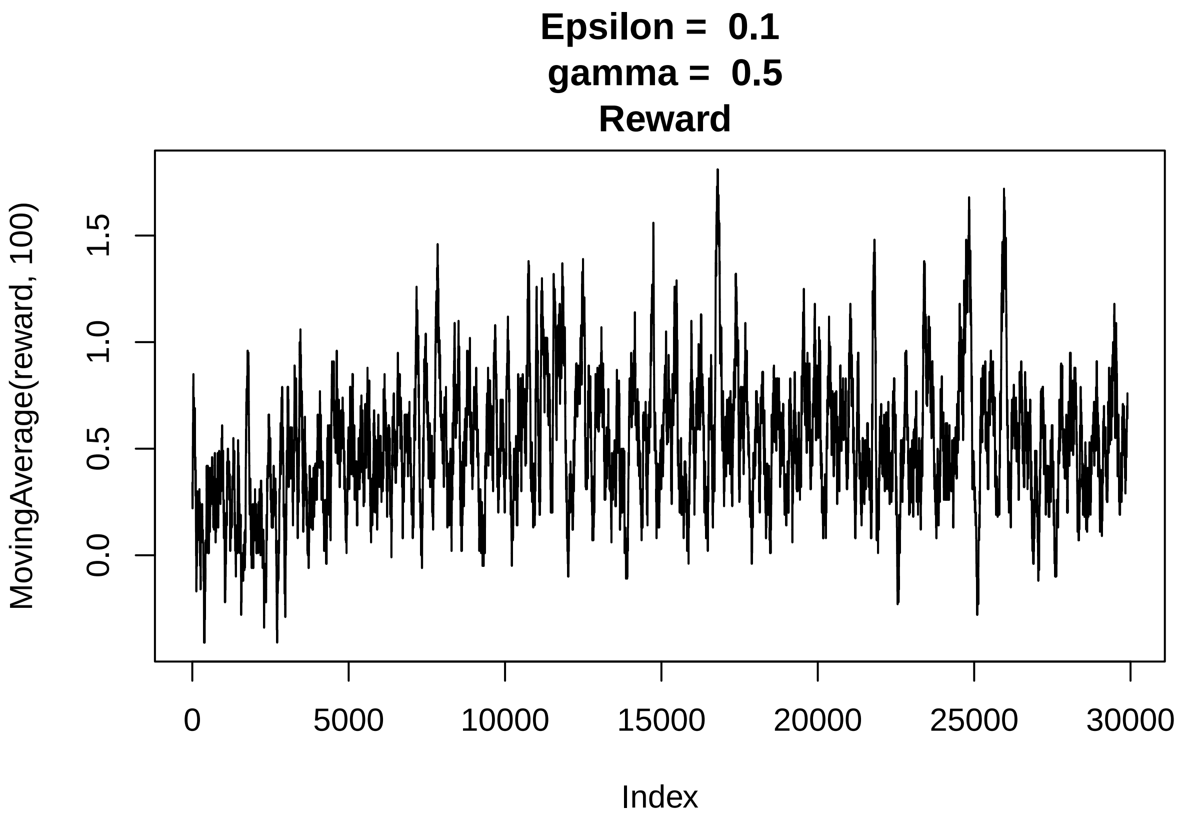

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
1
2
3
4
5
6
7
8
−1
5
10
10.0
7.5
5.0
2.5
0.0

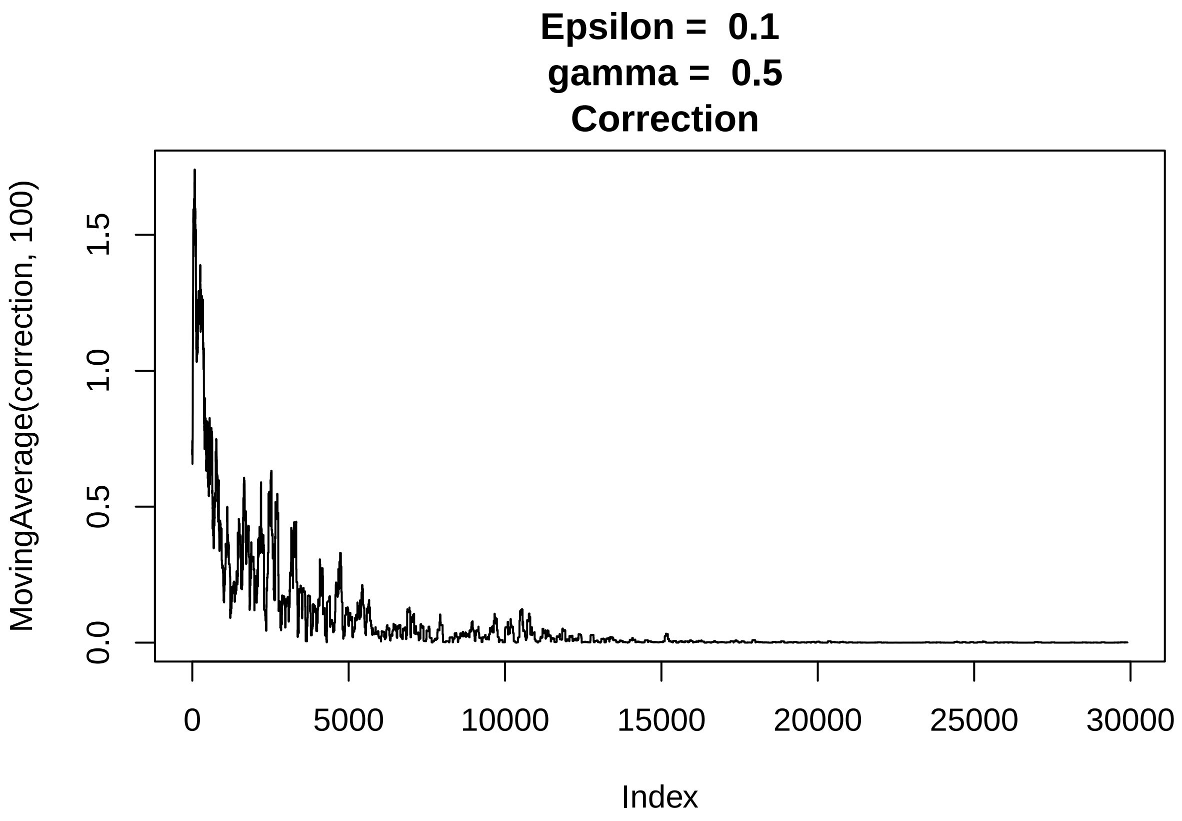

Epsilon = 0.5
gamma = 0.95
Reward
MovingAverage(reward, 100)
0.5
0.0
−0.5
−1.0
0
5000
10000
15000
20000
25000
30000
Index

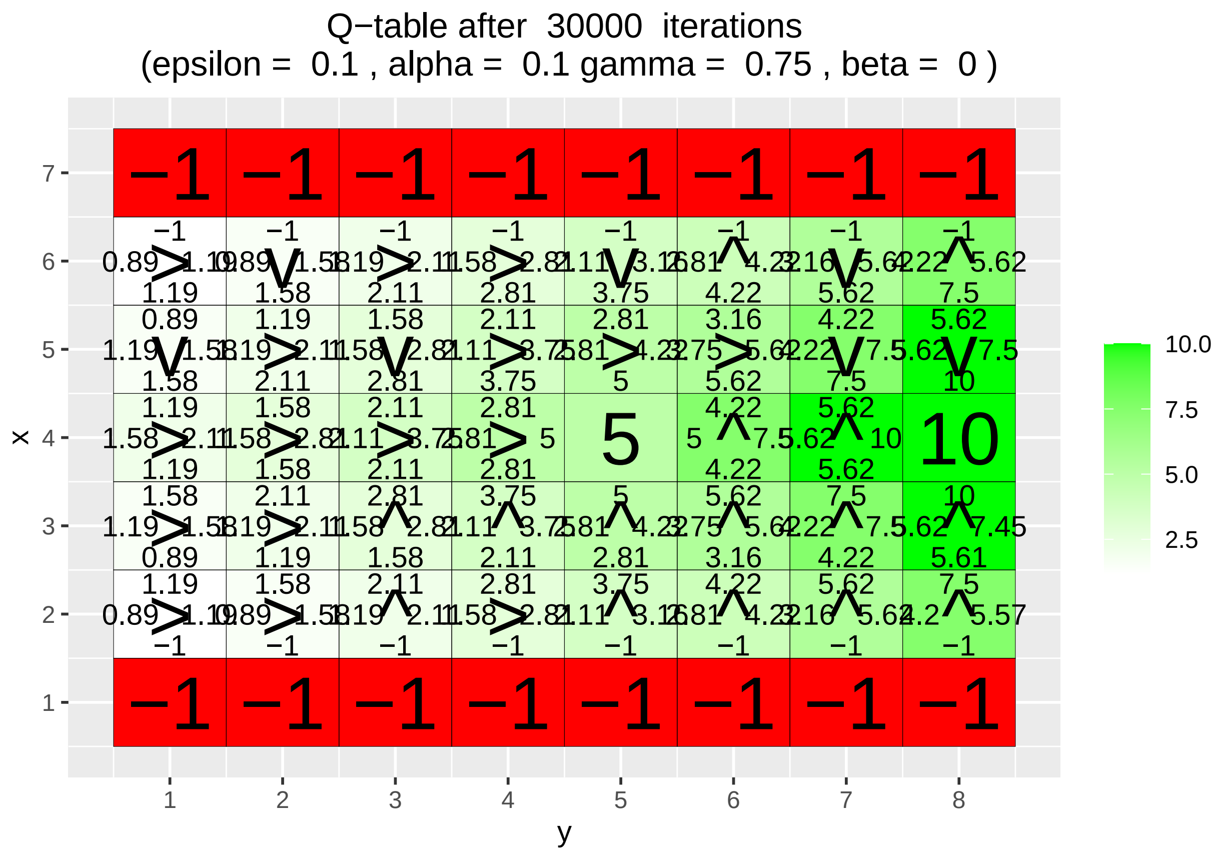


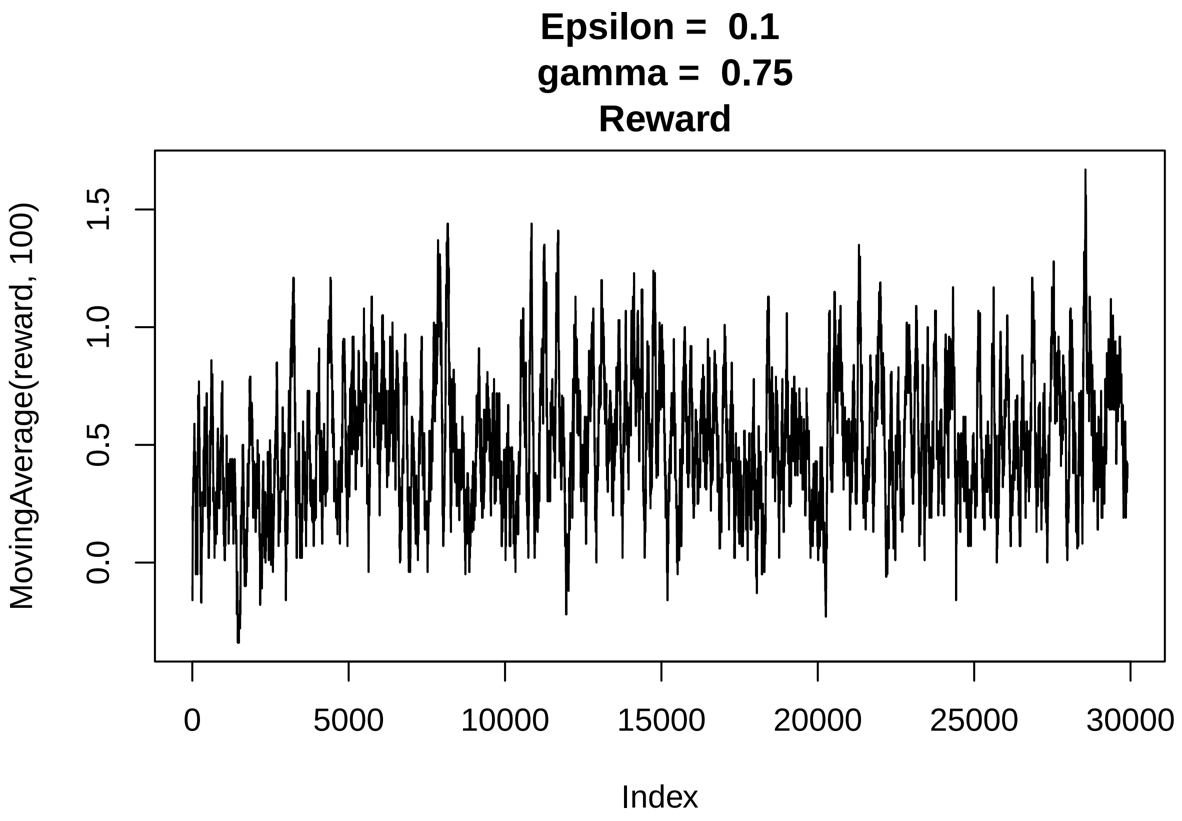
```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  for(i in 1:30000){
    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)

  title = paste(" Epsilon = ", .1, "\n",
               " gamma = ", j, "\n",
               "Reward"
  )
  plot(MovingAverage(reward,100),type = "l",main=title)

  title = paste(" Epsilon = ", .1, "\n",
               " gamma = ", j, "\n",
               "Correction"
  )
  plot(MovingAverage(correction,100),type = "l",main=title)

}
```

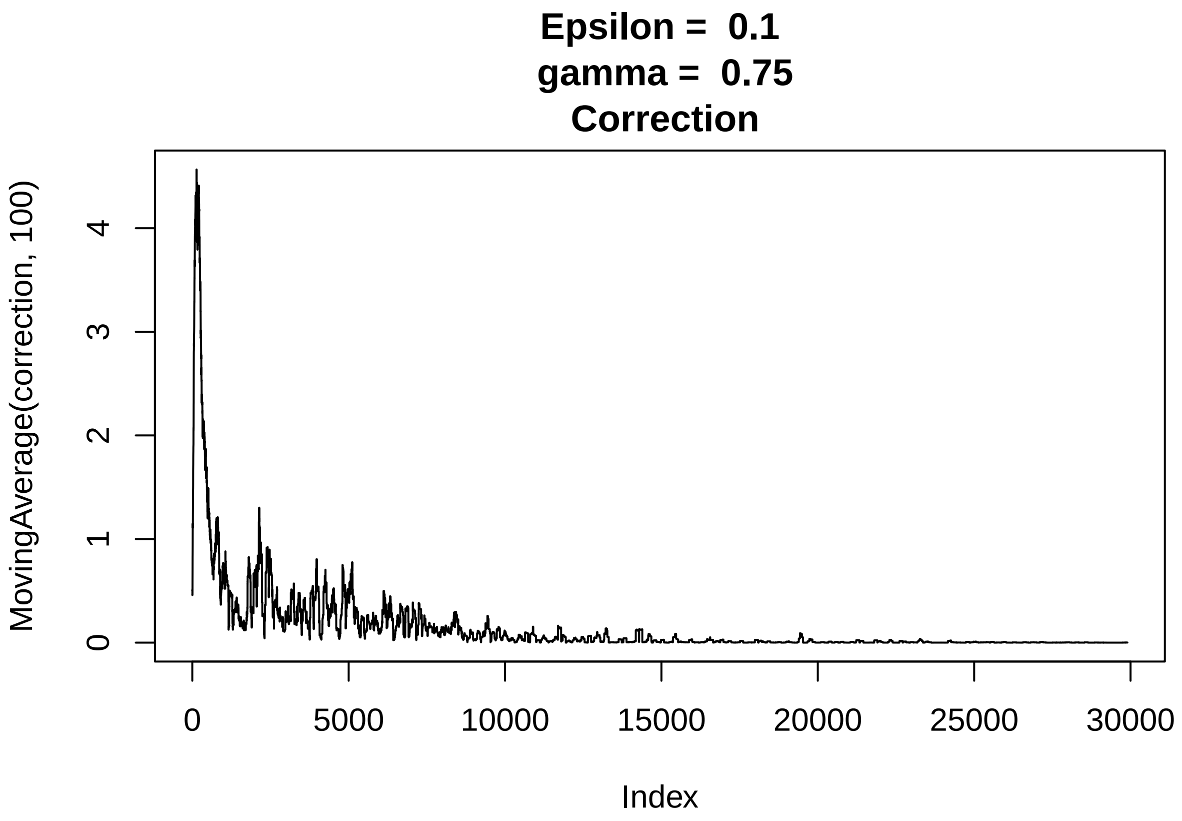

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10.0
7.5
5.0
2.5
5
10


Epsilon = 0.1
gamma = 0.5
Reward
MovingAverage(reward, 100)
0.0
0.5
1.0
1.5
0
5000
10000
15000
20000
25000
30000
Index

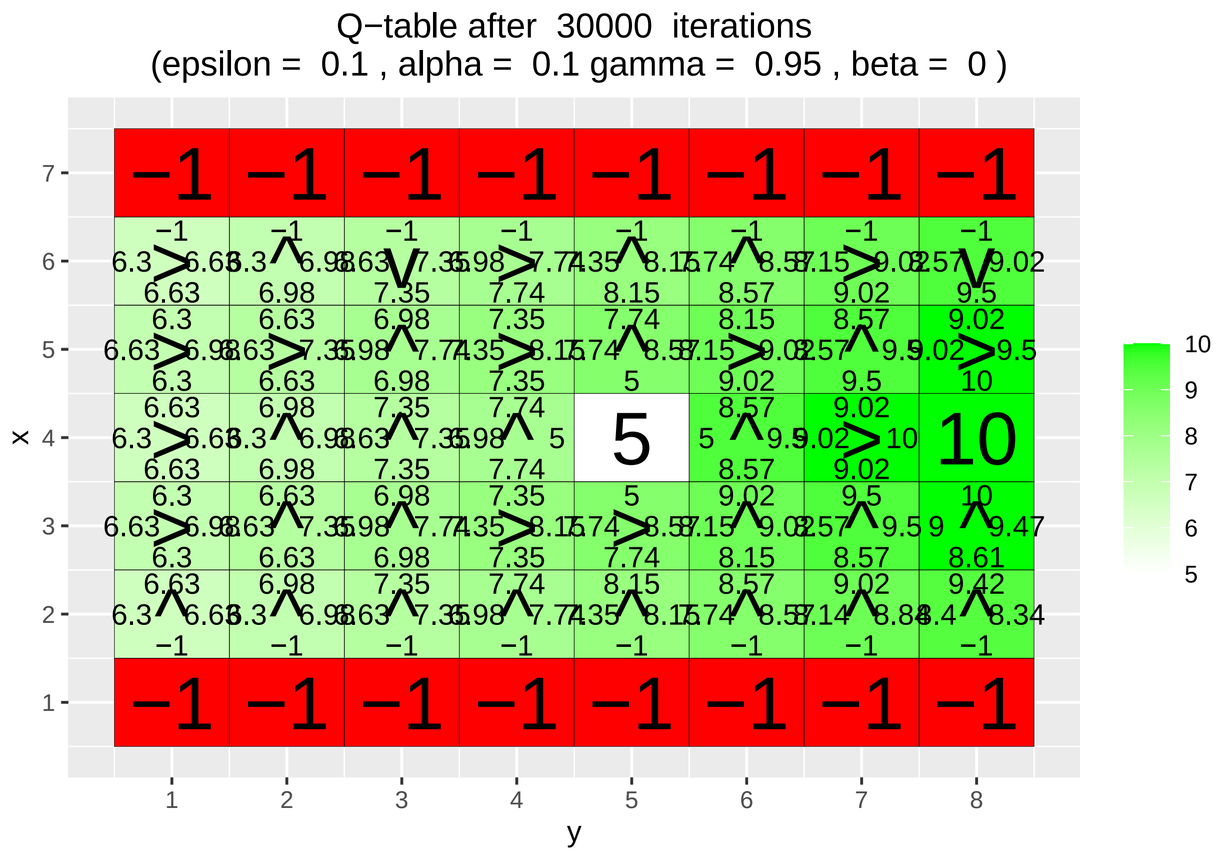

Epsilon = 0.1
gamma = 0.5
Correction
MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
0
5000
10000
15000
20000
25000
30000
Index

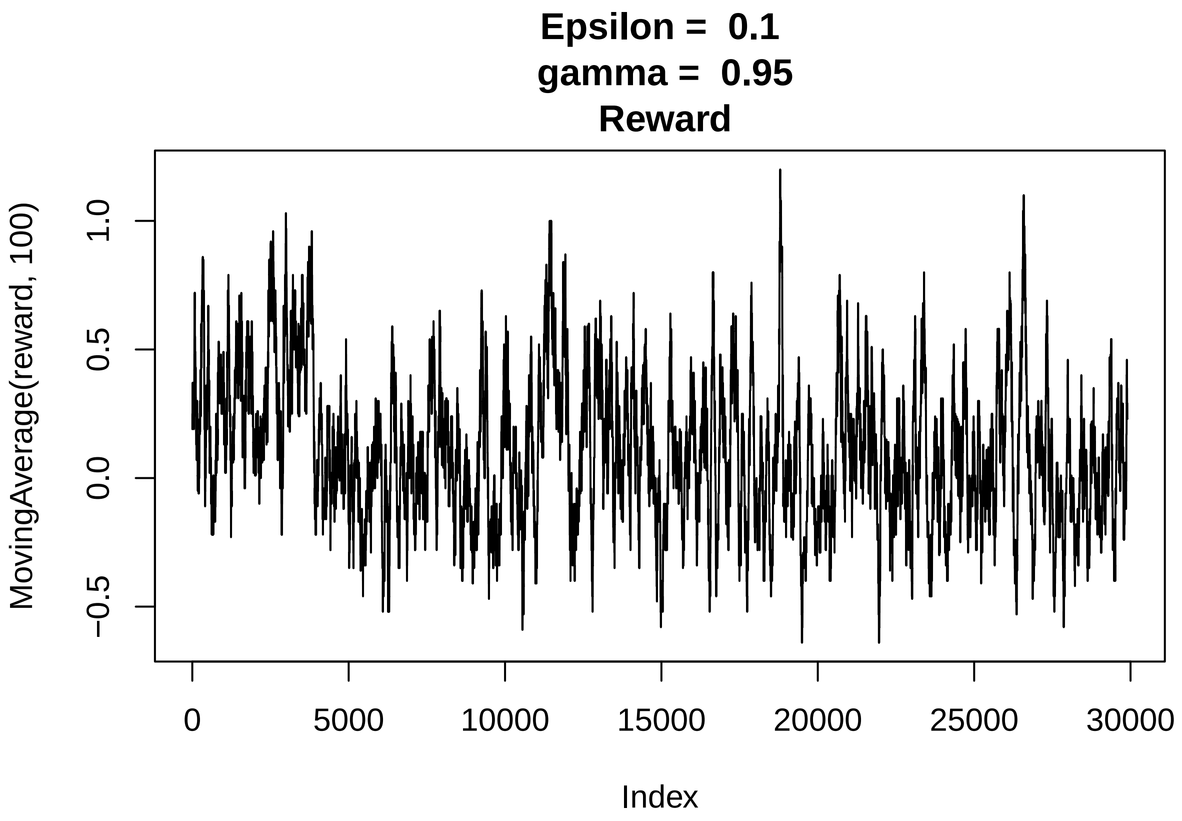

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5

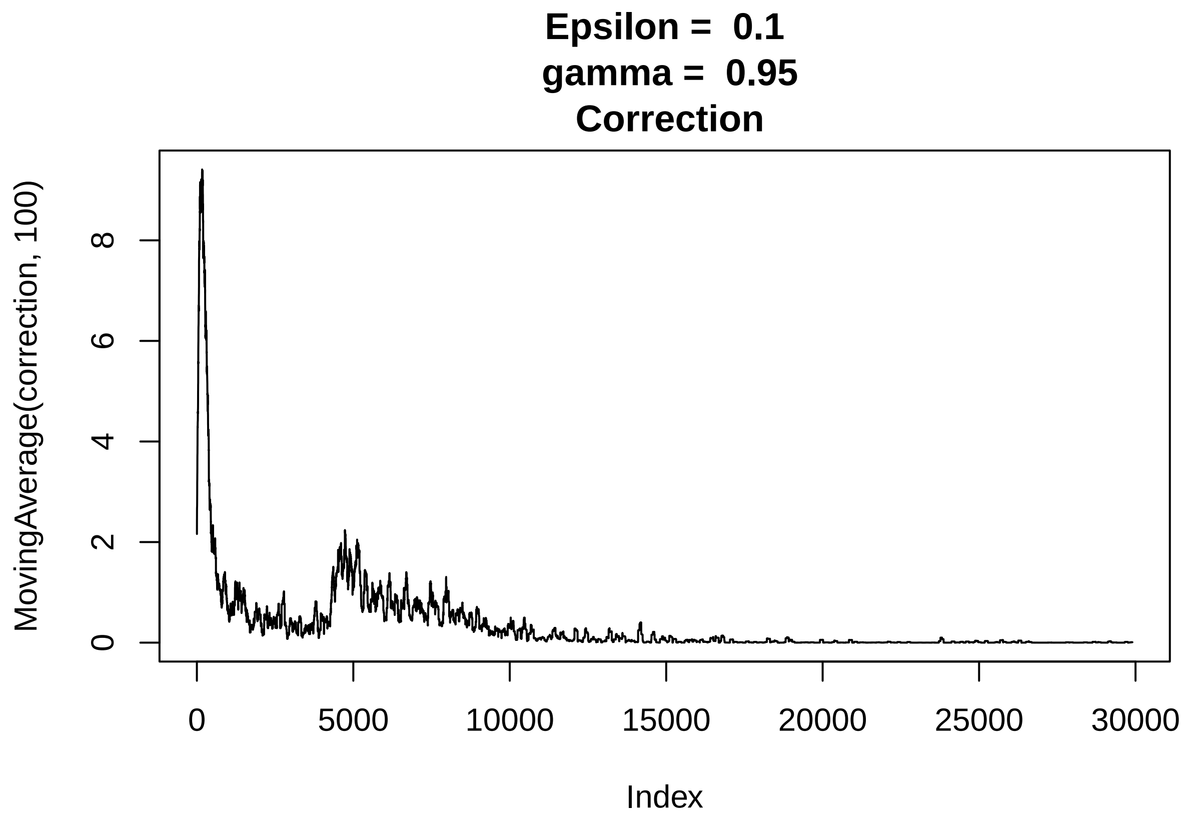

Epsilon = 0.1
gamma = 0.75
Reward
MovingAverage(reward, 100)
0.0
0.5
1.0
1.5
0
5000
10000
15000
20000
25000
30000
Index



Epsilon = 0.1
gamma = 0.75
Correction
MovingAverage(correction, 100)
0
1
2
3
4
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
5
10


Epsilon = 0.1
gamma = 0.95
Reward
MovingAverage(reward, 100)
1.0
0.5
0.0
−0.5
0
5000
10000
15000
20000
25000
30000
Index

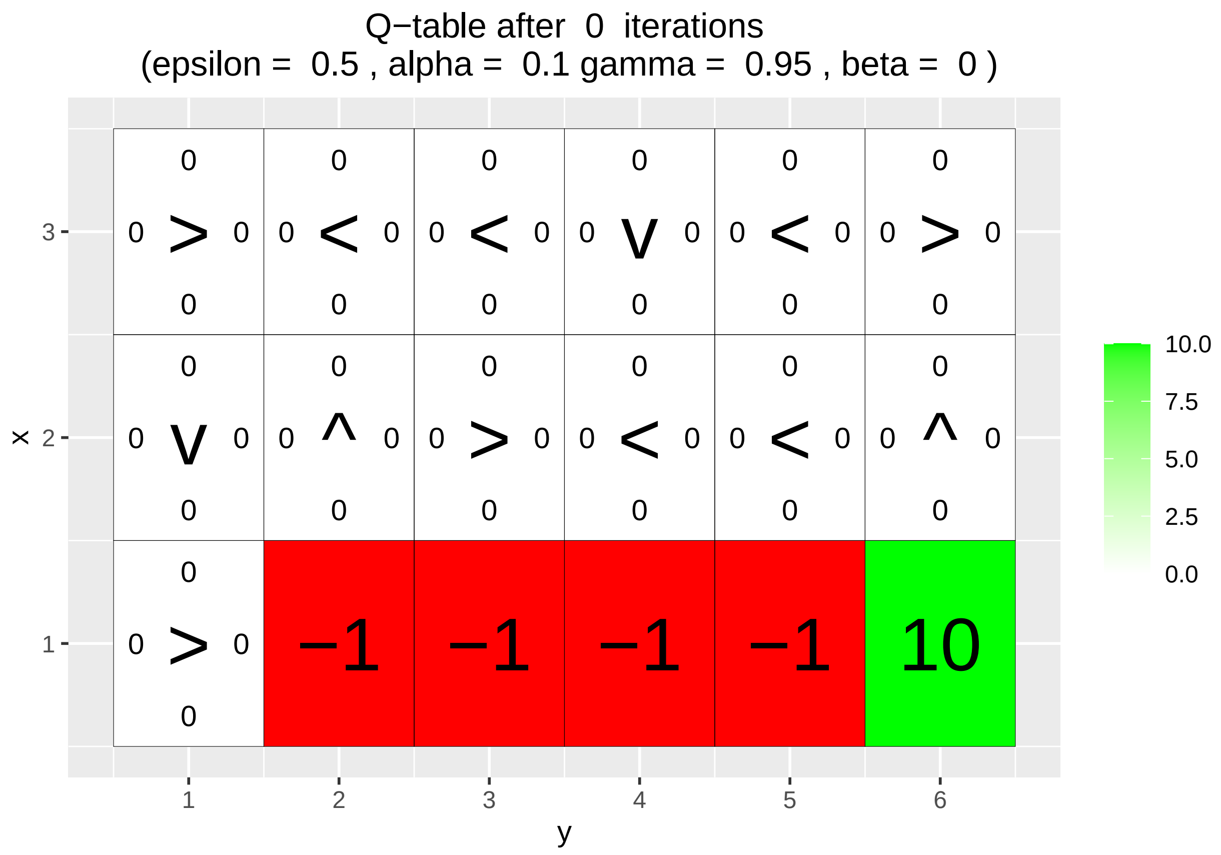


Gamma: A low gamma penalizes the agent based on the length of the path. Since 10 is further away from the start than 5. A lower gamma means the agent will focus on finding the path to 5. We also see that the value of the squares is much lower further away from the rewards when gamma is low

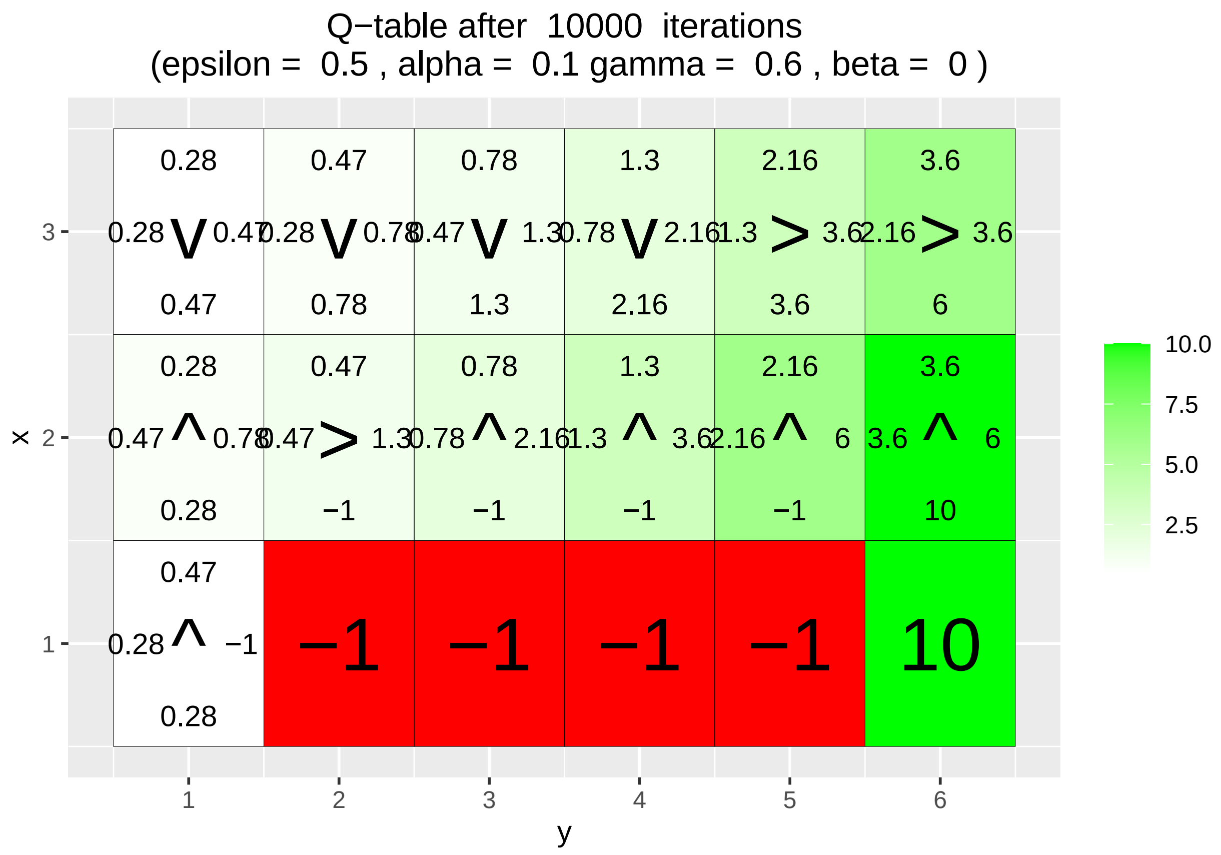
Espilon: “A higher epsilon makes the algorithm converge faster, e = .5 -> ~3000it. e = .1 -> ~15000it”

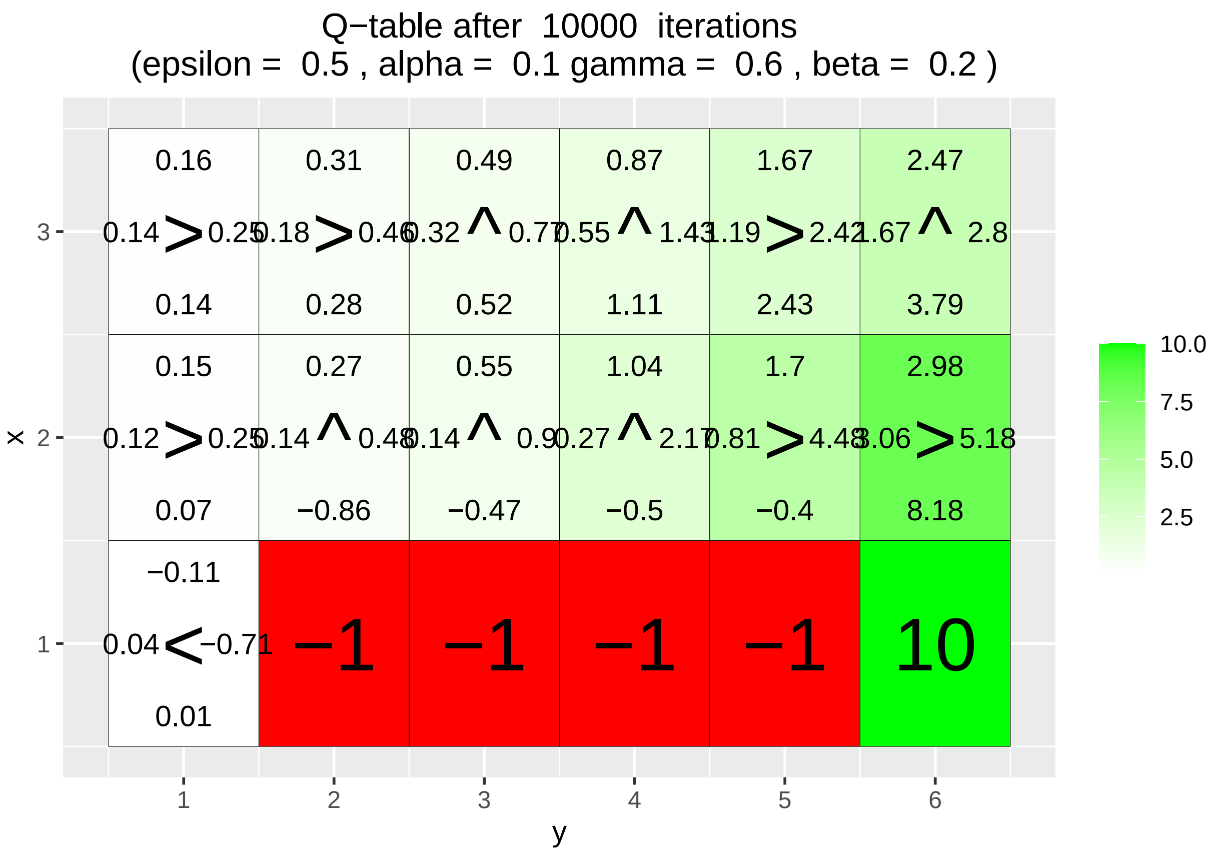
Environment C (the effect of beta).

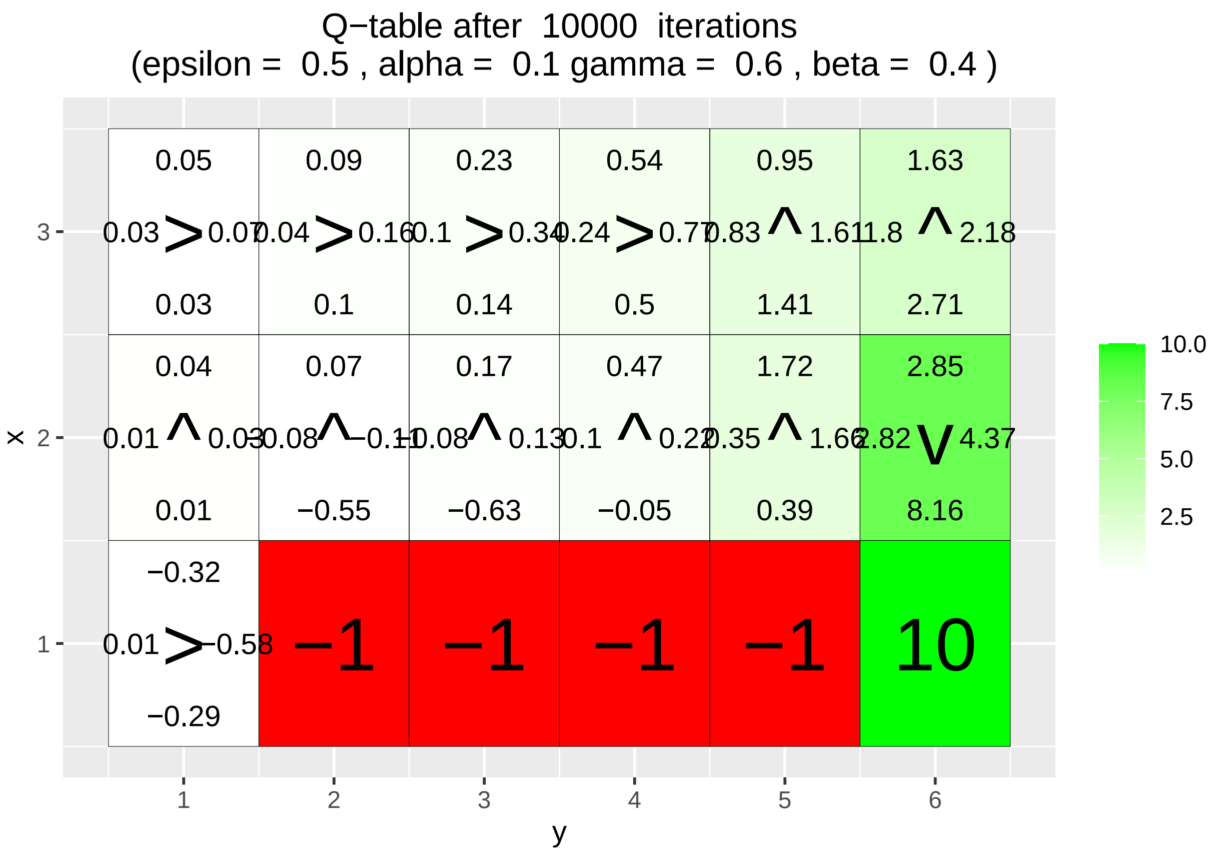
```r
set.seed(1337)
H <- 3
W <- 6

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()
```



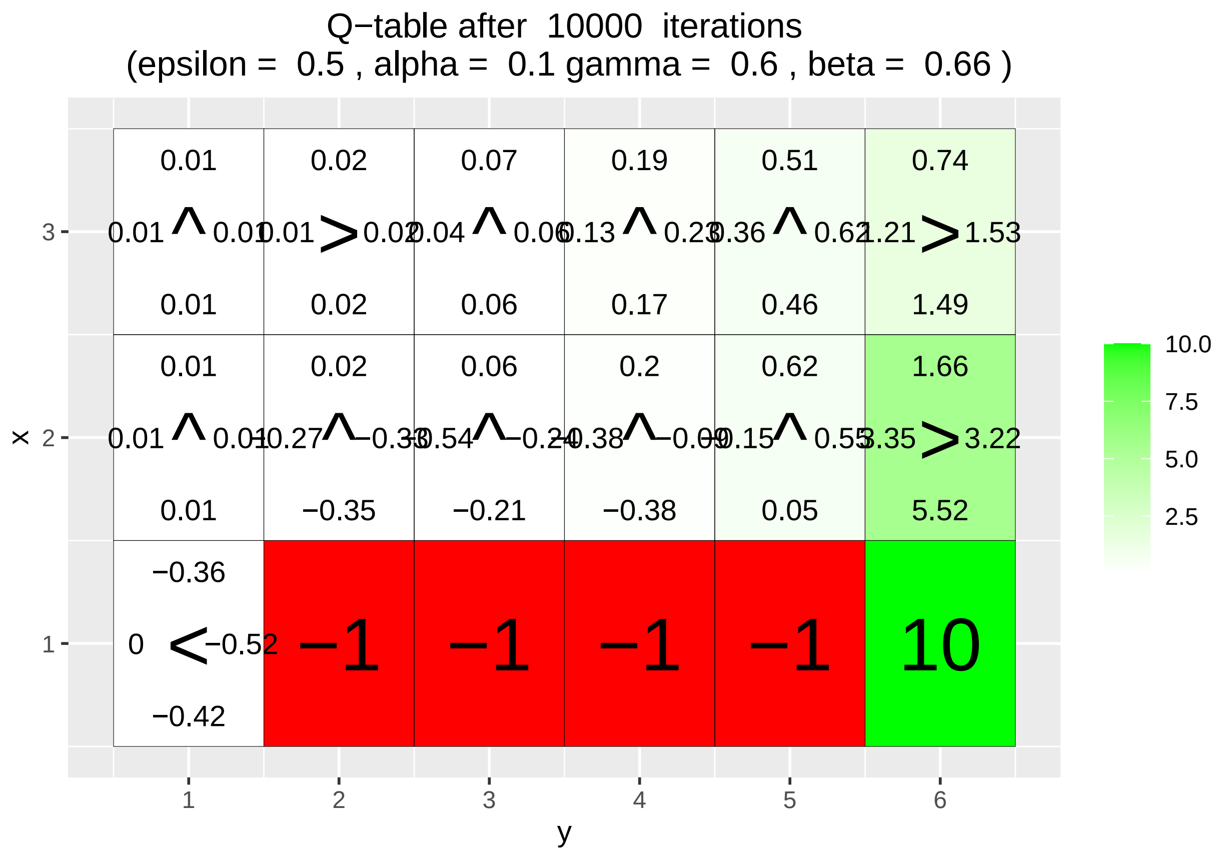
```r
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

  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)
x
y
1
2
3
4
5
6
10.0
7.5
5.0
2.5
−1
10


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)
0.16
0.14 > 0.25
0.14
0.31
0.18 > 0.46
0.28
0.49
0.32 ^ 0.77
0.52
0.87
0.55 ^ 1.43
1.11
1.67
1.19 > 2.42
2.43
2.47
1.67 ^ 2.8
3.79
0.15
0.12 > 0.25
0.07
0.27
0.14 ^ 0.48
−0.86
0.55
0.14 ^ 0.9
−0.47
1.04
0.27 ^ 2.17
−0.5
1.7
0.81 > 4.48
−0.4
2.98
3.06 > 5.18
8.18
−0.11
0.04 < −0.71
0.01
−1
−1
−1
−1
10
x
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)
0.05
0.03 > 0.07
0.03
0.09
0.04 > 0.16
0.1
0.23
0.1 > 0.34
0.14
0.54
0.24 > 0.77
0.5
0.95
0.83 ^ 1.61
1.41
1.63
1.8 ^ 2.18
2.71
0.04
0.01 ^ 0.03
0.01
0.07
0.08 ^ −0.11
−0.55
0.17
0.08 ^ 0.13
−0.63
0.47
0.1 ^ 0.22
−0.05
1.72
0.35 ^ 1.66
0.39
2.85
2.82 V 4.37
8.16
−0.32
0.01 > −0.58
−0.29
−1
−1
−1
−1
10
x
3
2
1
y
1
2
3
4
5
6
10.0
7.5
5.0
2.5



A high beta can be interpreted as the result of our actions being uncertain. This means we are incentivised to take a longer route, far away for the negative rewards when beta is high. In this run, gamma is substantial and there is some tension between wanting to take a short path and wanting to take a safe path. In general, a high beta will take us on a longer path, further from the negative rewards

Environment D

The agent does learn policies that lead to the goal. However, it is obvious that it doesn't understand the problem. The probabilities of going into the goal from an adjacent square are sometimes very clsoe to zero and the agent may heavily favor taking one route over another even though they are equal in length

For the Q-learning algorithm to handle new different environments the state has to be described in relative terms I.E. we are (dX, dY) squares from the goal, rather than: we are in square (x,y). This solution would, I think, work very well in these very specific circumstances where we do not have any obstacles. Although that problem is simple enough you could easily just calculate an optimal solution.

Envrionment E

The agent shows a very clear bias for moving upwards. This can obviously be attributed to the mismatch in training and validation data. The agent is simply not tested on the same type of data it has been trained on.

Again, when we train on such small datasets, and they show such clear differences the results are boudn to be very different. The first model shows no clear bias in what action to take,m wheras he second one does.