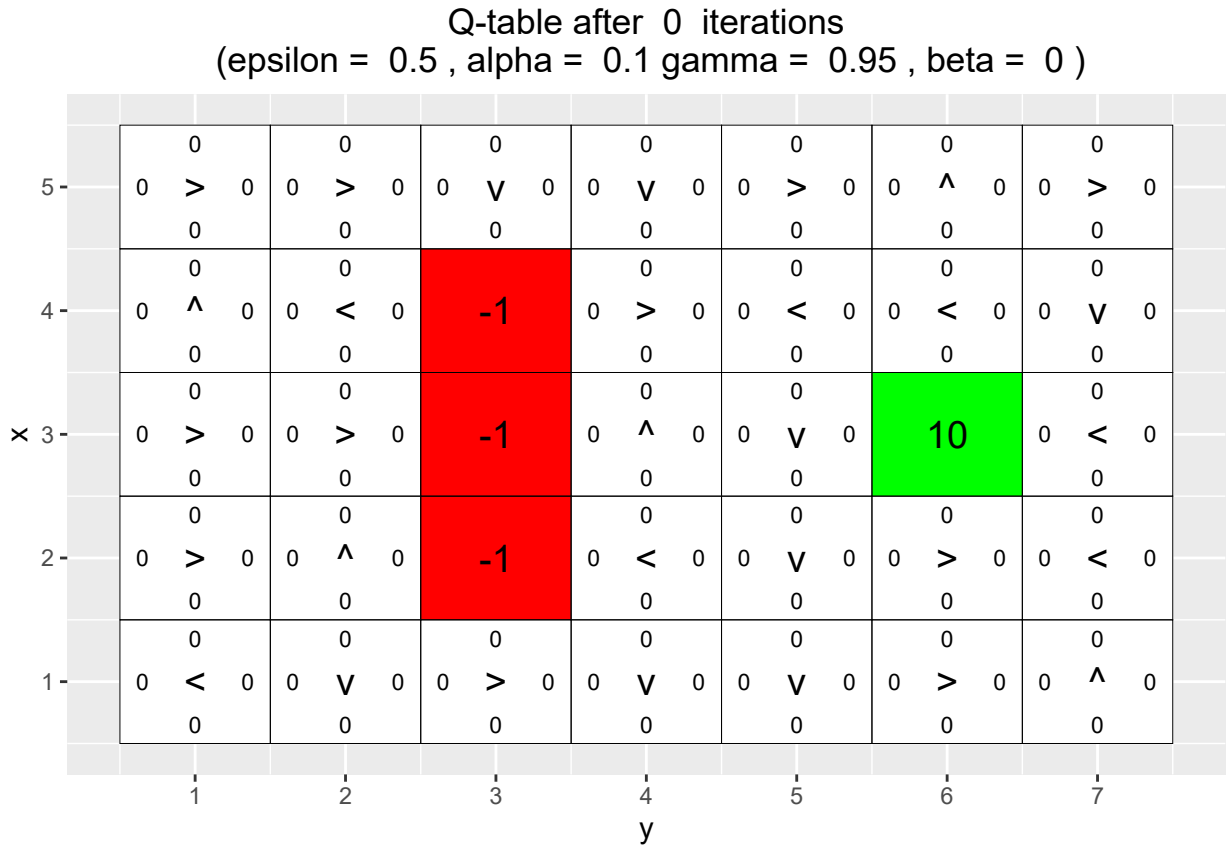
lab3_report

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

  maxValues <- which(q_table[x, y, 1:4] == max(q_table[x,y,1:4]))
  if(length(maxValues)>1){
    nextMove <- sample(maxValues, 1)
    return(nextMove)
  }else{
    return(maxValues)
  }


}

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

  randGen <- runif(1,0,1)
  if(randGen <= 1-epsilon){
    return(GreedyPolicy(x,y))
  } else {
    nextMove <- sample(1:4,1)
    return(nextMove)
  }


}


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

  currState <- start_state
  episode_correction=0

  repeat{
    # Follow policy, execute action, get reward.
    x <- currState[1]
    y <- currState[2]
    nextAction <- EpsilonGreedyPolicy(x, y, epsilon)
    nextState <- transition_model(x, y, nextAction, beta)
    next_x <- nextState[1]
    next_y <- nextState[2]
    reward <- reward_map[next_x,next_y]
    # Q-table update.
    temporal_diff = reward + gamma * max(q_table[next_x,next_y, ])
    q_table[x,y,nextAction] <<- q_table[x,y,nextAction] +
      alpha*(temporal_diff - q_table[x,y,nextAction])
```

```
    episode_correction = episode_correction + temporal_diff

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

}
```

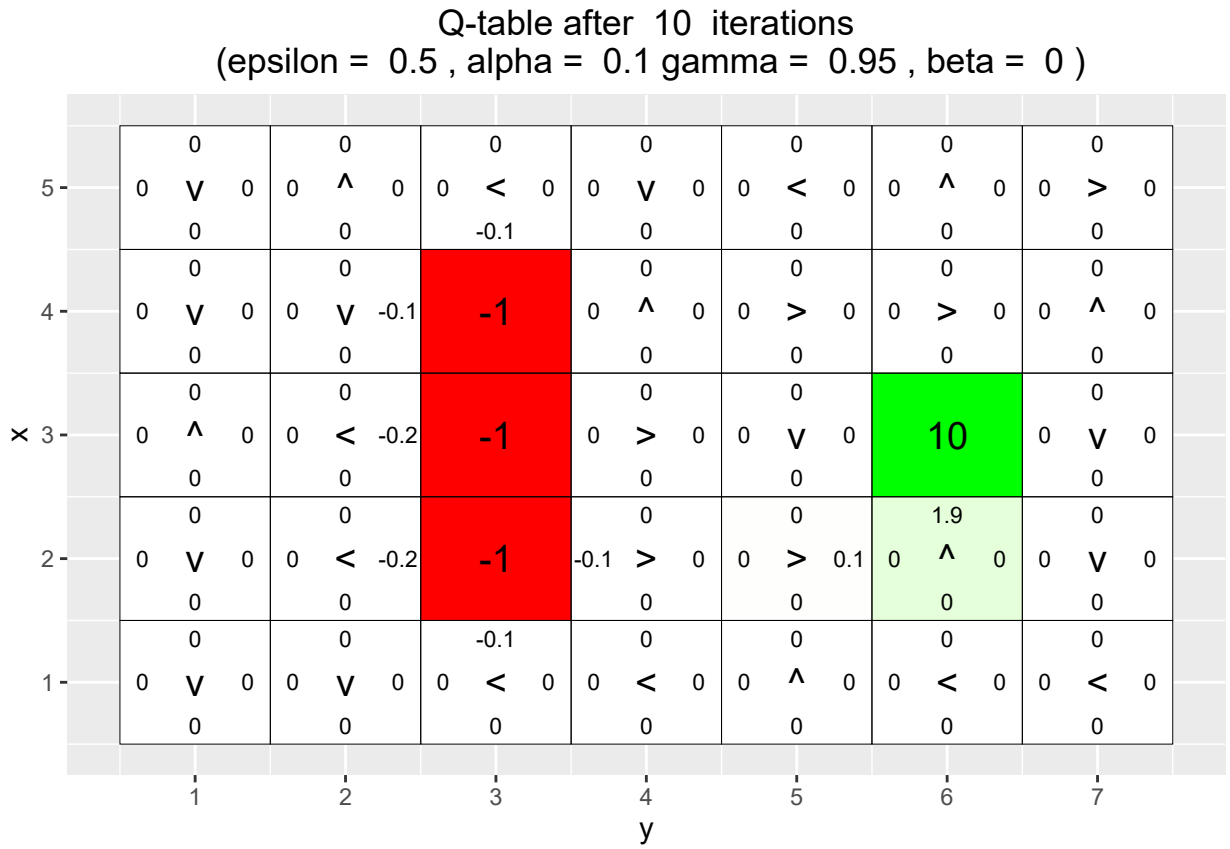
Greedy policy, etc

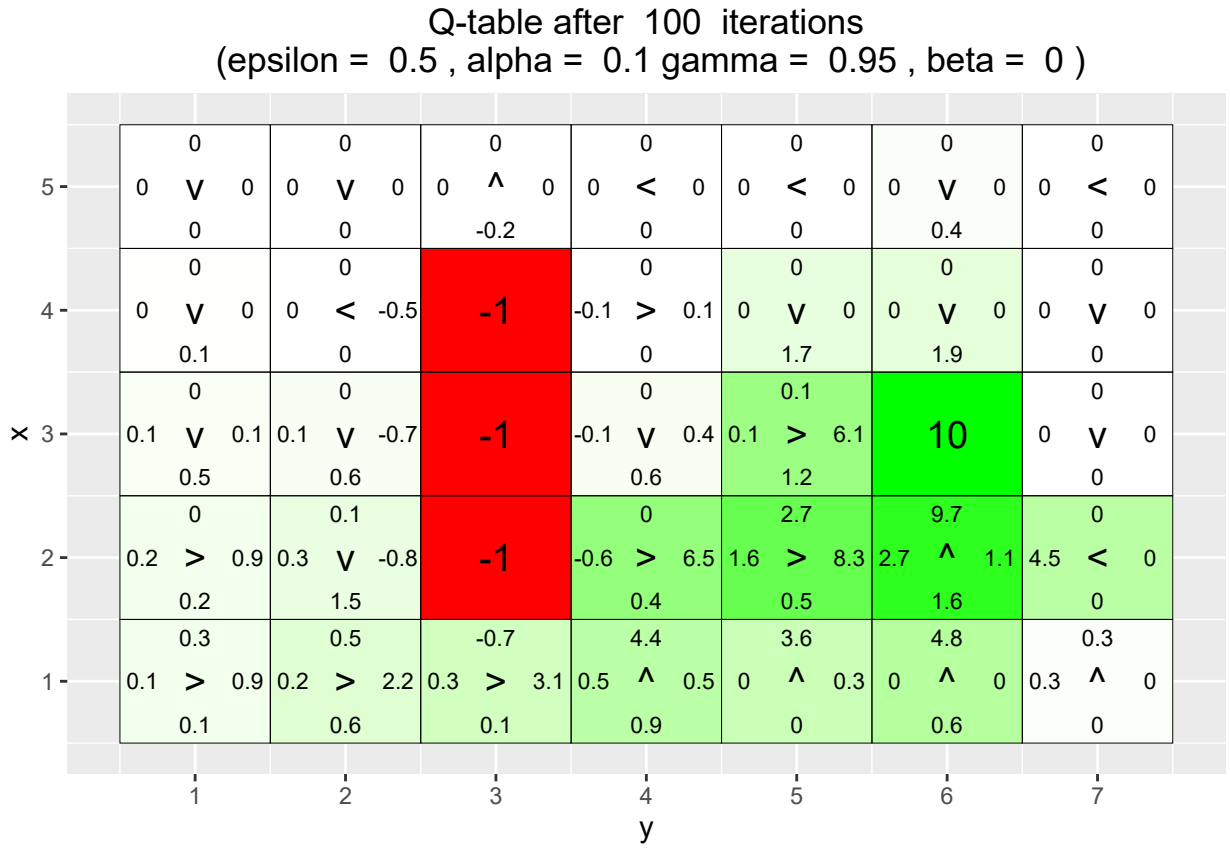
```
# Environment A (learning)

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()
```

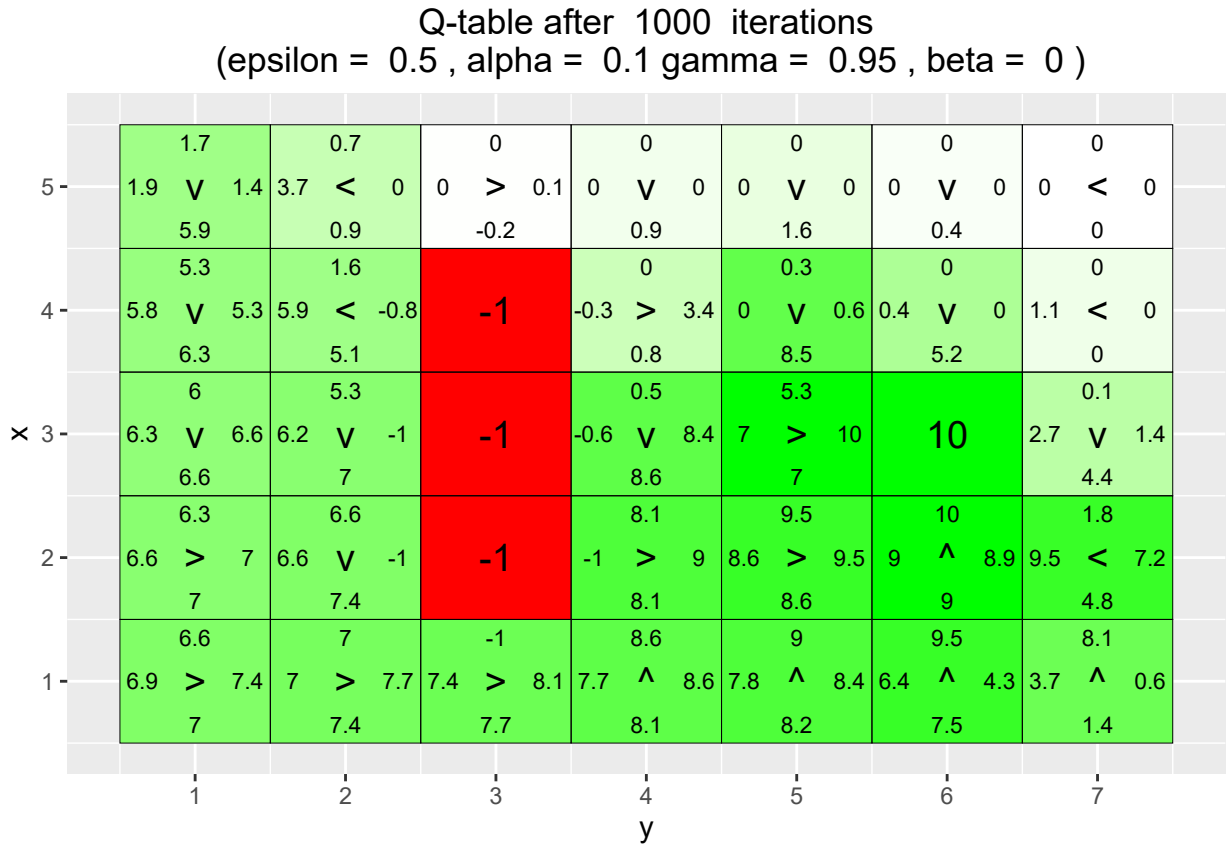


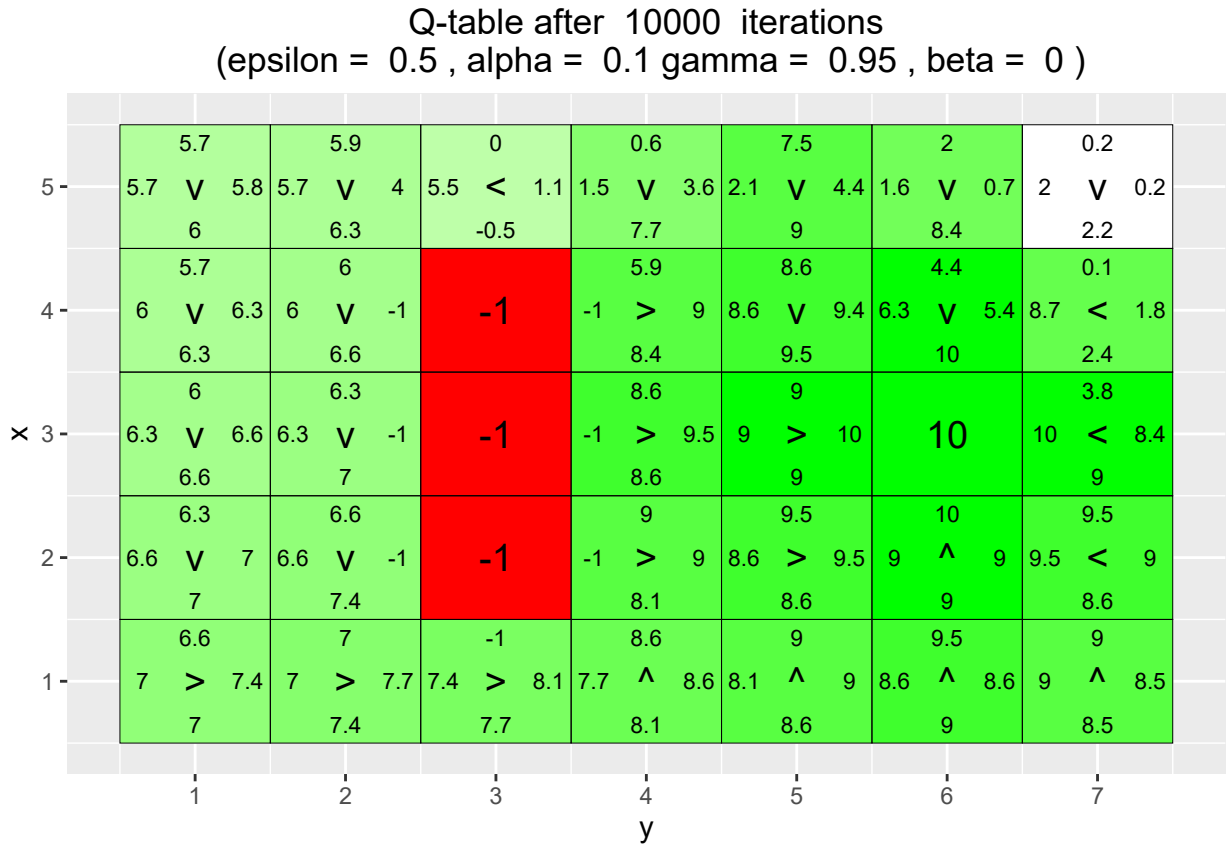

```
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))
```

```
  if(any(i==c(10,100,1000,10000)))
    vis_environment(i)
}
```




Q-table after 100 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
1
2
3
4
5
6
7
0 v 0 0 0 0
0 v 0 0 0 0
0 ^ 0 -0.2
0 < 0 0
0 < 0 0
0 v 0 0.4
0 < 0 0
0 v 0 0.1
0 < -0.5 0
-1
-0.1 > 0.1 0
0 v 0 1.7
0 v 0 1.9
0 v 0 0
0.1 v 0.1 0.5
0.1 v -0.7 0.6
-1
-0.1 v 0.4 0.6
0.1 > 6.1 1.2
10
0 v 0 0
0.2 > 0.9 0.2
0.3 v -0.8 1.5
-1
-0.6 > 6.5 0.4
1.6 > 8.3 0.5
2.7 ^ 1.1 1.6
4.5 < 0 0
0.1 > 0.9 0.1
0.2 > 2.2 0.6
0.3 > 3.1 0.1
0.5 ^ 0.5 0.9
0 ^ 0.3 0
0 ^ 0 0.6
0.3 ^ 0 0


Q-table after 1000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
5
4
3
2
1
1
2
3
4
5
6
7
1.7
1.9 V 1.4
5.9
0.7
3.7 < 0
0.9
0
0 > 0.1
-0.2
0
0 V 0
0.9
0
0 V 0
1.6
0
0 V 0
0.4
0
0 < 0
0
5.3
5.8 V 5.3
6.3
1.6
5.9 < -0.8
5.1
-1
0
-0.3 > 3.4
0.8
0.3
0 V 0.6
8.5
0
0.4 V 0
5.2
0
1.1 < 0
0
6
6.3 V 6.6
6.6
5.3
6.2 V -1
7
-1
0.5
-0.6 V 8.4
8.6
5.3
7 > 10
7
10
0.1
2.7 V 1.4
4.4
6.3
6.6 > 7
7
6.6
6.6 V -1
7.4
-1
8.1
-1 > 9
8.1
9.5
8.6 > 9.5
8.6
10
9 ^ 8.9
9
1.8
9.5 < 7.2
4.8
6.6
6.9 > 7.4
7
7
7 > 7.7
7.4
-1
7.4 > 8.1
7.7
8.6
7.7 ^ 8.6
8.1
9
7.8 ^ 8.4
8.2
9.5
6.4 ^ 4.3
7.5
8.1
3.7 ^ 0.6
1.4



Question 2.2:

What has the agent learned after the first 10 episodes? To not enter -1 and to go towards 10

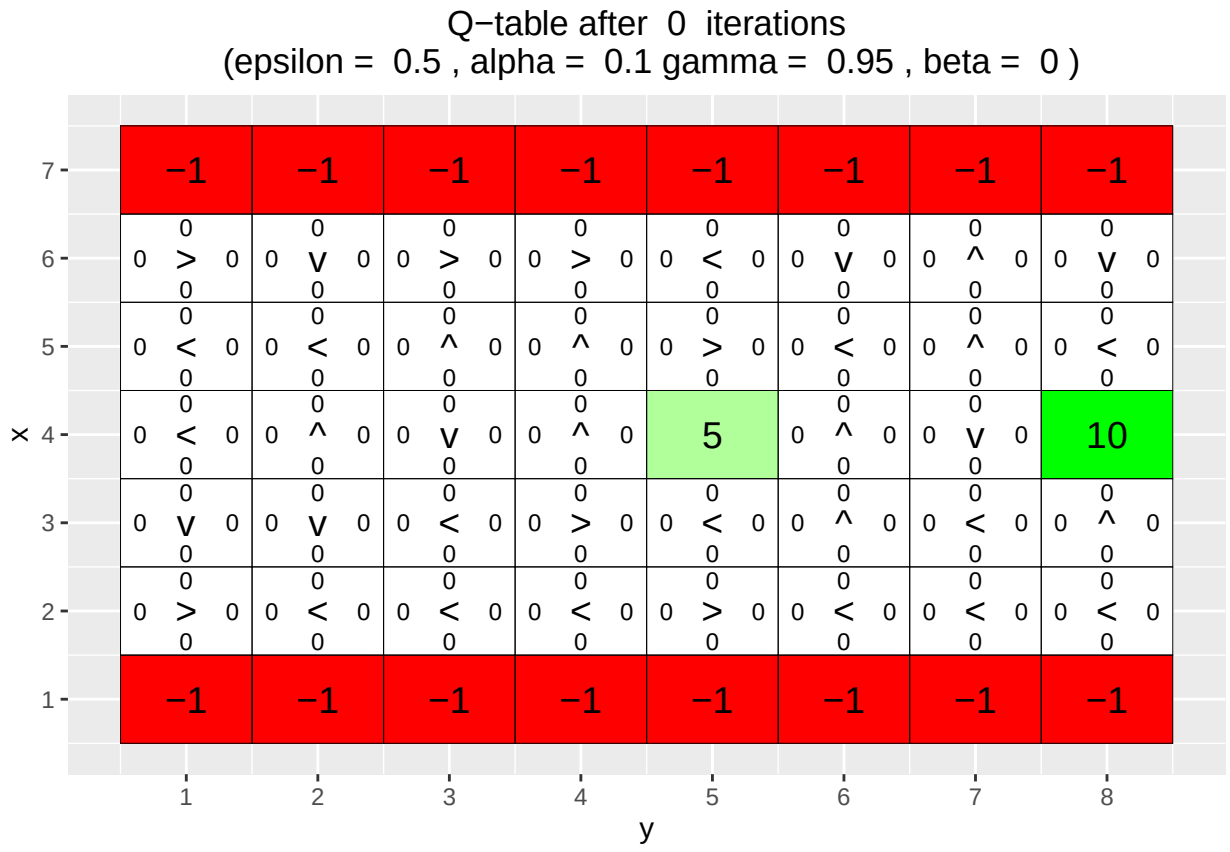
Is the final greedy policy (after 10000 episodes) optimal? Why / Why not? Less explored areas still recommend strange paths, not optimal?

Does the agent learn that there are multiple paths to get to the positive reward ? If not, what could be done to make the agent learn this? It does not learn that there are 2 equal paths to get to the reward. It becomes biased after the first randomly chosen path. To be able to understand that there are multiple paths it needs to get multiple correct randomized choices toward the other path in a row which is quite unlikely. /Train more - more randomness?

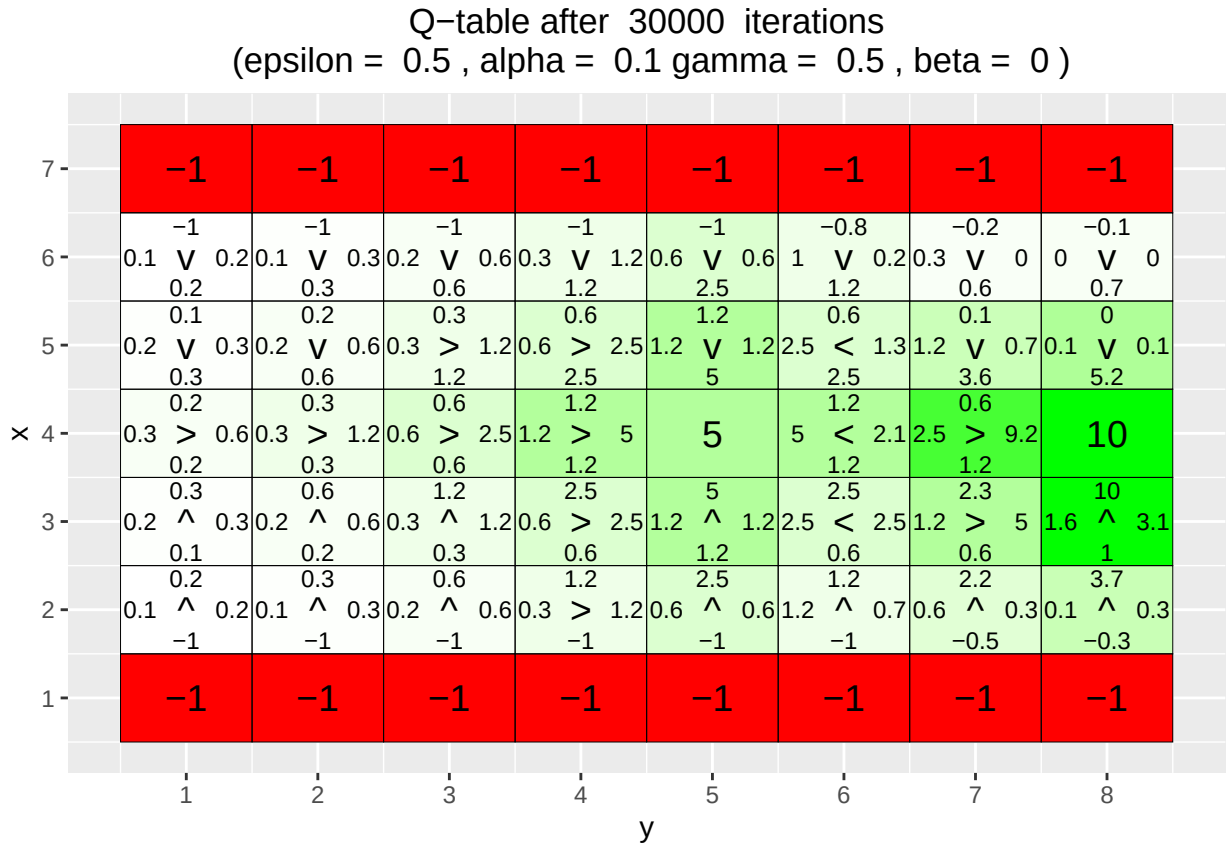
Assignment 2.3

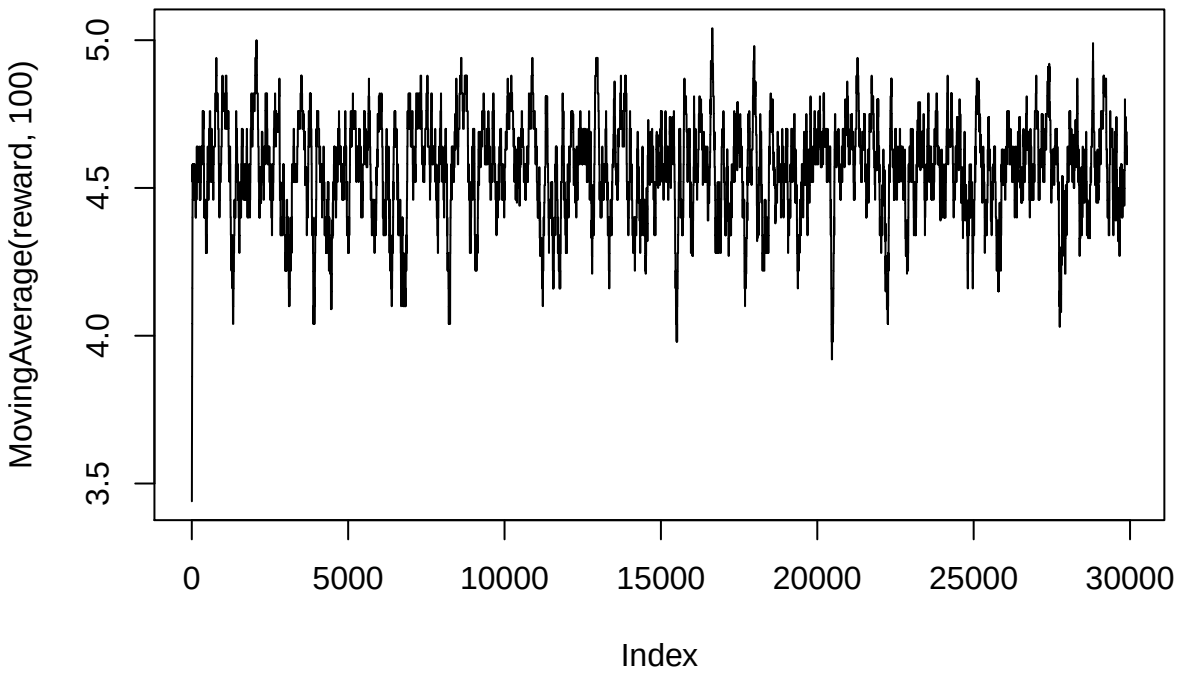
```
H <- 7
W <- 8

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))
```

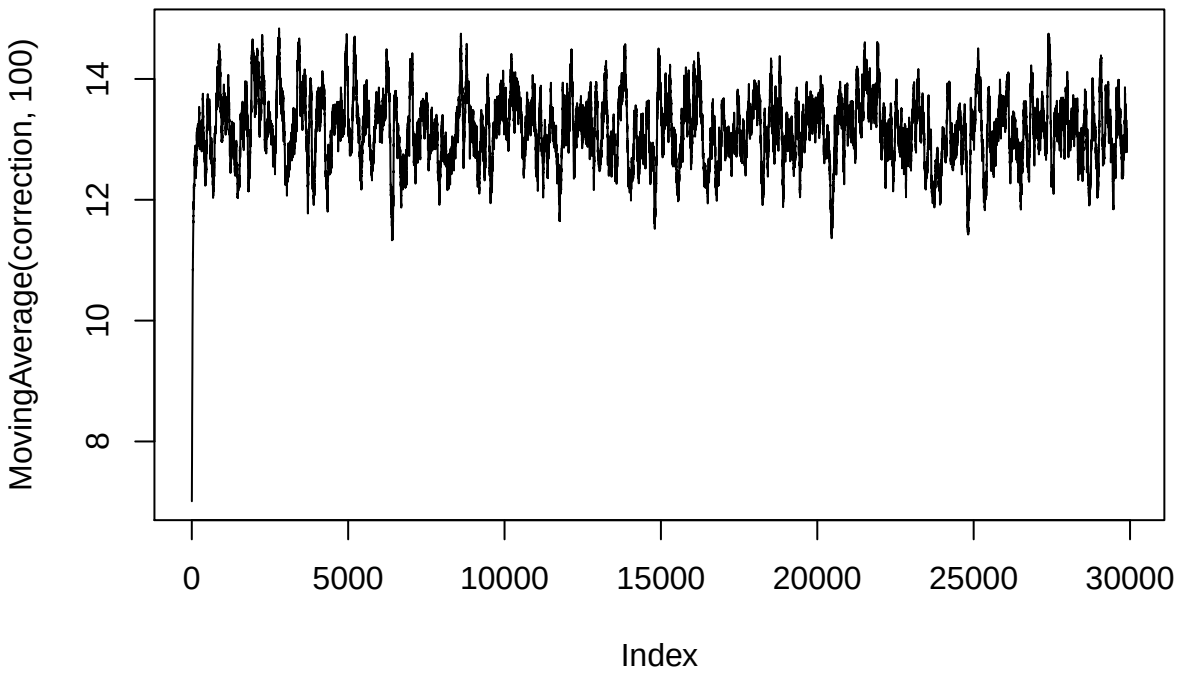
```
vis_environment()
```

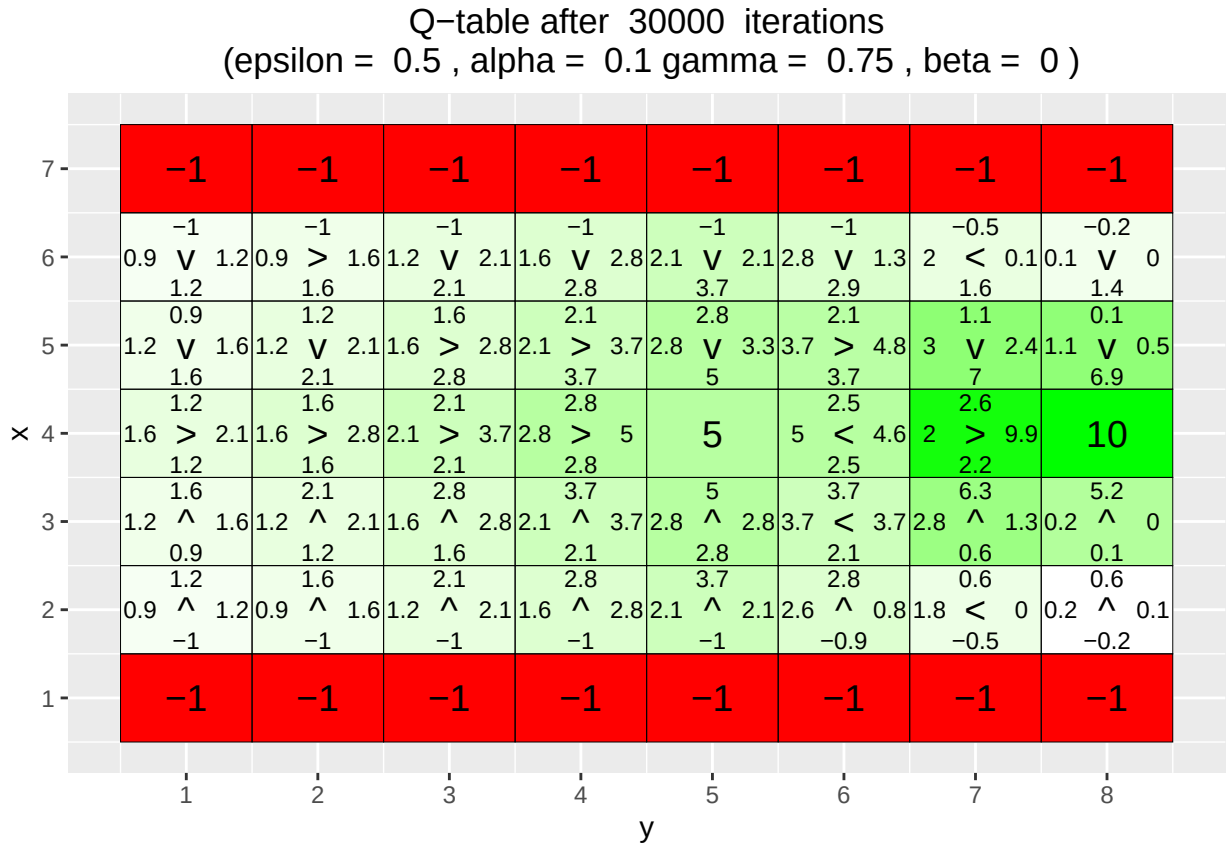


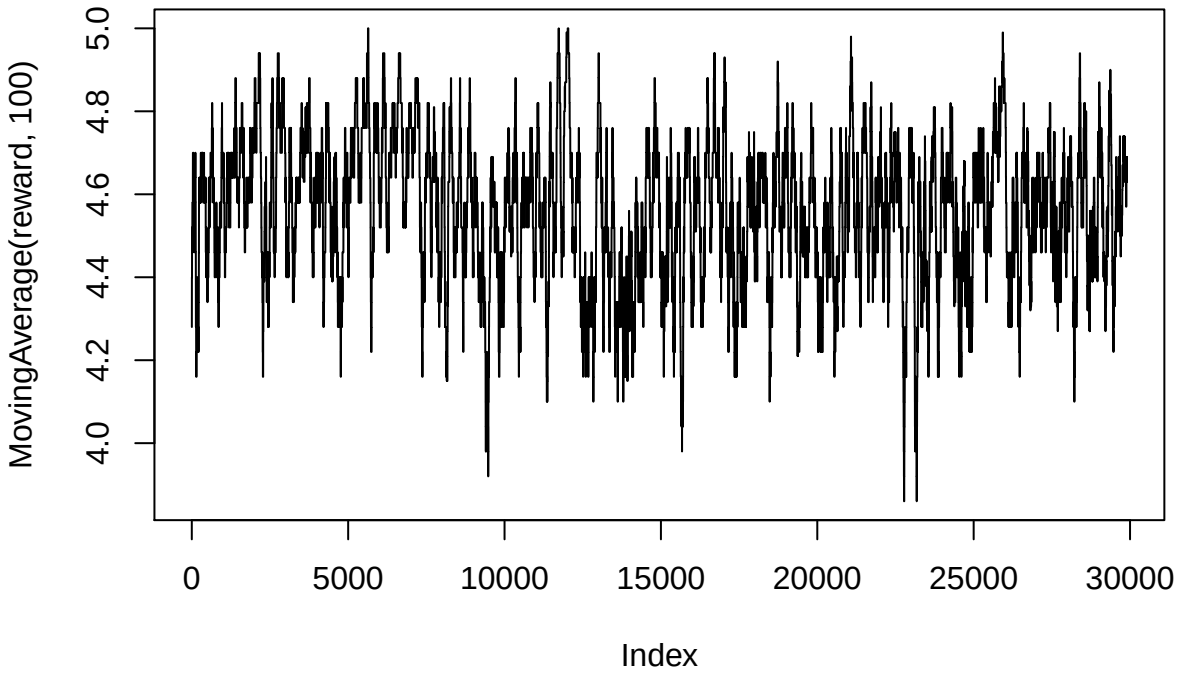

```
MovingAverage <- function(x, n){

  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")
}
```

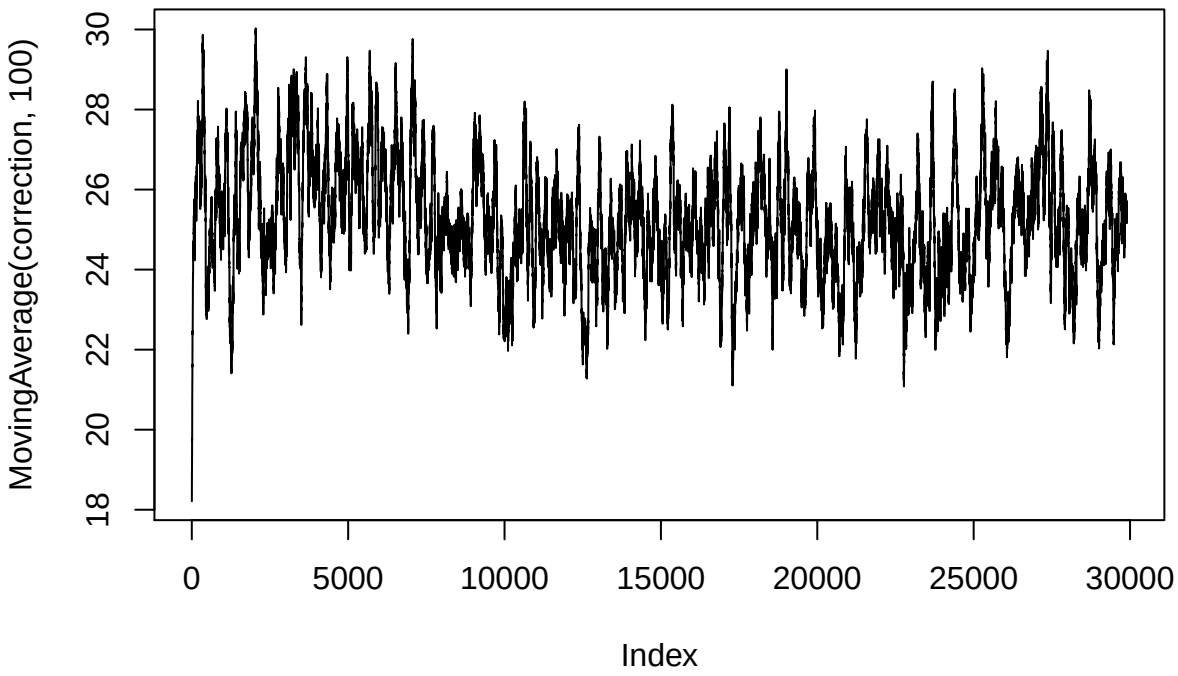

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
7 −1 −1 −1 −1 −1 −1 −1 −1
6 −1 0.1 V 0.2 0.2 | −1 0.1 V 0.3 0.3 | −1 0.2 V 0.6 0.6 | −1 0.3 V 1.2 1.2 | −1 0.6 V 0.6 2.5 | −0.8 1 V 0.2 1.2 | −0.2 0.3 V 0 0.6 | −0.1 0 V 0 0.7
5 0.1 0.2 V 0.3 0.3 | 0.2 0.2 V 0.6 0.6 | 0.3 0.3 > 1.2 1.2 | 0.6 0.6 > 2.5 2.5 | 1.2 1.2 V 1.2 5 | 0.6 2.5 < 1.3 2.5 | 0.1 1.2 V 0.7 3.6 | 0 0.1 V 0.1 5.2
4 0.2 0.3 > 0.6 0.2 | 0.3 0.3 > 1.2 0.3 | 0.6 0.6 > 2.5 0.6 | 1.2 1.2 > 5 1.2 | 5 | 1.2 5 < 2.1 1.2 | 0.6 2.5 > 9.2 1.2 | 10
3 0.3 0.2 ^ 0.3 0.1 | 0.6 0.2 ^ 0.6 0.2 | 1.2 0.3 ^ 1.2 0.3 | 2.5 0.6 > 2.5 0.6 | 5 1.2 ^ 1.2 1.2 | 2.5 2.5 < 2.5 0.6 | 2.3 1.2 > 5 0.6 | 10 1.6 ^ 3.1 1
2 0.2 0.1 ^ 0.2 −1 | 0.3 0.1 ^ 0.3 −1 | 0.6 0.2 ^ 0.6 −1 | 1.2 0.3 > 1.2 −1 | 2.5 0.6 ^ 0.6 −1 | 1.2 1.2 ^ 0.7 −1 | 2.2 0.6 ^ 0.3 −0.5 | 3.7 0.1 ^ 0.3 −0.3
1 −1 −1 −1 −1 −1 −1 −1 −1
1 2 3 4 5 6 7 8

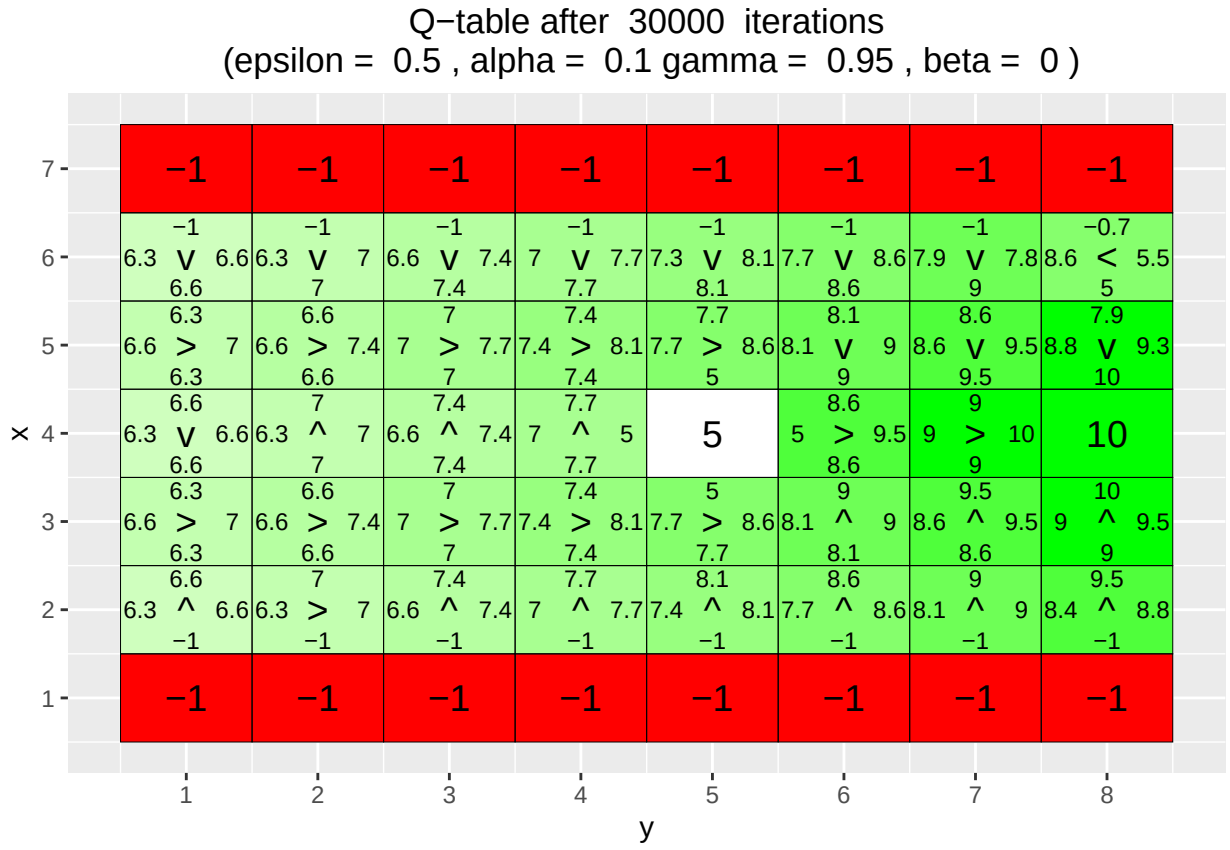

MovingAverage(reward, 100)
3.5
4.0
4.5
5.0
0
5000
10000
15000
20000
25000
30000
Index

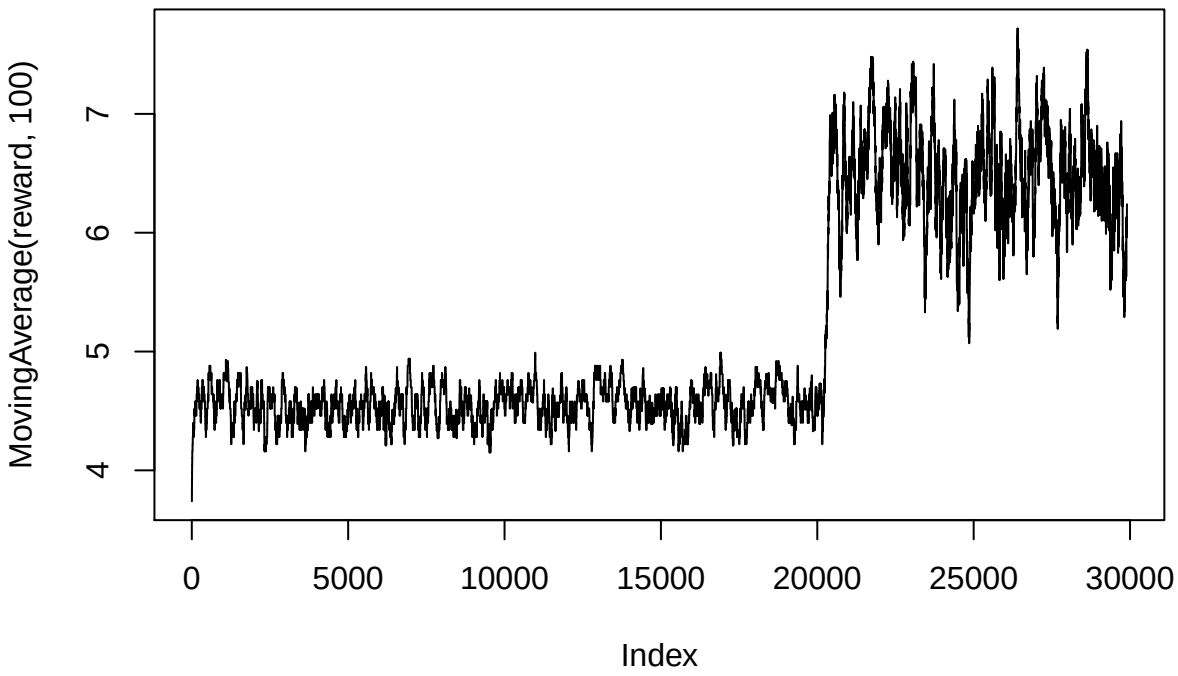

MovingAverage(correction, 100)
8
10
12
14
0
5000
10000
15000
20000
25000
30000
Index

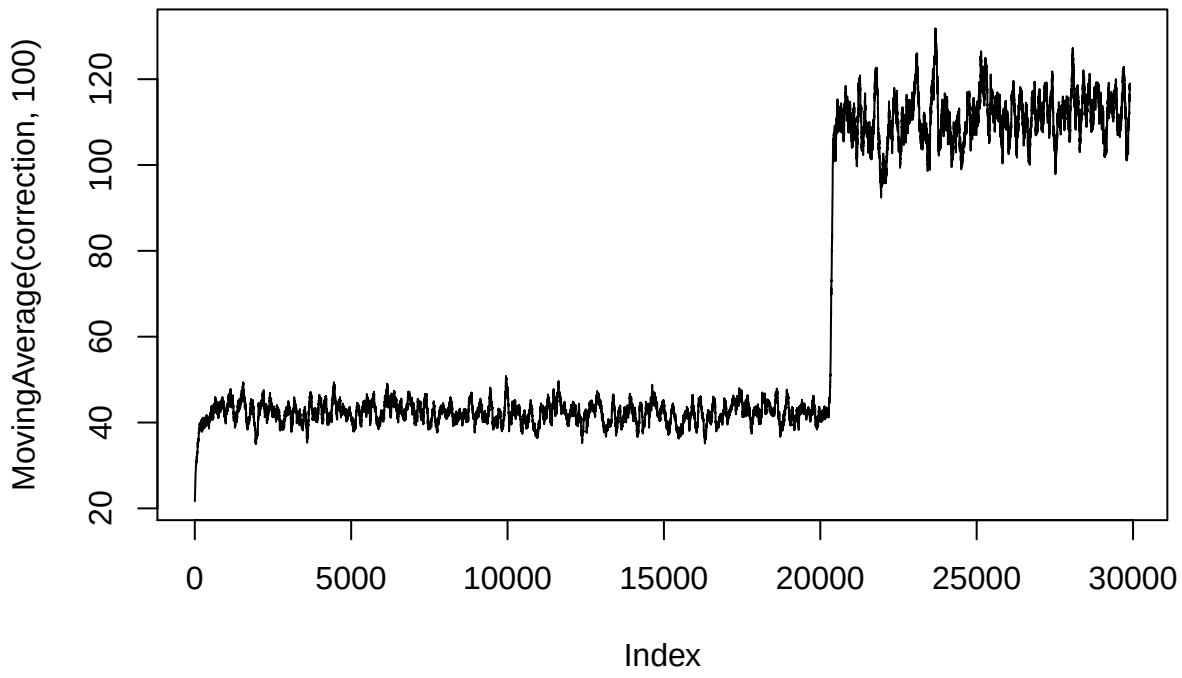

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
−1 −1 −1 −1 −1 −1 −1 −1
−1 −1 −1 −1 −1 −1 −0.5 −0.2
0.9 V 1.2 0.9 > 1.6 1.2 V 2.1 1.6 V 2.8 2.1 V 2.1 2.8 V 1.3 2 < 0.1 0.1 V 0
1.2 1.6 2.1 2.8 3.7 2.9 1.6 1.4
0.9 1.2 1.6 2.1 2.8 2.1 1.1 0.1
1.2 V 1.6 1.2 V 2.1 1.6 > 2.8 2.1 > 3.7 2.8 V 3.3 3.7 > 4.8 3 V 2.4 1.1 V 0.5
1.6 2.1 2.8 3.7 5 3.7 7 6.9
1.2 1.6 2.1 2.8 2.5 2.6
1.6 > 2.1 1.6 > 2.8 2.1 > 3.7 2.8 > 5 5 5 < 4.6 2 > 9.9 10
1.2 1.6 2.1 2.8 2.5 2.2
1.6 2.1 2.8 3.7 5 3.7 6.3 5.2
1.2 ^ 1.6 1.2 ^ 2.1 1.6 ^ 2.8 2.1 ^ 3.7 2.8 ^ 2.8 3.7 < 3.7 2.8 ^ 1.3 0.2 ^ 0
0.9 1.2 1.6 2.1 2.8 2.1 0.6 0.1
1.2 1.6 2.1 2.8 3.7 2.8 0.6 0.6
0.9 ^ 1.2 0.9 ^ 1.6 1.2 ^ 2.1 1.6 ^ 2.8 2.1 ^ 2.1 2.6 ^ 0.8 1.8 < 0 0.2 ^ 0.1
−1 −1 −1 −1 −1 −0.9 −0.5 −0.2
−1 −1 −1 −1 −1 −1 −1 −1


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

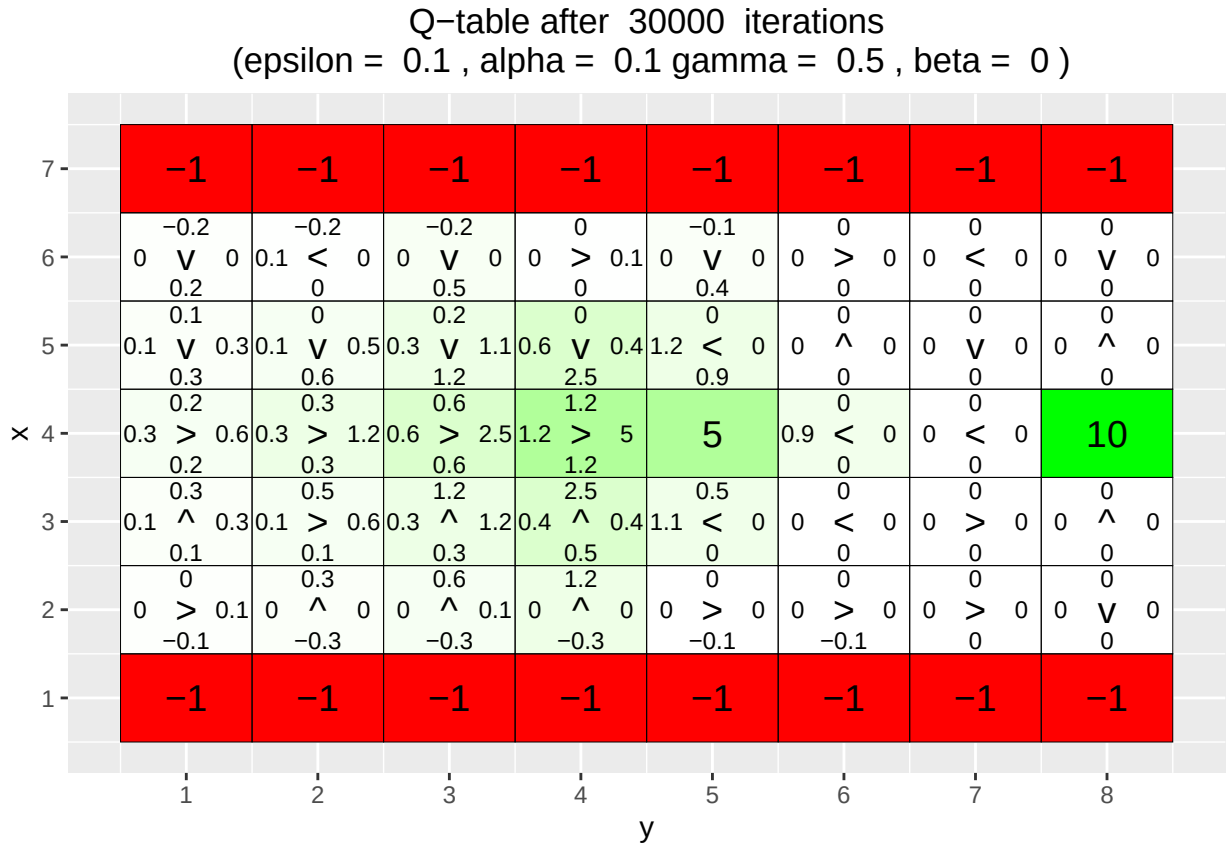

MovingAverage(correction, 100)
18
20
22
24
26
28
30
0
5000
10000
15000
20000
25000
30000
Index

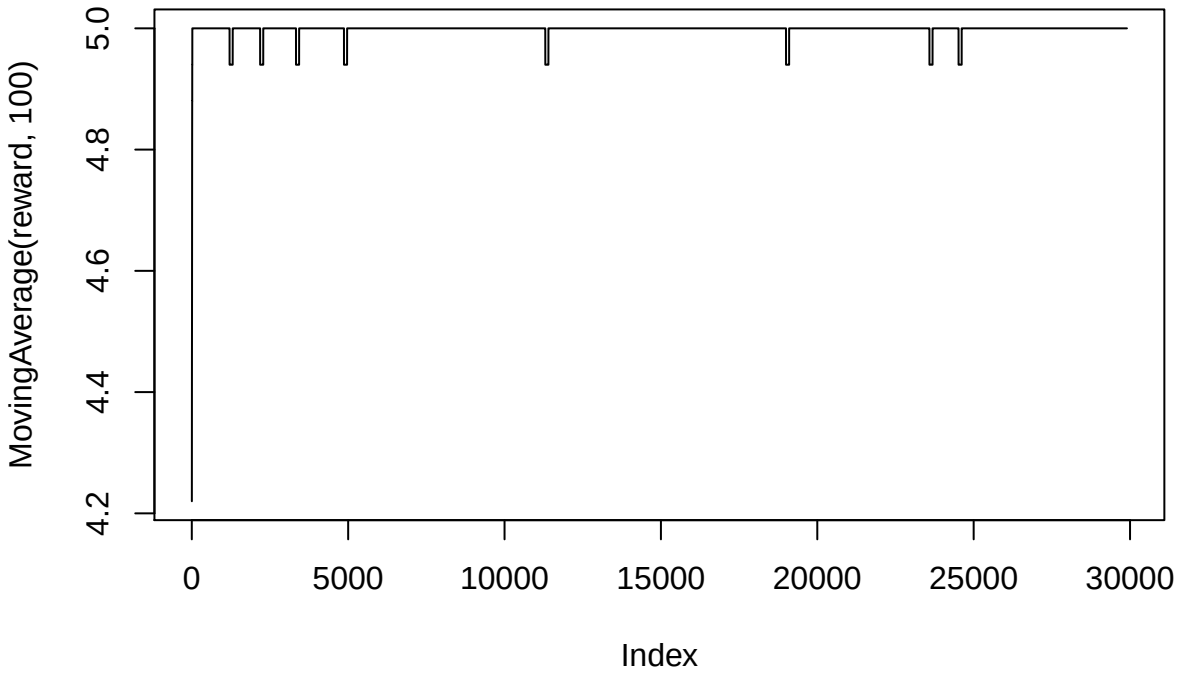

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
−1 −1 −1 −1 −1 −1 −1 −1
−1 −1 −1 −1 −1 −1 −1 −0.7
6.3 V 6.6 6.3 V 7 6.6 V 7.4 7 V 7.7 7.3 V 8.1 7.7 V 8.6 7.9 V 7.8 8.6 < 5.5
6.6 7 7.4 7.7 8.1 8.6 9 5
6.3 6.6 7 7.4 7.7 8.1 8.6 7.9
6.6 > 7 6.6 > 7.4 7 > 7.7 7.4 > 8.1 7.7 > 8.6 8.1 V 9 8.6 V 9.5 8.8 V 9.3
6.3 6.6 7 7.4 5 9 9.5 10
6.6 7 7.4 7.7 8.6 9
6.3 V 6.6 6.3 ^ 7 6.6 ^ 7.4 7 ^ 5 5 5 > 9.5 9 > 10 10
6.6 7 7.4 7.7 8.6 9
6.3 6.6 7 7.4 5 9 9.5 10
6.6 > 7 6.6 > 7.4 7 > 7.7 7.4 > 8.1 7.7 > 8.6 8.1 ^ 9 8.6 ^ 9.5 9 ^ 9.5
6.3 6.6 7 7.4 7.7 8.1 8.6 9
6.6 7 7.4 7.7 8.1 8.6 9 9.5
6.3 ^ 6.6 6.3 > 7 6.6 ^ 7.4 7 ^ 7.7 7.4 ^ 8.1 7.7 ^ 8.6 8.1 ^ 9 8.4 ^ 8.8
−1 −1 −1 −1 −1 −1 −1 −1
−1 −1 −1 −1 −1 −1 −1 −1

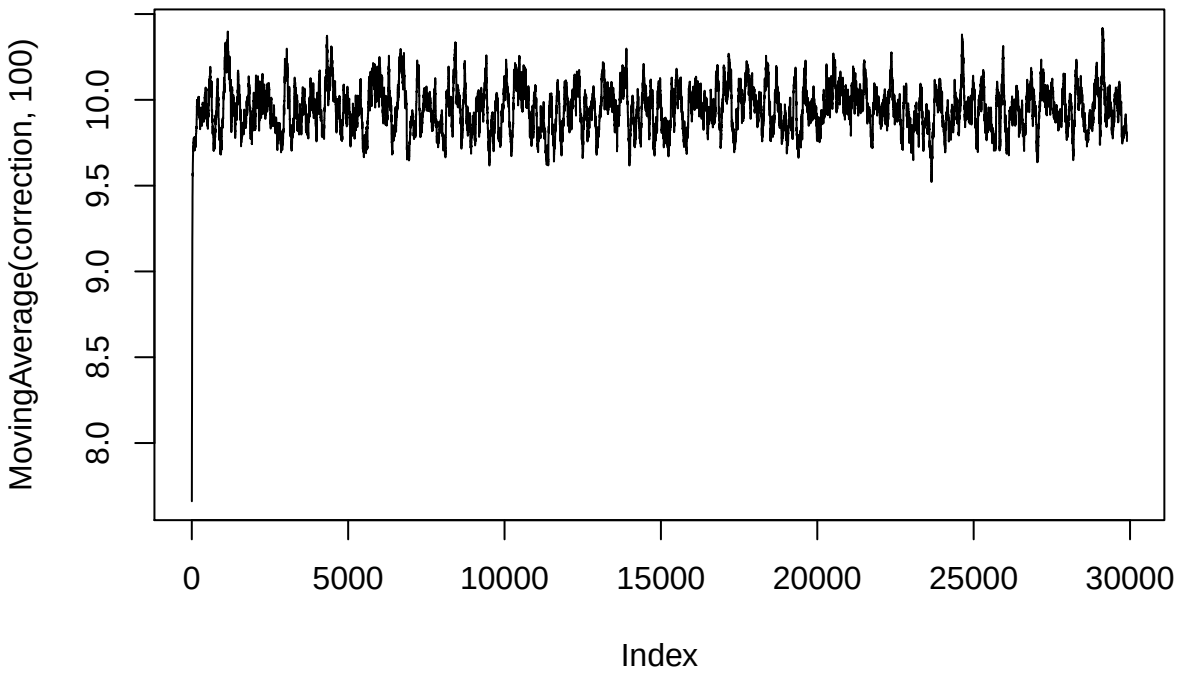

MovingAverage(reward, 100)
4
5
6
7
0
5000
10000
15000
20000
25000
30000
Index

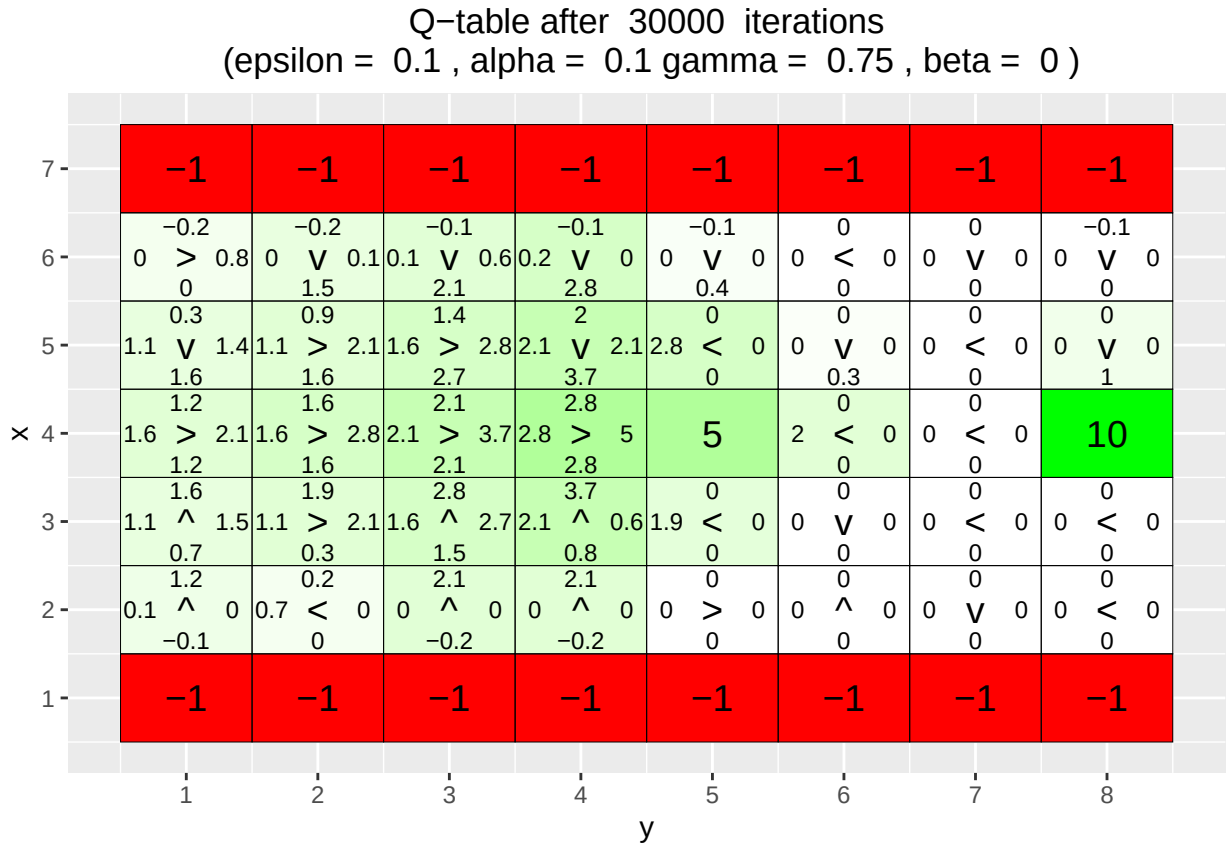


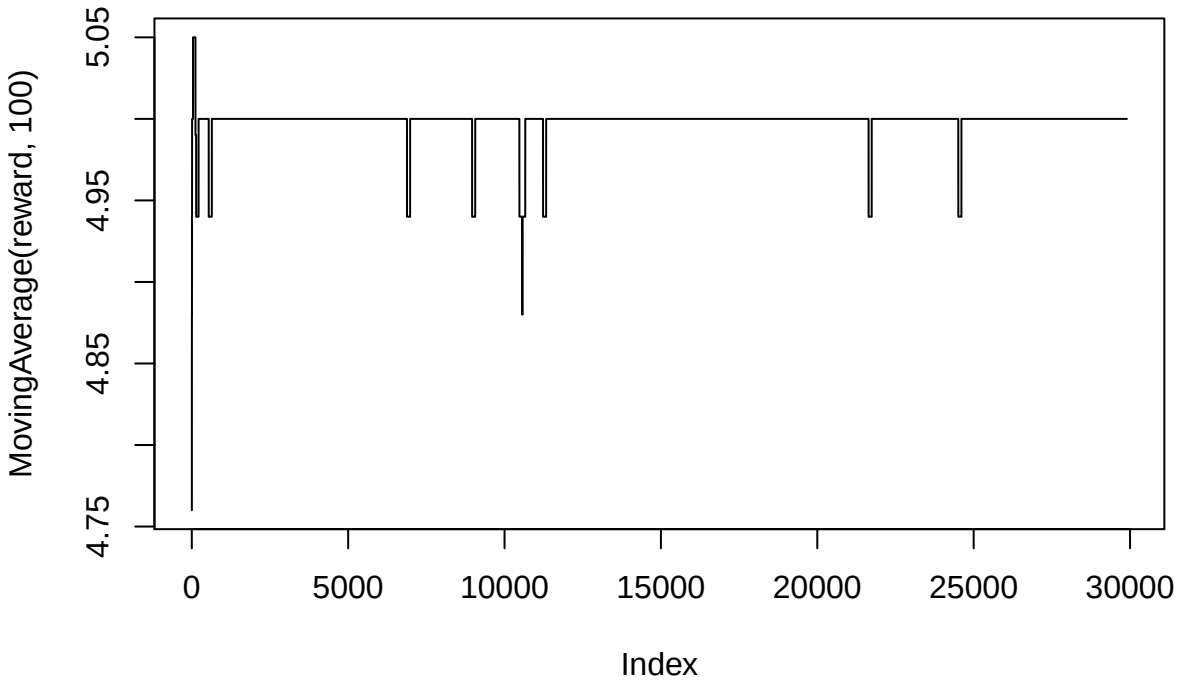

```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  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")
}
```

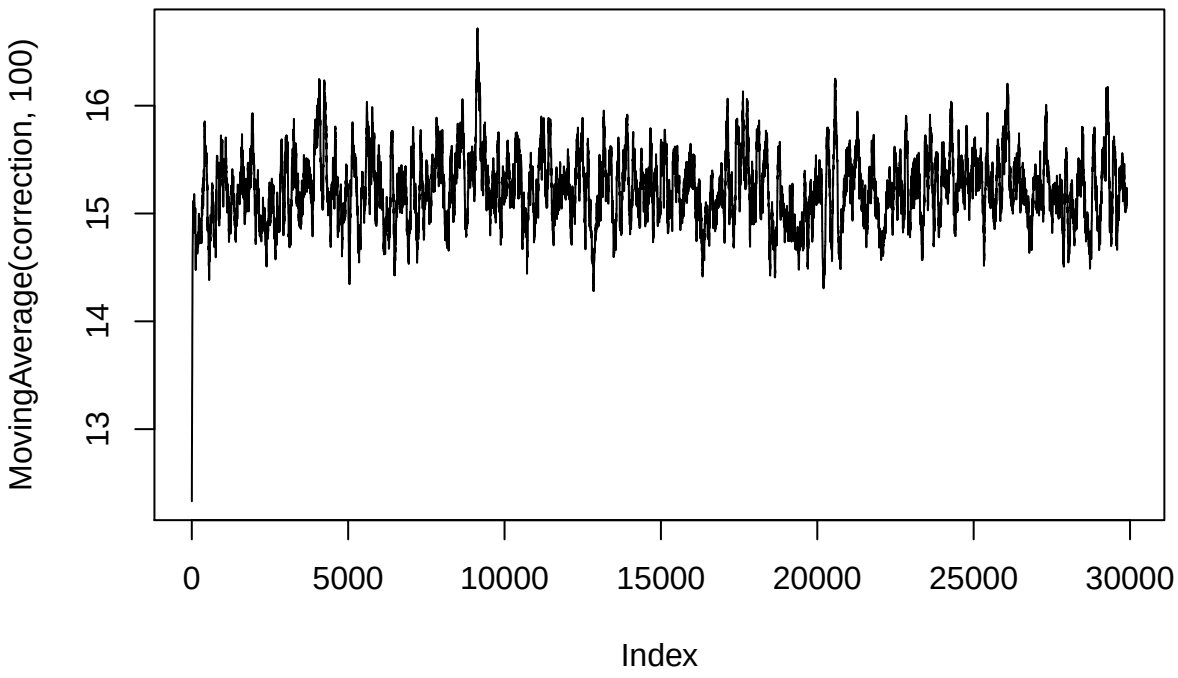

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
−1
−0.2 0 V 0 0.2
−0.2 0.1 < 0 0
−0.2 0 V 0 0.5
0 0 > 0.1 0
−0.1 0 V 0 0.4
0 0 > 0 0
0 0 < 0 0
0 0 V 0 0
0.1 0.1 V 0.3 0.3
0 0.1 V 0.5 0.6
0.2 0.3 V 1.1 1.2
0 0.6 V 0.4 2.5
0 1.2 < 0 0.9
0 0 ^ 0 0
0 0 V 0 0
0 0 ^ 0 0
0.2 0.3 > 0.6 0.2
0.3 0.3 > 1.2 0.3
0.6 0.6 > 2.5 0.6
1.2 1.2 > 5 1.2
5
0 0.9 < 0 0
0 0 < 0 0
10
0.3 0.1 ^ 0.3 0.1
0.5 0.1 > 0.6 0.1
1.2 0.3 ^ 1.2 0.3
2.5 0.4 ^ 0.4 0.5
0.5 1.1 < 0 0
0 0 < 0 0
0 0 > 0 0
0 0 ^ 0 0
0 0 > 0.1 −0.1
0.3 0 ^ 0 −0.3
0.6 0 ^ 0.1 −0.3
1.2 0 ^ 0 −0.3
0 0 > 0 −0.1
0 0 > 0 −0.1
0 0 > 0 0
0 0 V 0 0

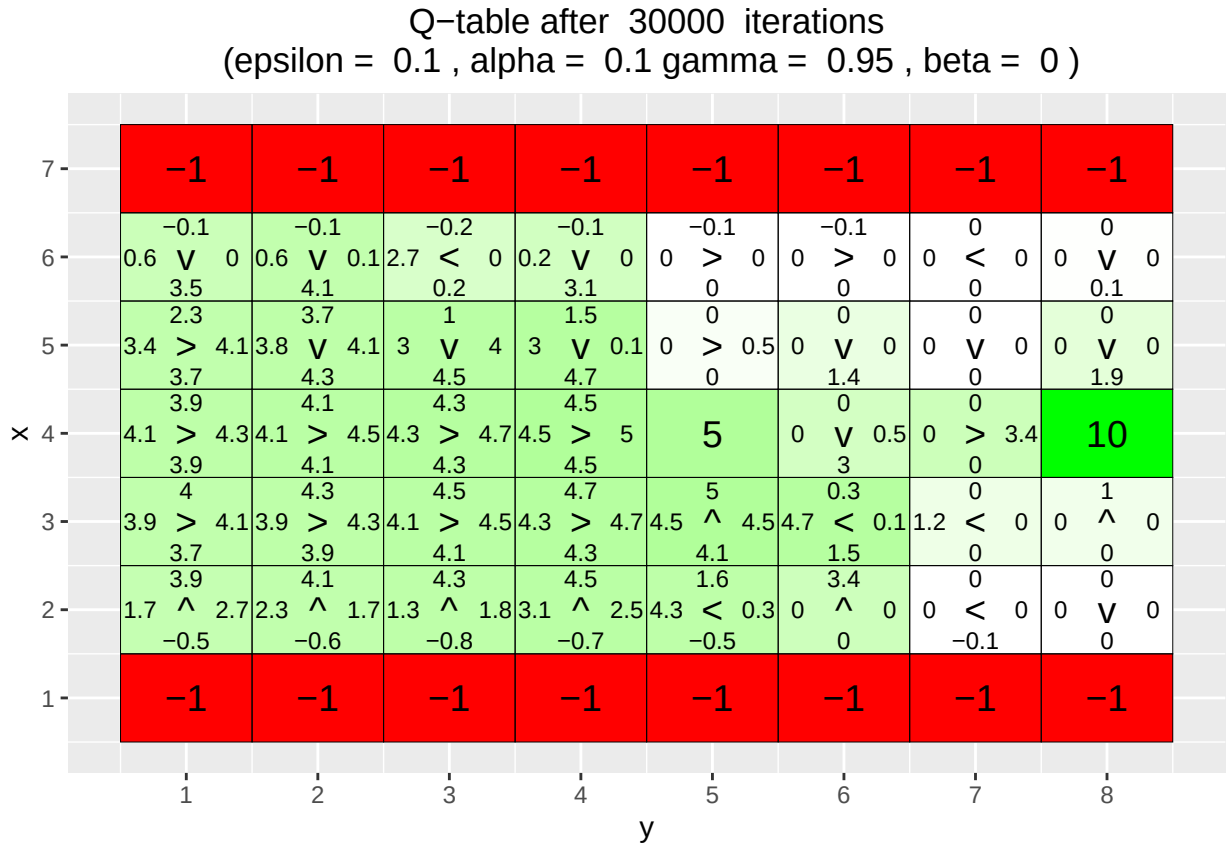

MovingAverage(reward, 100)
4.2
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index

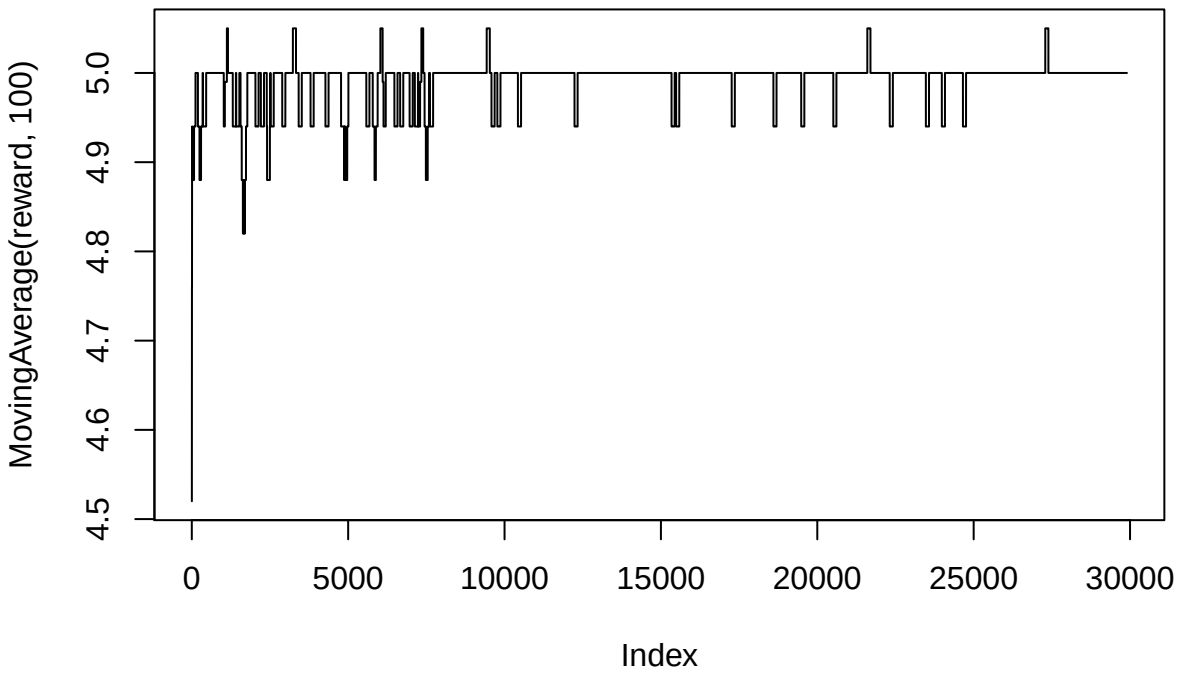

MovingAverage(correction, 100)
8.0
8.5
9.0
9.5
10.0
0
5000
10000
15000
20000
25000
30000
Index

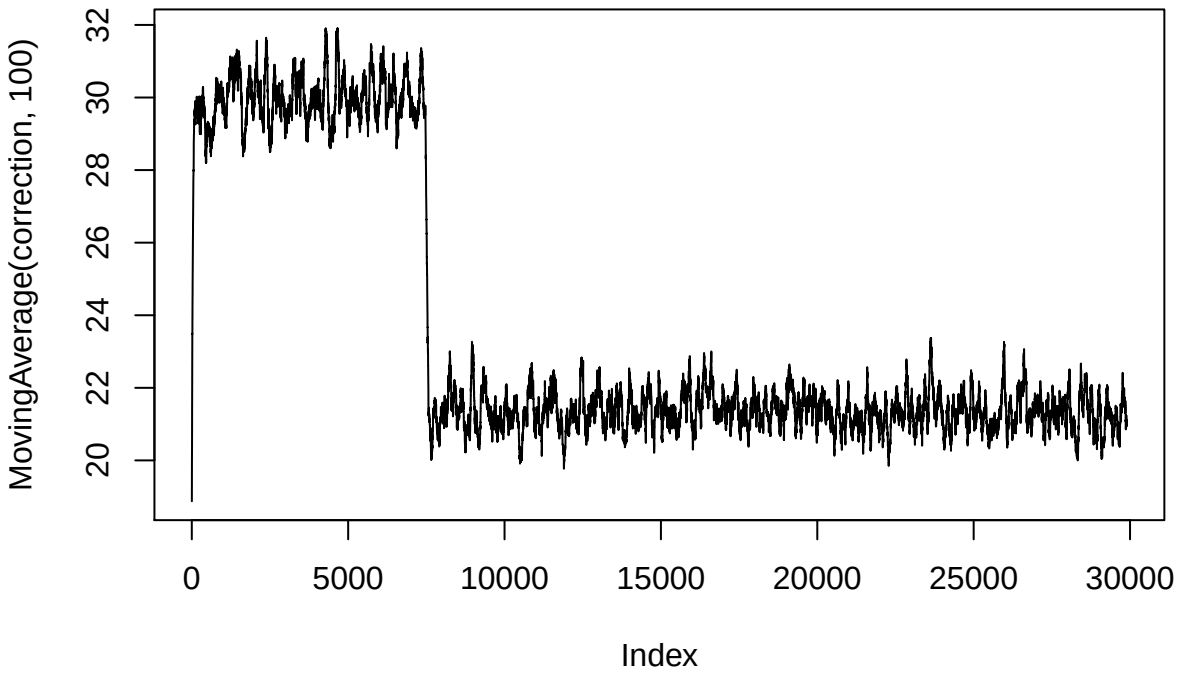

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
−1 −1 −1 −1 −1 −1 −1 −1
−0.2 −0.2 −0.1 −0.1 −0.1 0 0 −0.1
0 > 0.8 0 ∨ 0.1 0.1 ∨ 0.6 0.2 ∨ 0 0 ∨ 0 0 < 0 0 ∨ 0 0 ∨ 0
0 1.5 2.1 2.8 0.4 0 0 0
0.3 0.9 1.4 2 0 0 0 0
1.1 ∨ 1.4 1.1 > 2.1 1.6 > 2.8 2.1 ∨ 2.1 2.8 < 0 0 ∨ 0 0 < 0 0 ∨ 0
1.6 1.6 2.7 3.7 0 0.3 0 1
1.2 1.6 2.1 2.8 0 0 0 0
1.6 > 2.1 1.6 > 2.8 2.1 > 3.7 2.8 > 5 5 2 < 0 0 < 0 10
1.2 1.6 2.1 2.8 0 0 0 0
1.6 1.9 2.8 3.7 0 0 0 0
1.1 ^ 1.5 1.1 > 2.1 1.6 ^ 2.7 2.1 ^ 0.6 1.9 < 0 0 ∨ 0 0 < 0 0 < 0
0.7 0.3 1.5 0.8 0 0 0 0
1.2 0.2 2.1 2.1 0 0 0 0
0.1 ^ 0 0.7 < 0 0 ^ 0 0 ^ 0 0 > 0 0 ^ 0 0 ∨ 0 0 < 0
−0.1 0 −0.2 −0.2 0 0 0 0
−1 −1 −1 −1 −1 −1 −1 −1


MovingAverage(reward, 100)
5.05
4.95
4.85
4.75
0
5000
10000
15000
20000
25000
30000
Index


MovingAverage(correction, 100)
13
14
15
16
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
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
−1 −1 −1 −1 −1 −1 −1 −1
−0.1 −0.1 −0.2 −0.1 −0.1 −0.1 0 0
0.6 V 0 0.6 V 0.1 2.7 < 0 0.2 V 0 0 > 0 0 > 0 0 < 0 0 V 0
3.5 4.1 0.2 3.1 0 0 0 0.1
2.3 3.7 1 1.5 0 0 0 0
3.4 > 4.1 3.8 V 4.1 3 V 4 3 V 0.1 0 > 0.5 0 V 0 0 V 0 0 V 0
3.7 4.3 4.5 4.7 0 1.4 0 1.9
3.9 4.1 4.3 4.5 0 0 0
4.1 > 4.3 4.1 > 4.5 4.3 > 4.7 4.5 > 5 5 0 V 0.5 0 > 3.4 10
3.9 4.1 4.3 4.5 3 0
4 4.3 4.5 4.7 5 0.3 0 1
3.9 > 4.1 3.9 > 4.3 4.1 > 4.5 4.3 > 4.7 4.5 ^ 4.5 4.7 < 0.1 1.2 < 0 0 ^ 0
3.7 3.9 4.1 4.3 4.1 1.5 0 0
3.9 4.1 4.3 4.5 1.6 3.4 0 0
1.7 ^ 2.7 2.3 ^ 1.7 1.3 ^ 1.8 3.1 ^ 2.5 4.3 < 0.3 0 ^ 0 0 < 0 0 V 0
−0.5 −0.6 −0.8 −0.7 −0.5 0 −0.1 0
−1 −1 −1 −1 −1 −1 −1 −1


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



Question 2.3

Your task is to investigate how the ϵ and γ parameters affect the learned policy. A higher value for ϵ will increase the chances of taking a random action which makes the algorithm more likely to reach the 10-reward.

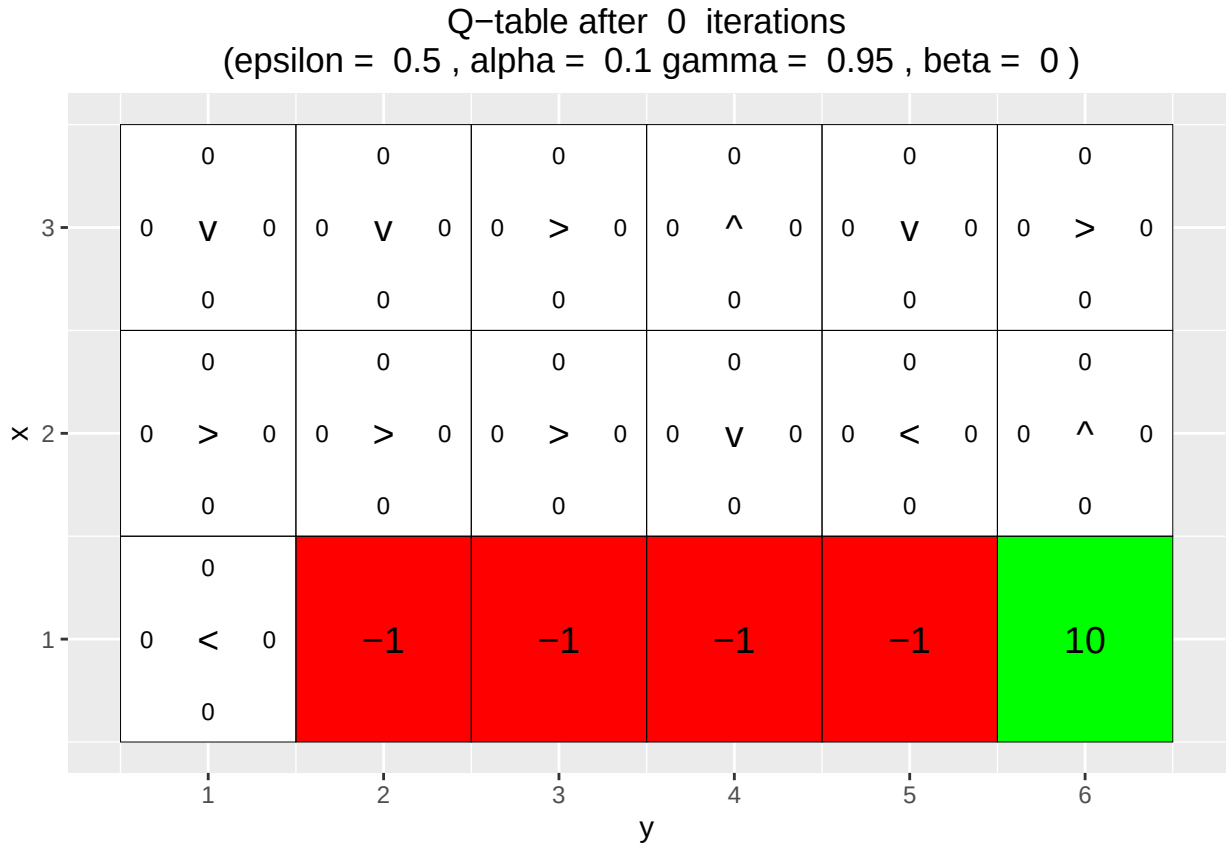
A higher value for γ will make the algorithm inclined to care for total expected reward no matter the length of the path.

Together a high value for both ϵ and γ is the only combination which leads to creating a policy which ends at reward 10.

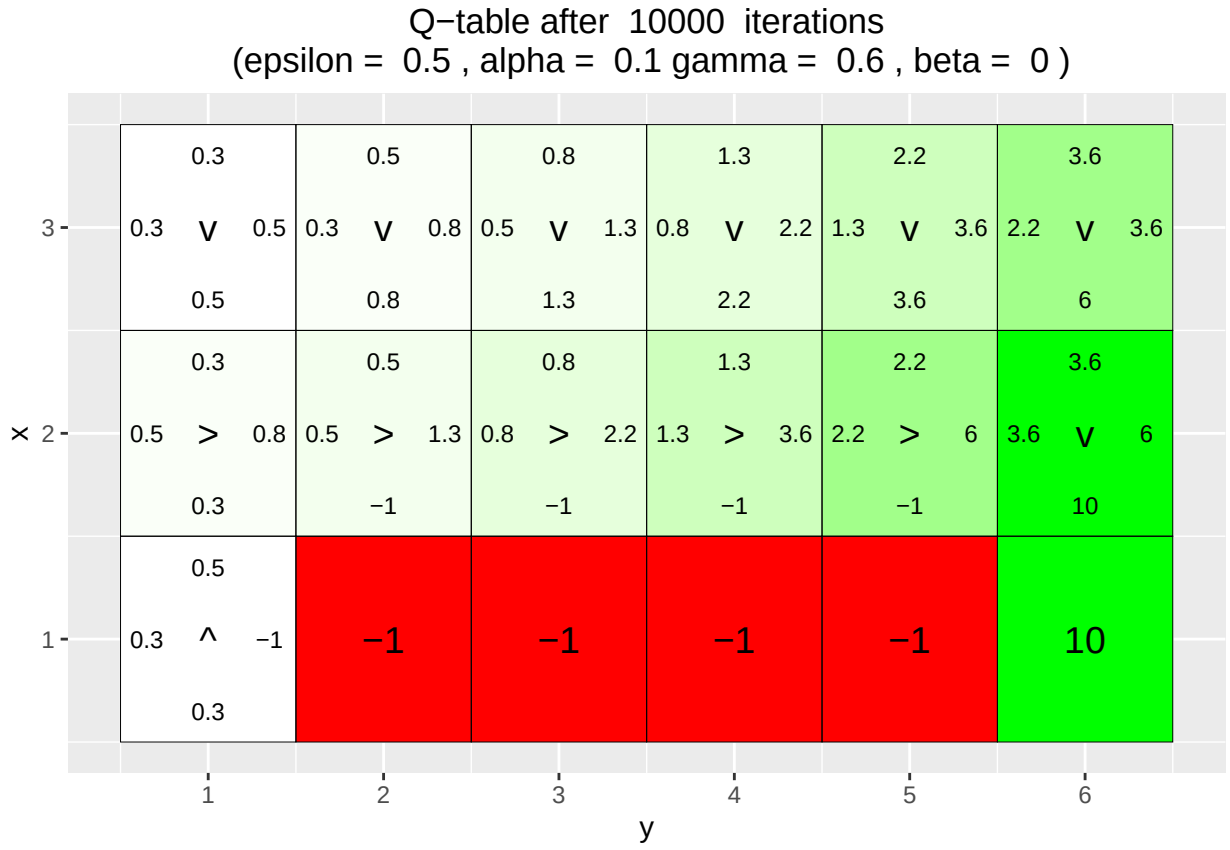
In the moving average plot of the reward for $\epsilon = 0.5$, $\gamma = 0.95$ and we see that there is a significant increase from around 4.5 to 7 when the algorithm finds the 10-reward.

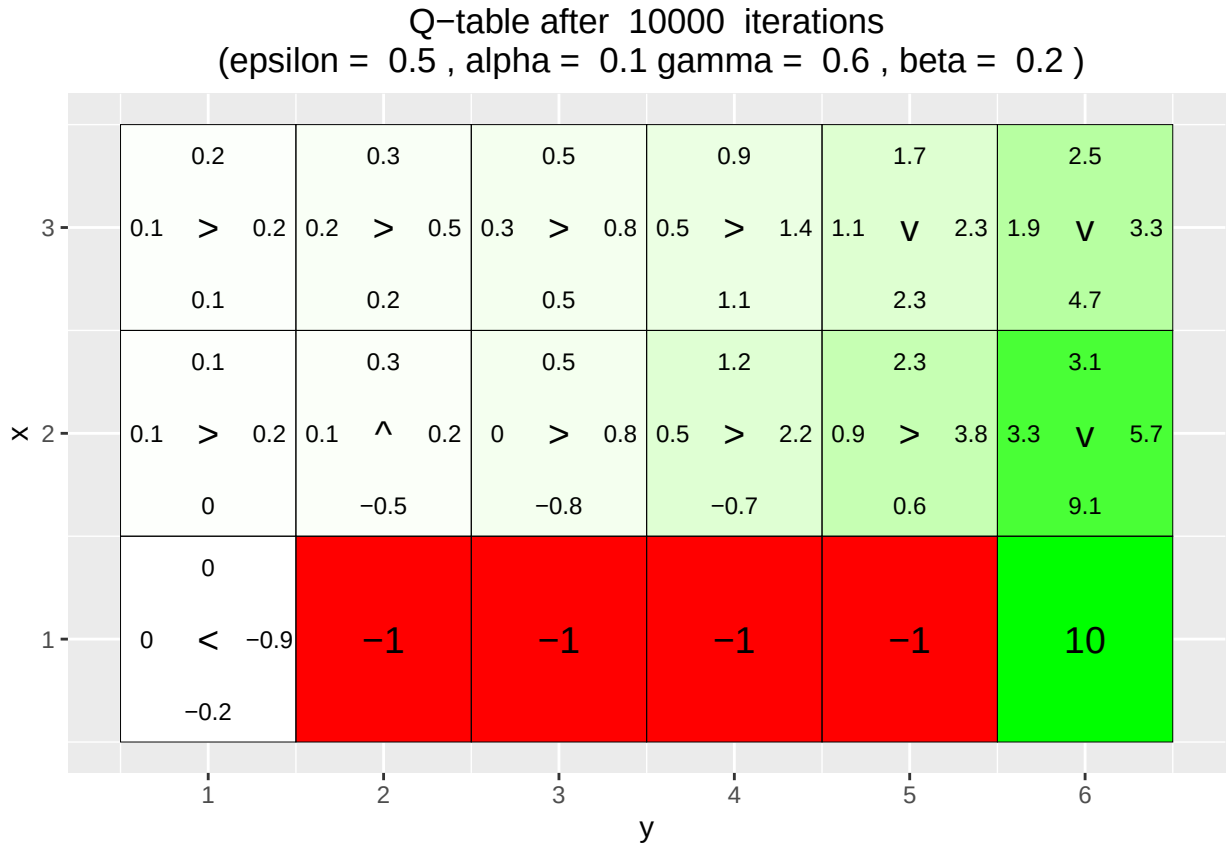
Assignment 2.4

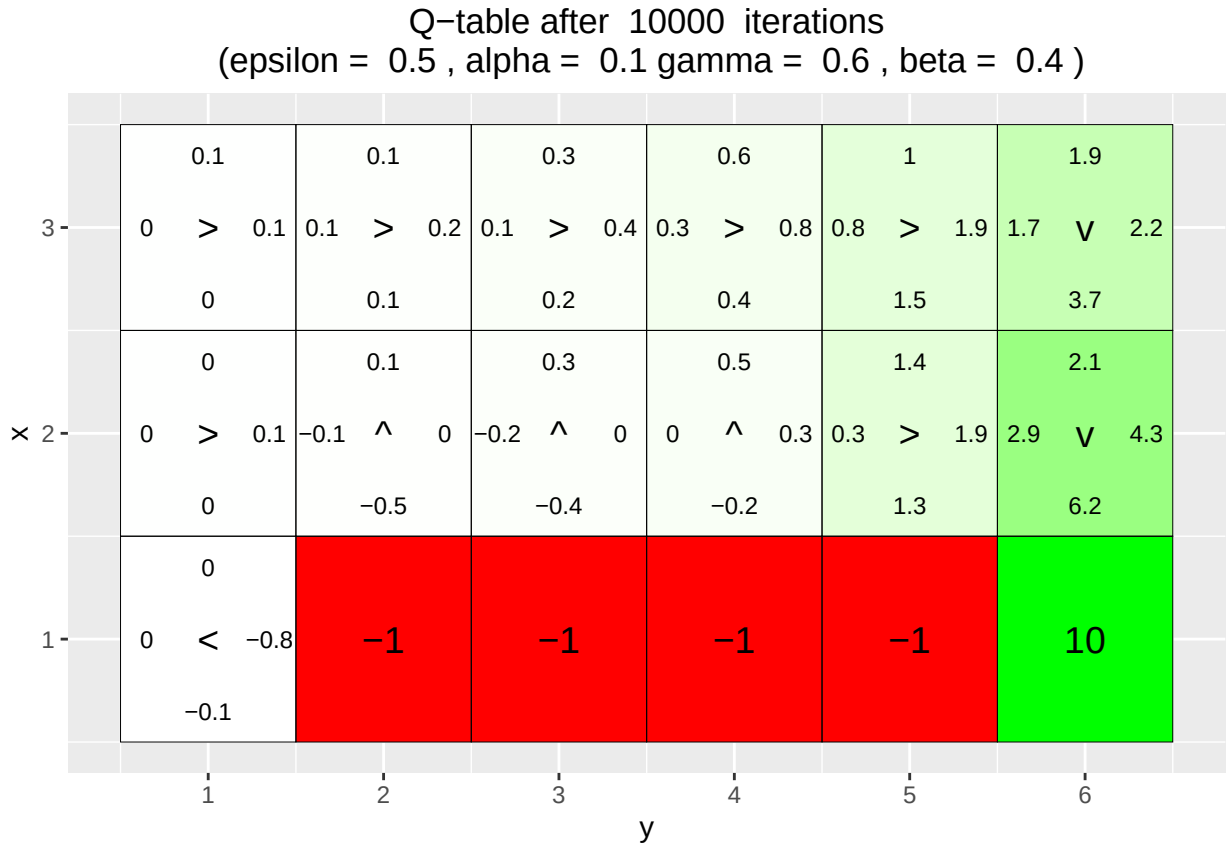
```
H <- 3
W <- 6

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()
```

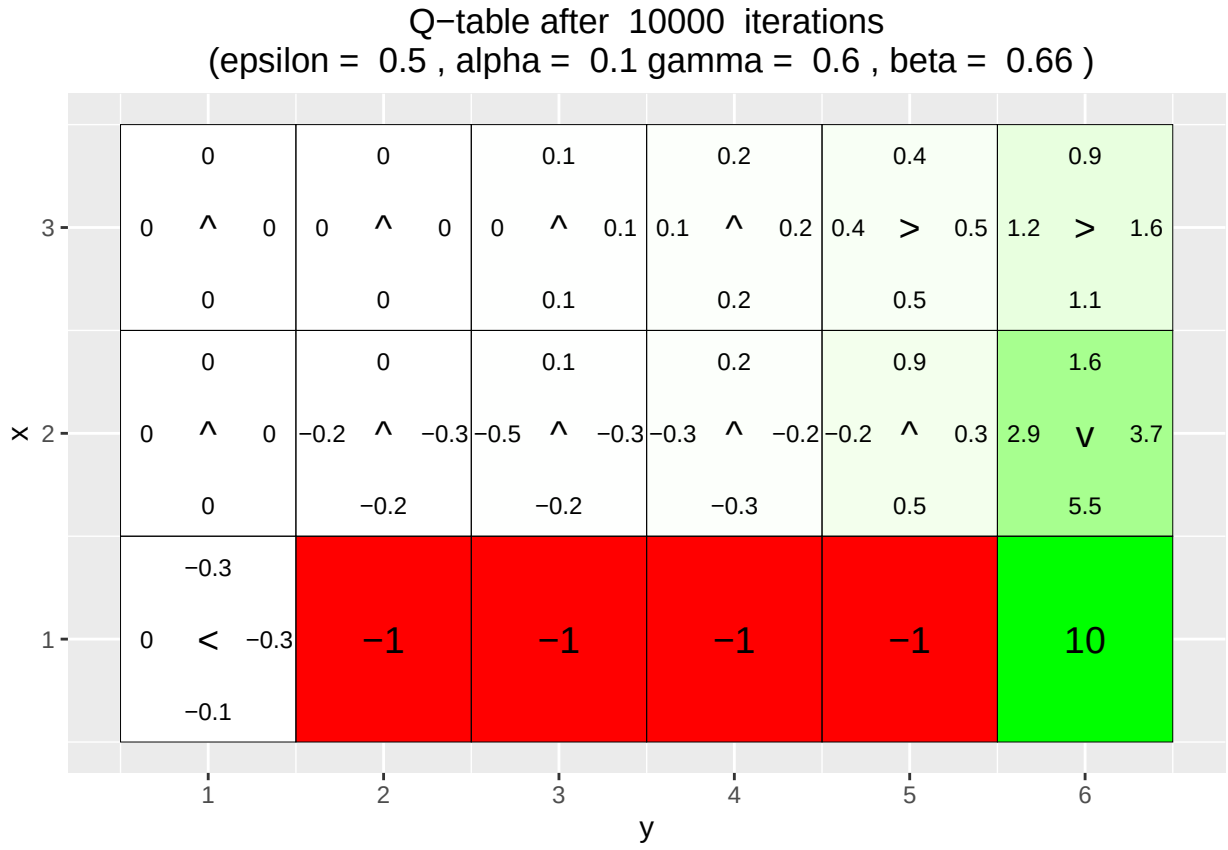



```
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

  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)
0.3
0.3 v 0.5
0.5
0.5
0.3 v 0.8
0.8
0.8
0.5 v 1.3
1.3
1.3
0.8 v 2.2
2.2
2.2
1.3 v 3.6
3.6
3.6
2.2 v 3.6
6
0.3
0.5 > 0.8
0.3
0.5
0.5 > 1.3
−1
0.8
0.8 > 2.2
−1
1.3
1.3 > 3.6
−1
2.2
2.2 > 6
−1
3.6
3.6 v 6
10
0.5
0.3 ^ −1
0.3
−1
−1
−1
−1
10
x
3
2
1
1
2
3
4
5
6
y


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)
x
y
3
2
1
1
2
3
4
5
6
0.2
0.1 > 0.2
0.1
0.3
0.2 > 0.5
0.2
0.5
0.3 > 0.8
0.5
0.9
0.5 > 1.4
1.1
1.7
1.1 v 2.3
2.3
2.5
1.9 v 3.3
4.7
0.1
0.1 > 0.2
0
0.3
0.1 ^ 0.2
−0.5
0.5
0 > 0.8
−0.8
1.2
0.5 > 2.2
−0.7
2.3
0.9 > 3.8
0.6
3.1
3.3 v 5.7
9.1
0
0 < −0.9
−0.2
−1
−1
−1
−1
10


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.4)
x
y
3
2
1
1
2
3
4
5
6
0.1
0 > 0.1
0
0.1
0.1 > 0.2
0.1
0.3
0.1 > 0.4
0.2
0.6
0.3 > 0.8
0.4
1
0.8 > 1.9
1.5
1.9
1.7 v 2.2
3.7
0
0 > 0.1
0
0.1
−0.1 ^ 0
−0.5
0.3
−0.2 ^ 0
−0.4
0.5
0 ^ 0.3
−0.2
1.4
0.3 > 1.9
1.3
2.1
2.9 v 4.3
6.2
0
0 < −0.8
−0.1
−1
−1
−1
−1
10



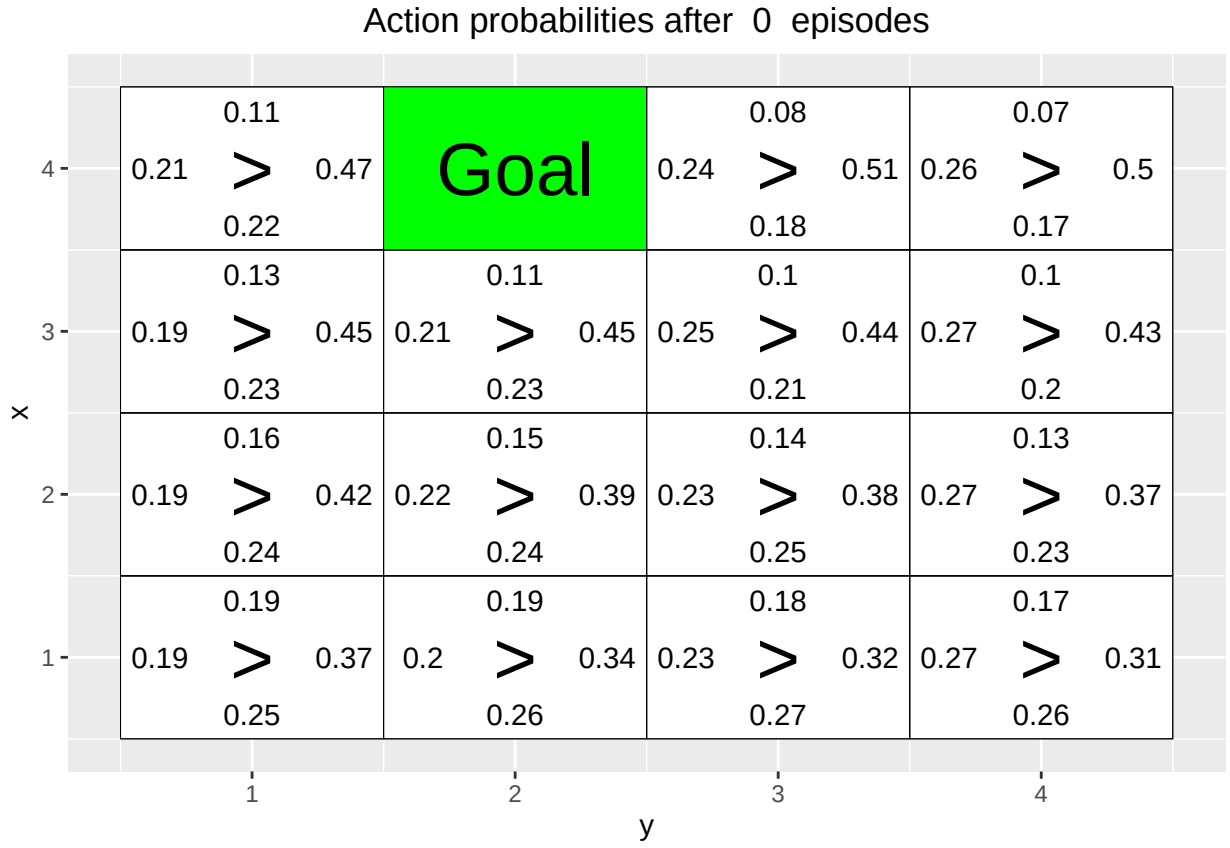
Question 2.4

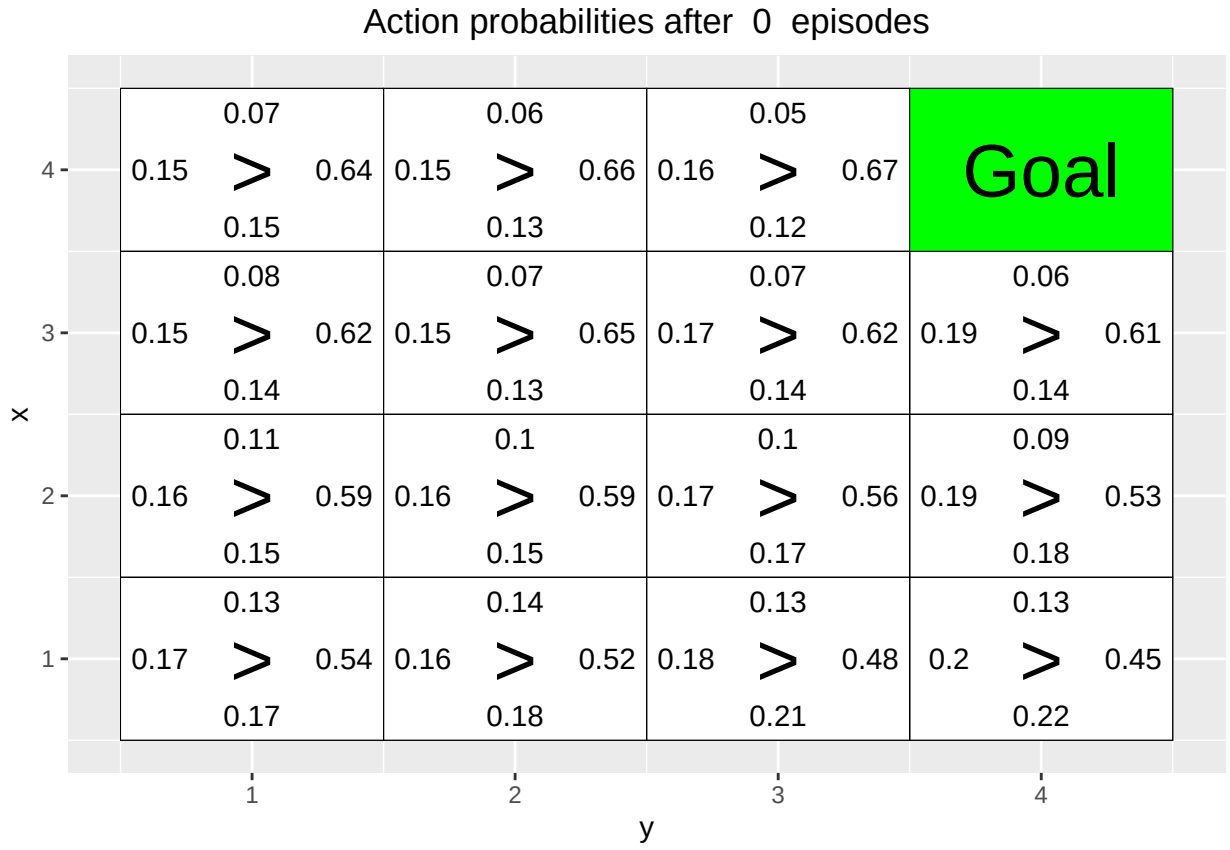
Your task is to investigate how the β parameter affects the learned policy. Since β makes the agent able to "slip" and go left or right from the actually preferred state, a higher β value makes the policy inclined to move away from the negative reward states, so the policy doesn't slip into a negative reward.

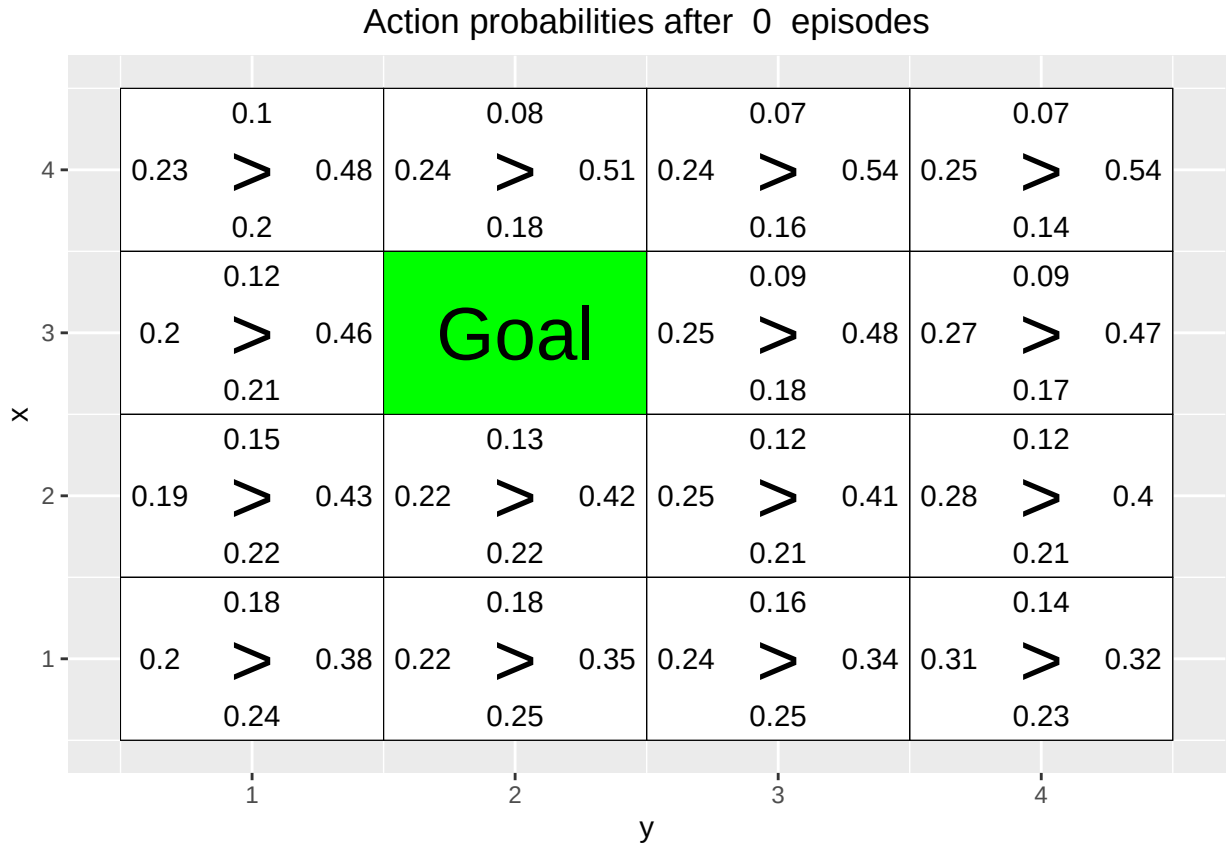
Assignment 2.5

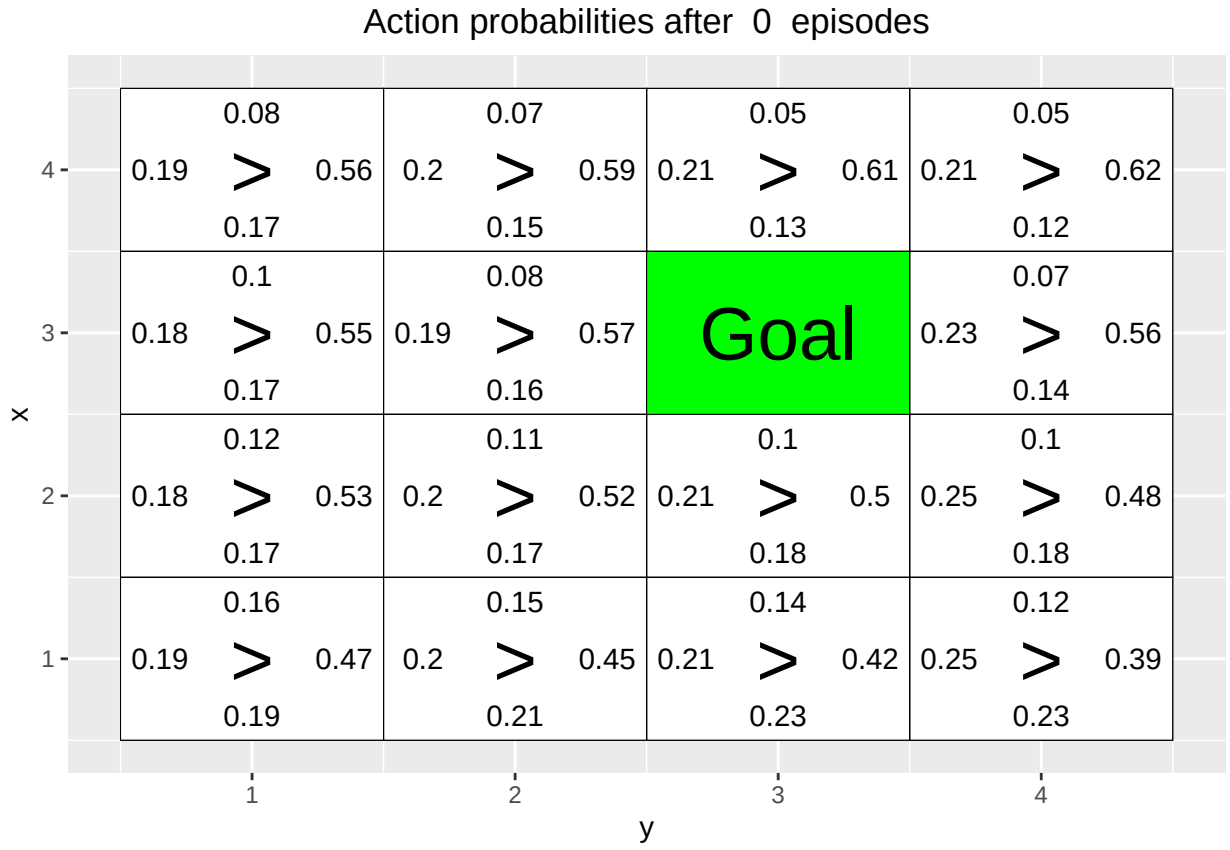
```
# Environment D (training with random goal positions)

H <- 4
W <- 4

# Define the neural network (two hidden layers of 32 units each).
model <- keras_model_sequential()
model %>%
  layer_dense(units = 32, input_shape = c(4), activation = 'relu') %>%
  layer_dense(units = 32, activation = 'relu') %>%
  layer_dense(units = 4, activation = 'softmax')

compile(model, loss = "categorical_crossentropy", optimizer = optimizer_sgd(lr=0.001))

initial_weights <- get_weights(model)

train_goals <- list(c(4,1), c(4,3), c(3,1), c(3,4), c(2,1), c(2,2), c(1,2), c(1,3))
```

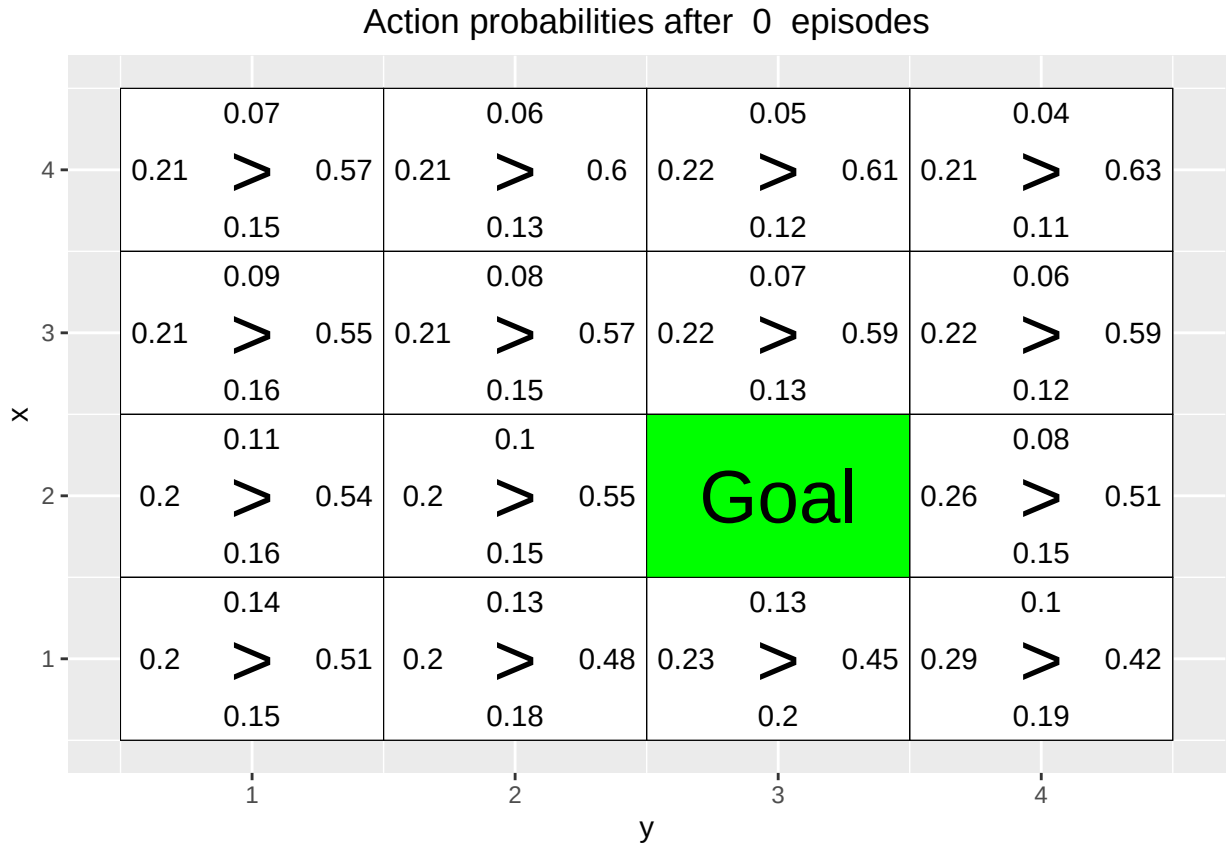
```
val_goals <- list(c(4,2), c(4,4), c(3,2), c(3,3), c(2,3), c(2,4), c(1,1), c(1,4))

show_validation <- function(episodes){

  for(goal in val_goals)
    vis_prob(goal, episodes)

}

set_weights(model,initial_weights)

show_validation(0)
```

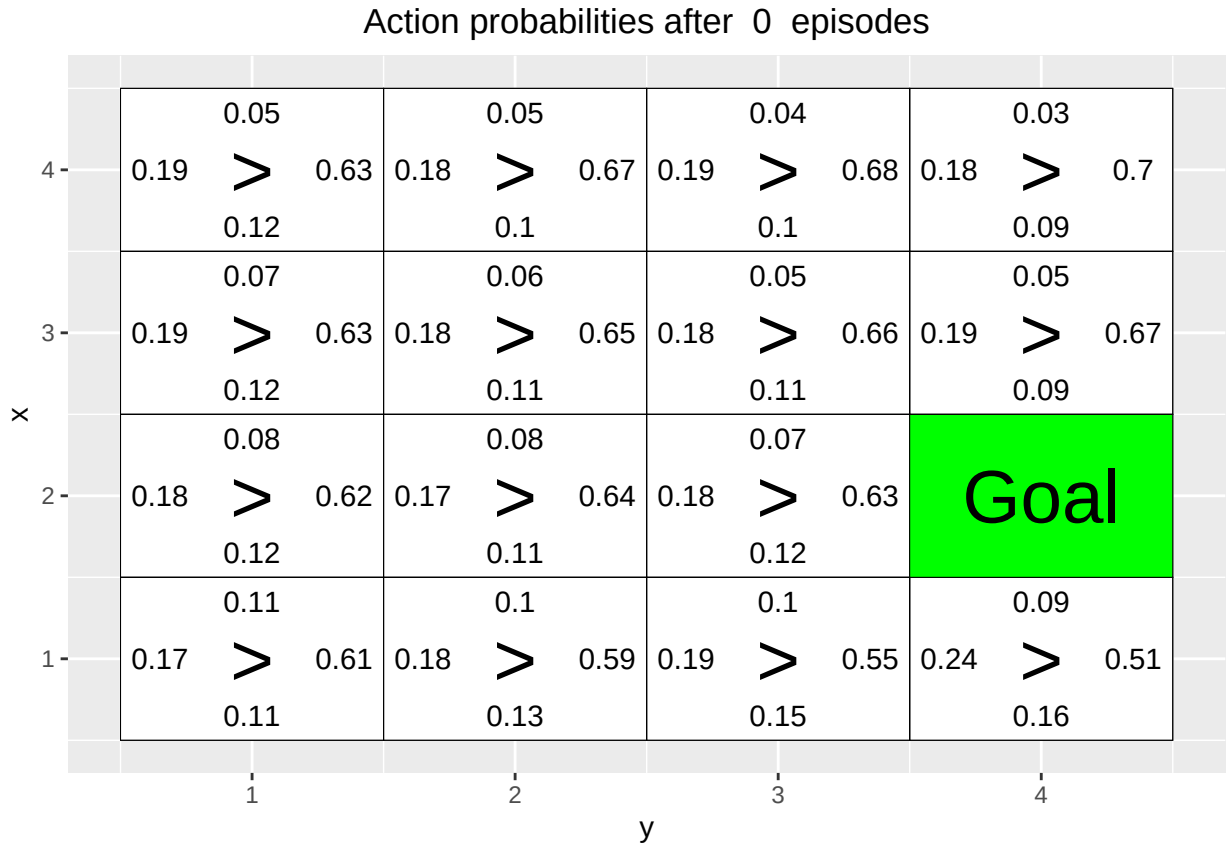


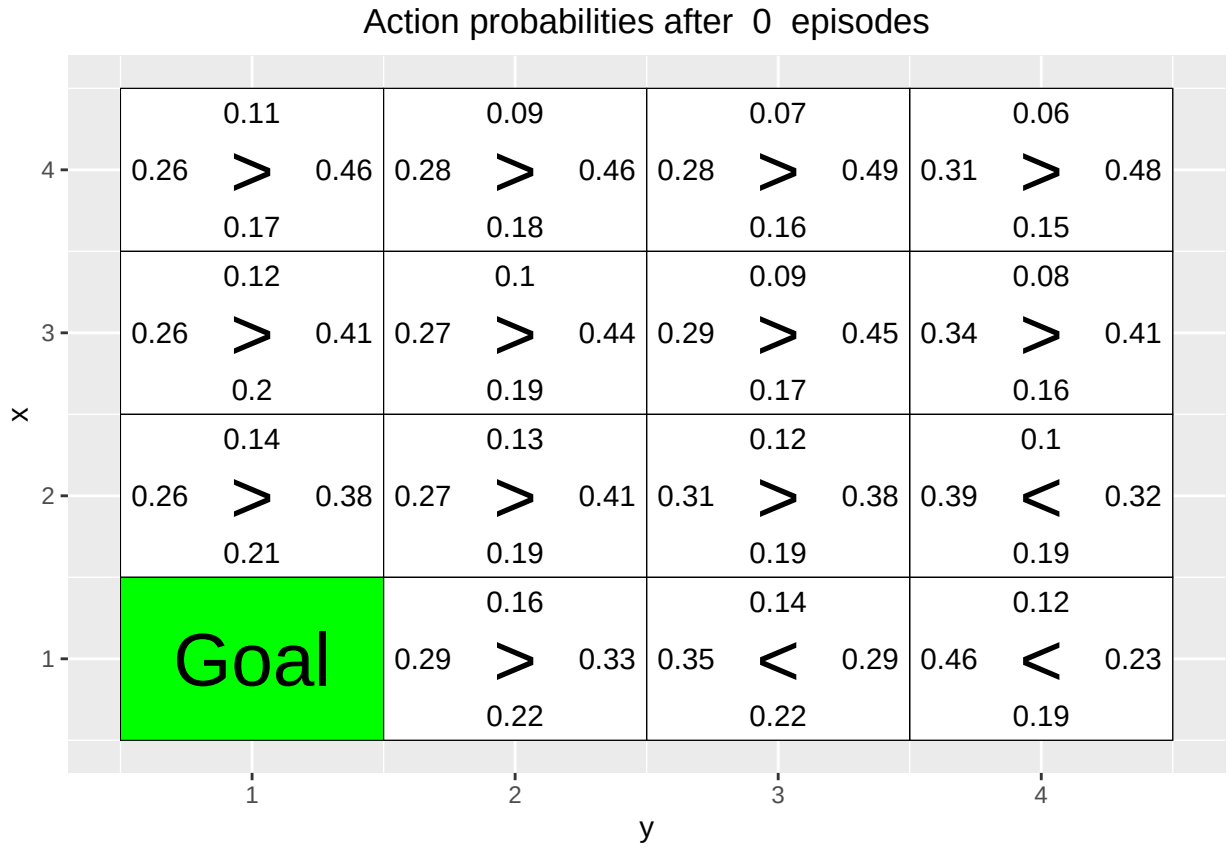

Action probabilities after 0 episodes
0.07
0.15 > 0.64
0.15
0.06
0.15 > 0.66
0.13
0.05
0.16 > 0.67
0.12
Goal
0.08
0.15 > 0.62
0.14
0.07
0.15 > 0.65
0.13
0.07
0.17 > 0.62
0.14
0.06
0.19 > 0.61
0.14
0.11
0.16 > 0.59
0.15
0.1
0.16 > 0.59
0.15
0.1
0.17 > 0.56
0.17
0.09
0.19 > 0.53
0.18
0.13
0.17 > 0.54
0.17
0.14
0.16 > 0.52
0.18
0.13
0.18 > 0.48
0.21
0.13
0.2 > 0.45
0.22
x
4
3
2
1
y
1
2
3
4

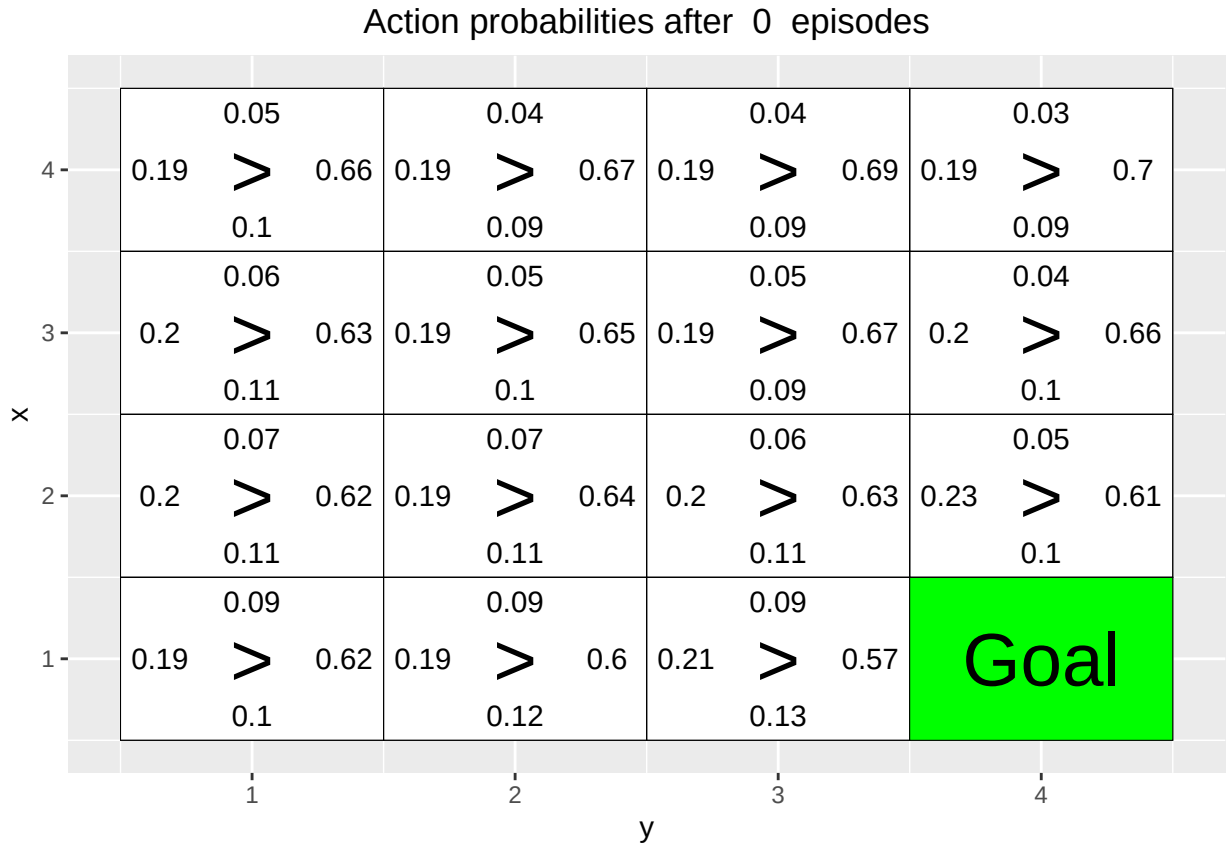

Action probabilities after 0 episodes
0.1
0.23 > 0.48
0.2
0.08
0.24 > 0.51
0.18
0.07
0.24 > 0.54
0.16
0.07
0.25 > 0.54
0.14
0.12
0.2 > 0.46
0.21
Goal
0.09
0.25 > 0.48
0.18
0.09
0.27 > 0.47
0.17
0.15
0.19 > 0.43
0.22
0.13
0.22 > 0.42
0.22
0.12
0.25 > 0.41
0.21
0.12
0.28 > 0.4
0.21
0.18
0.2 > 0.38
0.24
0.18
0.22 > 0.35
0.25
0.16
0.24 > 0.34
0.25
0.14
0.31 > 0.32
0.23
x
4
3
2
1
y
1
2
3
4


Action probabilities after 0 episodes
0.08
0.19 > 0.56
0.17
0.07
0.2 > 0.59
0.15
0.05
0.21 > 0.61
0.13
0.05
0.21 > 0.62
0.12
0.1
0.18 > 0.55
0.17
0.08
0.19 > 0.57
0.16
Goal
0.07
0.23 > 0.56
0.14
0.12
0.18 > 0.53
0.17
0.11
0.2 > 0.52
0.17
0.1
0.21 > 0.5
0.18
0.1
0.25 > 0.48
0.18
0.16
0.19 > 0.47
0.19
0.15
0.2 > 0.45
0.21
0.14
0.21 > 0.42
0.23
0.12
0.25 > 0.39
0.23
x
4
3
2
1
y
1
2
3
4


Action probabilities after 0 episodes
0.07
0.21 > 0.57
0.15
0.06
0.21 > 0.6
0.13
0.05
0.22 > 0.61
0.12
0.04
0.21 > 0.63
0.11
0.09
0.21 > 0.55
0.16
0.08
0.21 > 0.57
0.15
0.07
0.22 > 0.59
0.13
0.06
0.22 > 0.59
0.12
0.11
0.2 > 0.54
0.16
0.1
0.2 > 0.55
0.15
Goal
0.08
0.26 > 0.51
0.15
0.14
0.2 > 0.51
0.15
0.13
0.2 > 0.48
0.18
0.13
0.23 > 0.45
0.2
0.1
0.29 > 0.42
0.19
x
4
3
2
1
y
1
2
3
4

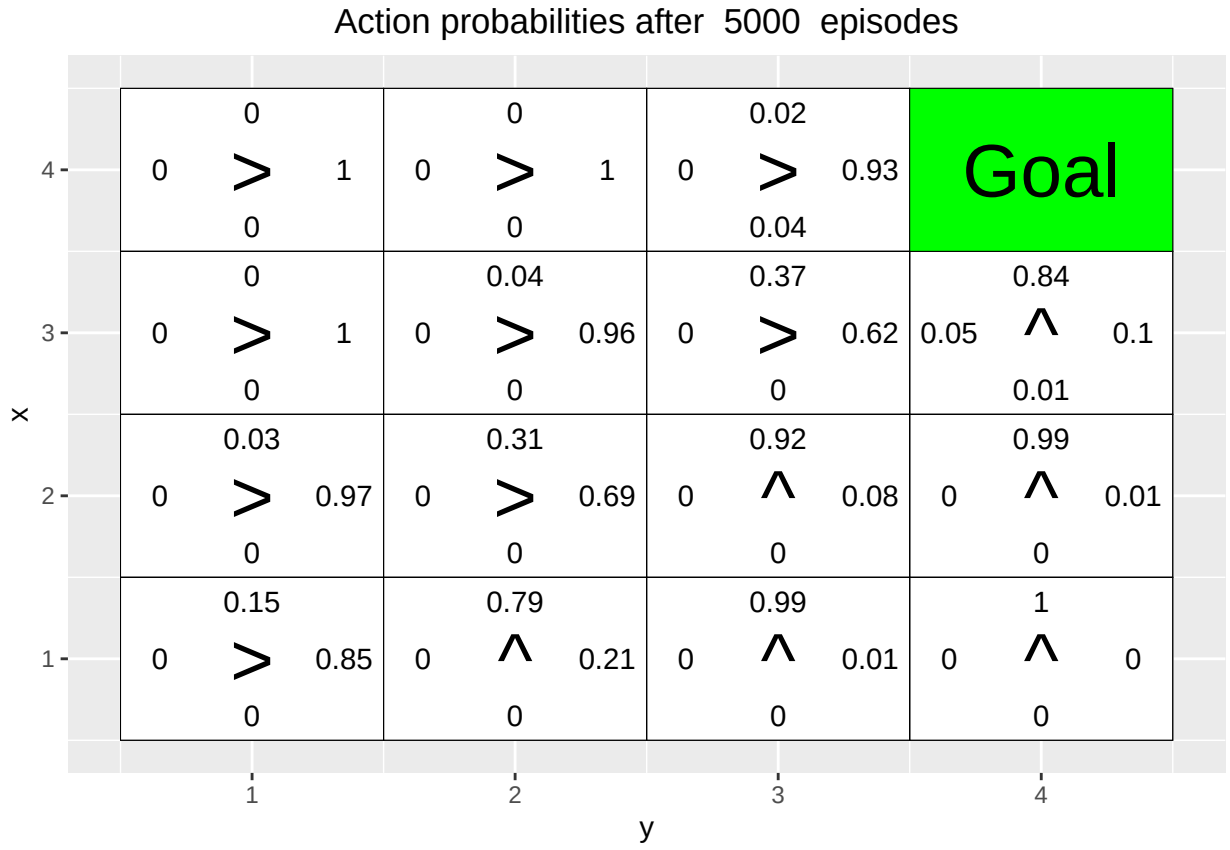

Action probabilities after 0 episodes
0.05
0.19 > 0.63
0.12
0.05
0.18 > 0.67
0.1
0.04
0.19 > 0.68
0.1
0.03
0.18 > 0.7
0.09
0.07
0.19 > 0.63
0.12
0.06
0.18 > 0.65
0.11
0.05
0.18 > 0.66
0.11
0.05
0.19 > 0.67
0.09
0.08
0.18 > 0.62
0.12
0.08
0.17 > 0.64
0.11
0.07
0.18 > 0.63
0.12
Goal
0.11
0.17 > 0.61
0.11
0.1
0.18 > 0.59
0.13
0.1
0.19 > 0.55
0.15
0.09
0.24 > 0.51
0.16
x
4
3
2
1
1
2
3
4
y

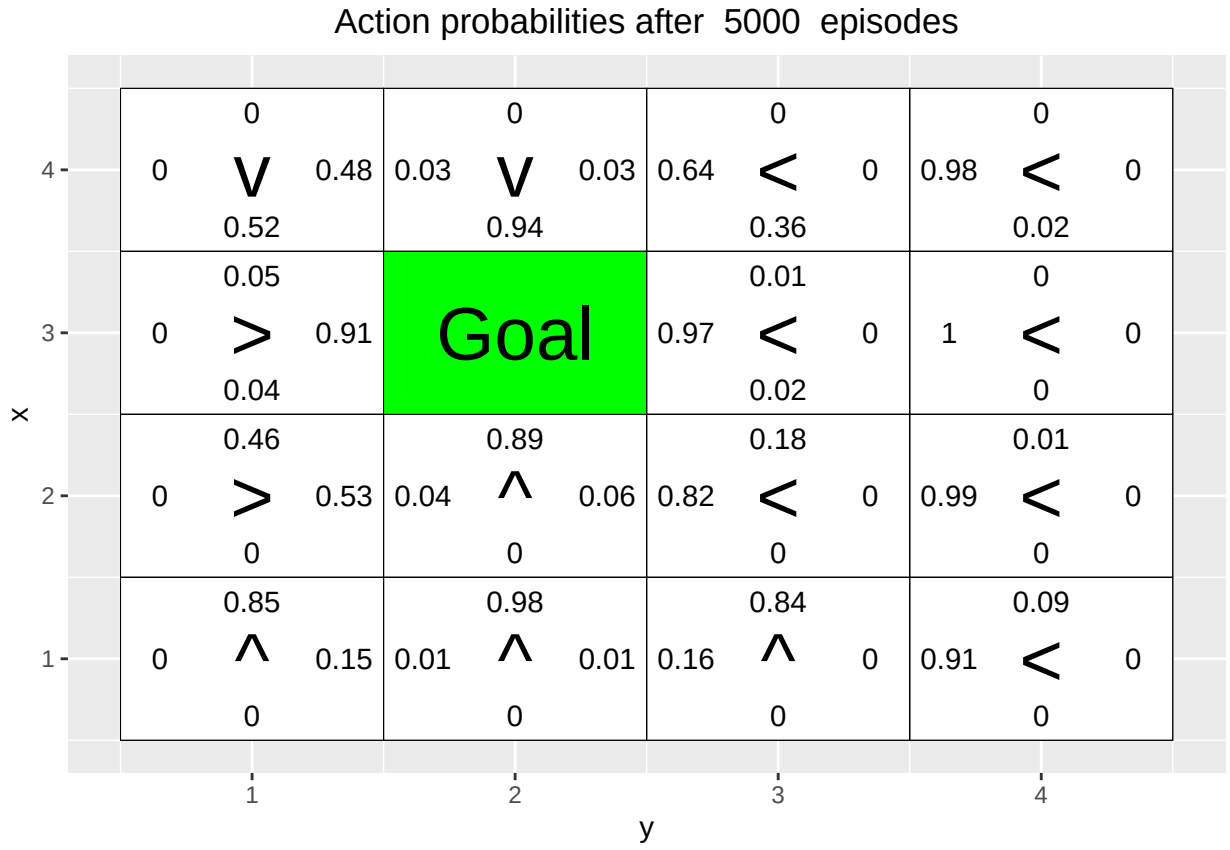

Action probabilities after 0 episodes
0.11
0.26 > 0.46
0.17
0.09
0.28 > 0.46
0.18
0.07
0.28 > 0.49
0.16
0.06
0.31 > 0.48
0.15
0.12
0.26 > 0.41
0.2
0.1
0.27 > 0.44
0.19
0.09
0.29 > 0.45
0.17
0.08
0.34 > 0.41
0.16
0.14
0.26 > 0.38
0.21
0.13
0.27 > 0.41
0.19
0.12
0.31 > 0.38
0.19
0.1
0.39 < 0.32
0.19
Goal
0.16
0.29 > 0.33
0.22
0.14
0.35 < 0.29
0.22
0.12
0.46 < 0.23
0.19
x
4
3
2
1
y
1
2
3
4

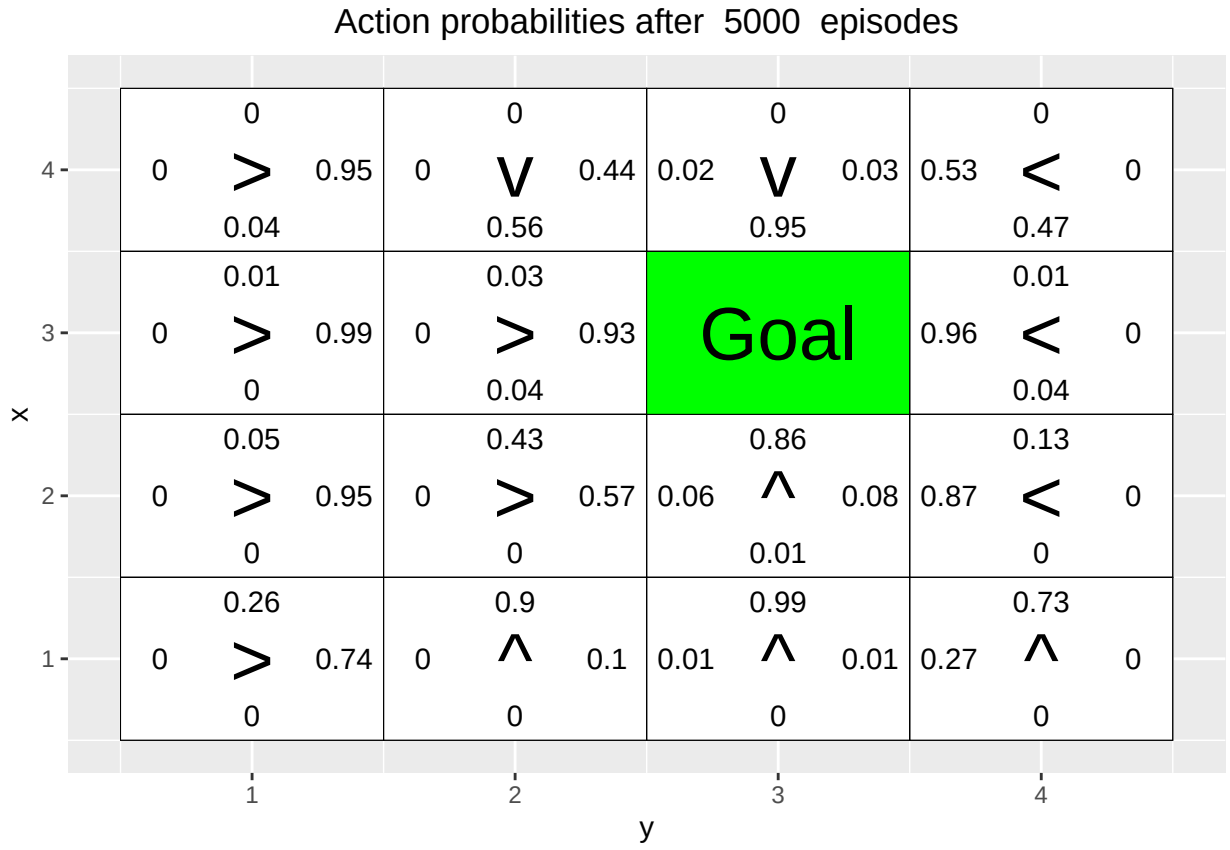


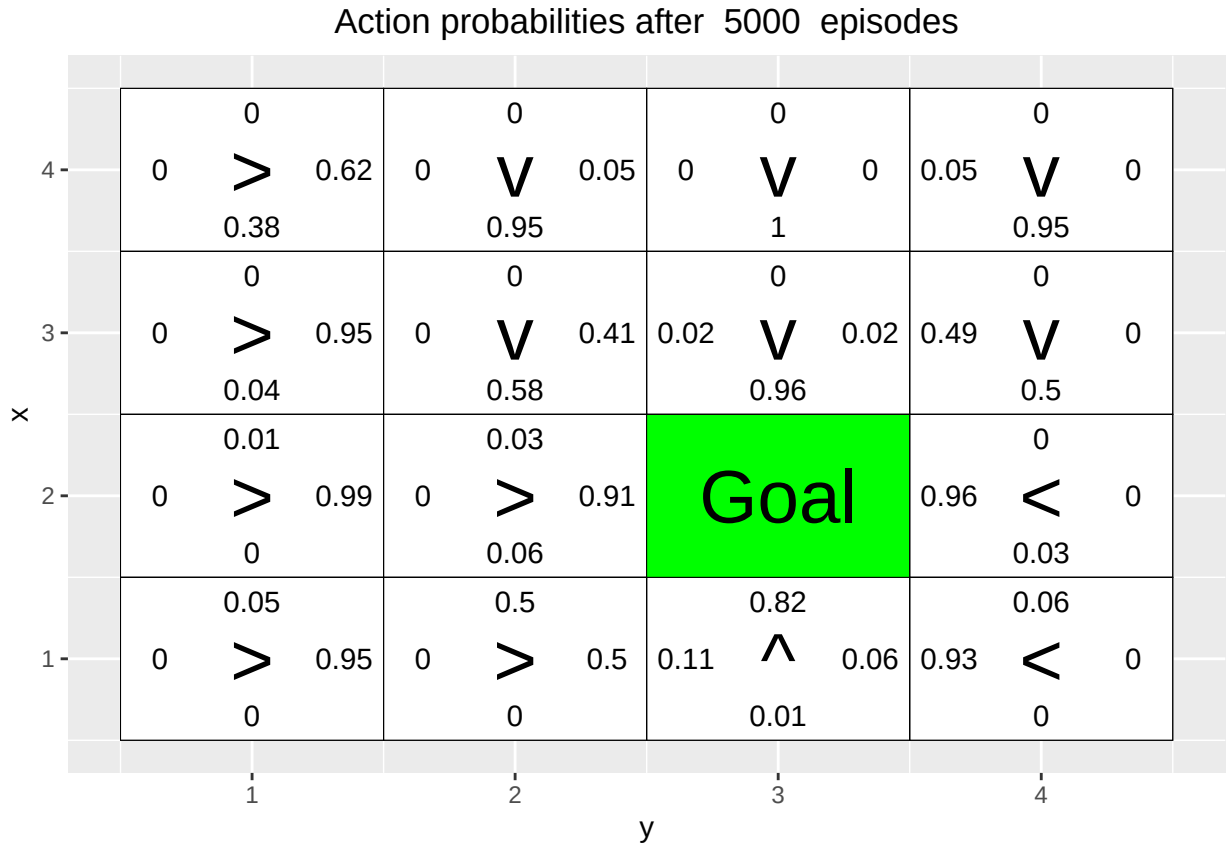

```
for(i in 1:5000){
  goal <- sample(train_goals, size = 1)
  reinforce_episode(unlist(goal))
}

show_validation(5000)
```

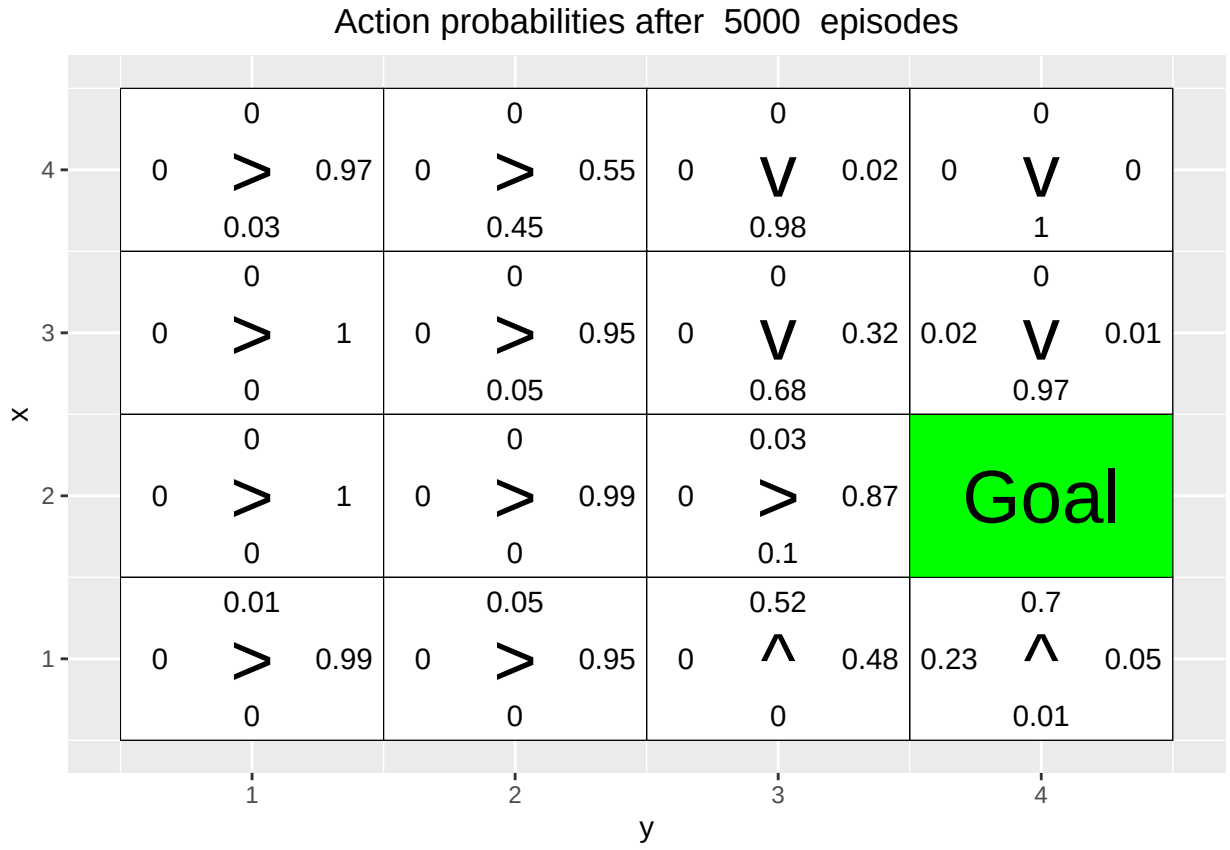

Action probabilities after 5000 episodes
0.05
0
>
0.9
0.05
Goal
0.01
0.97
<
0
0.02
0
1
<
0
0
0.52
0
^
0.48
0
0.9
0.04
^
0.06
0
0.2
0.8
<
0
0
0
1
<
0
0
0.89
0
^
0.11
0
0.99
0.01
^
0.01
0
0.87
0.13
^
0
0
0.11
0.89
<
0
0
0.97
0
^
0.03
0
1
0
^
0
0
0.98
0.02
^
0
0
0.65
0.35
^
0
0
x
4
3
2
1
1
2
3
4
y

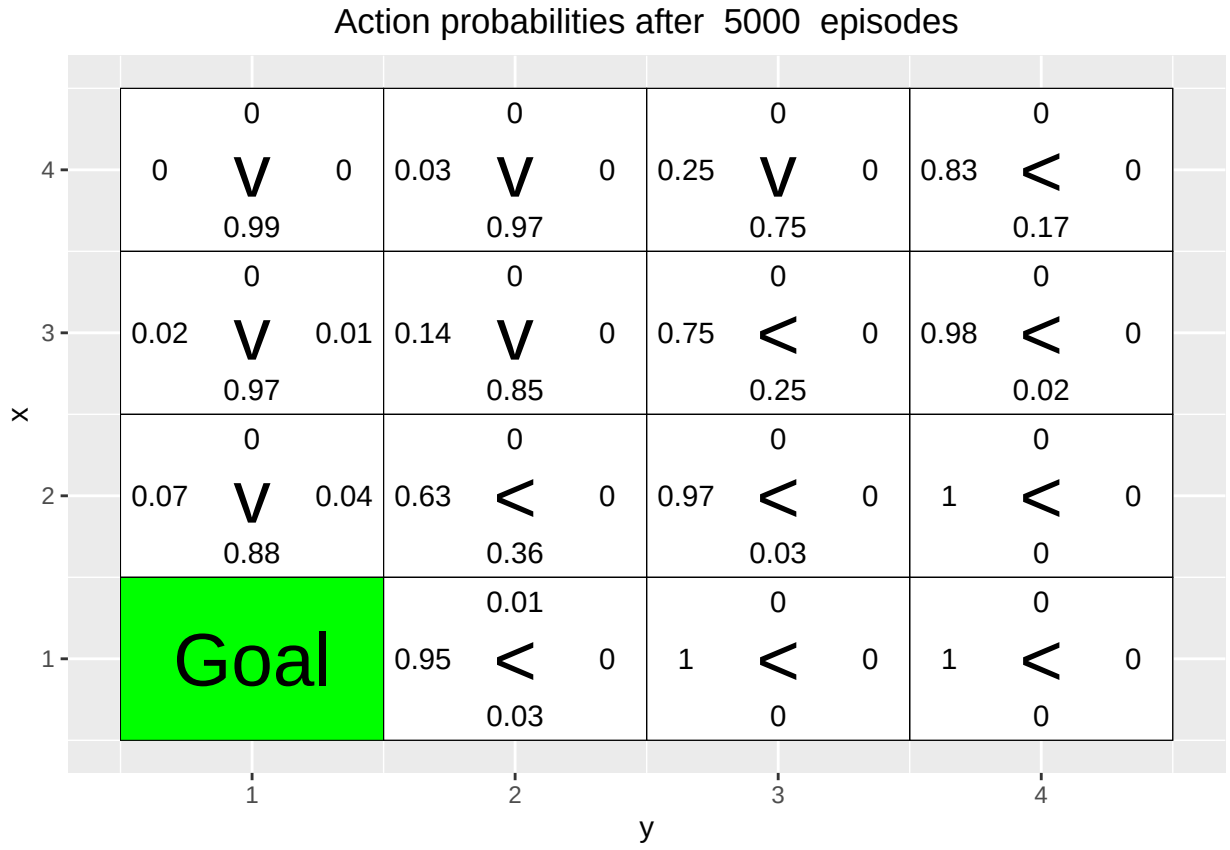

Action probabilities after 5000 episodes
0
0 > 1
0
0
0 > 1
0
0.02
0 > 0.93
0.04
Goal
0
0 > 1
0
0.04
0 > 0.96
0
0.37
0 > 0.62
0
0.84
0.05 ^ 0.1
0.01
0.03
0 > 0.97
0
0.31
0 > 0.69
0
0.92
0 ^ 0.08
0
0.99
0 ^ 0.01
0
0.15
0 > 0.85
0
0.79
0 ^ 0.21
0
0.99
0 ^ 0.01
0
1
0 ^ 0
0
x
4
3
2
1
y
1
2
3
4

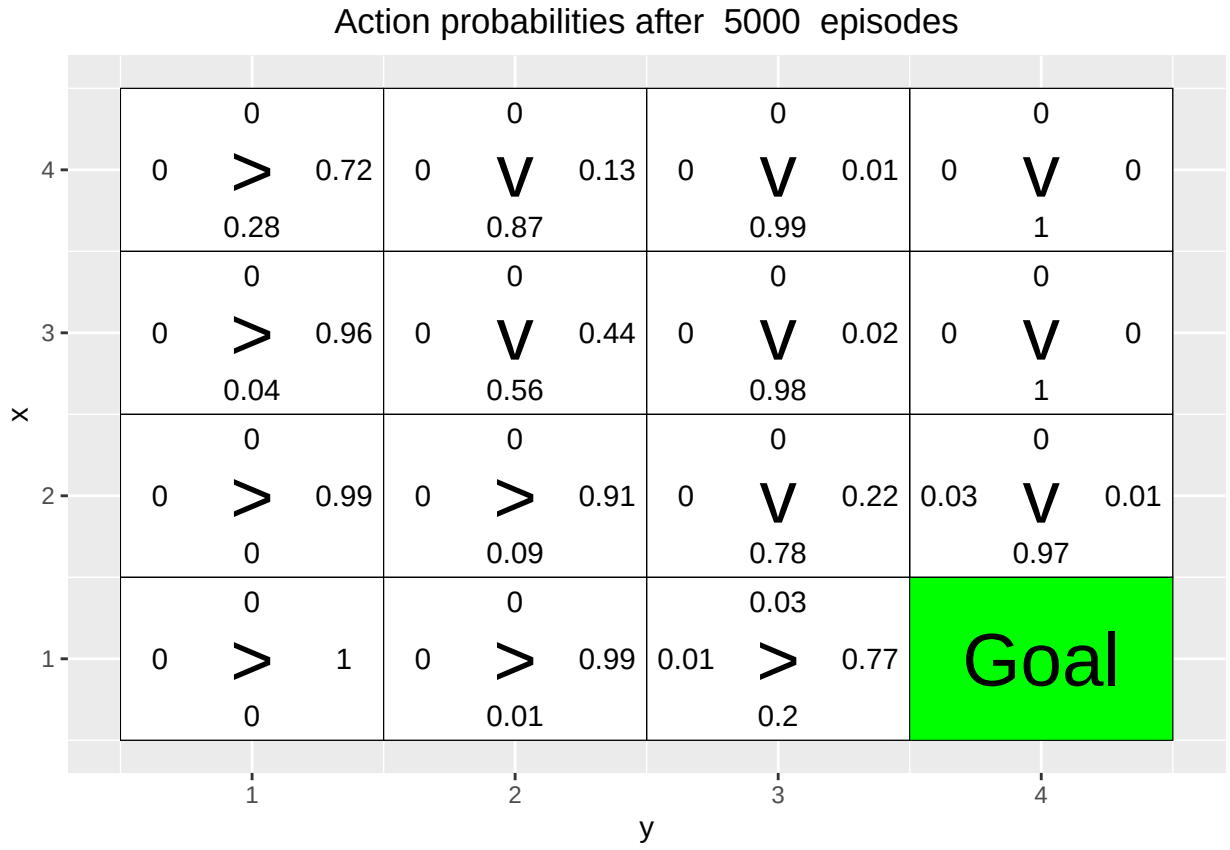

Action probabilities after 5000 episodes
0
0 V 0.48
0.52
0
0.03 V 0.03
0.94
0
0.64 < 0
0.36
0
0.98 < 0
0.02
0.05
0 > 0.91
0.04
Goal
0.01
0.97 < 0
0.02
0
1 < 0
0
0.46
0 > 0.53
0
0.89
0.04 ^ 0.06
0
0.18
0.82 < 0
0
0.01
0.99 < 0
0
0.85
0 ^ 0.15
0
0.98
0.01 ^ 0.01
0
0.84
0.16 ^ 0
0
0.09
0.91 < 0
0
x
4
3
2
1
y
1
2
3
4


Action probabilities after 5000 episodes
0
0 > 0.95
0.04
0
0 V 0.44
0.56
0
0.02 V 0.03
0.95
0
0.53 < 0
0.47
0.01
0 > 0.99
0
0.03
0 > 0.93
0.04
Goal
0.01
0.96 < 0
0.04
0.05
0 > 0.95
0
0.43
0 > 0.57
0
0.86
0.06 ^ 0.08
0.01
0.13
0.87 < 0
0
0.26
0 > 0.74
0
0.9
0 ^ 0.1
0
0.99
0.01 ^ 0.01
0
0.73
0.27 ^ 0
0
x
4
3
2
1
1
2
3
4
y


Action probabilities after 5000 episodes
0
0 > 0.62
0.38
0
0 V 0.05
0.95
0
0 V 0
1
0
0.05 V 0
0.95
0
0 > 0.95
0.04
0
0 V 0.41
0.58
0
0.02 V 0.02
0.96
0
0.49 V 0
0.5
0.01
0 > 0.99
0
0.03
0 > 0.91
0.06
Goal
0
0.96 < 0
0.03
0.05
0 > 0.95
0
0.5
0 > 0.5
0
0.82
0.11 ^ 0.06
0.01
0.06
0.93 < 0
0
x
4
3
2
1
1
2
3
4
y


Action probabilities after 5000 episodes
4
3
2
1
x
1
2
3
4
y
0 0 0 0
0 > 0.97 0 > 0.55 0 V 0.02 0 V 0
0.03 0.45 0.98 1
0 0 0 0
0 > 1 0 > 0.95 0 V 0.32 0.02 V 0.01
0 0.05 0.68 0.97
0 0 0.03 0
0 > 1 0 > 0.99 0 > 0.87 Goal
0 0 0.1
0.01 0.05 0.52 0.7
0 > 0.99 0 > 0.95 0 ^ 0.48 0.23 ^ 0.05
0 0 0 0.01


Action probabilities after 5000 episodes
4
3
2
1
x
1
2
3
4
y
0
0 V 0
0.99
0
0.03 V 0
0.97
0
0.25 V 0
0.75
0
0.83 < 0
0.17
0
0.02 V 0.01
0.97
0
0.14 V 0
0.85
0
0.75 < 0
0.25
0
0.98 < 0
0.02
0
0.07 V 0.04
0.88
0
0.63 < 0
0.36
0
0.97 < 0
0.03
0
1 < 0
0
Goal
0.01
0.95 < 0
0.03
0
1 < 0
0
0
1 < 0
0

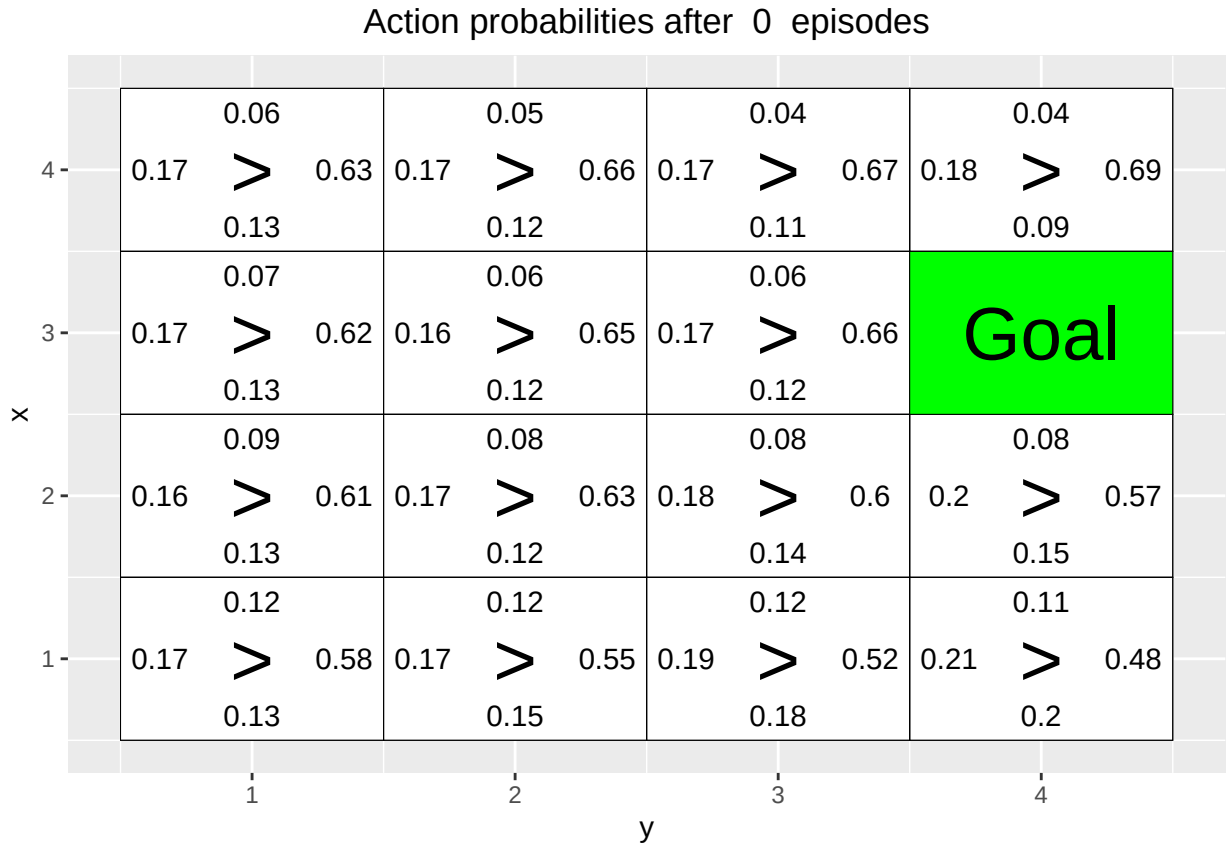


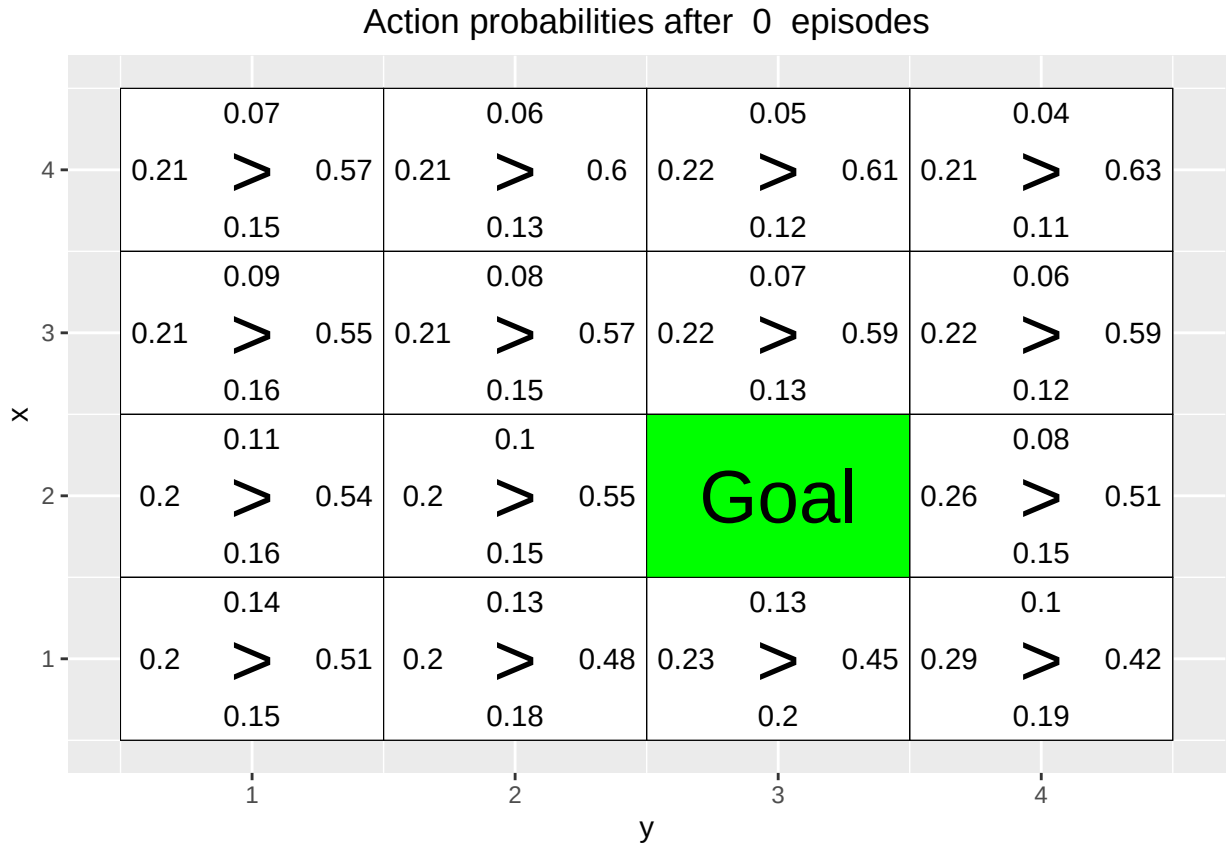
Has the agent learned a good policy? Why / Why not?

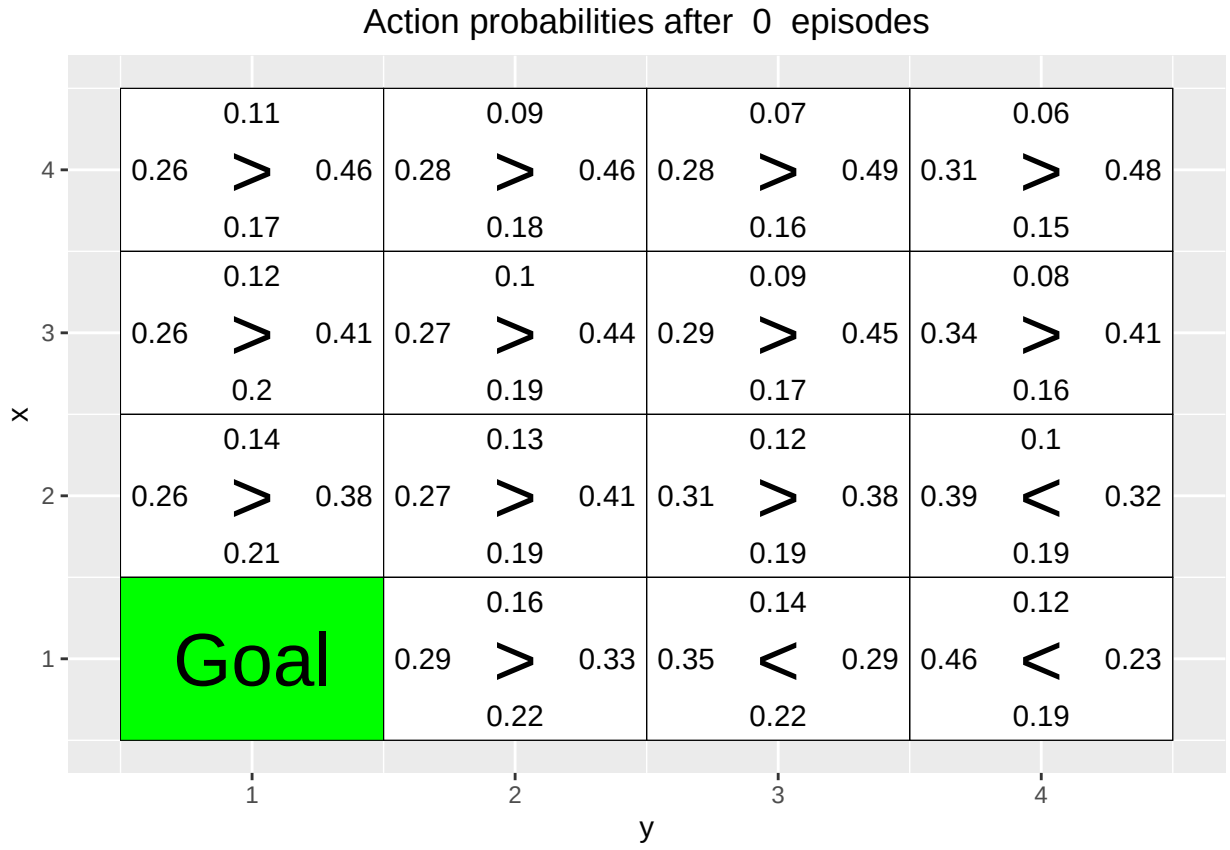
It has learned a good policy, since all the paths go directly to goal. Although worth mentioning some states have >30% prob of not going to goal even though they are next to goal.

Could you have used the Q-learning algorithm to solve this task? It would be possible to model the problem into the Q-learning algorithm if we could add goal coordinates in the q-table. Although it would not be possible to train towards one goal and then validate towards another because the goal might be in unseen territories.

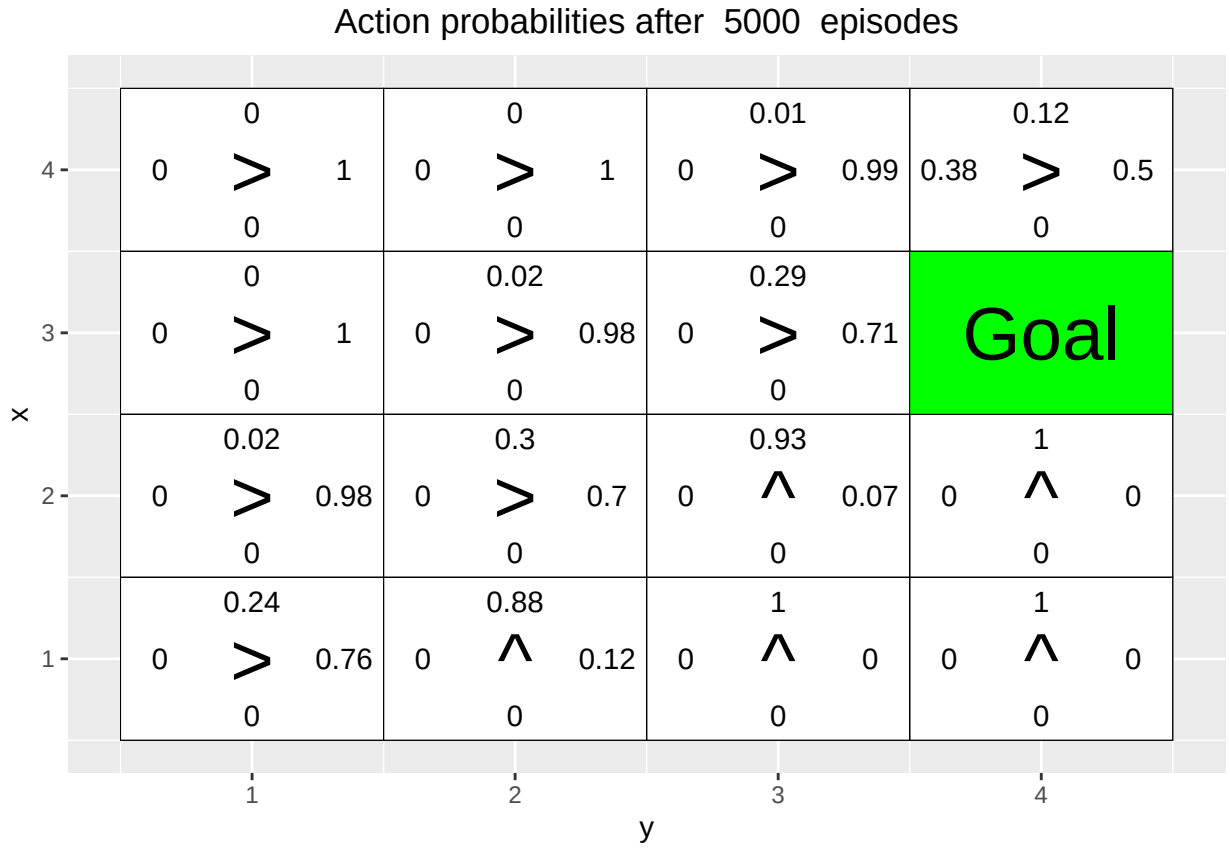
```
train_goals <- list(c(4,1), c(4,2), c(4,3), c(4,4))
val_goals <- list(c(3,4), c(2,3), c(1,1))

set_weights(model,initial_weights)

show_validation(0)
```

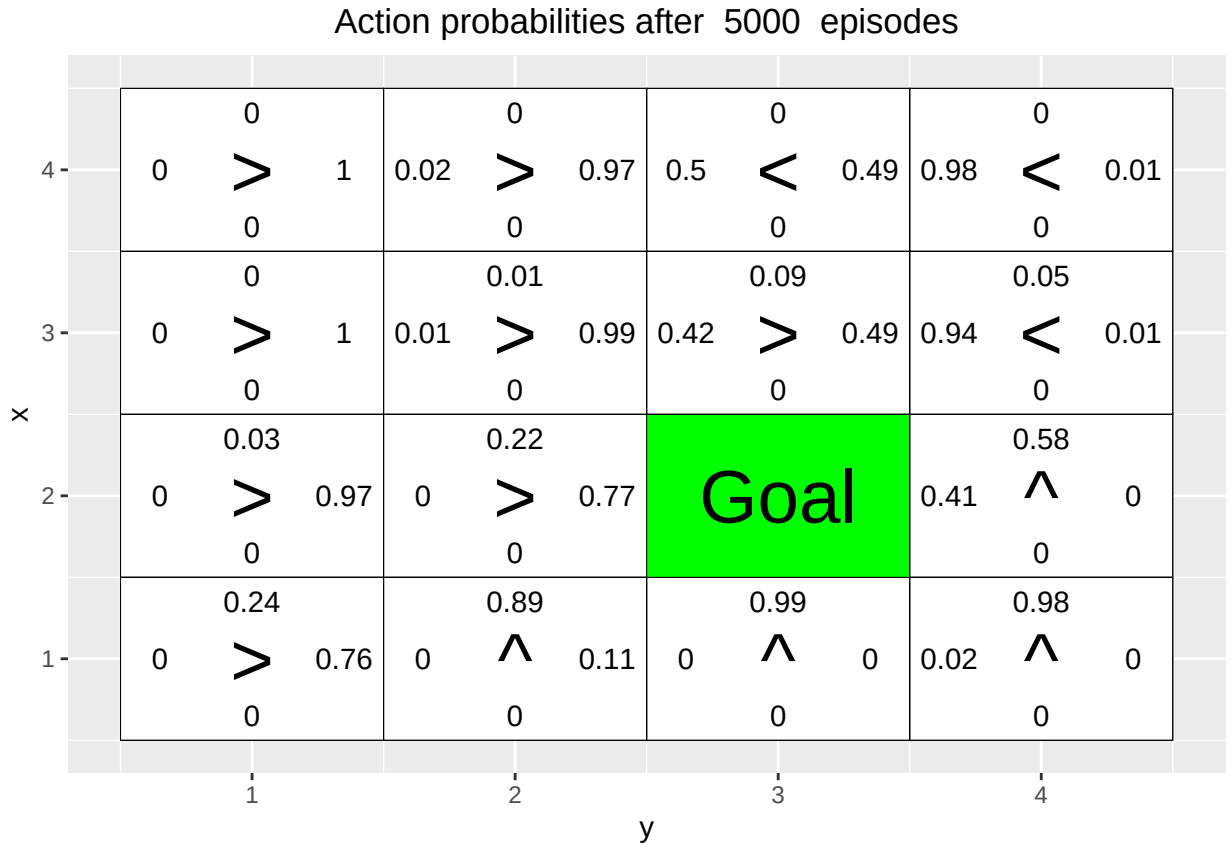

Action probabilities after 0 episodes
0.06
0.17 > 0.63
0.13
0.05
0.17 > 0.66
0.12
0.04
0.17 > 0.67
0.11
0.04
0.18 > 0.69
0.09
0.07
0.17 > 0.62
0.13
0.06
0.16 > 0.65
0.12
0.06
0.17 > 0.66
0.12
Goal
0.09
0.16 > 0.61
0.13
0.08
0.17 > 0.63
0.12
0.08
0.18 > 0.6
0.14
0.08
0.2 > 0.57
0.15
0.12
0.17 > 0.58
0.13
0.12
0.17 > 0.55
0.15
0.12
0.19 > 0.52
0.18
0.11
0.21 > 0.48
0.2
x
4
3
2
1
y
1
2
3
4

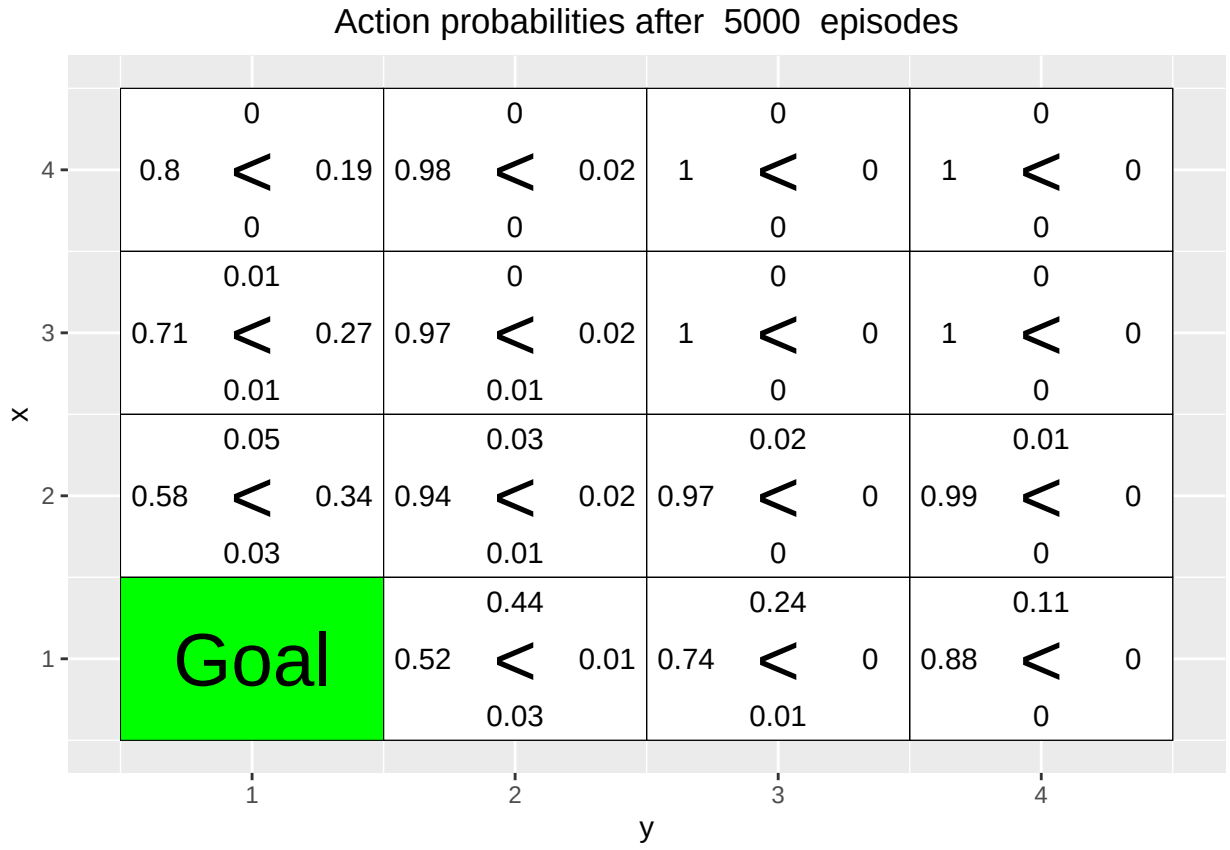

Action probabilities after 0 episodes
0.07
0.21 > 0.57
0.15
0.06
0.21 > 0.6
0.13
0.05
0.22 > 0.61
0.12
0.04
0.21 > 0.63
0.11
0.09
0.21 > 0.55
0.16
0.08
0.21 > 0.57
0.15
0.07
0.22 > 0.59
0.13
0.06
0.22 > 0.59
0.12
0.11
0.2 > 0.54
0.16
0.1
0.2 > 0.55
0.15
Goal
0.08
0.26 > 0.51
0.15
0.14
0.2 > 0.51
0.15
0.13
0.2 > 0.48
0.18
0.13
0.23 > 0.45
0.2
0.1
0.29 > 0.42
0.19
x
4
3
2
1
y
1
2
3
4




```
for(i in 1:5000){
  goal <- sample(train_goals, size = 1)
  reinforce_episode(unlist(goal))
}

show_validation(5000)
```


Action probabilities after 5000 episodes
0
0 > 1
0
0
0 > 1
0
0.01
0 > 0.99
0
0.12
0.38 > 0.5
0
0
0 > 1
0
0.02
0 > 0.98
0
0.29
0 > 0.71
0
Goal
0.02
0 > 0.98
0
0.3
0 > 0.7
0
0.93
0 ^ 0.07
0
1
0 ^ 0
0
0.24
0 > 0.76
0
0.88
0 ^ 0.12
0
1
0 ^ 0
0
1
0 ^ 0
0
x
4
3
2
1
y
1
2
3
4


Action probabilities after 5000 episodes
0
0 > 1
0
0
0.02 > 0.97
0
0
0.5 < 0.49
0
0
0.98 < 0.01
0
0
0 > 1
0
0
0.01 > 0.99
0
0.01
0.42 > 0.49
0
0.09
0.94 < 0.01
0
0.05
0
0.03
0 > 0.97
0
0
0.22
0 > 0.77
0
Goal
0.41 ^ 0
0
0.58
0
0.24
0 > 0.76
0
0
0.89
0 ^ 0.11
0
0.99
0 ^ 0
0
0.98
0.02 ^ 0
0
x
4
3
2
1
y
1
2
3
4



Has the agent learned a good policy? Why / Why not ? No it has not learned a good policy. Because it cannot make a policy with all paths leading to goal.

If the results obtained for environments D and E differ, explain why. The results differ. This is because when training on environment E there are no goals which require the agent to go "down". This means that it never learns this operation and this is why all paths can not lead to goal if the goal is not in the top row.