lab3

Shipeng Liu

2023-10-02

Q-Learning

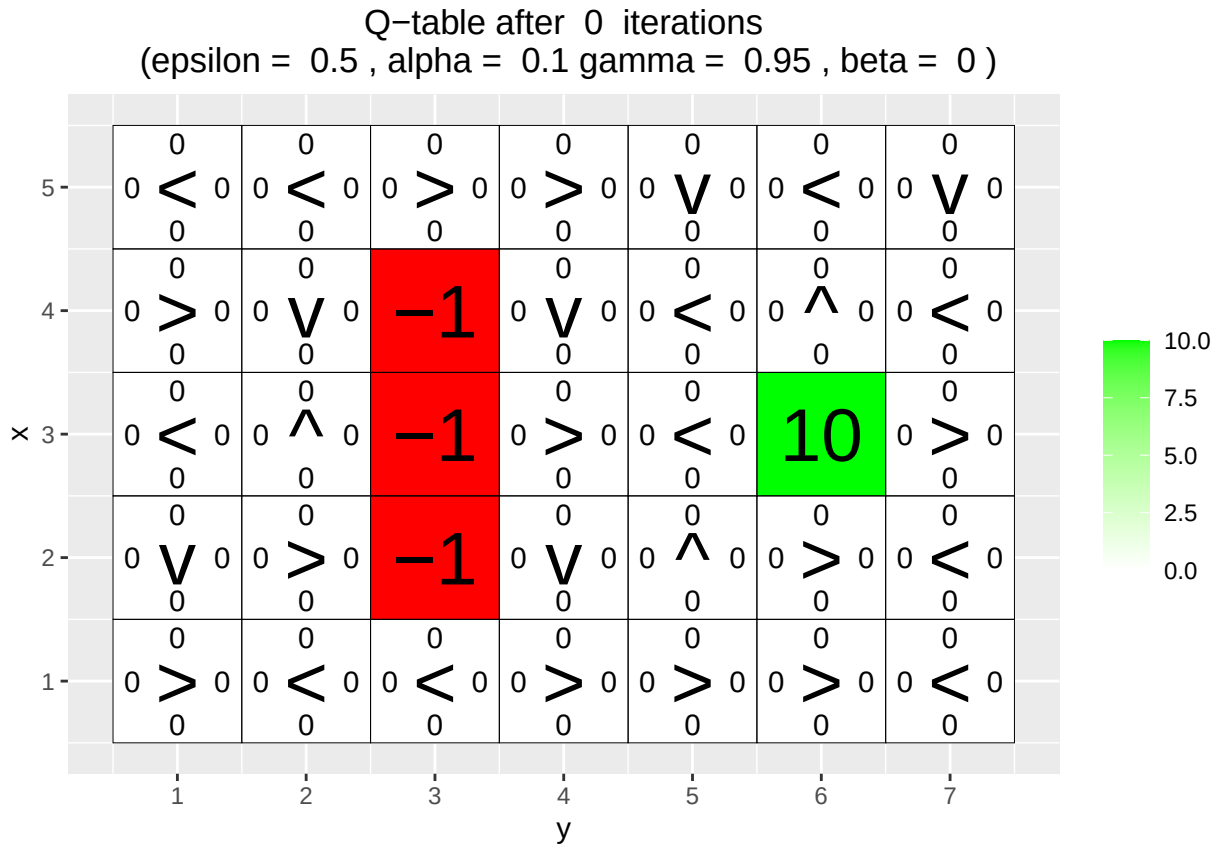
```
arrows <- c("^", ">", "v", "<")
action_deltas <- list(c(1,0), # up
                      c(0,1), # right
                      c(-1,0), # down
                      c(0,-1)) # left

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) +
```

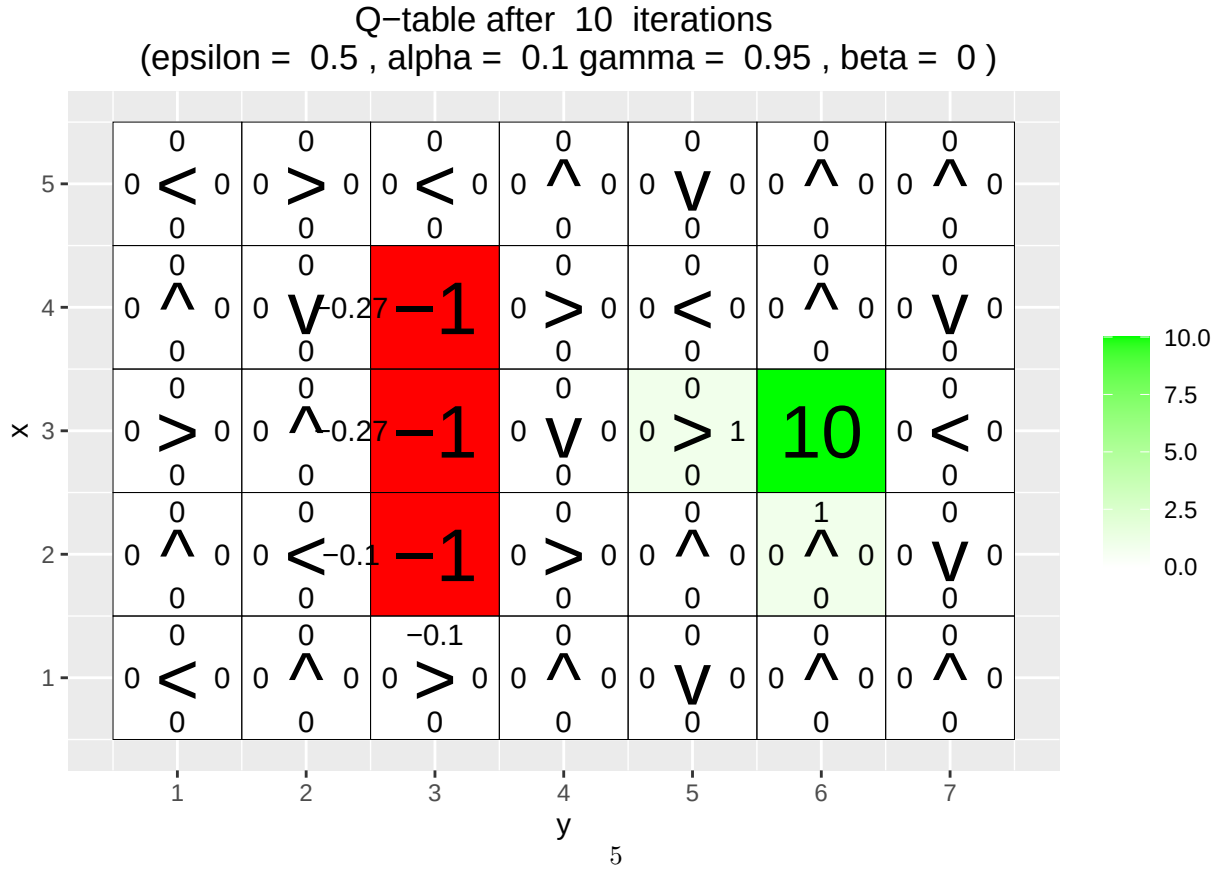
```
          geom_text(aes(label = val2),size = 4,nudge_x = .35,na.rm = TRUE) +
          geom_text(aes(label = val3),size = 4,nudge_y = -.35,na.rm = TRUE) +
          geom_text(aes(label = val4),size = 4,nudge_x = -.35,na.rm = TRUE) +
          geom_text(aes(label = val5),size = 10) +
          geom_tile(fill = '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)))

}

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.

  q_state=q_table[x,y,1:4]
  max_q=which(max(q_state)==q_state)

  if(length(max_q)==1){
    return(max_q)
  }
  return(sample(max_q, 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(1,0,1)<epsilon){
    return(GreedyPolicy(x,y))
  }
  return(sample(1:4, 1))
```

```
}

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.

  episode_correction=0
  reward=0

  x=start_state[1]
  y=start_state[2]
```

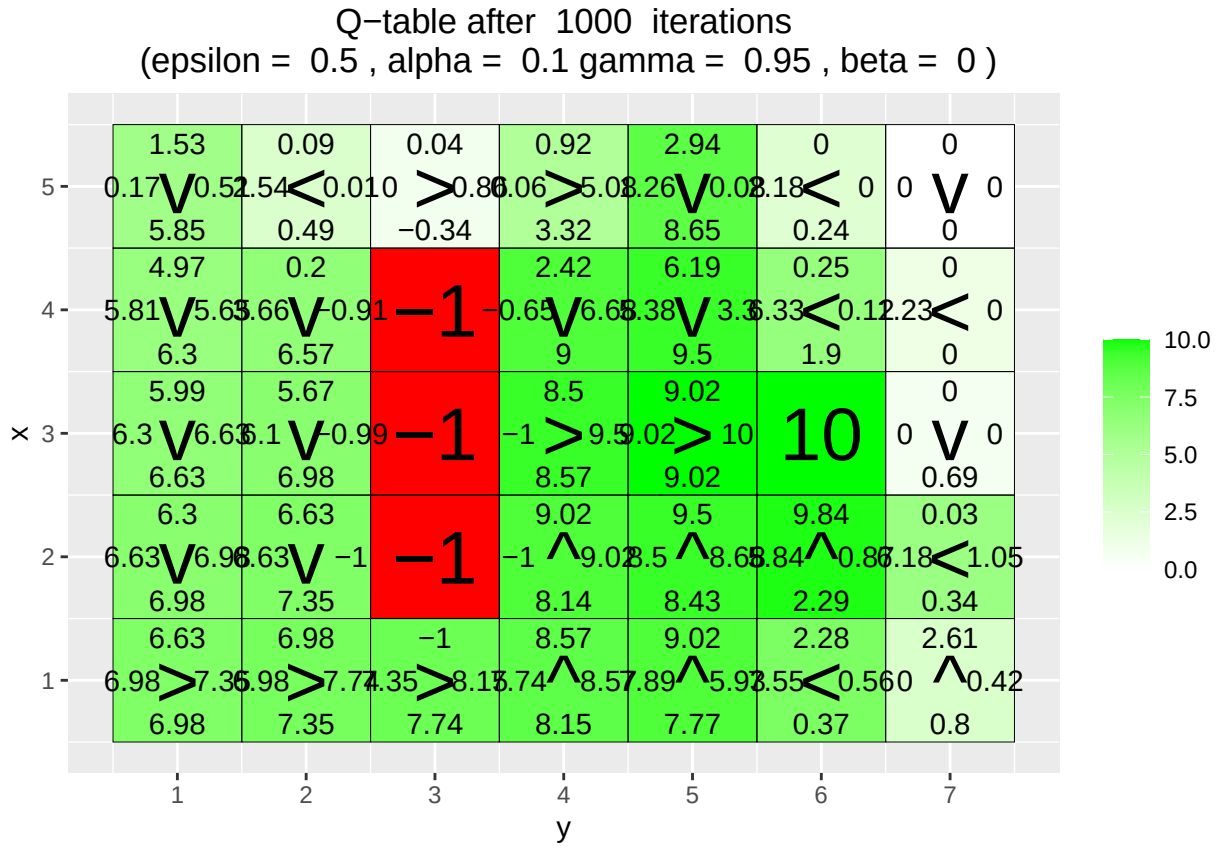
```
  repeat{
    # Follow policy, execute action, get reward.
    chosenAction=EpsilonGreedyPolicy(x,y,epsilon)   #Chosen action
    newState=transition_model(x,y,chosenAction,beta)  #new S

    new_x=newState[1]
    new_y=newState[2]
    reward=reward_map[new_x,new_y]     #new reward

    # Q-table update.
    tempDifference=reward+gamma*max(q_table[new_x,new_y,1:4])-q_table[x,y,chosenAction]
    q_table[x,y,chosenAction]<<-q_table[x,y,chosenAction]+alpha*tempDifference

    episode_correction=episode_correction+tempDifference

    x=new_x
    y=new_y

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

}
```

Environment A






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
0.0

Q1 What has the agent learned after the first 10 episodes ?

A According to the chart, the agent only learned not to take the actions which will walk on the red tile (with -1 reward).

Q2 Is the final greedy policy (after 10000 episodes) optimal for all states, i.e. not only for the initial state ? Why / Why not ?

A Yes. Because the Q-learning is kind of like dynamic programming, by decomposing big problems into small problems (intermediate states), finally solves the big problem (initial state). Hence if we find the optimal policy for initial state, it must be the optimal policy for all other states on the path (Except for the states that have not been passed in the random walk).

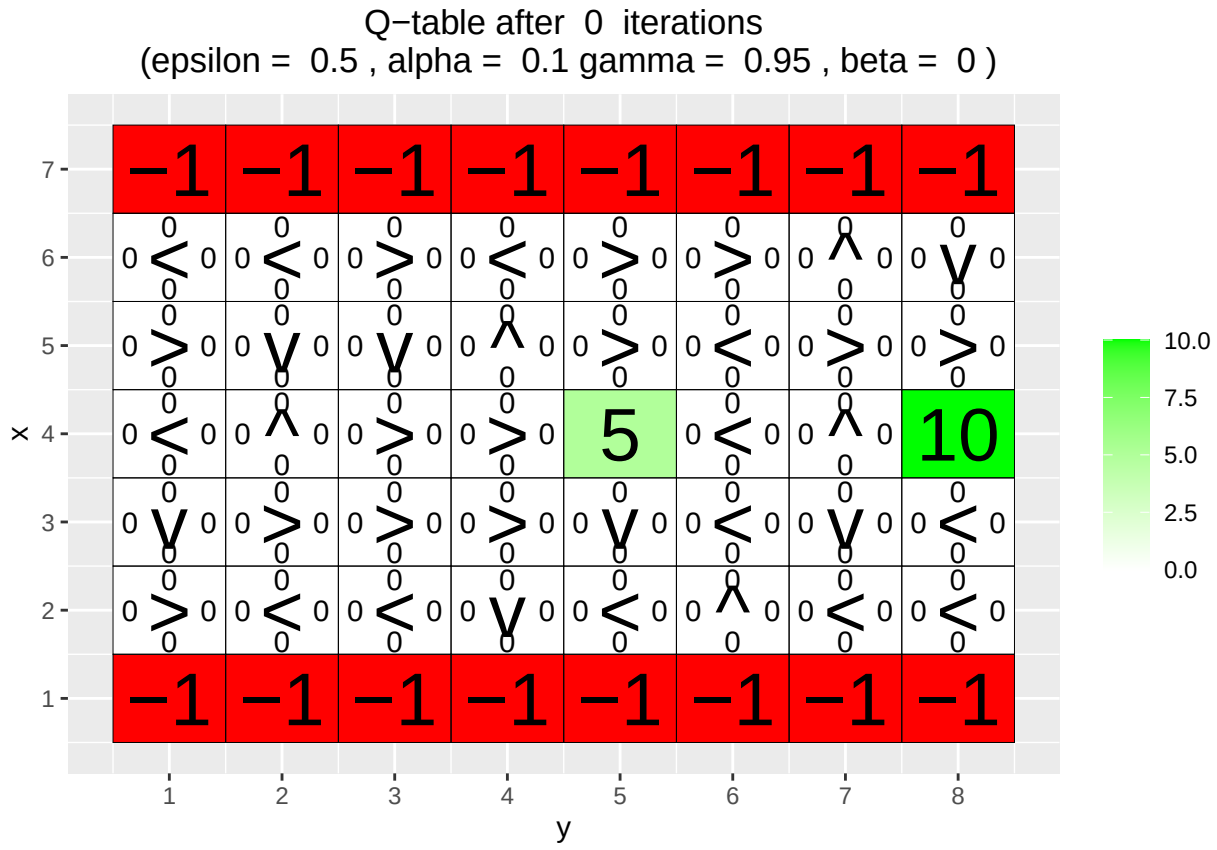
Q3 Do the learned values in the Q-table reflect the fact that there are multiple paths (above and below the negative rewards) to get to the positive reward ? If not, what could be done to make it happen ?

A Yes, from the chart there are two path (above and below the negative rewards) to get to the positive rewards. Because on the early episode there are some iterations walk upper and finally get the positive reward (Perhaps as many as the path below), which cause larger Q value both above and below the negative rewards. But if we choose different random seed, the result might be different.

Environment B

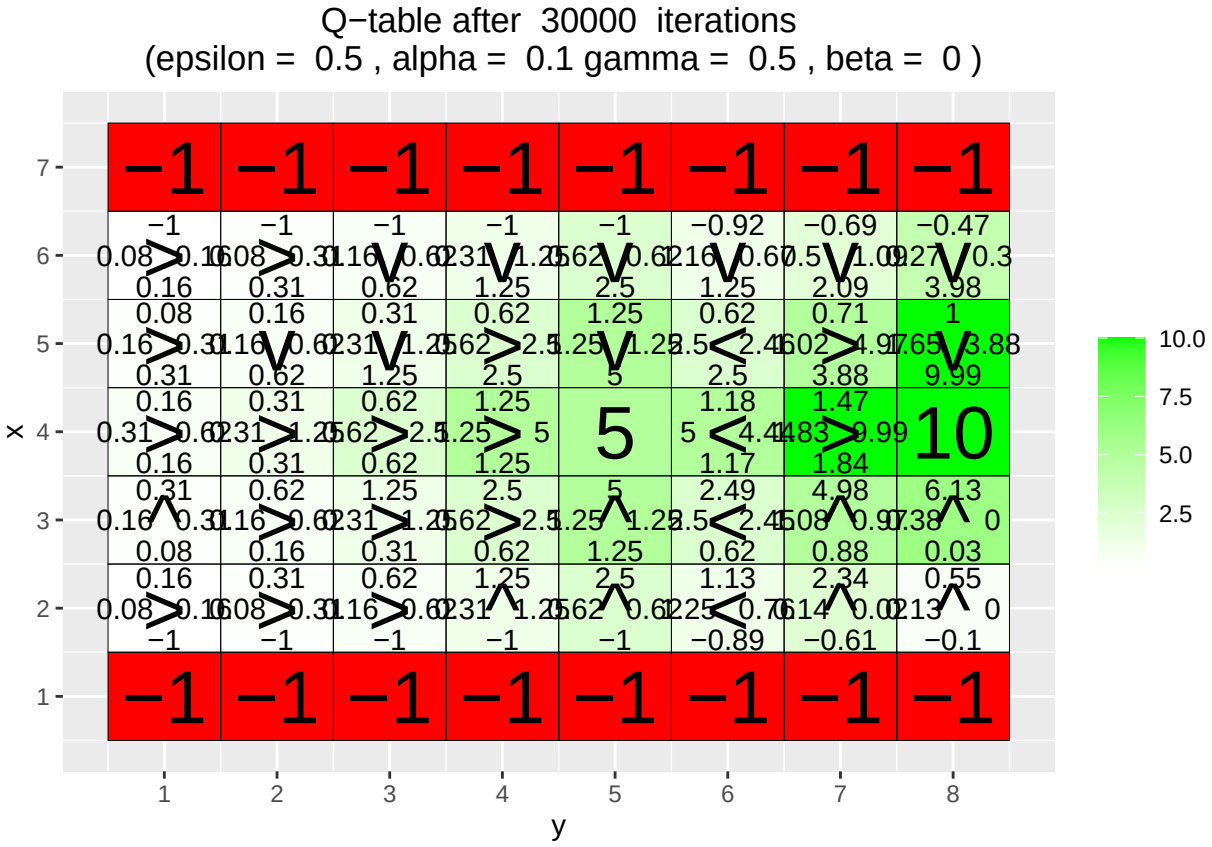
```
set.seed(12345)
# Environment B (the effect of epsilon and gamma)
```

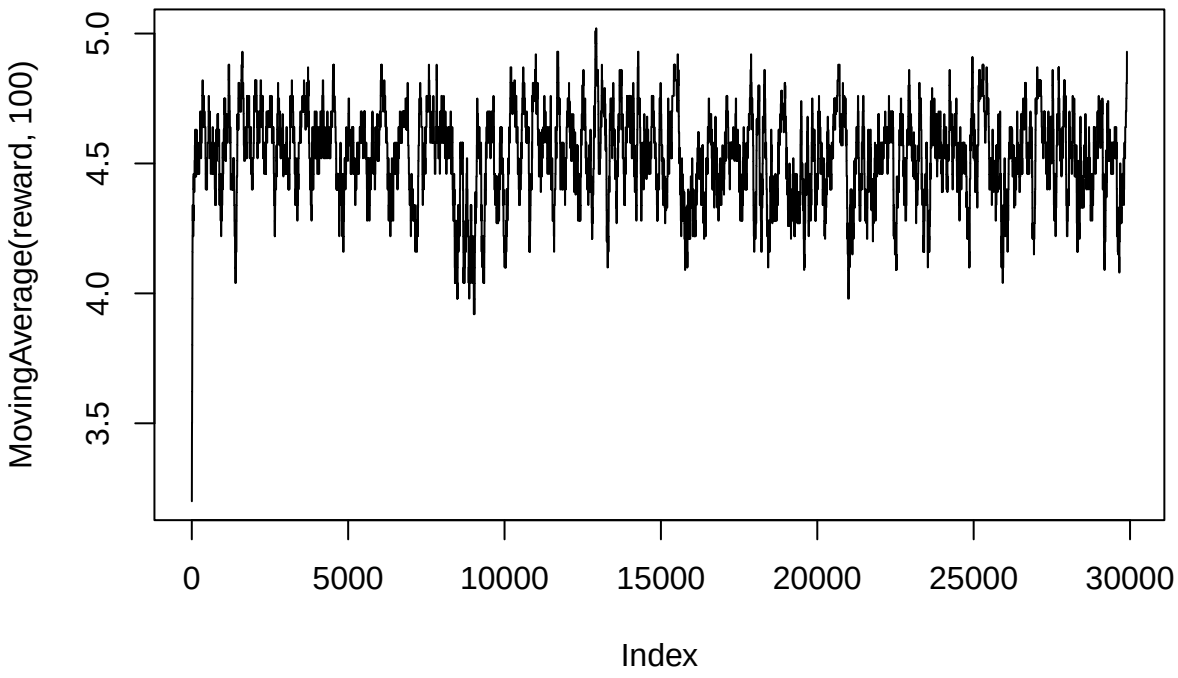
```
H <- 7
W <- 8

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()
```

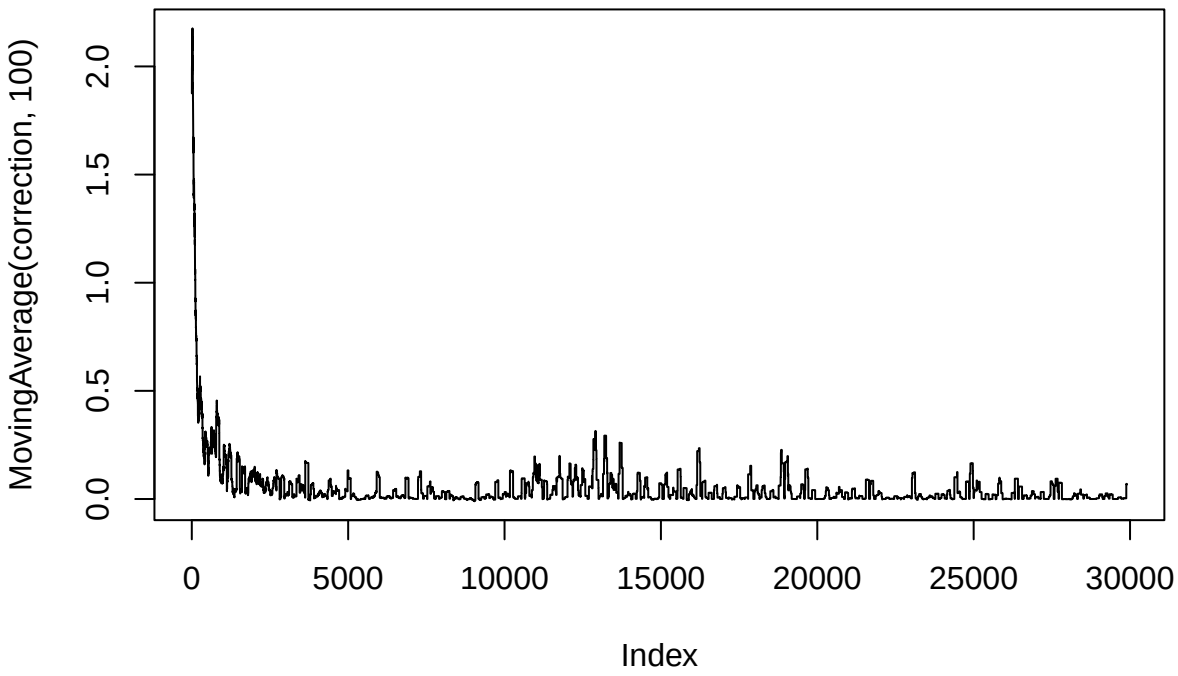


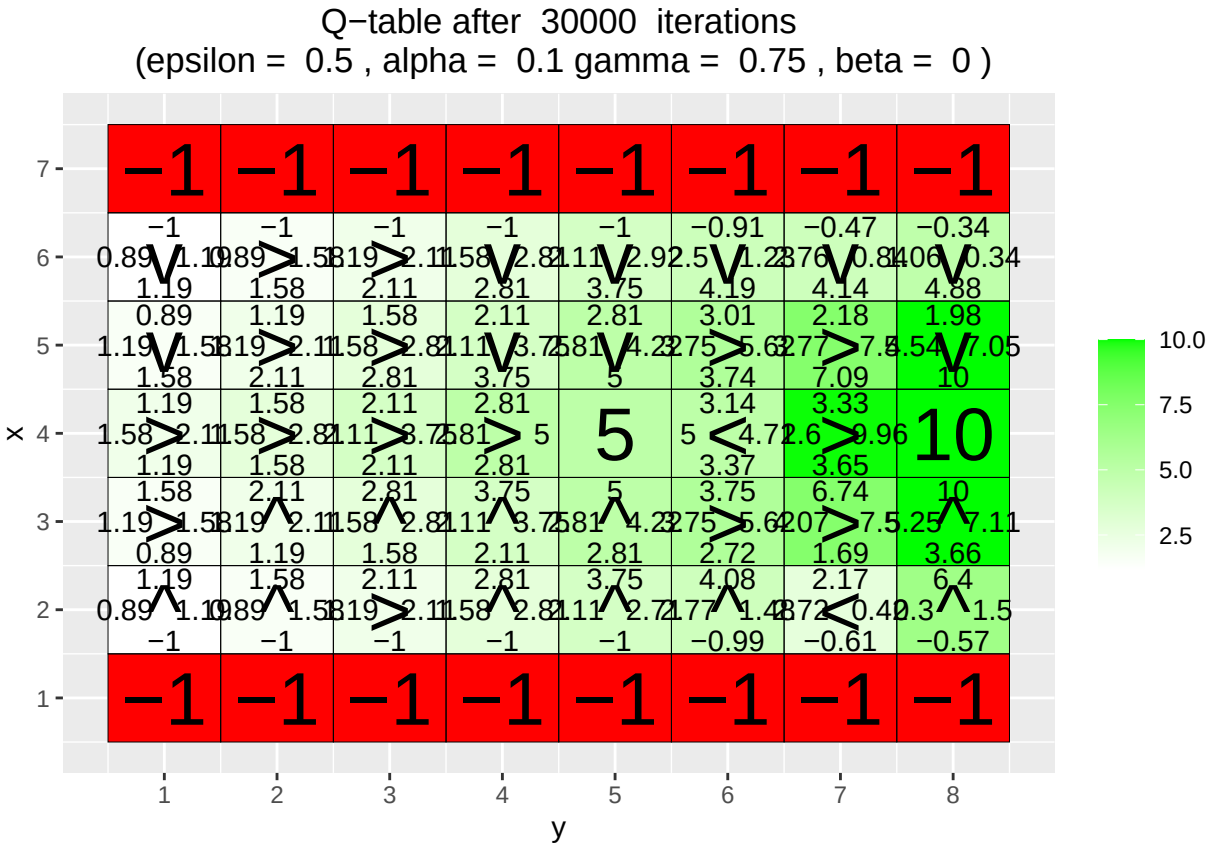

```
MovingAverage <- function(x, n){

  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
```

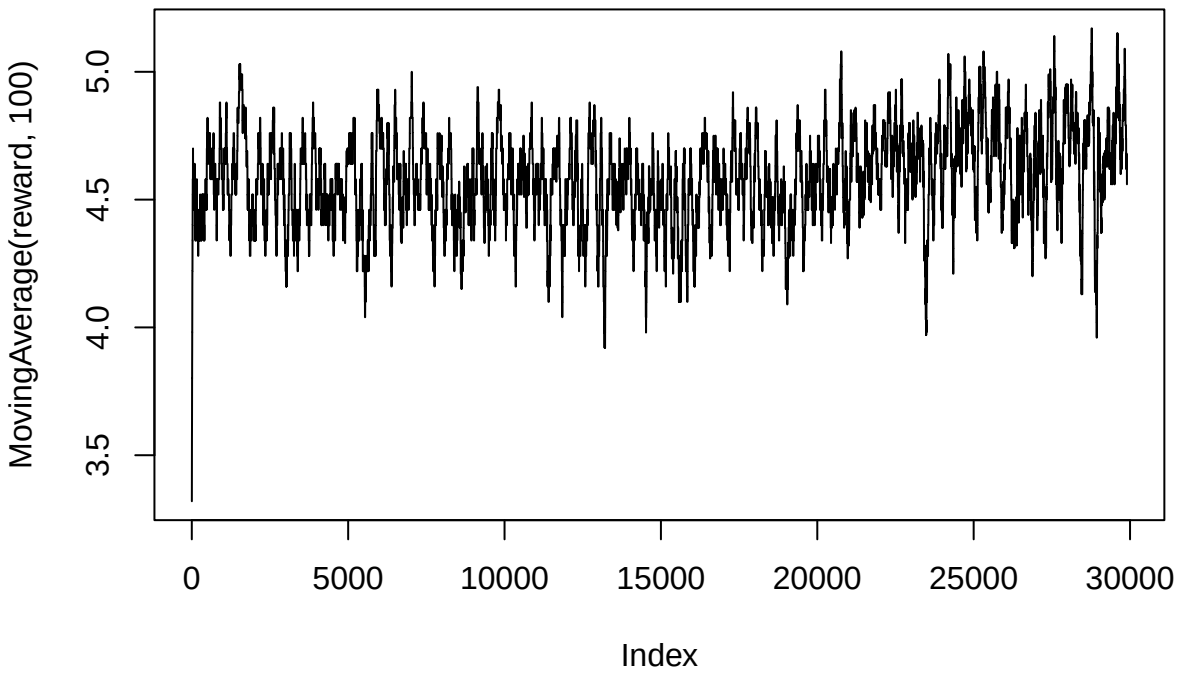
```
  for(i in 1:30000){
    foo <- q_learning(gamma = j, start_state = c(4,1))
    reward <- c(reward,foo[1])
    correction <- c(correction,foo[2])
  }

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

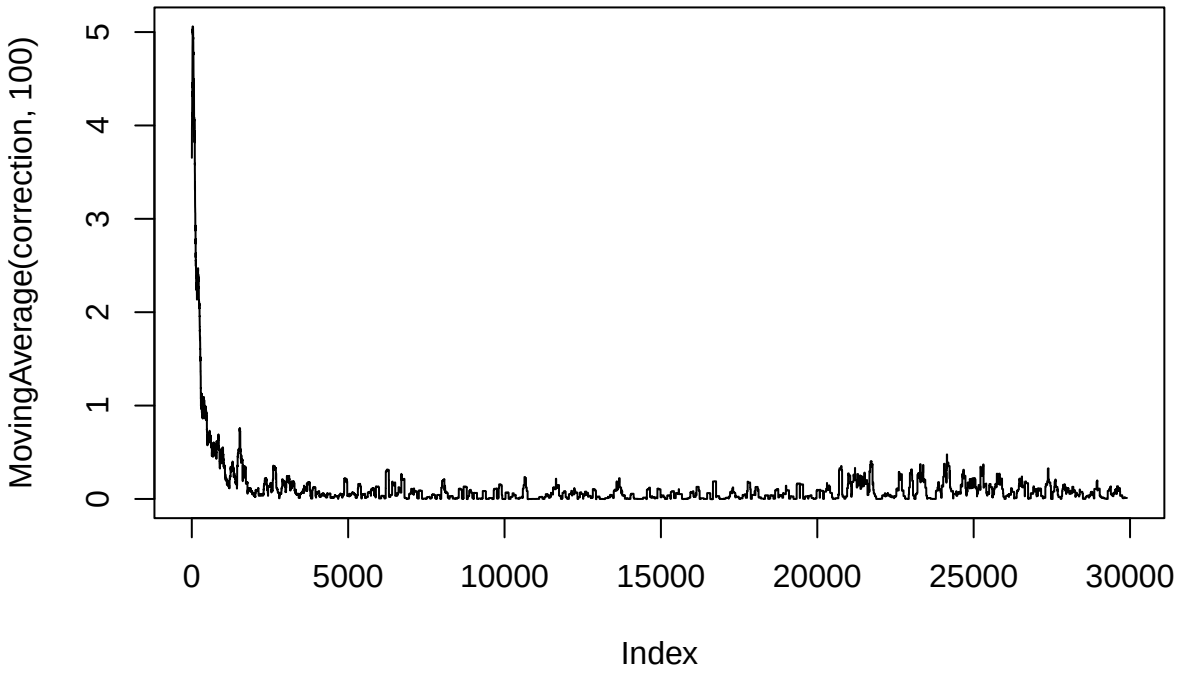


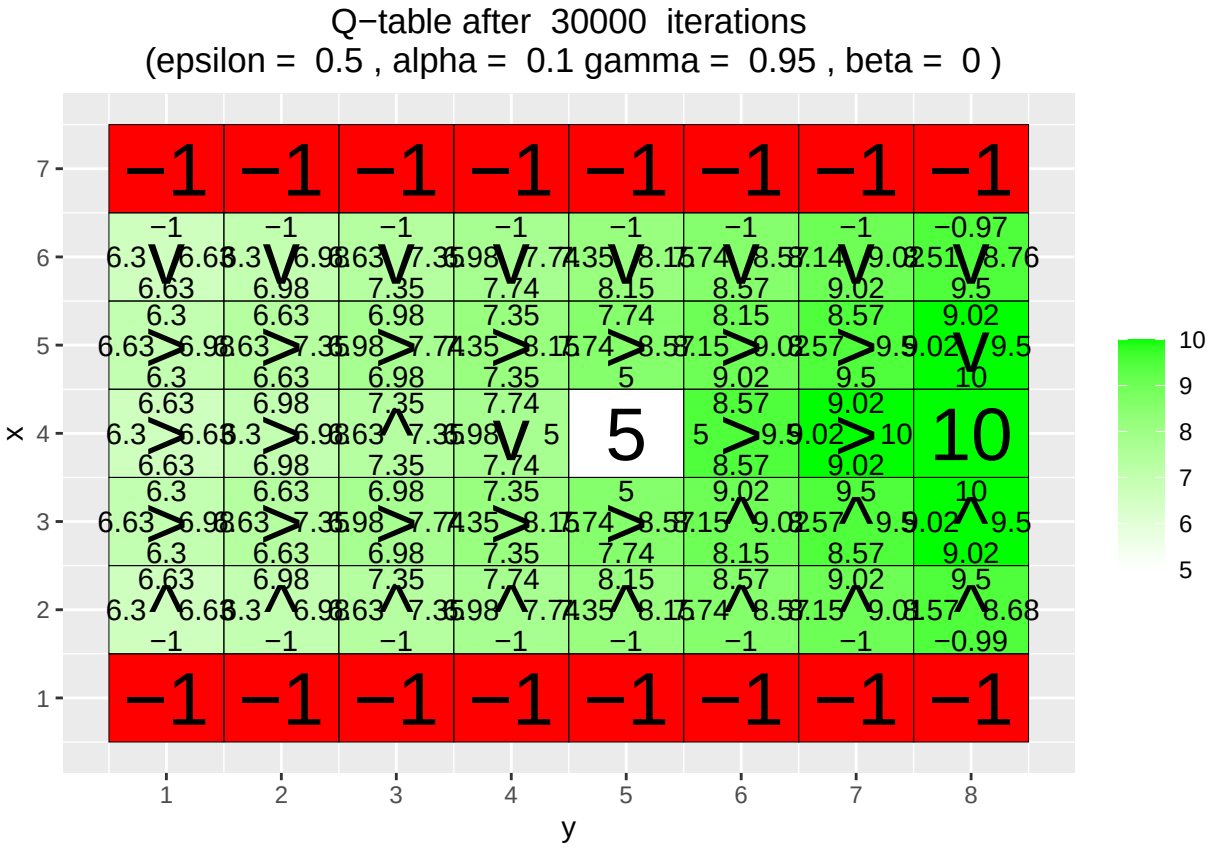

MovingAverage(reward, 100)
3.5
4.0
4.5
5.0
0
5000
10000
15000
20000
25000
30000
Index

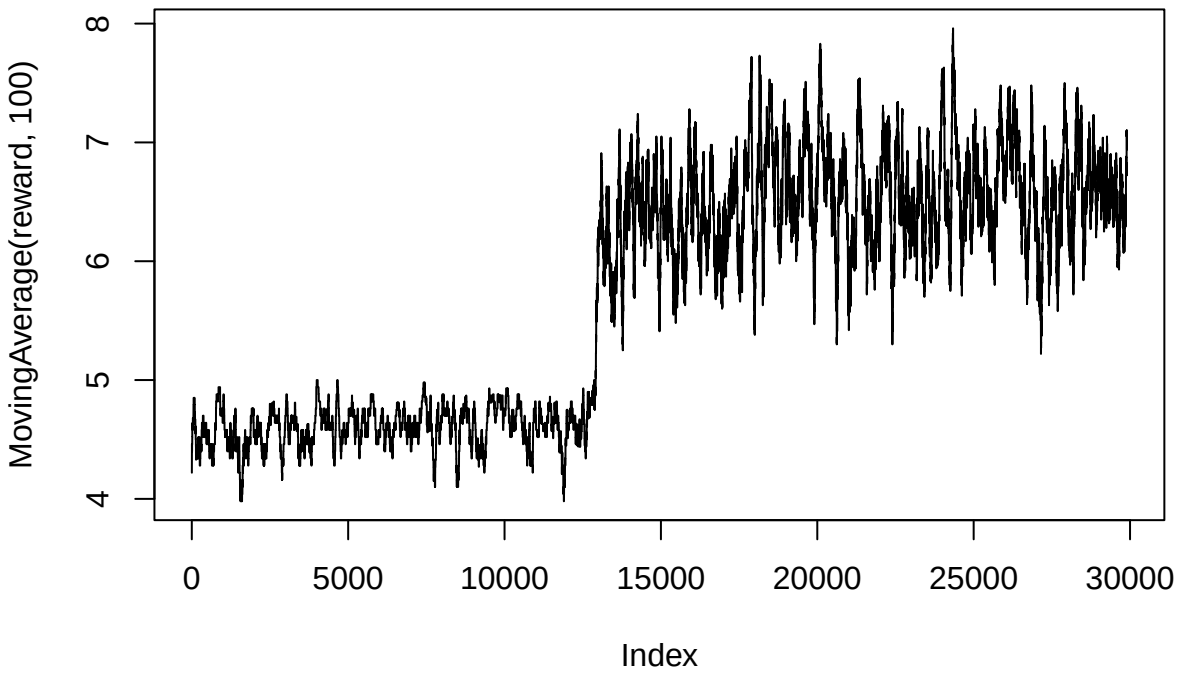

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index

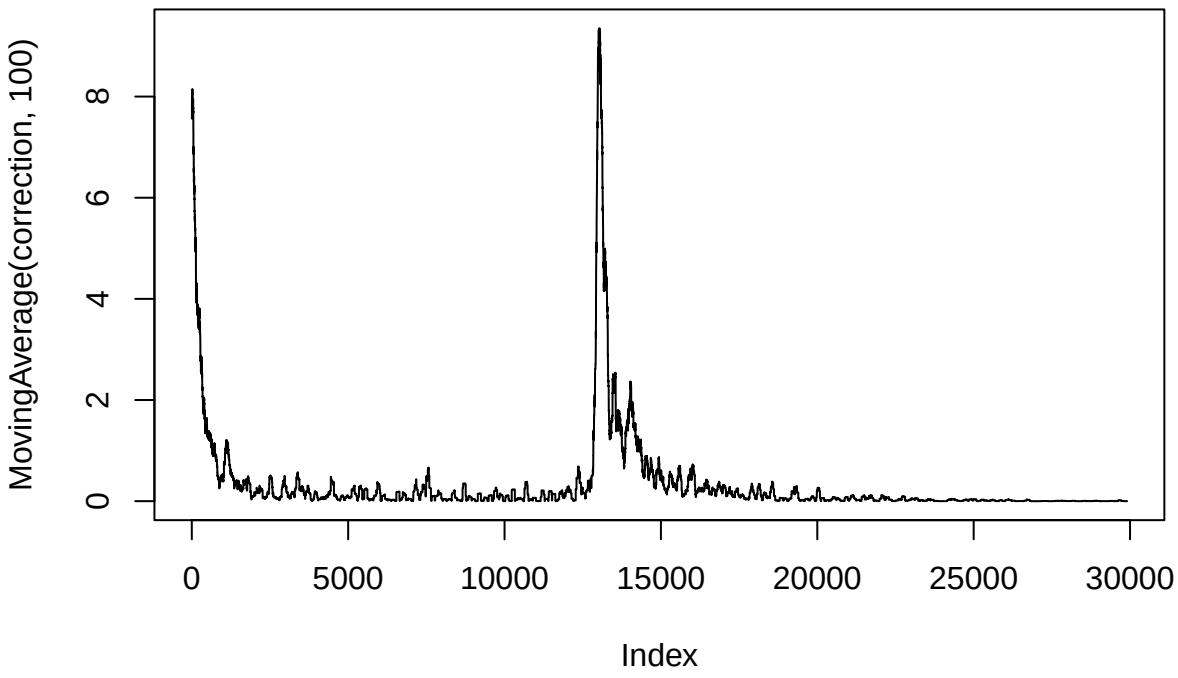

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5


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

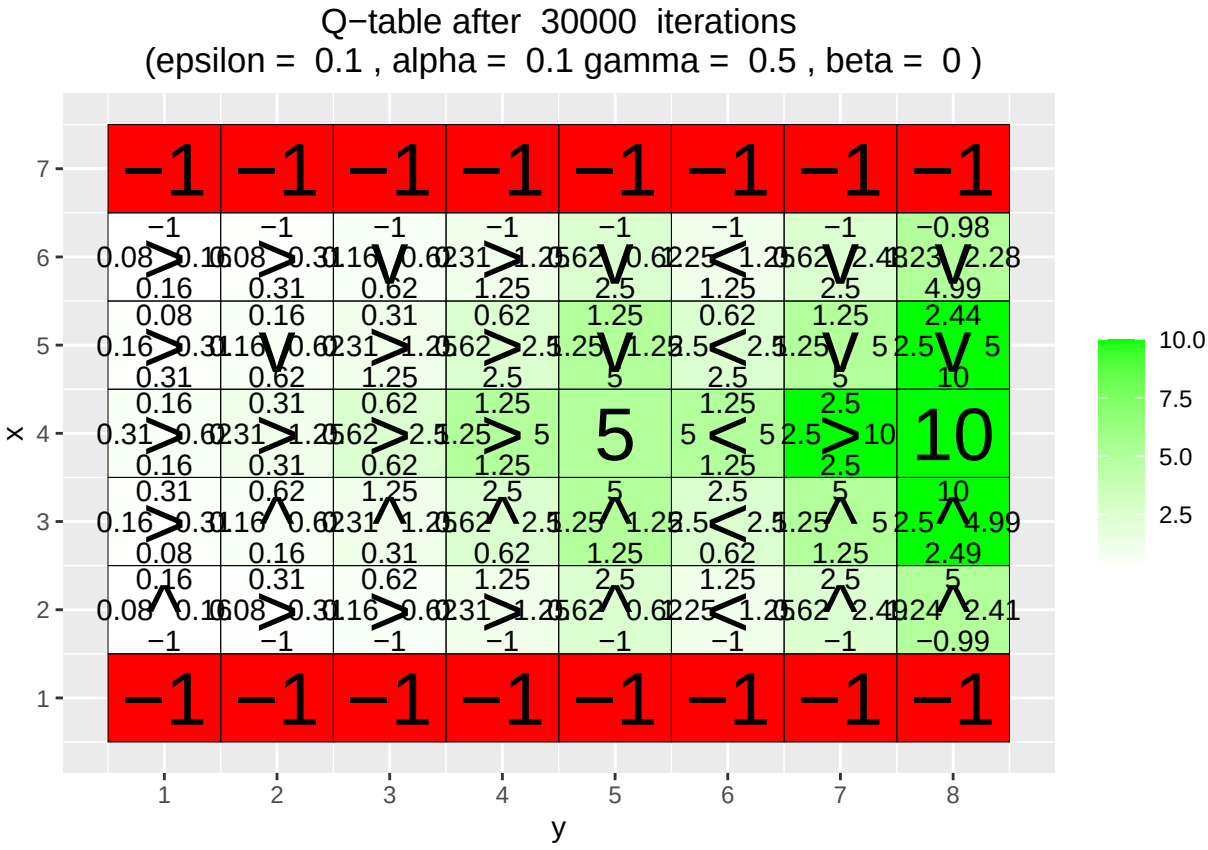

MovingAverage(correction, 100)
0
1
2
3
4
5
0
5000
10000
15000
20000
25000
30000
Index

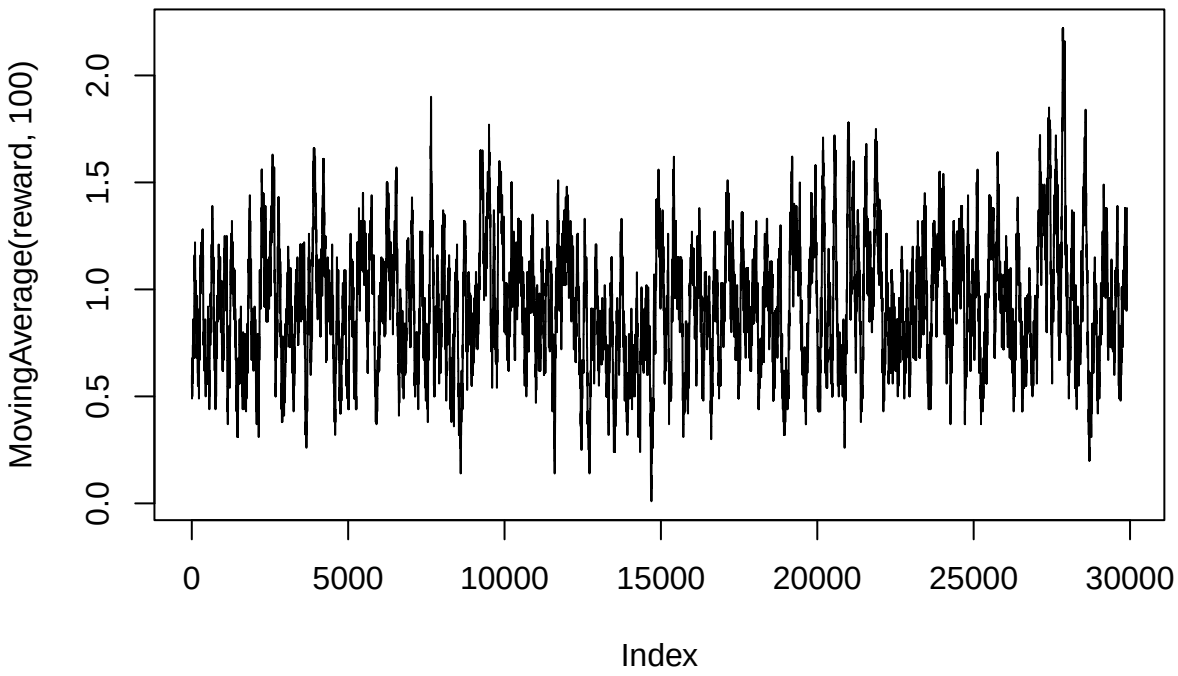

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10
9
8
7
6
5

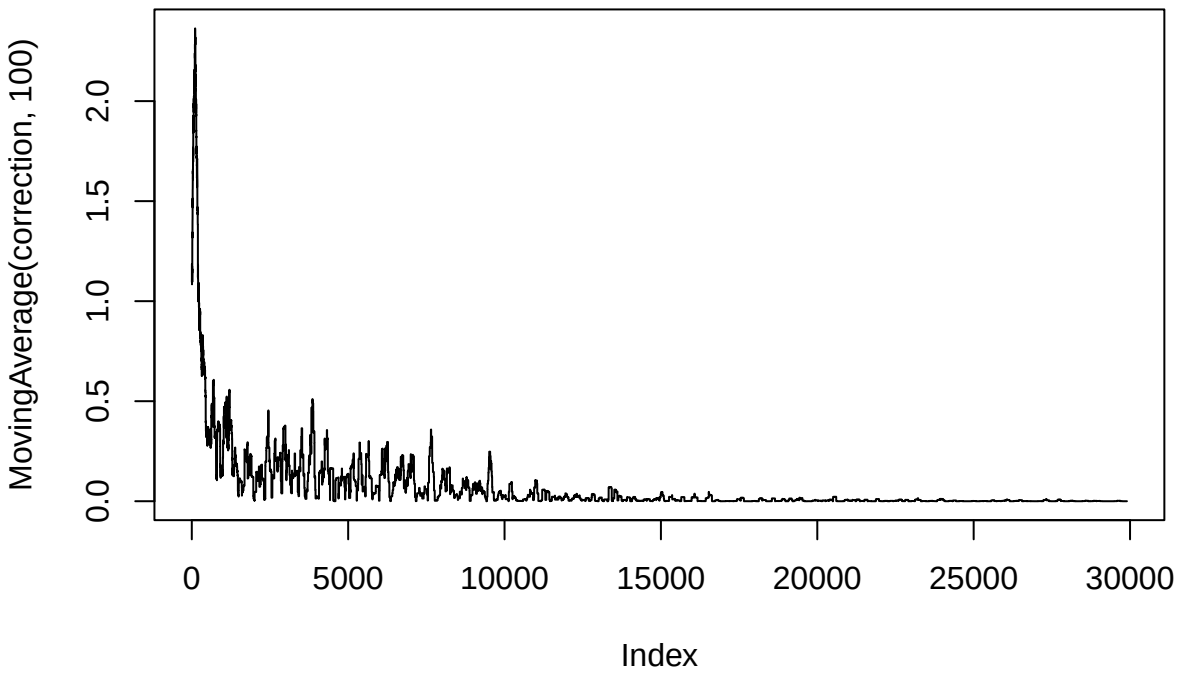

MovingAverage(reward, 100)
4
5
6
7
8
0
5000
10000
15000
20000
25000
30000
Index

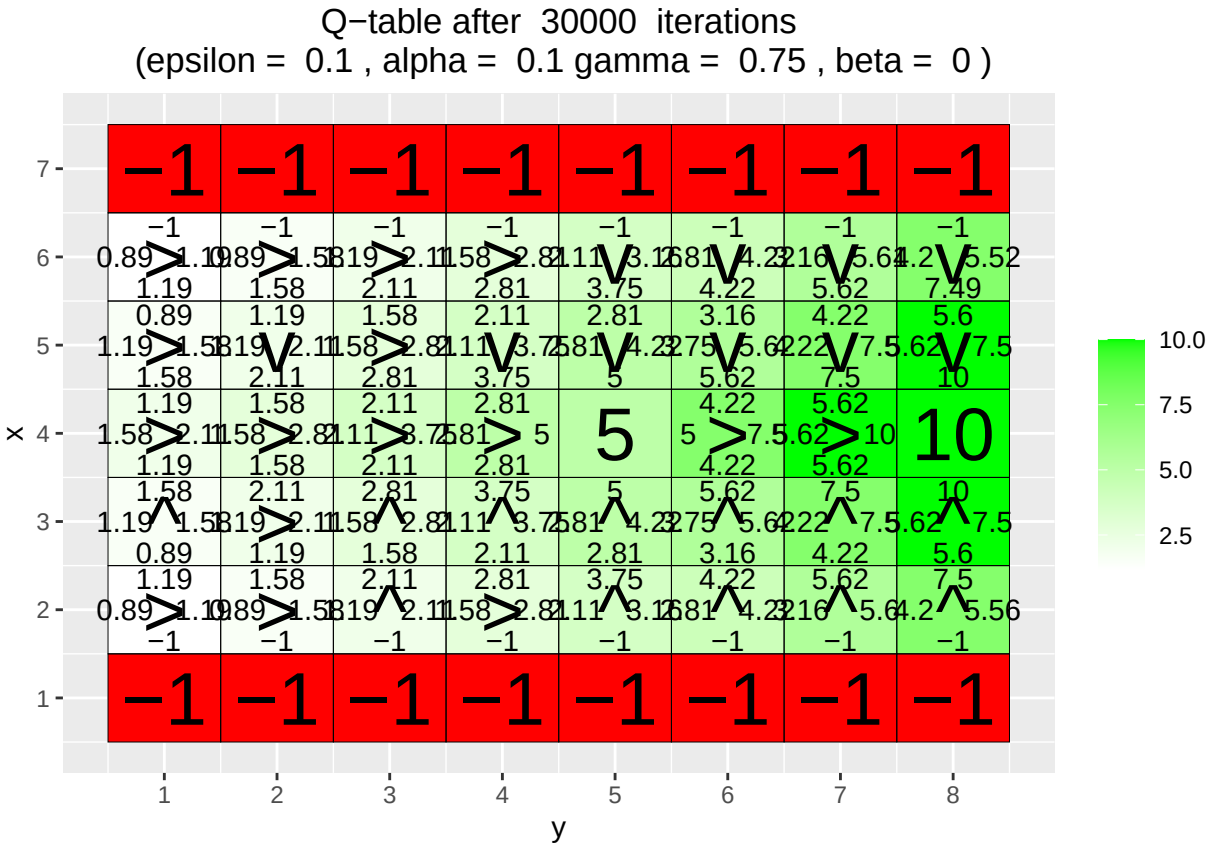


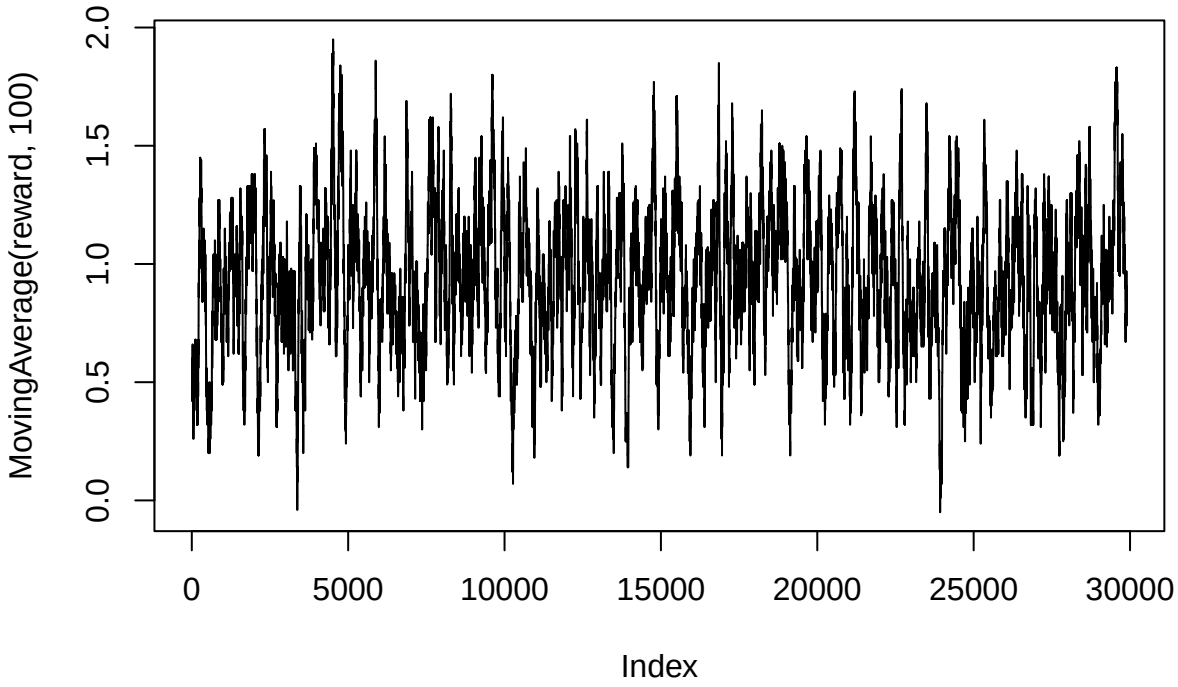

```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  for(i in 1:30000){
    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")
}
```

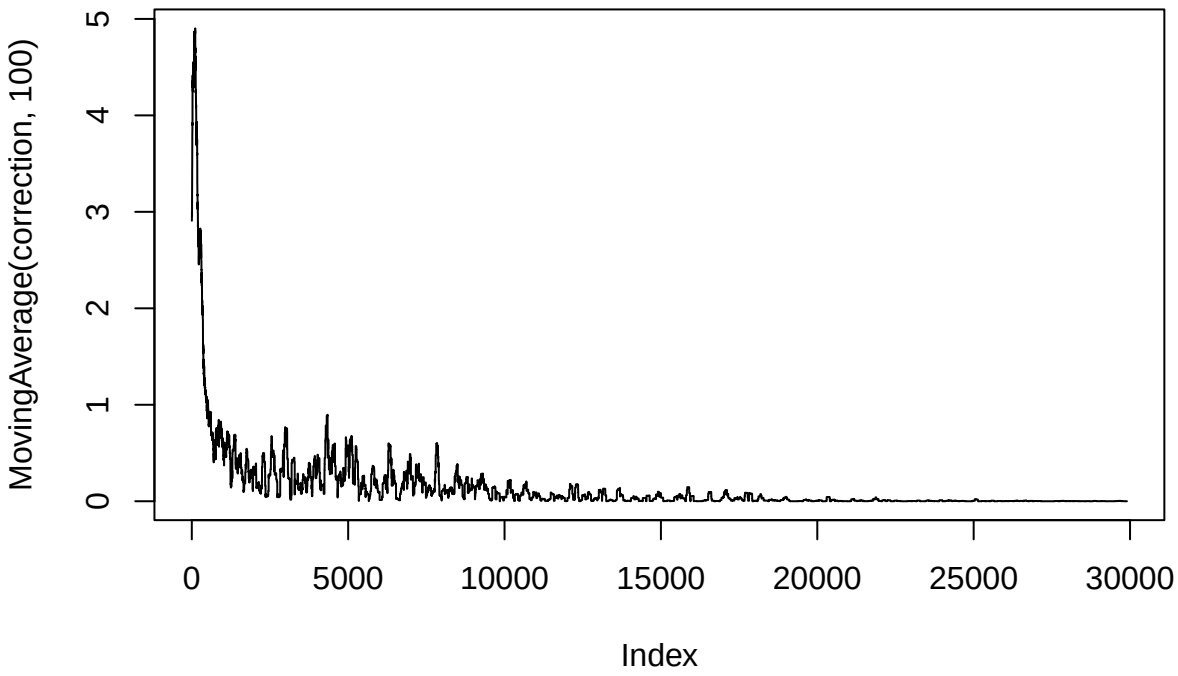

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10.0
7.5
5.0
2.5

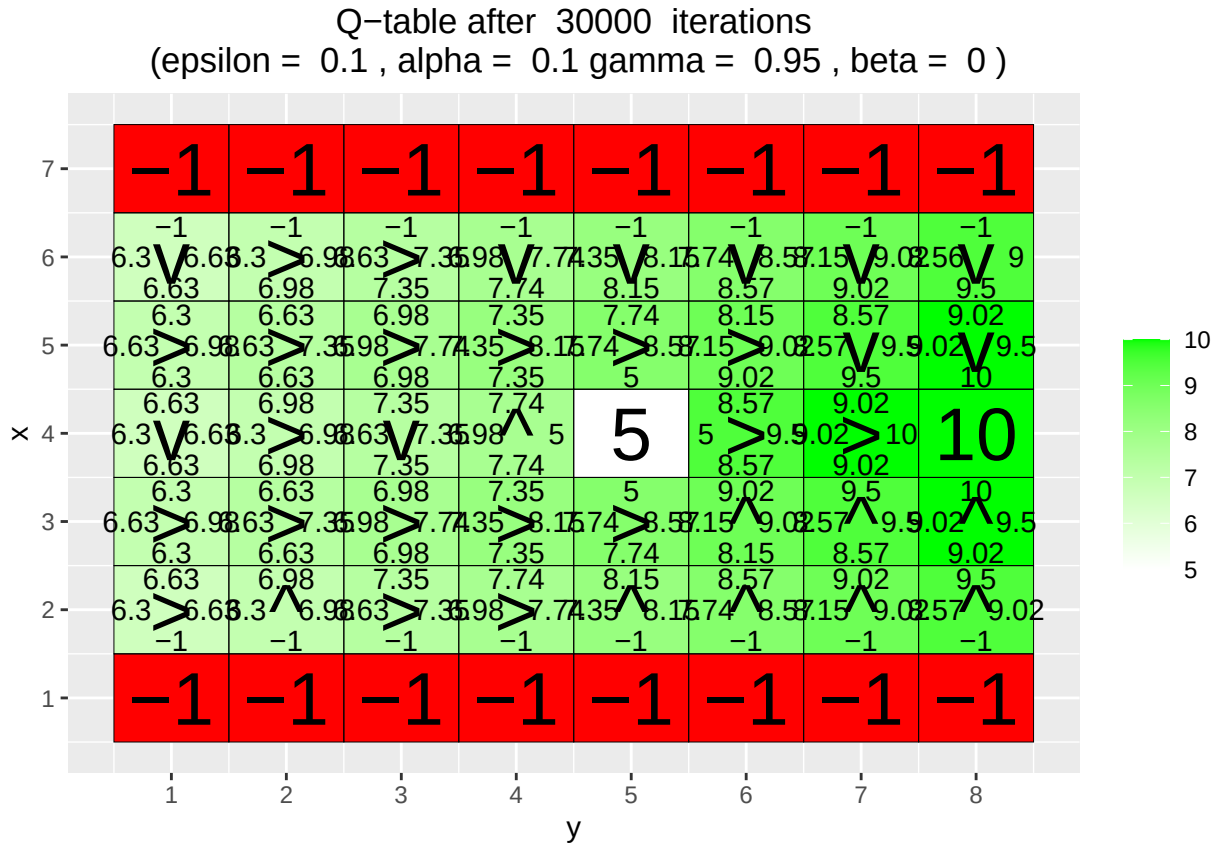

MovingAverage(reward, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index

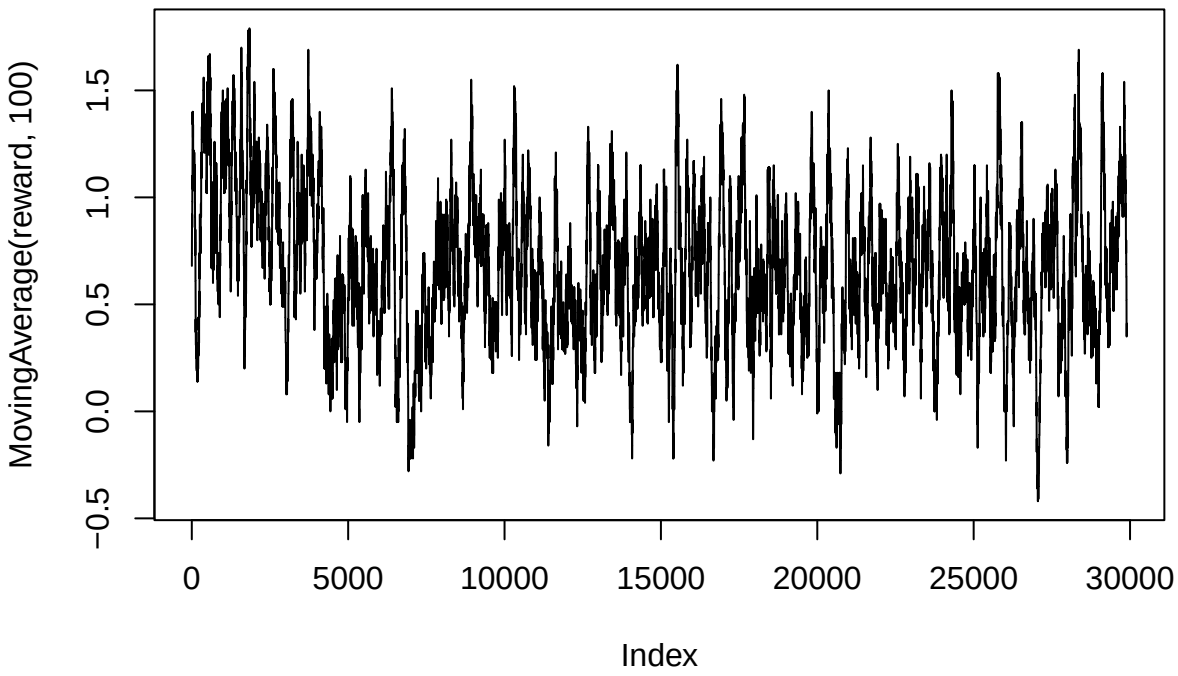

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index

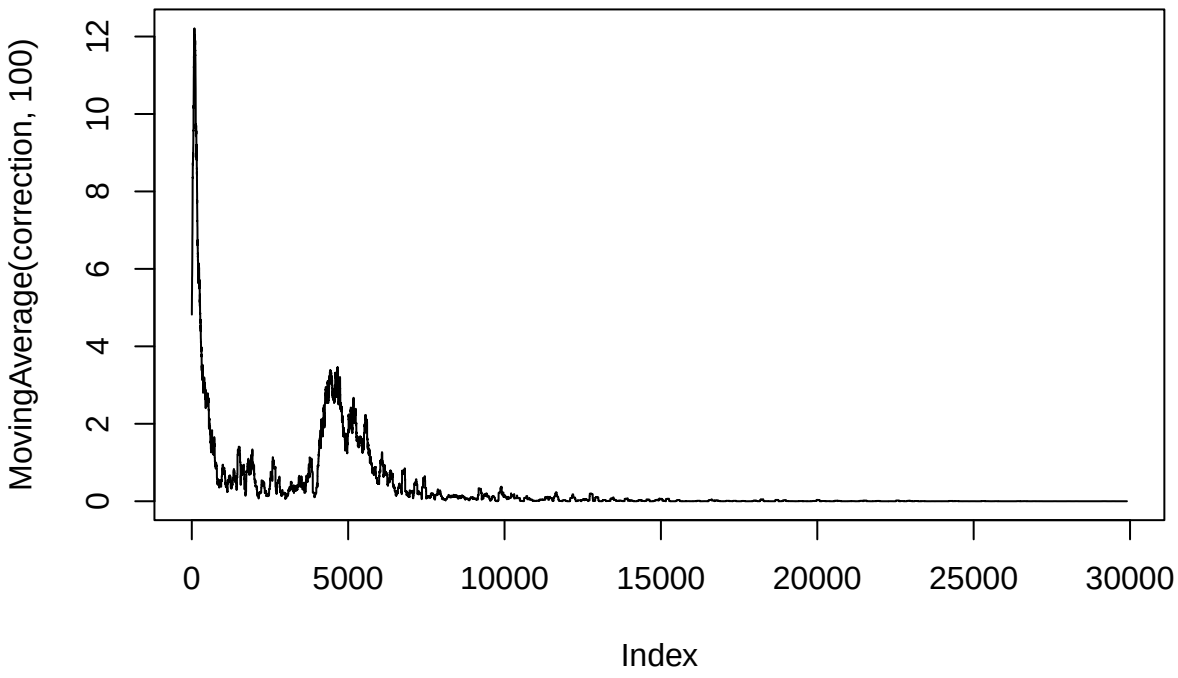

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5


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


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


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


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



How ϵ affect learned policy

The parameter ϵ affect the probability of choose the random action instead of follow the policy with the larger Q value. Hence we make an assumption that smaller ϵ will make it easier to find the global optimal (but maybe with lower average reward).

Compare the chart with ϵ is 0.1 & 0.5 with equal gamma. Compare to $\epsilon = 0.1$, when $\epsilon = 0.5$, the mean of the reward received in the episode is larger, with lower variance. Meanwhile, the the temporal difference correction converge slower, which means find the optimal policy slower.

But there are no evidence that smaller ϵ will make it easier to find the global optimal.

Summery

1. Larger ϵ cause larger average reward.

2. Smaller ϵ make the algorithm converge faster (but maybe not global optimal).

3. When ϵ is small, the agent will find policy base on what it has learned and rarely explore the state it never been.(sometimes if we use another random seed.)

How γ affect learned policy

$0 < \gamma \leq 1$ describes our preference between present and future rewards, i.e. lower γ means the future reward affect less on state values and action values and vice versa. Hence we assume that larger γ cause larger state value and action value when the tiles are far away from the positive rewards.

Compare the plot in the same ϵ and $\gamma = 0.5, 0.75, 0.95$, our hypothesis can be seen to be correct. When $\gamma = 0.5, 0.75$ there is few difference in Moving Average of reward and temporal difference correction. And the agent will hardly reach 10 tile from the initial state.

But when $\gamma = 0.95$, even the tiles extremely close to 5 tile tend to reach the 10 tile instead of 5 tile. And when $\epsilon = 0.5$, from the chart of moving average of reward and temporal difference correction, around 12500 episode, the agent jump out of local optimal and find the global optimal. Hence, when γ is larger it is easier to find the global optimal.

Summery

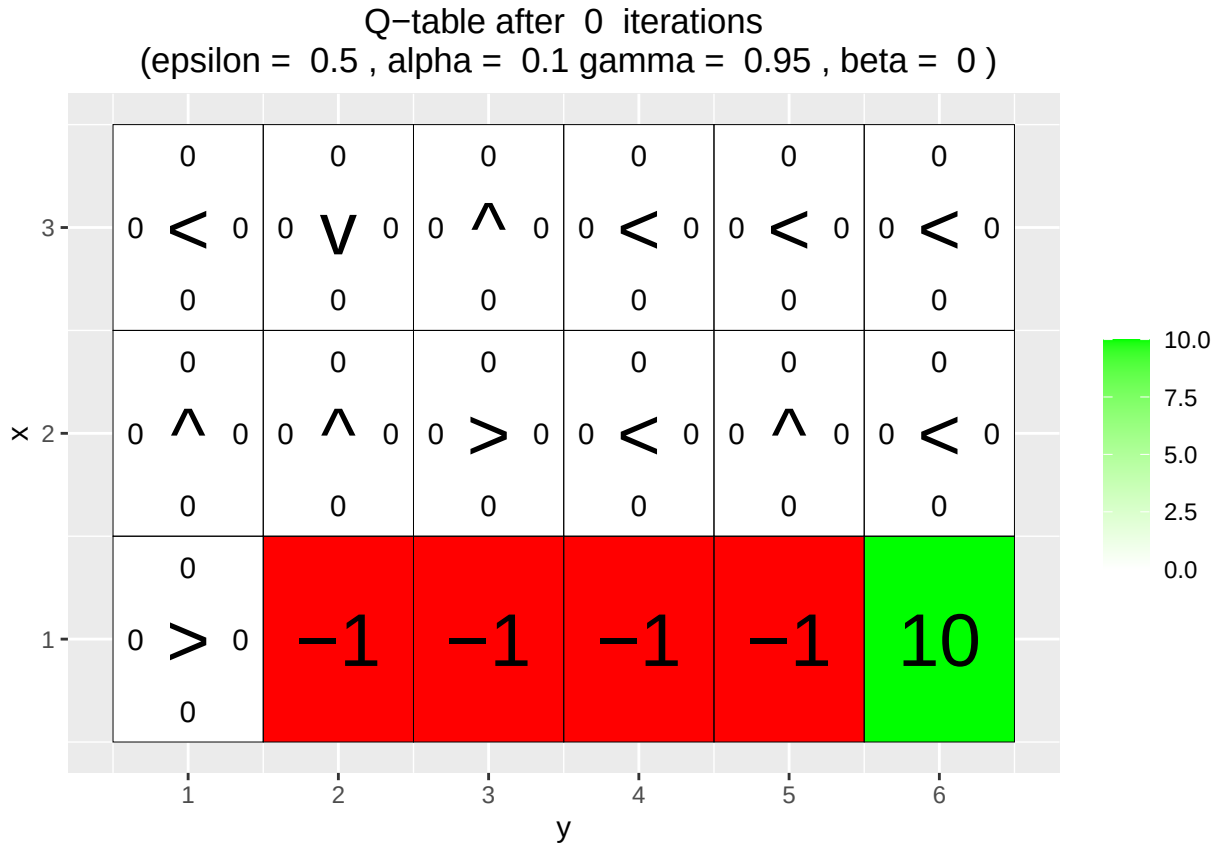
1. Lower γ means the future reward affect less on state values and action values and vice versa.

2. larger γ cause larger state value and action value when the tiles are far away from the positive rewards.

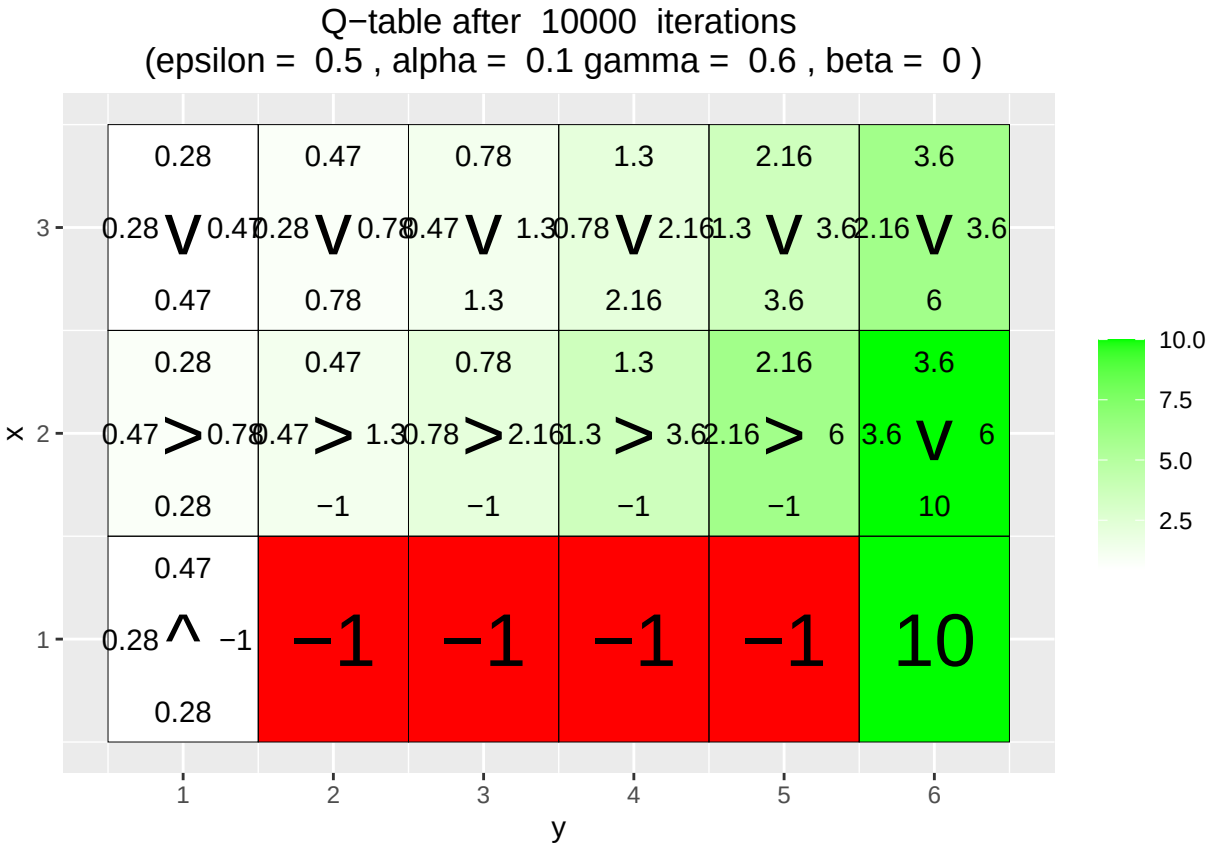
2. larger γ make it easier to find the global optimal.

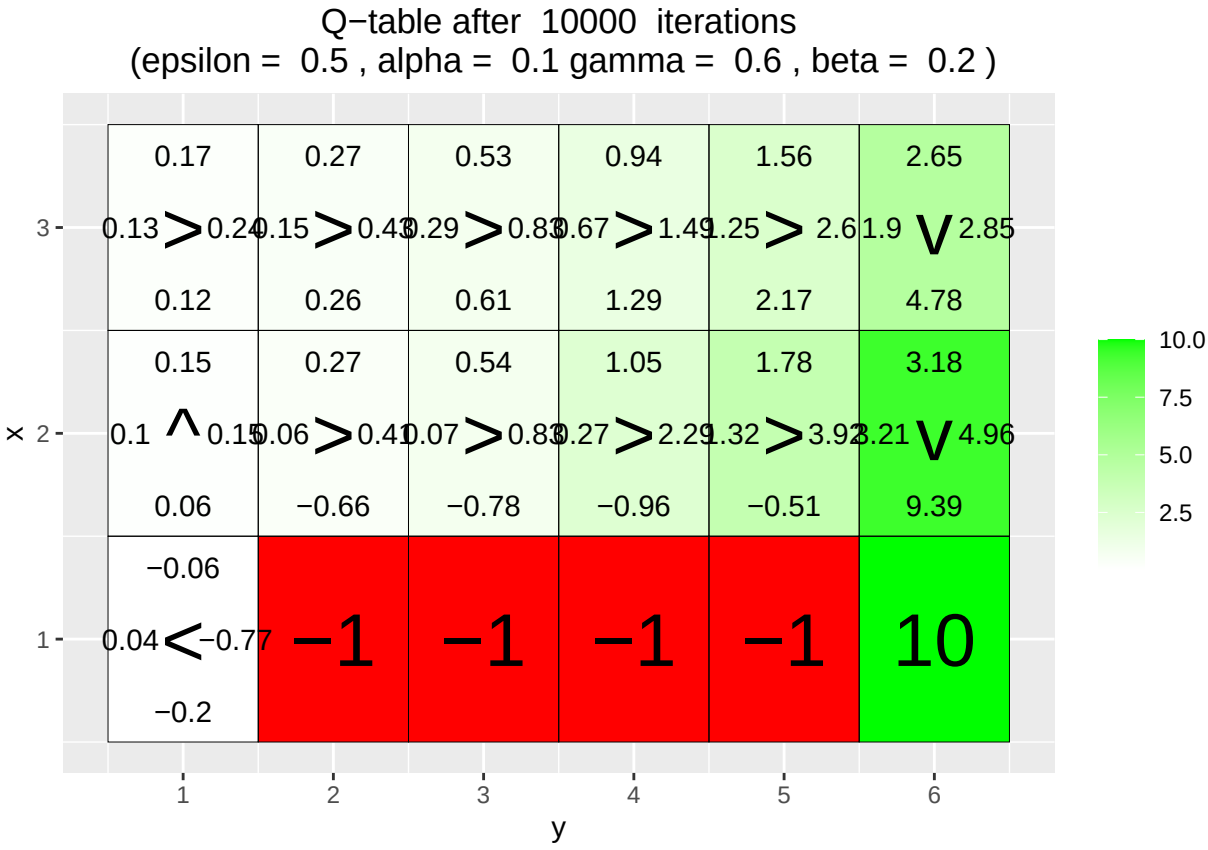
Environment C

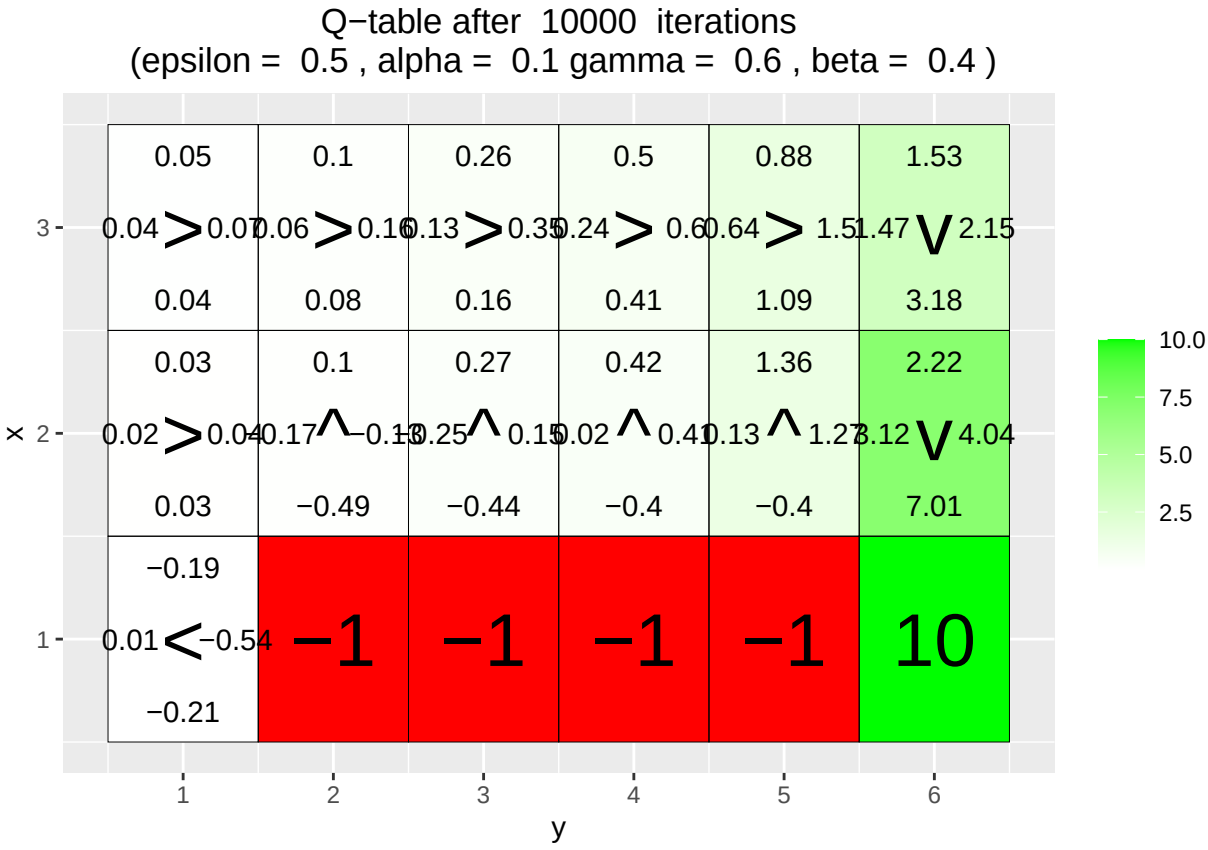
```
# Environment C (the effect of beta).

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))

vis_environment()
```

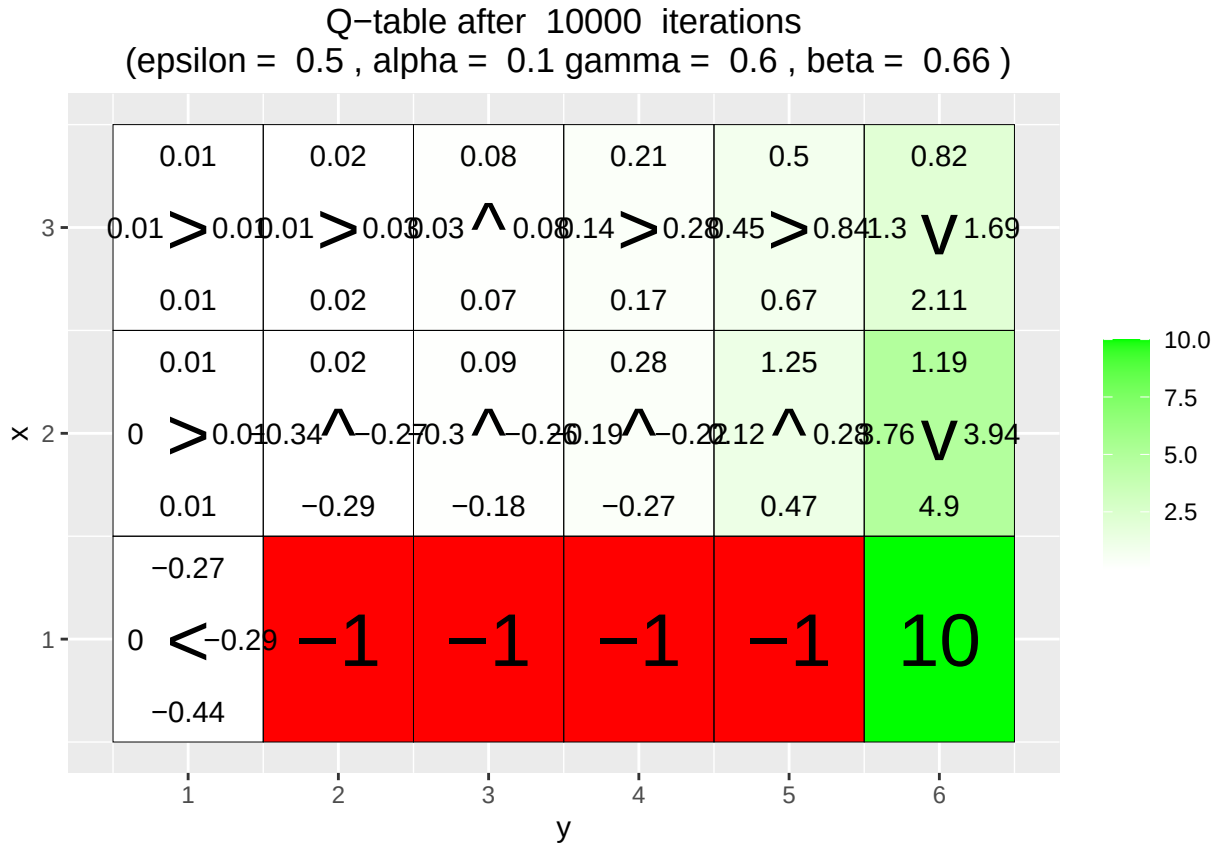



```
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

  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 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
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)
0.17
0.13 > 0.24
0.12
0.27
0.15 > 0.43
0.26
0.53
0.29 > 0.83
0.61
0.94
0.67 > 1.49
1.29
1.56
1.25 > 2.6
2.17
2.65
1.9 V 2.85
4.78
0.15
0.1 ^ 0.15
0.06
0.27
0.06 > 0.41
−0.66
0.54
0.07 > 0.83
−0.78
1.05
0.27 > 2.29
−0.96
1.78
1.32 > 3.92
−0.51
3.18
3.21 V 4.96
9.39
−0.06
0.04 < −0.77
−0.2
−1
−1
−1
−1
10
x
3
2
1
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)
0.05
0.1
0.26
0.5
0.88
1.53
0.04 > 0.07
0.06 > 0.16
0.13 > 0.35
0.24 > 0.6
0.64 > 1.5
1.47 V 2.15
0.04
0.08
0.16
0.41
1.09
3.18
0.03
0.1
0.27
0.42
1.36
2.22
0.02 > 0.04
0.17 ^ −0.13
0.25 ^ 0.15
0.02 ^ 0.41
0.13 ^ 1.27
3.12 V 4.04
0.03
−0.49
−0.44
−0.4
−0.4
7.01
−0.19
0.01 < −0.54
−1
−1
−1
−1
10
−0.21
x
3
2
1
1
2
3
4
5
6
y
10.0
7.5
5.0
2.5



How β affect learned policy

β is slipping factor, The agent will follow the action with probability $1-\beta$ and slip to the right or left with probability $\frac{\beta}{2}$ each.

When $\beta = 0$, the agent always walk as the action and never slip. As the β grow larger, it is clearly that the policy tend to stay away from the -1 tiles. when $\beta = 0.2$, the policy on the state (2,1) tend to move to (3,1), and when $\beta = 0.4$, even the policy in the state (2,2) and (2,3) tend to move upward, instead of follow the shortest path to the tile 10. Which is because the policy tend to avoid the danger of slipping into negative rewards as the β grow larger.

However when $\beta = 0.66$, it is more likely that the next action is moving upward from the initial state, into the -1 tile, which may because as β grow larger, when the next action is moving upward, the probability of slipping out of the tile -1 is larger than moving to the (2,2) or (3,2). This cause a strange situation, which is that proactively entering the -1 tile could be the best choose than moving left from initial state.

Environment D

Q1 Has the agent learned a good policy? Why / Why not ?

A Yes, the agent learned a good policy since each state take the optimal route to the goal.

Q2 Could you have used the Q-learning algorithm to solve this task ?

A No, in order to solve the task, Q-learning must maintain a huge Q-table for each goal position. Each Q-table can't be apply on the new goal position, i.e. it only depends on the goal that it trained on.

Environment E

Q1 Has the agent learned a good policy? Why / Why not ?

A No, most of the tiles show upwards trend even though the goal positions are below them or on their left and right sides.

Q2 If the results obtained for environments D and E differ, explain why.

A The REINFORCE algorithm use trained parameters to predict each step in each state. The distribution of training samples has some bias from the validation distribution, which leads to a large bias in the model we trained. If we want to get better result, we should use data close to the distribution we want to predict as a training set.