Advanced Machine Learning Lab 3

Simge Cinar

2024-10-06

First let's define the necessary functions

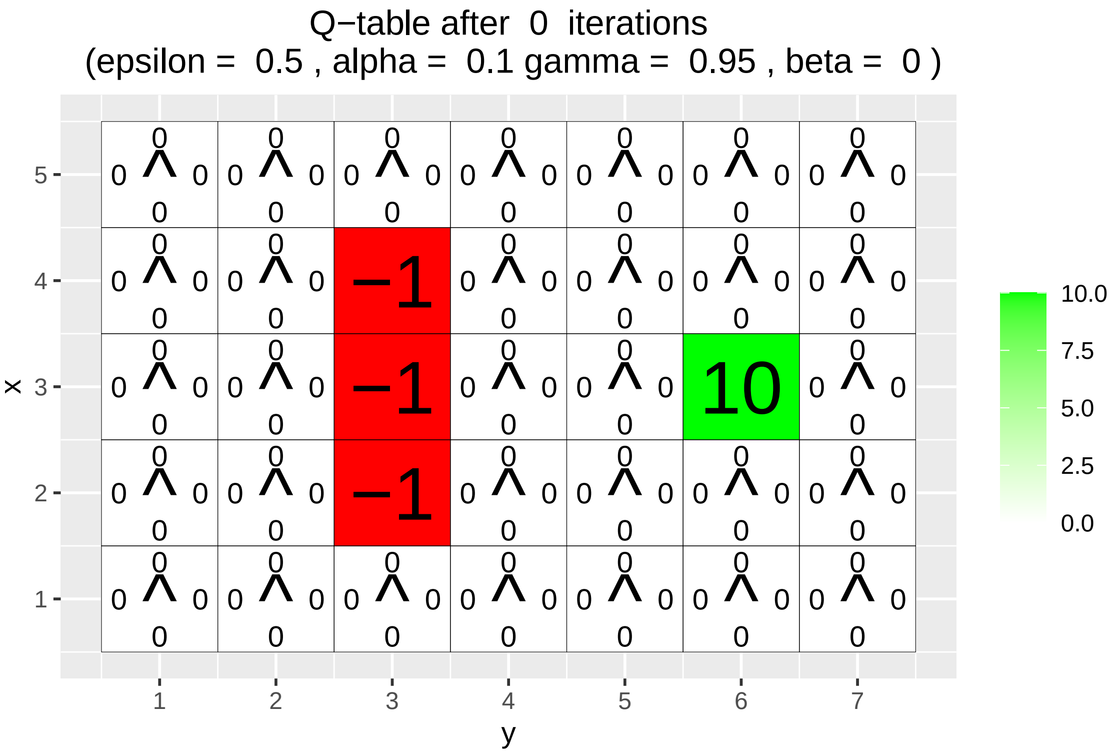
```r
GreedyPolicy <- function(x, y){

  # 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. SIMGE:
  q_values <- q_table[x, y, ] # gives the values of 4 actions for a specific location
  greedy_action <- which.max(q_values) # chooses the best action with maximum value
  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}.

  # Your code here. SIMGE:
  # explore with prob epsilon, greedy action wih prob 1-epsilon
  rand_num <- runif(1)
  action <- ifelse(rand_num < epsilon, sample(1:4, 1), GreedyPolicy(x,y))
  return(action)
}
```

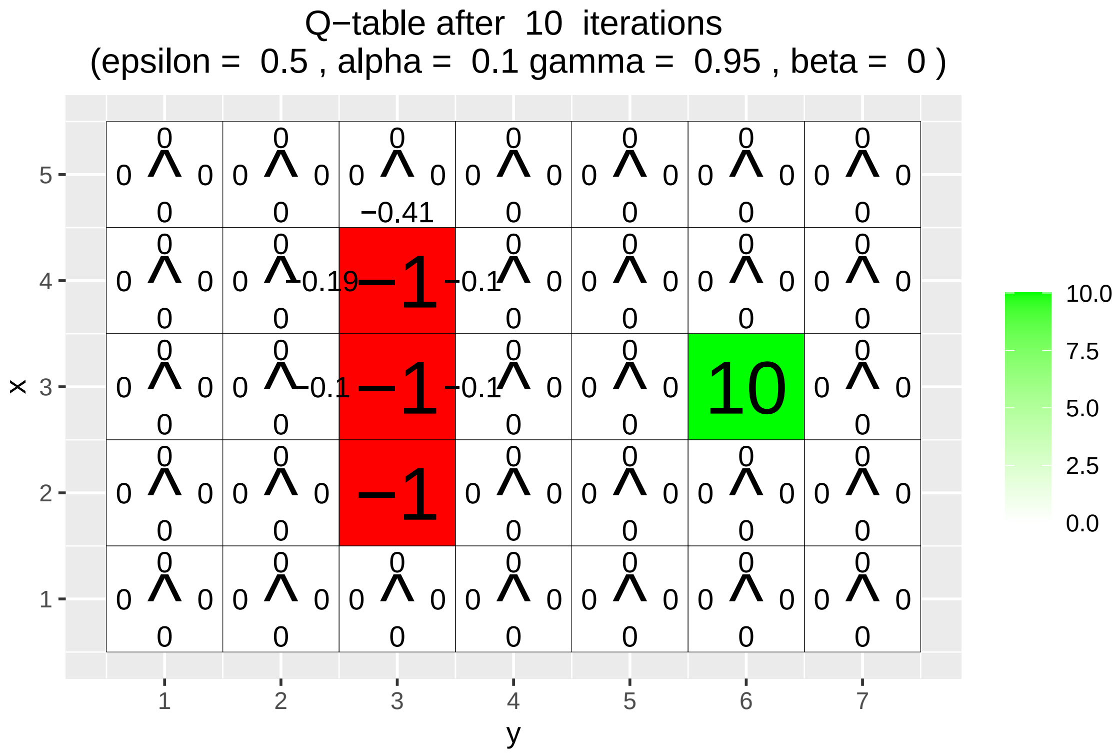
```r
q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95,
                       beta = 0){

  # 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 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. SIMGE:

  # Initialize the variables
  state_x <- start_state[1]
  state_y <- start_state[2]
  episode_correction <- 0

  repeat{
    # Get the action
    action <- EpsilonGreedyPolicy(state_x, state_y, epsilon)

    # Get the new states after taking the action, goes to selection location with prob 1-beta
    new_state <- transition_model(state_x, state_y, action, beta)
    state_x_new <- new_state[1]
    state_y_new <- new_state[2]

    # Update the reward map
    reward <- reward_map[state_x_new, state_y_new]

    # Update Q-table
    step_correction <- reward + gamma * max(q_table[state_x_new, state_y_new, ]) - q_table[state_x, stat
    q_table[state_x, state_y, action] <<- q_table[state_x, state_y, action] + alpha * step_correction

    # Update total temporal difference correction
    episode_correction <- episode_correction + abs(step_correction)

    # Reset the state
    state_x <- state_x_new
    state_y <- state_y_new

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

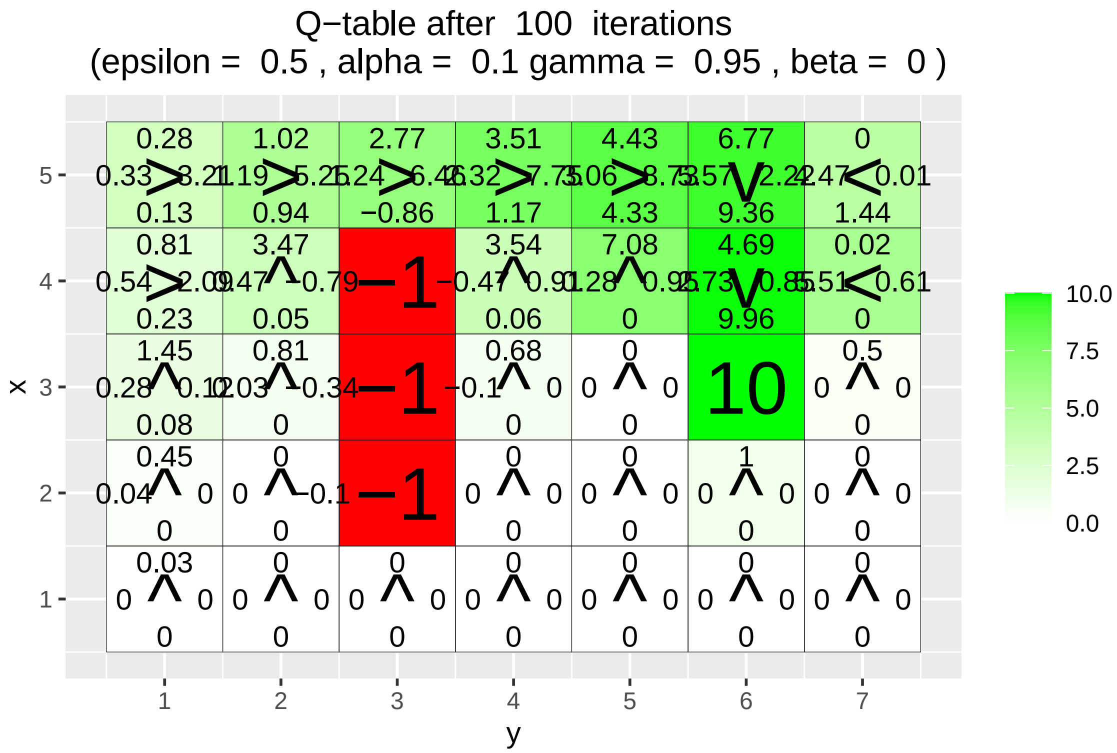
Question 2.2 - Environment A

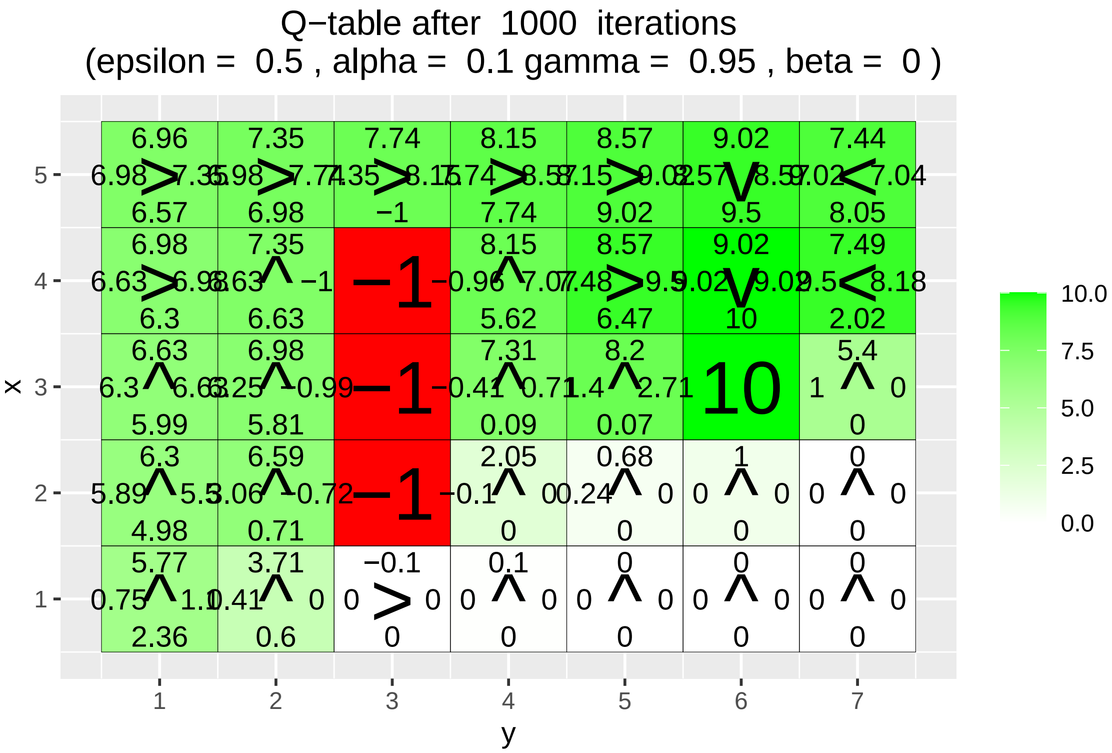
```r
H <- 5
W <- 7

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()
```

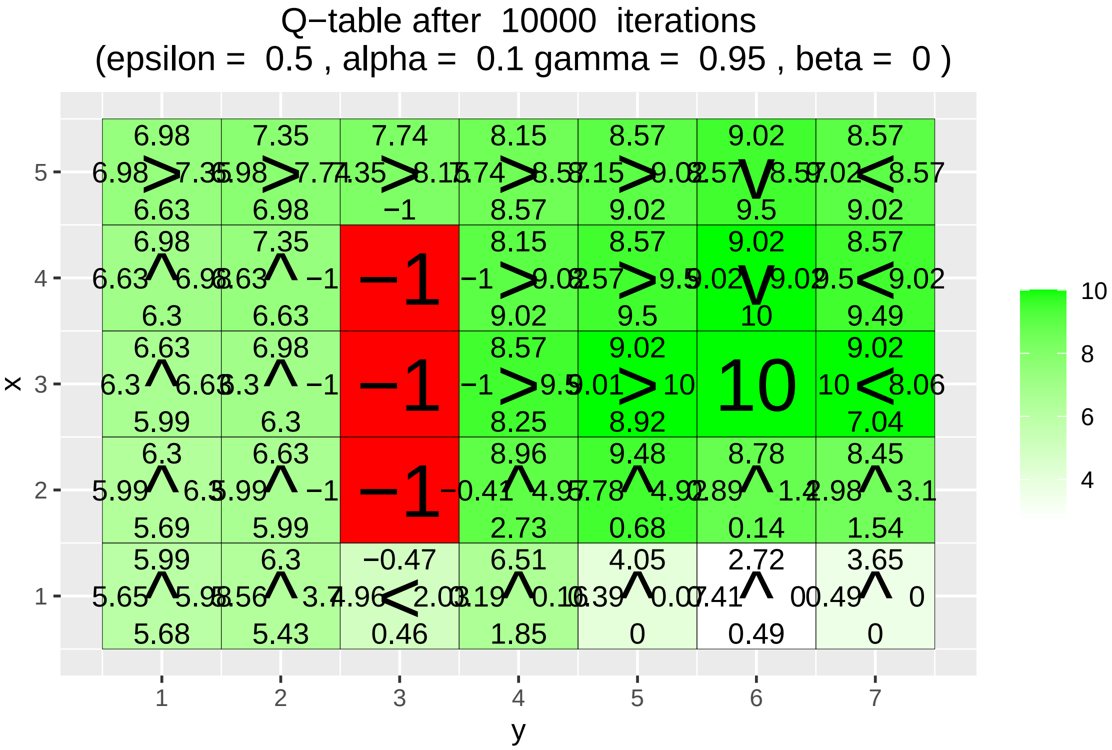


```r
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))

  if(any(i==c(10,100,1000,10000)))
    vis_environment(i)
    cat("\n\n")
}
```


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)





What has the agent learned after the first 10 episodes ?

The agent doesn't learn much after 10 iterations. It only learned the some parts of the obstacles in the reward map.

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

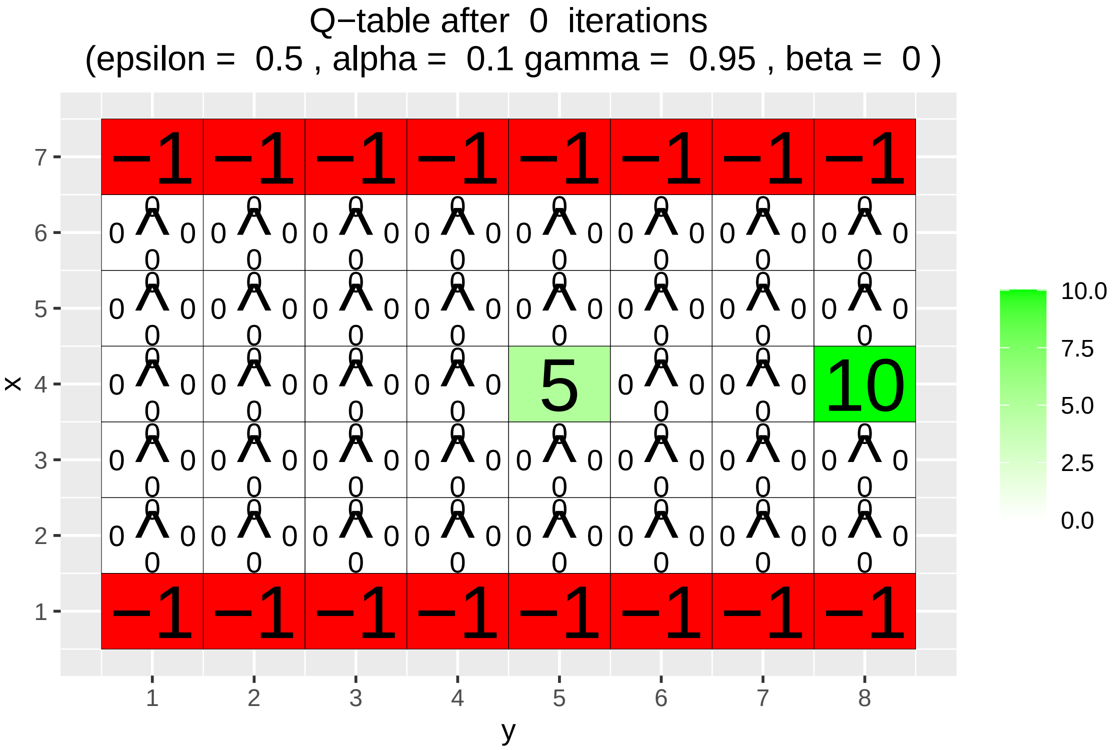
No, even though the agent learns to find the reward, it tends to take a longer path when it starts below the obstacle, specifically from the state (1,3).

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 ?

The Q-table assigns higher values to safer paths but agent learns only one path to reach to goal when it is left side of the block. Decreasing epsilon value more in time could give the agent explore more in the beginning and exploit more in the end.

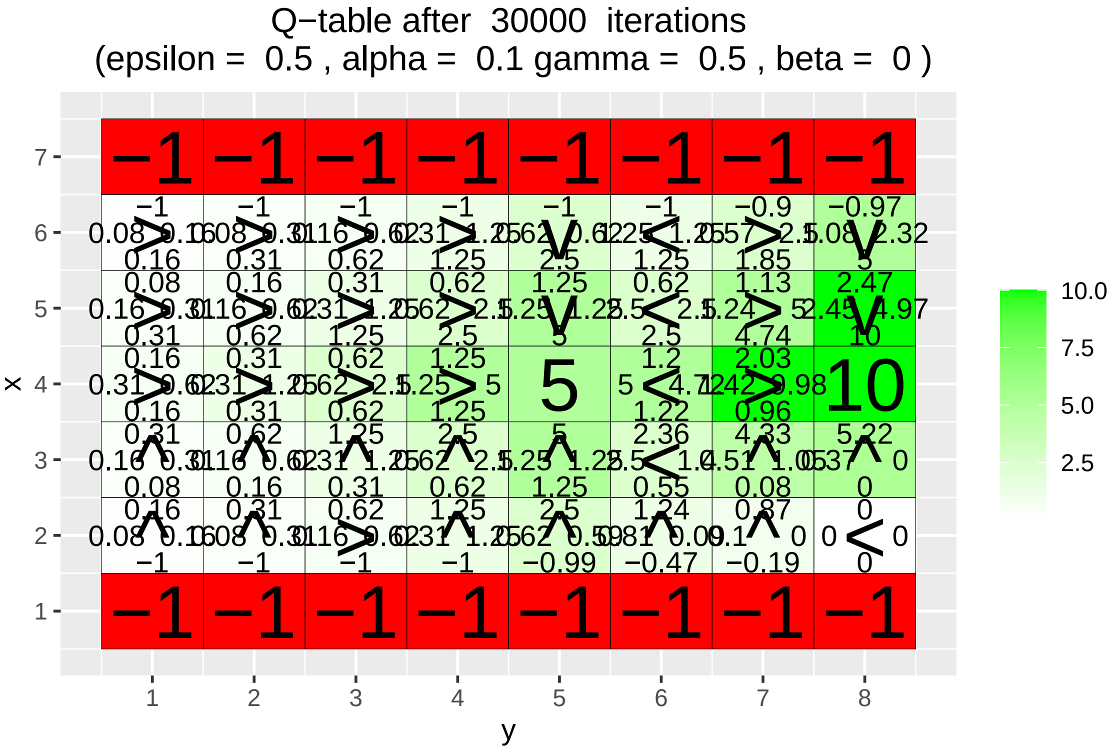
Question 2.3 - Environment B

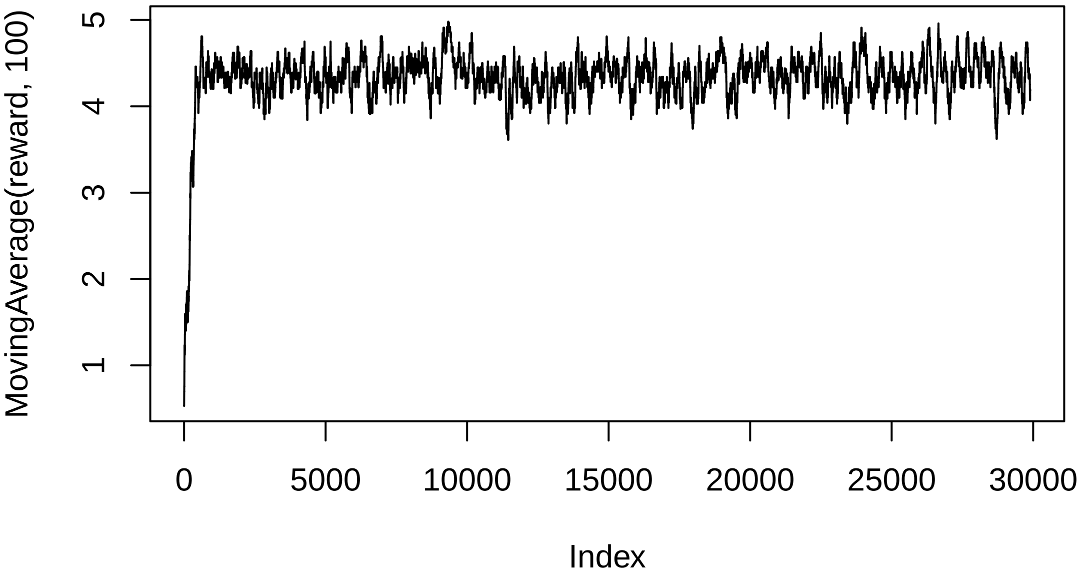
```
H <- 7
W <- 8

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

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

vis_environment()
```

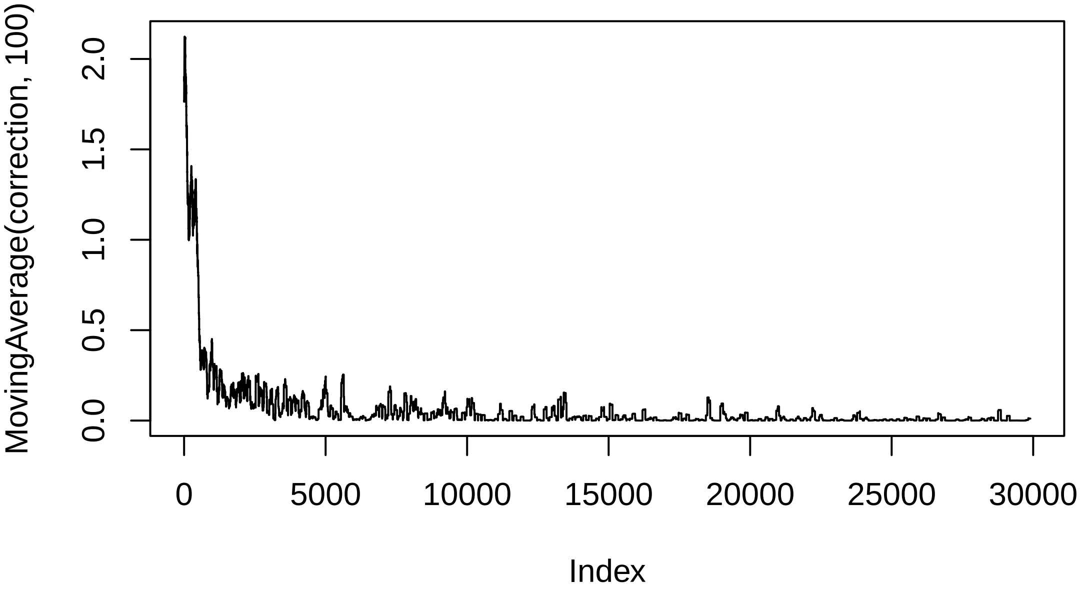


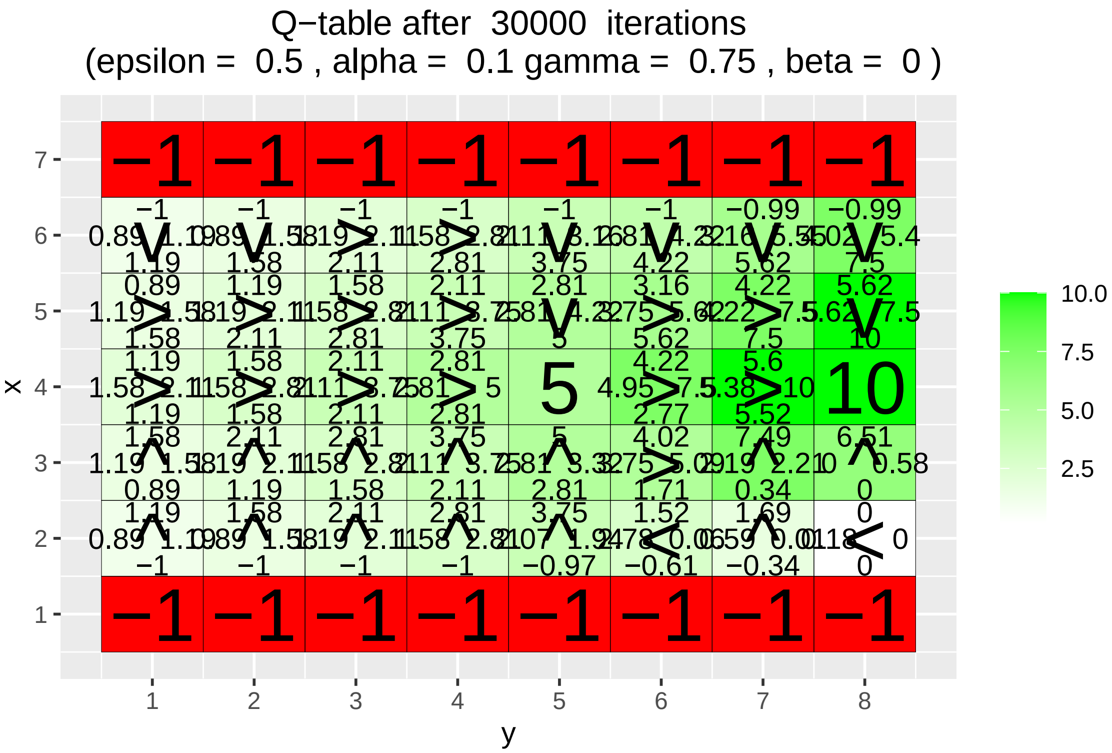
```
MovingAverage <- function(x, n){

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

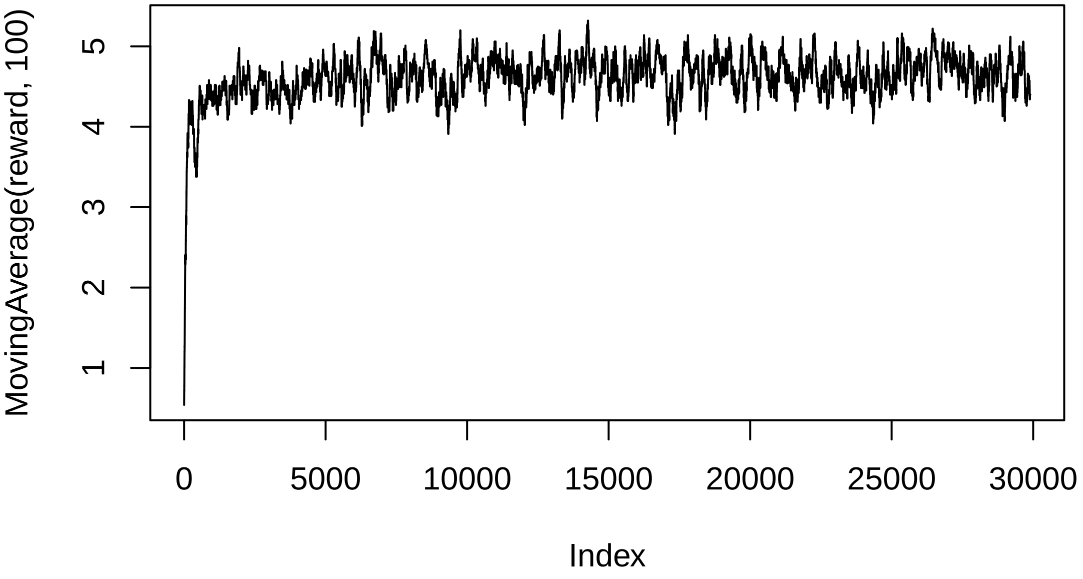
```r
  return (rsum)
}

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)
  cat("\n\n")
  plot(MovingAverage(reward,100),type = "l")
  cat("\n\n")
  plot(MovingAverage(correction,100),type = "l")
  cat("\n\n")
}
```

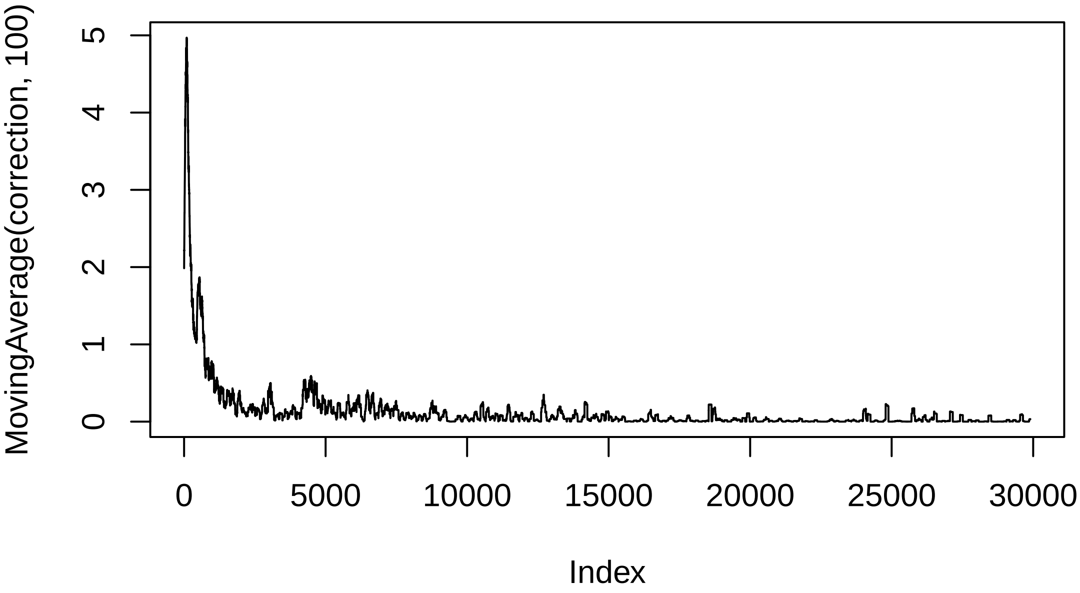


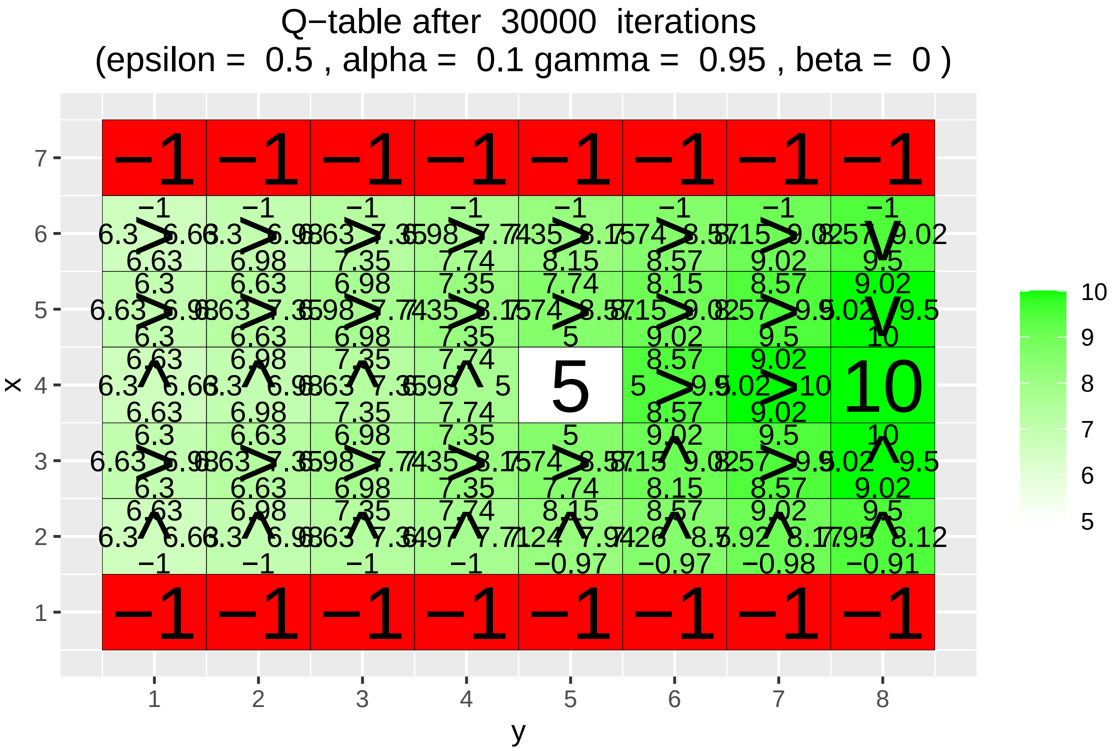

MovingAverage(reward, 100)
1
2
3
4
5
0
5000
10000
15000
20000
25000
30000
Index

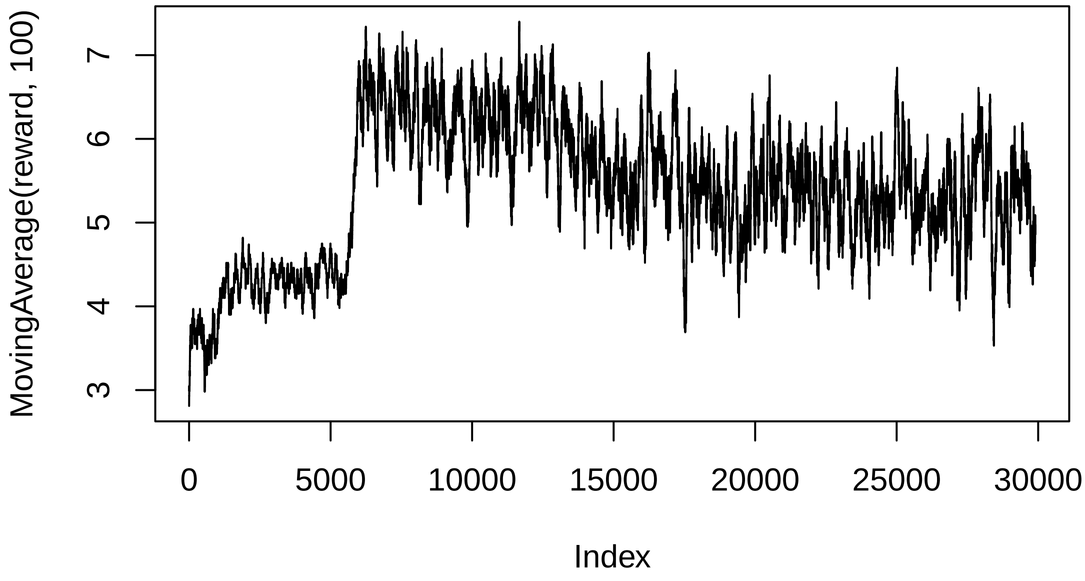

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index

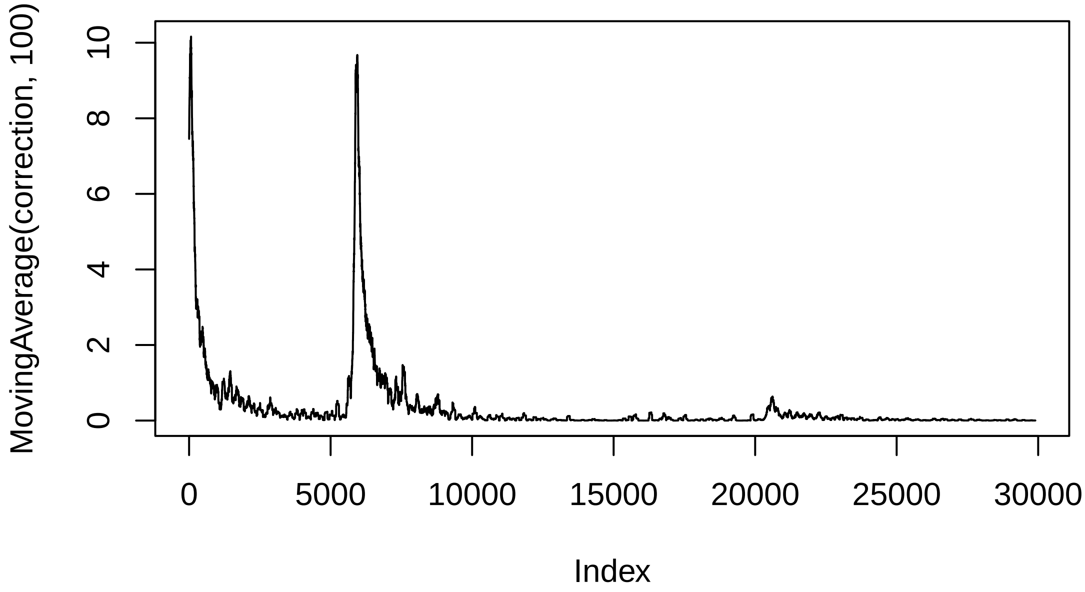

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
1
2
3
4
5
6
7
8
−1
5
10
10.0
7.5
5.0
2.5


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

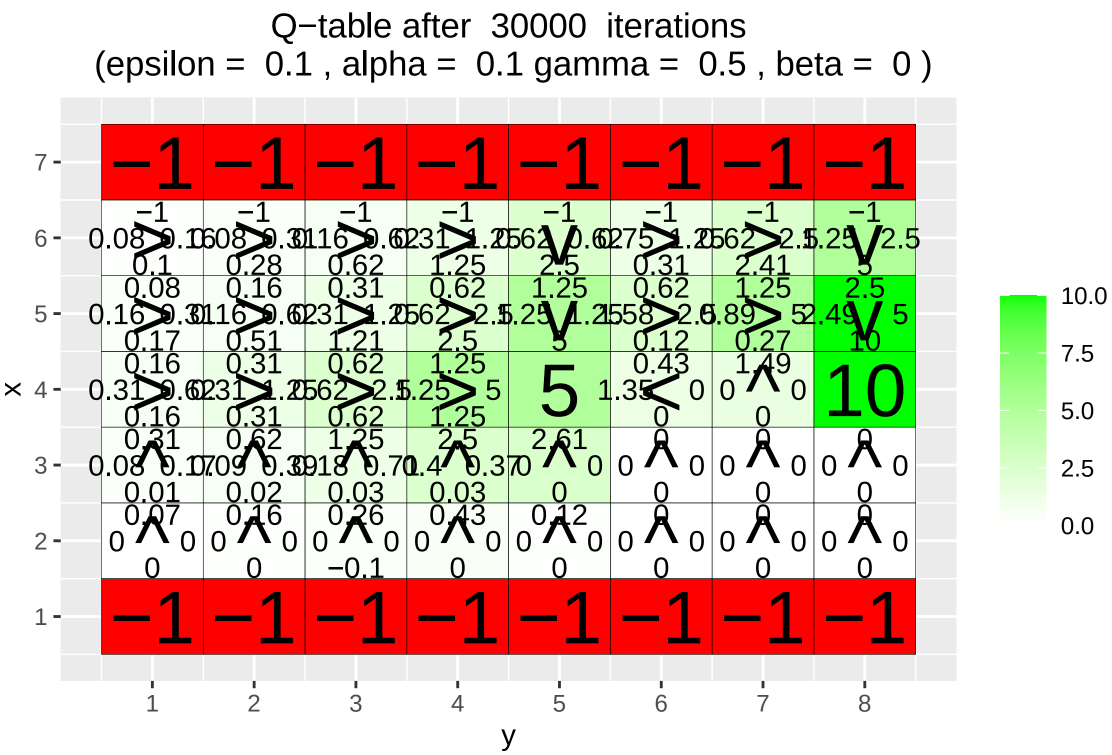

MovingAverage(correction, 100)
0
1
2
3
4
5
0
5000
10000
15000
20000
25000
30000
Index

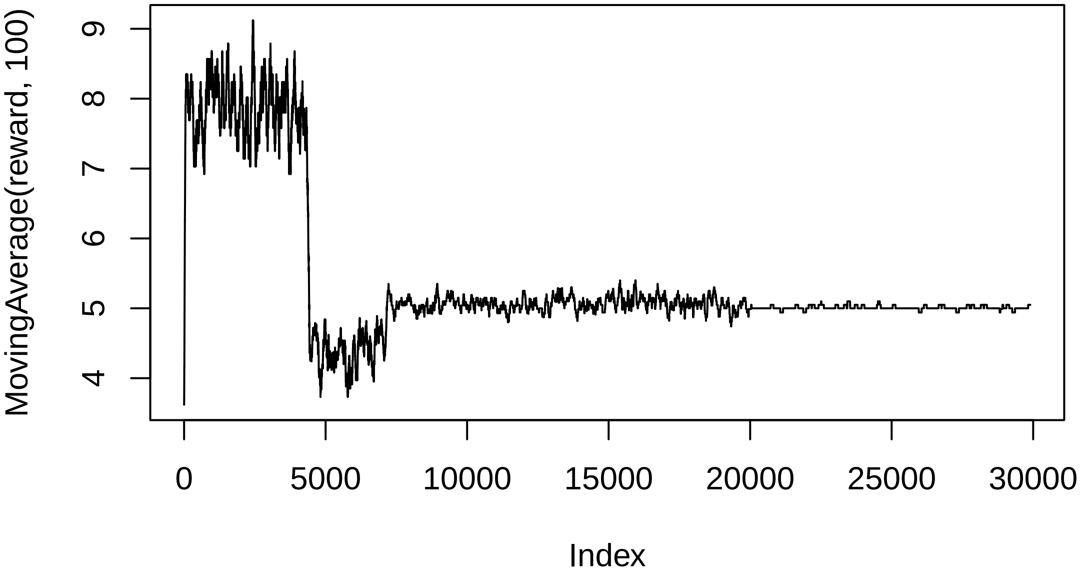

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
1
2
3
4
5
6
7
8
−1
5
10
10
9
8
7
6
5

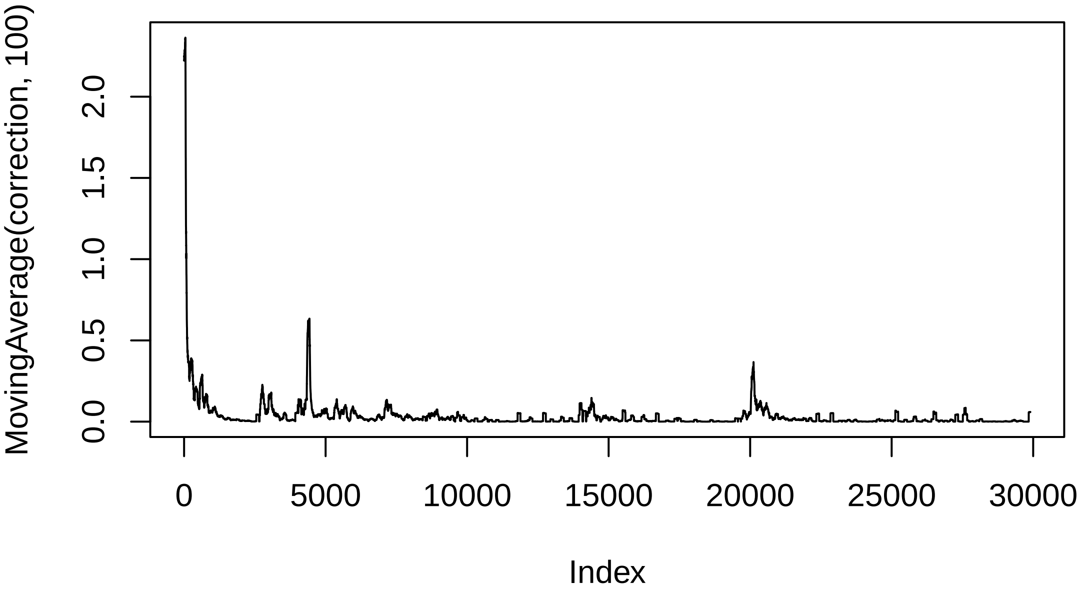


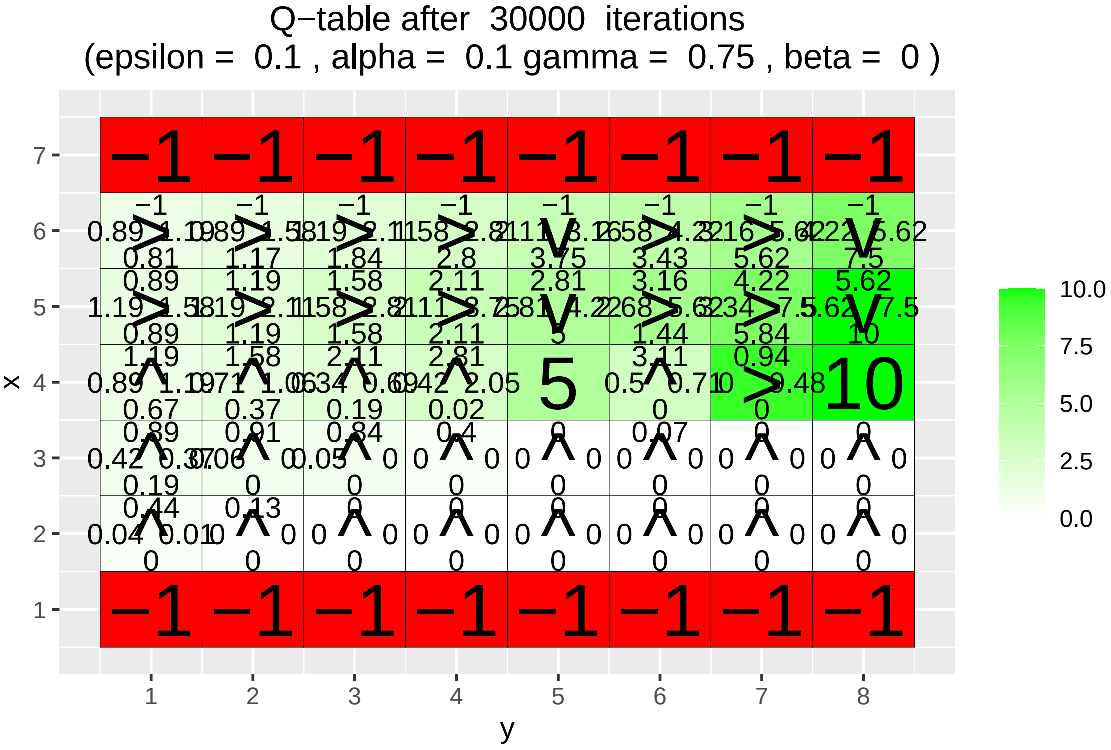


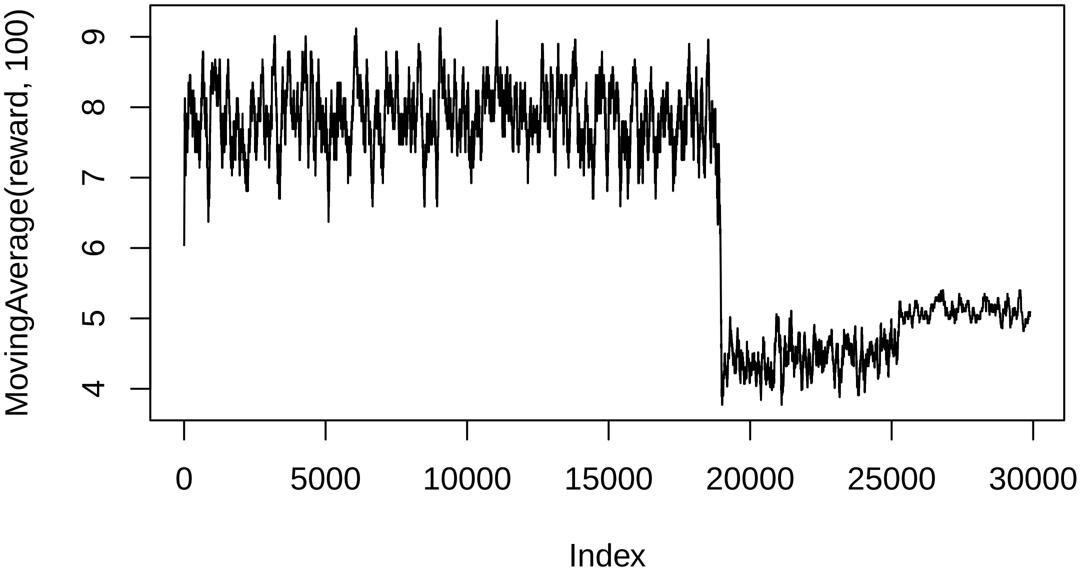
```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  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)
  cat("\n\n")
  plot(MovingAverage(reward,100),type = "l")
  cat("\n\n")
  plot(MovingAverage(correction,100),type = "l")
  cat("\n\n")
}
```

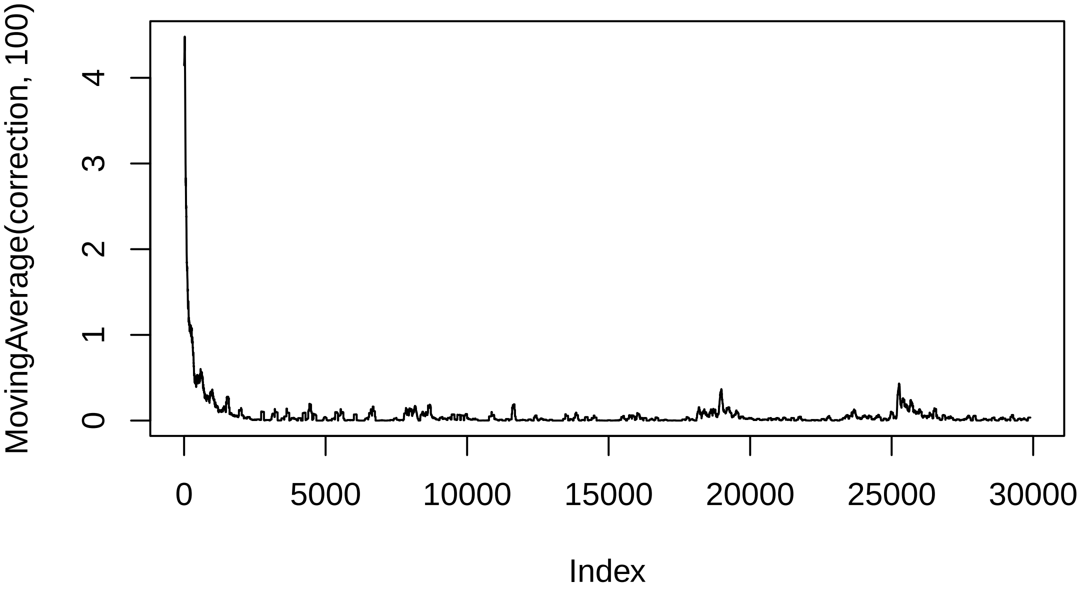

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
1
2
3
4
5
6
7
8
10.0
7.5
5.0
2.5
0.0
−1
5
10

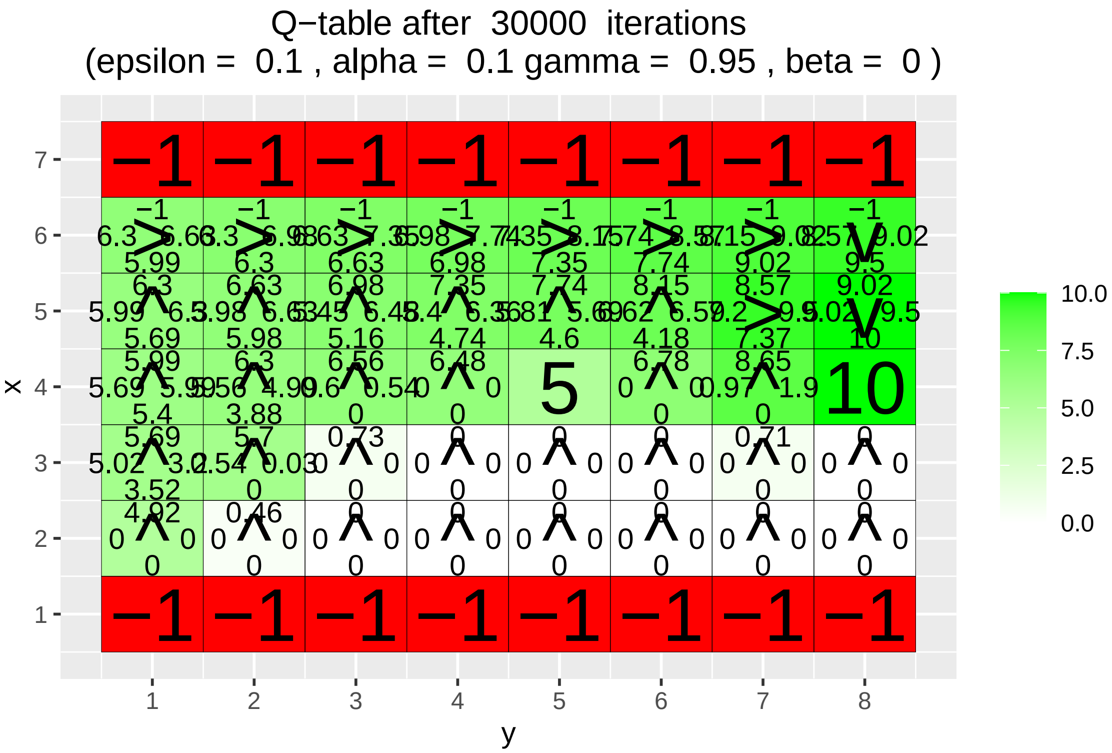

MovingAverage(reward, 100)
Index
4
5
6
7
8
9
0
5000
10000
15000
20000
25000
30000

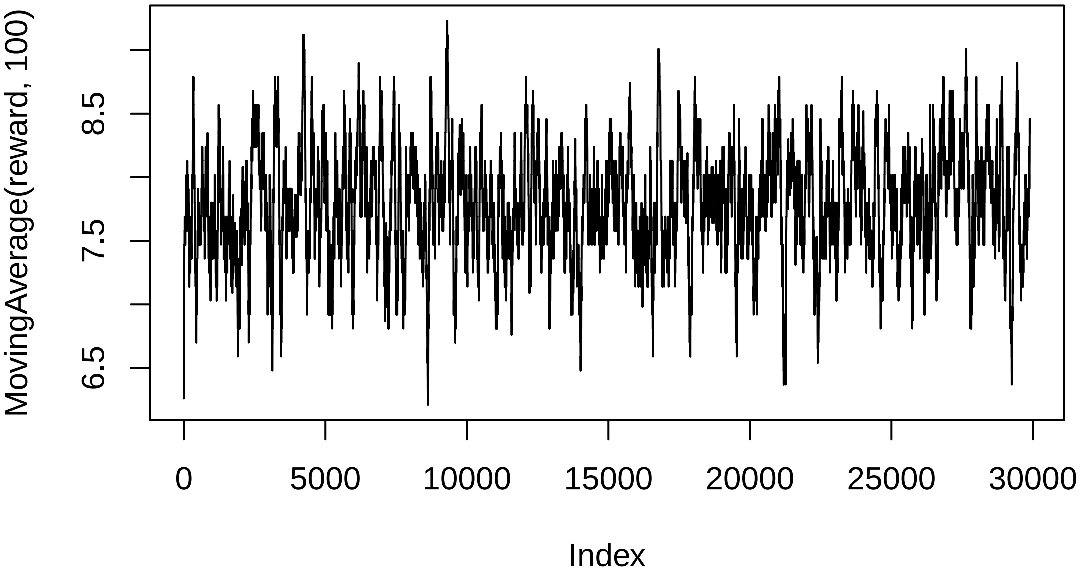

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index

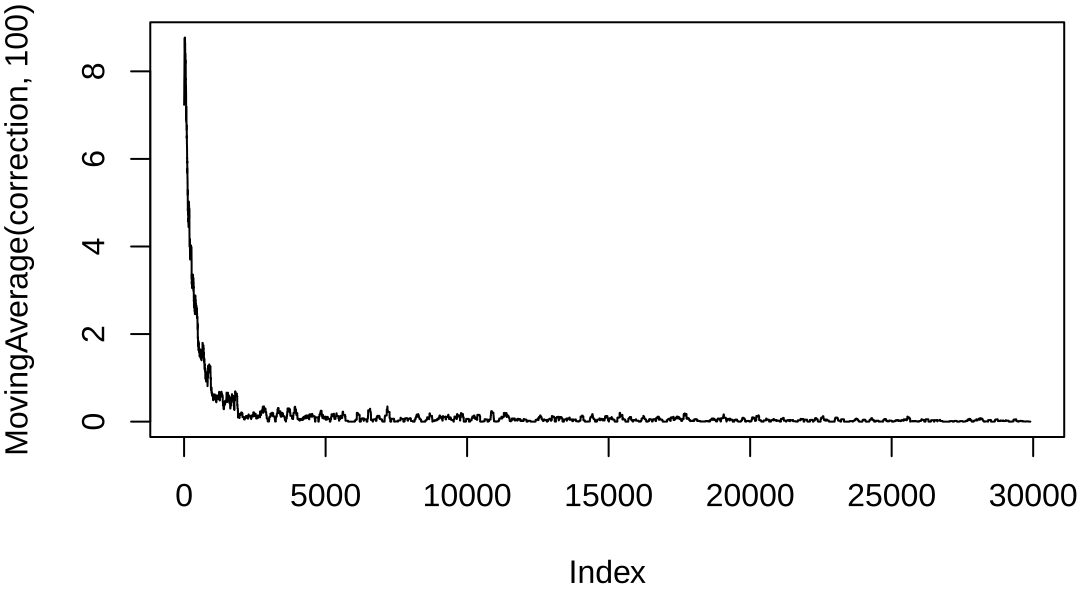

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
7
6
5
4
3
2
1
y
1
2
3
4
5
6
7
8
−1
5
10
10.0
7.5
5.0
2.5
0.0


MovingAverage(reward, 100)
4
5
6
7
8
9
0
5000
10000
15000
20000
25000
30000
Index


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


Q−table after 30000 iterations
(epsilon = 0.1 , 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


MovingAverage(reward, 100)
6.5
7.5
8.5
0
5000
10000
15000
20000
25000
30000
Index



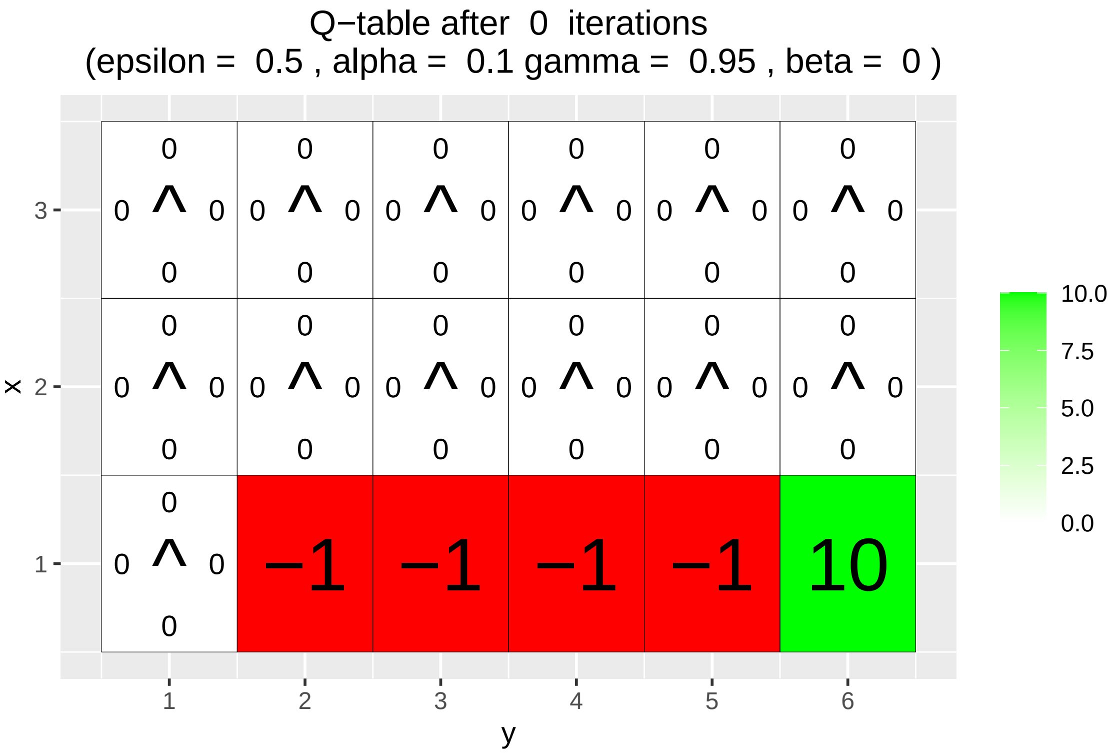
Investigate how ϵ and γ parameters affect the learned policy by running 30000 episodes of Q-learning with $\epsilon = 0.1, 0.5$, $\gamma = 0.5, 0.75, 0.95$, $\beta = 0$ and and $\alpha = 0.1$.

ϵ parameter determines how often the agent explores new actions versus exploiting the best-known action and γ parameter determines how much future rewards are taken into account. When $\epsilon = 0.5$ and $\gamma = 0.5$, the agent learns the path for the reward 5, it can only learn the path to reward 10 when it is near the goal with reward 10. In some states, even though goal with reward 10 is closer, the path leads to reward 5, such as in state (2,7). While keeping the ϵ parameter same and increasing the γ parameter, we observe that agent takes into account future reward more and in the case where $\gamma = 0.95$, all states leads to goal with reward 10.

When $\epsilon = 0.1$, the agent chooses random action 10% of the time. It can be observed that even though the agent can learn the path to goal with reward 10 when γ value is high, it chooses the longer path.

Question 2.4 - Environment C

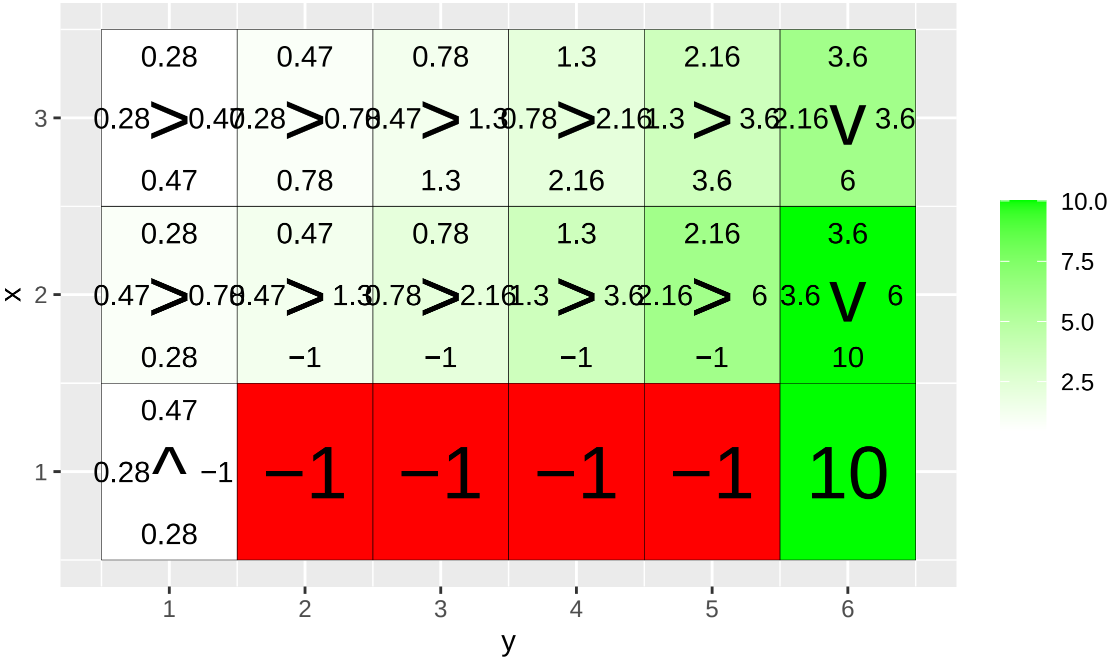
```
H <- 3
W <- 6

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()
```



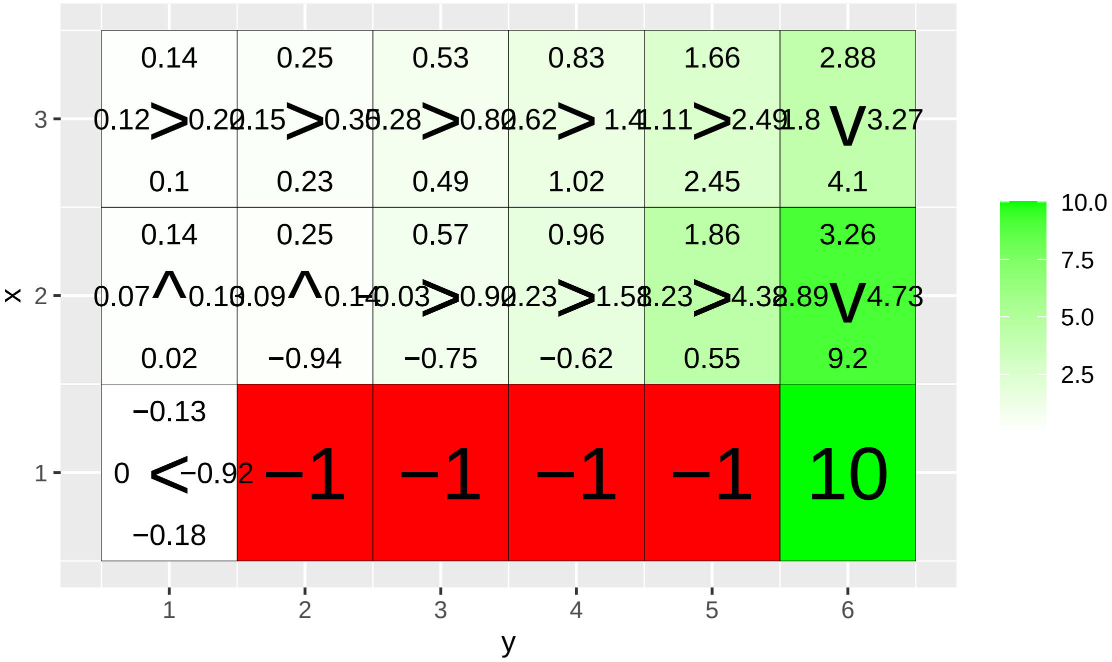
```r
cat("\n\n")
```

```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)
  cat("\n\n")
}
```

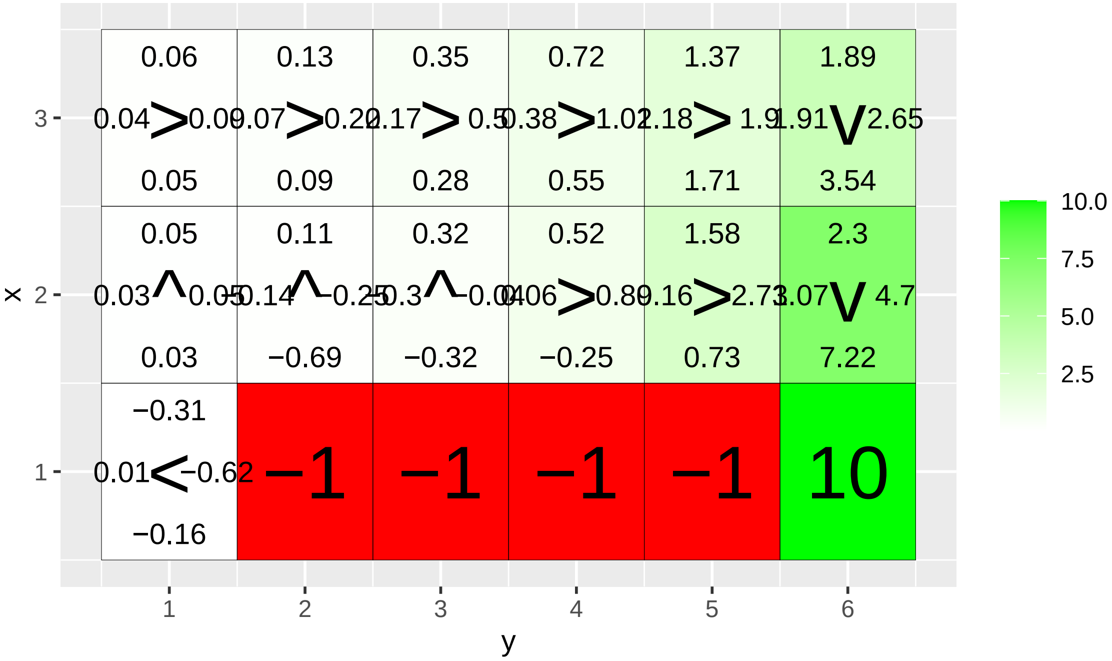
Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0)



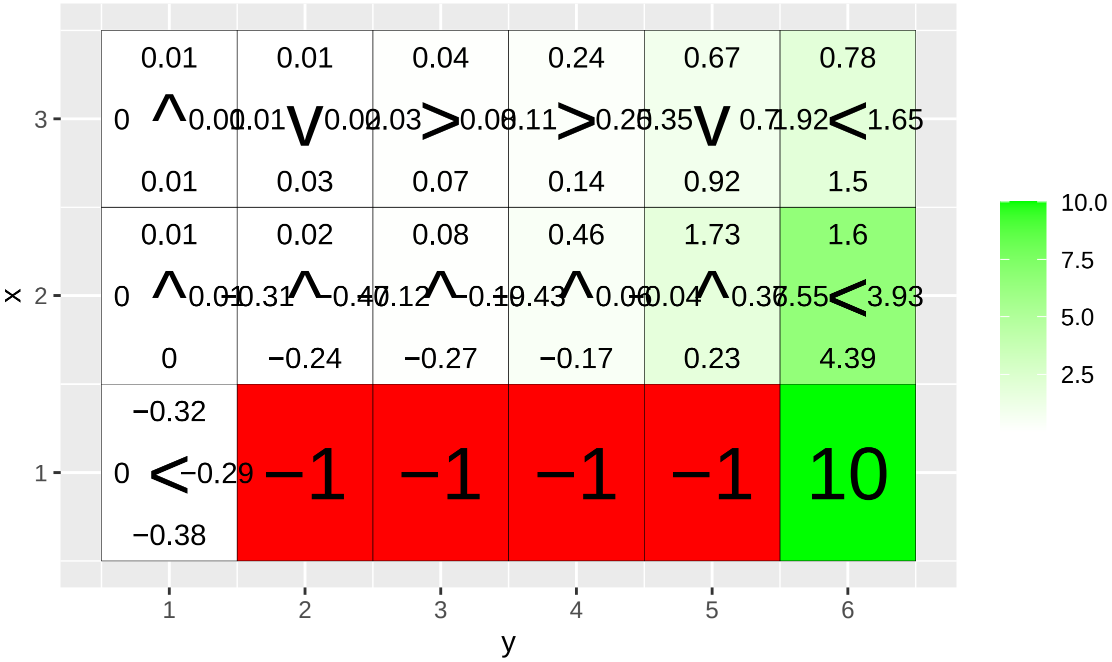
Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)



Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.4)



Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.66)



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 $\alpha = 0.1$.

The agent follows the intended action with probability $(1 - \beta)$. With high β value, the agent frequently slips and making the environment more stochastic. It can be observed that when $\beta = 0.2$, it cannot find the reward on the state (1,1) - left side of the obstacle.

Question 2.6 - Environment D

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

We can say that the agent learned a good policy because almost all initial states lead to goal, except for validation data 1 state (x = 4, y = 1)

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

This problem can be solved using Q-learning since it's not too big but we need a new reward map each time the goal changes it's location and it's not memory efficient.

Question 2.7 - Environment E

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

No it couldn't learn a good policy because even the states near the goal cannot lead to the goal

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

Yes, they differ because in environment E, the model is trained with only top row goal positions hence generally the best policy is to move up for the agent