AML_Lab3

Yifan Ding, Chao Fu, Yunan Dong, Ravinder Reddy

10 October, 2021

Contributions

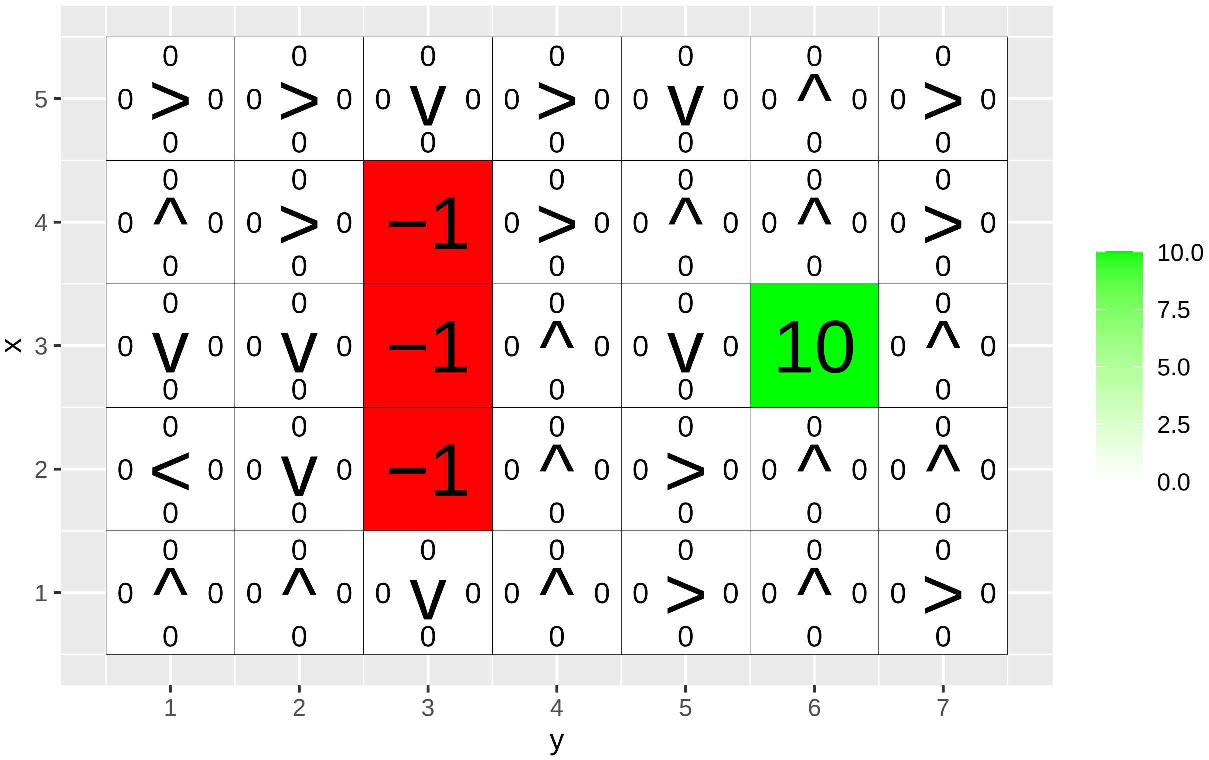
We solved all questions individually and discussed together, we wrote our comment together.

1. Q-Learning

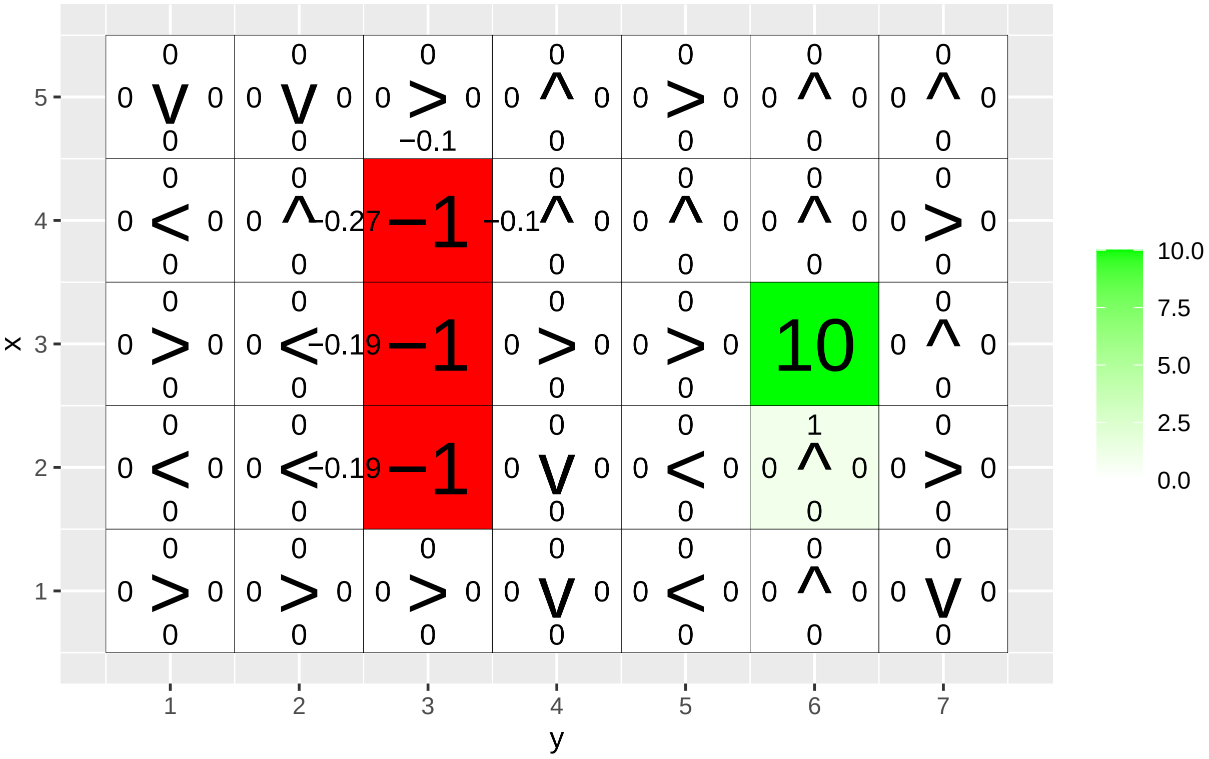
```r
# Implement Greedy algorithm
GreedyPolicy <- function(x, y){
  max_q <- max(q_table[x, y, ])
  index <- which(q_table[x, y, ] == max_q)
  a <- ifelse(length(index) > 1, sample(index, 1), index)
  return(a)
}

# Implement Epsilon Greedy algorithm
EpsilonGreedyPolicy <- function(x, y, epsilon){
  a <- ifelse(runif(1) >= epsilon, GreedyPolicy(x, y), sample(1 : 4, 1))
  return(a)
}

# Implement Q-Learning algorithm
q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95,
                       beta = 0){
  p <- start_state
  episode_correction <- 0
  repeat{
    # Follow policy, execute action, get reward.
    a_1 <- EpsilonGreedyPolicy(p[1], p[2], epsilon)
    p1 <- transition_model(p[1], p[2], a_1, beta)
    reward <- reward_map[p1[1], p1[2]]
    # Q-table update.
    temporal_diff <- reward + gamma * max(q_table[p1[1], p1[2], ]) - q_table[p[1], p[2], a_1]
    q_table[p[1], p[2], a_1] <<- q_table[p[1], p[2], a_1] + alpha * temporal_diff
    p <- p1
    episode_correction <- episode_correction + temporal_diff
    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))
  }

}
```

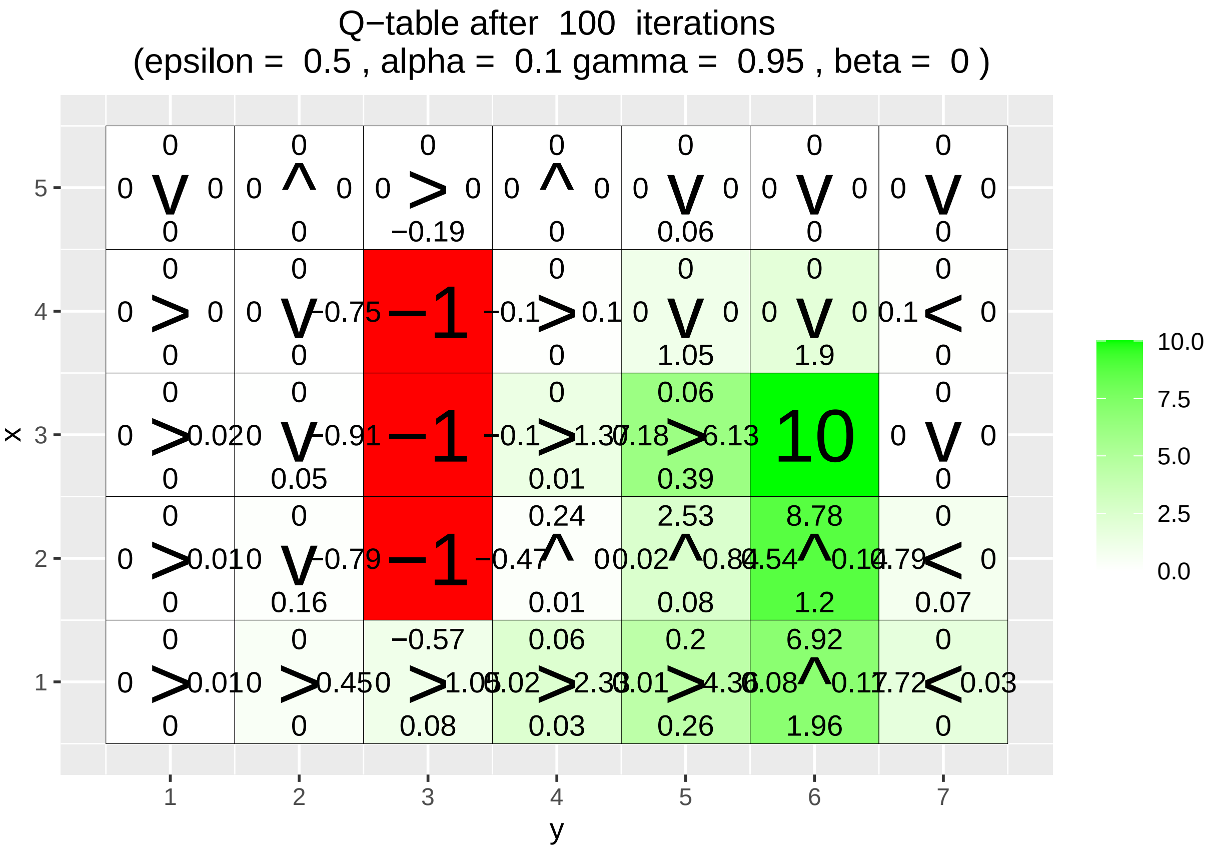
2. Environment A

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

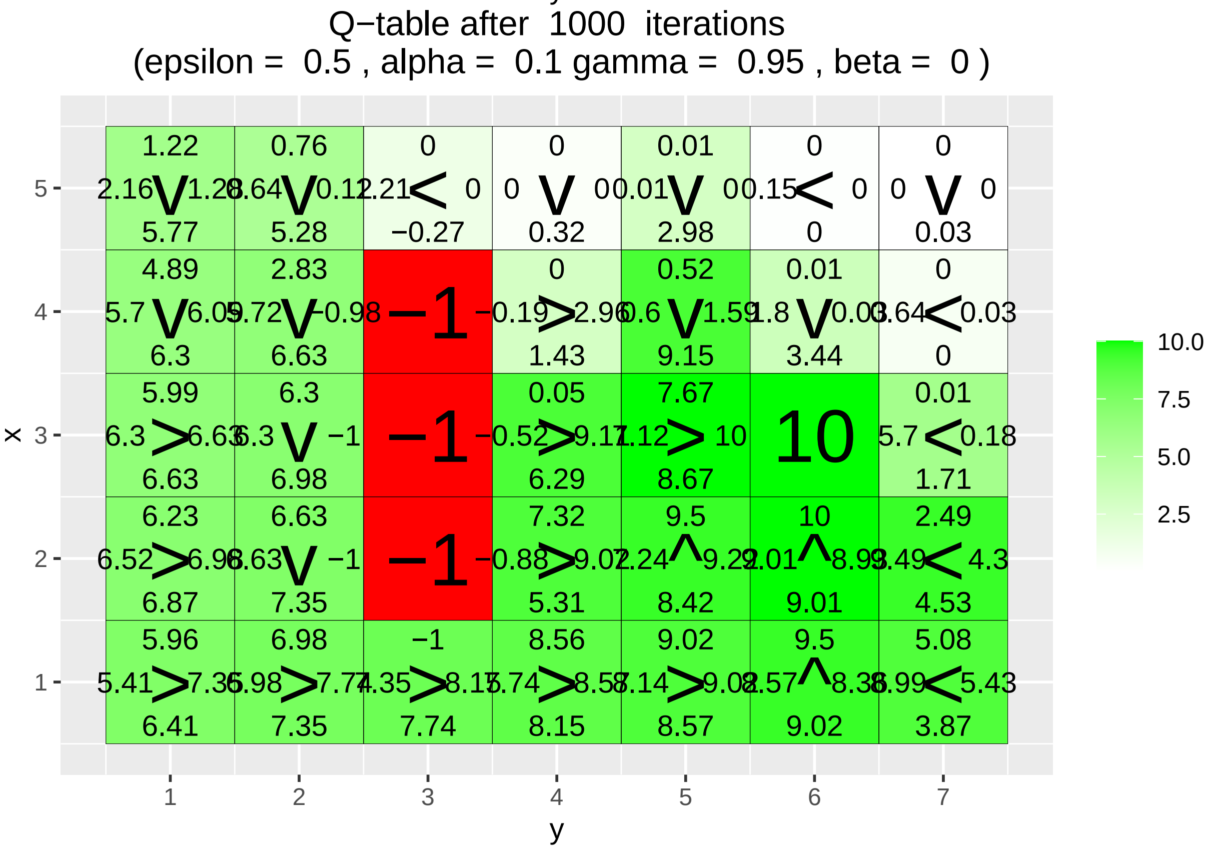


Q−table after 10 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)

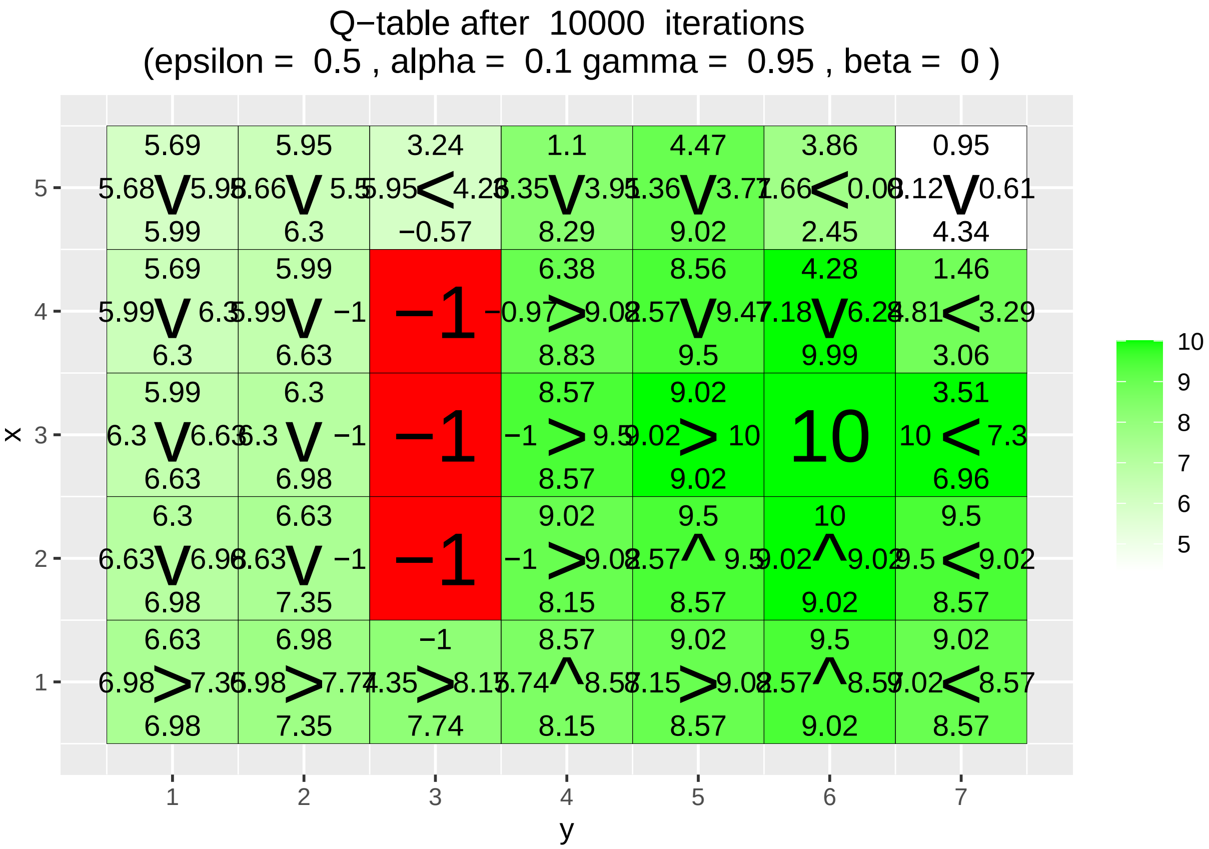


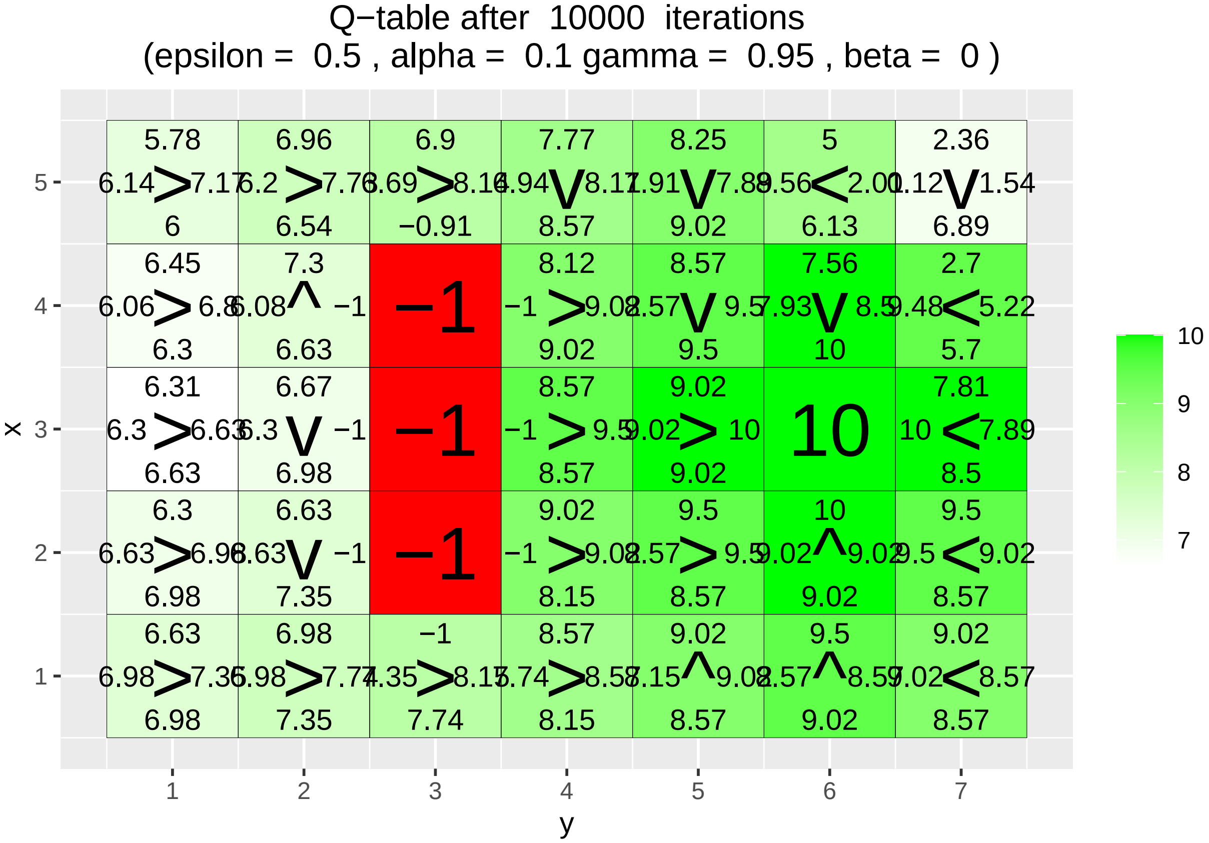


Q−table after 100 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)



Q−table after 1000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
1
2
3
4
5
y
1
2
3
4
5
6
7
−1
−1
−1
10
10
9
8
7
6
5


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
1
2
3
4
5
y
1
2
3
4
5
6
7
−1
−1
−1
10
10
9
8
7

Comment

(1) What has the agent learned after the first 10 episodes ?

The agent learned few information about the position of the goal (positive reward)and barrier(negative reward). When the agent is next to the goal, it will move to it. The agent will avoid to move to the barrier when it is next to it.

(2) Is the final greedy policy (after 10000 episodes) optimal? Why / Why not ?

The final greedy policy is not optimal for all states(after 10000 episodes).

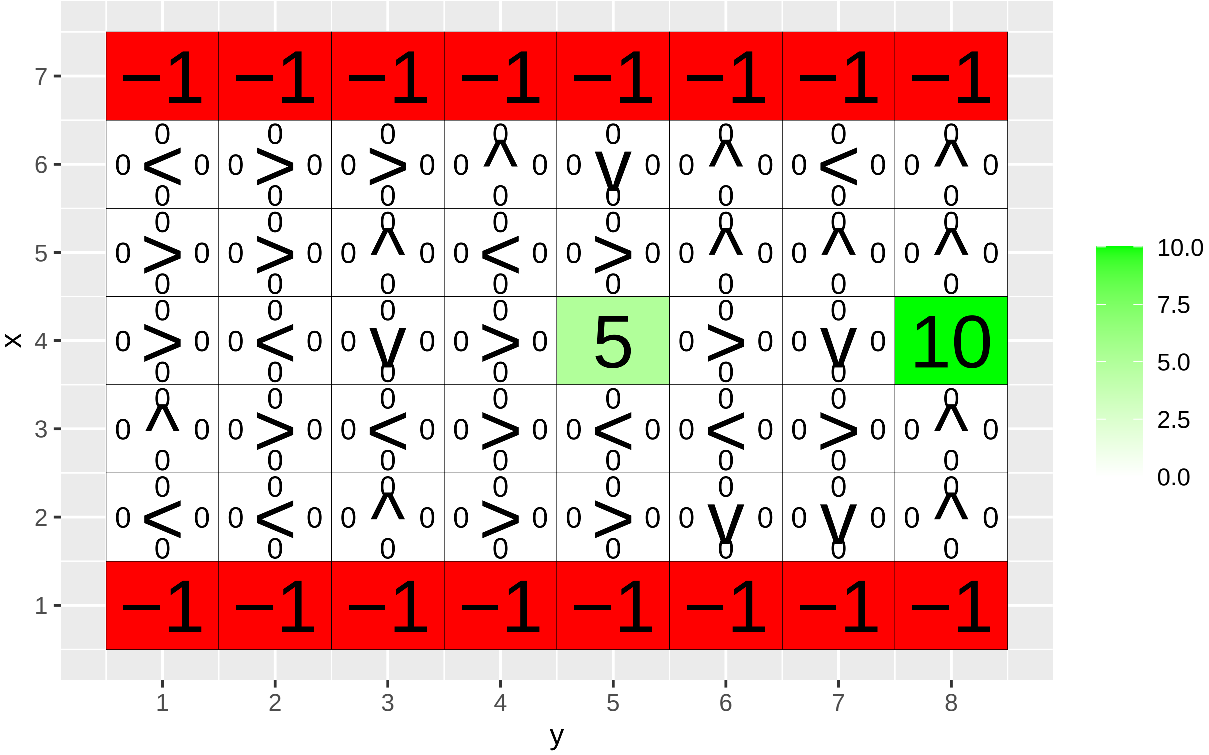
In state(5, 3), the best policy is go right and then down to the goal with reward 10. But the agent will choose to go left and then take a long way to reach the goal.

(3) 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 ?

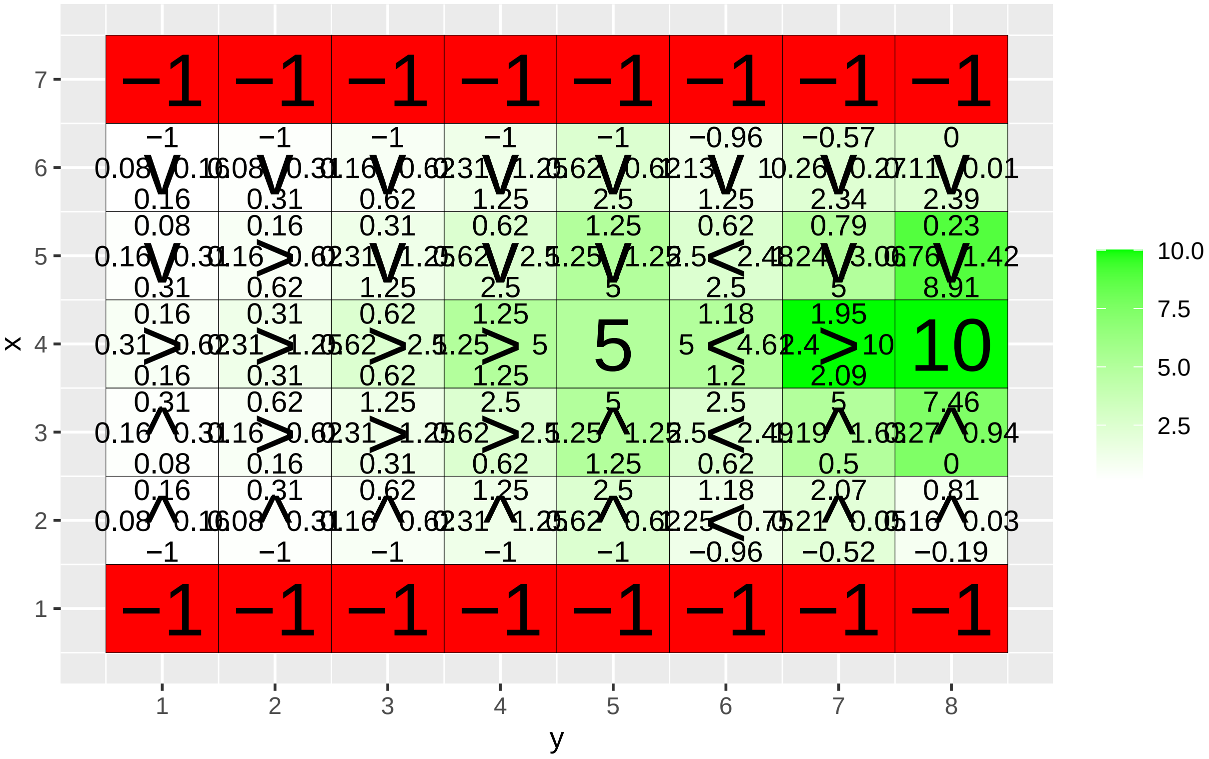
The agent can learn that there are multiple paths to the positive reward. From the 5th graph, it can be seen that many states have multiple maximum of q values, such as (2, 4) and (3, 1). Hence, the agent can choose any direction in those states. The 6th graph is another run with 10000 iterations, it can seen that optimal path from (3, 1) to the goal is different with the 5th graph.

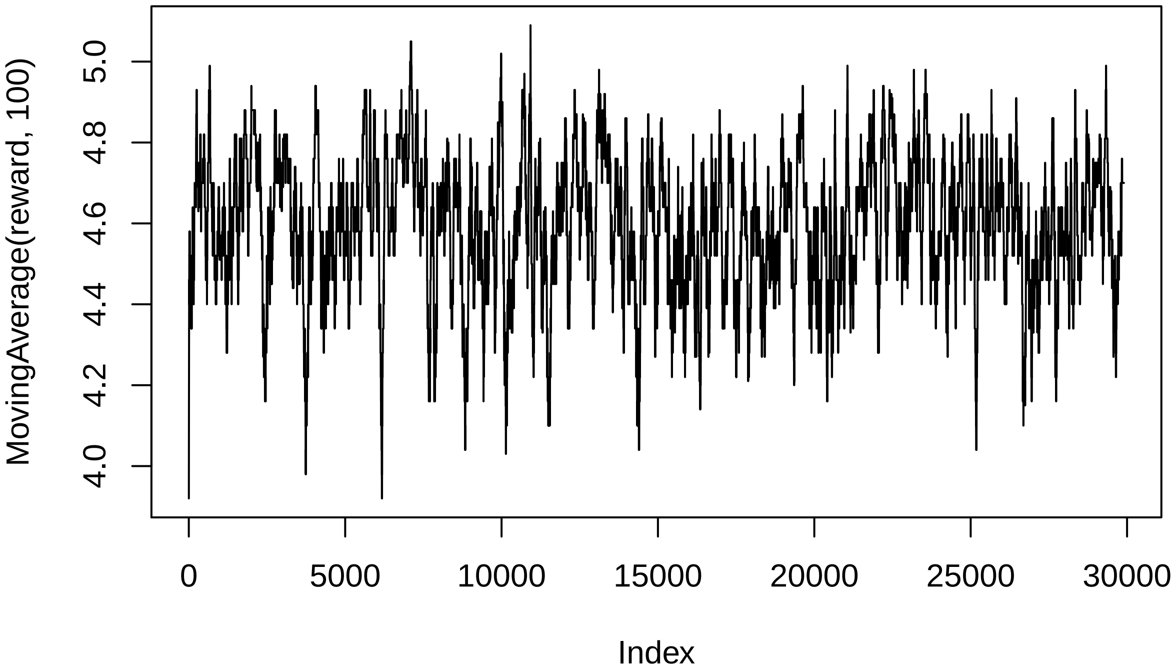
3. Envrionment B

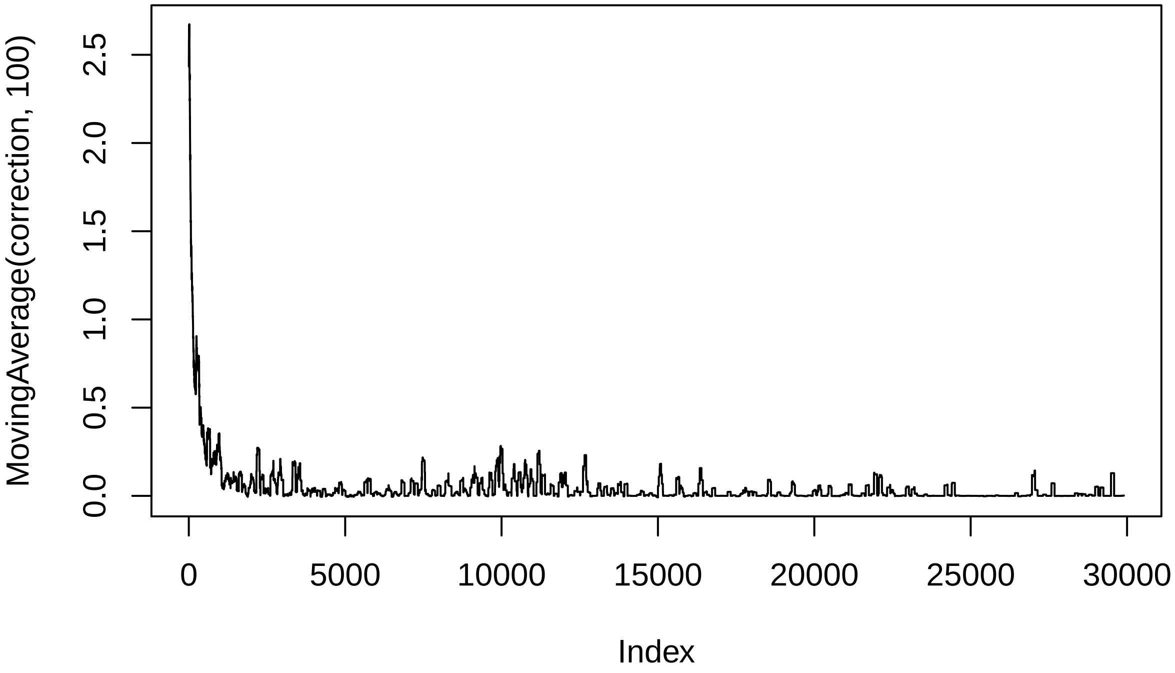
Q−table after 0 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)

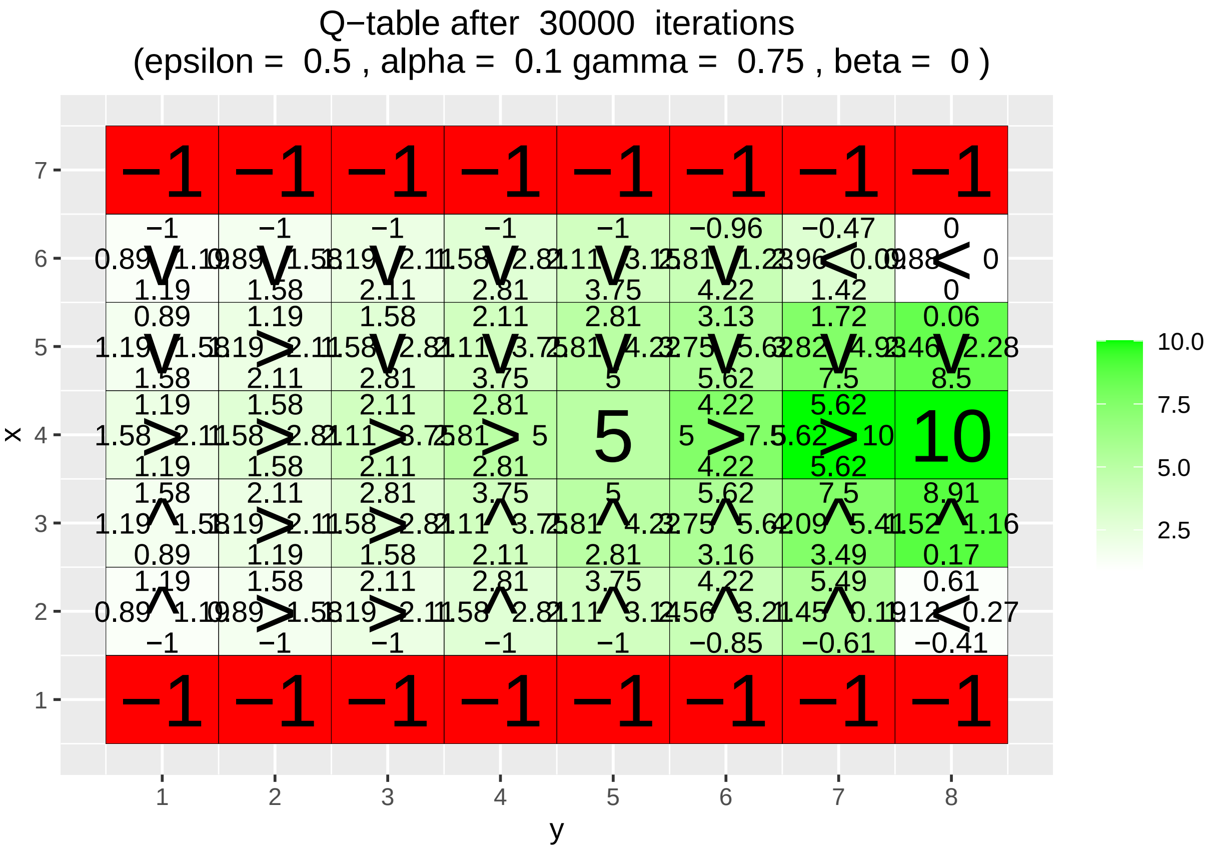


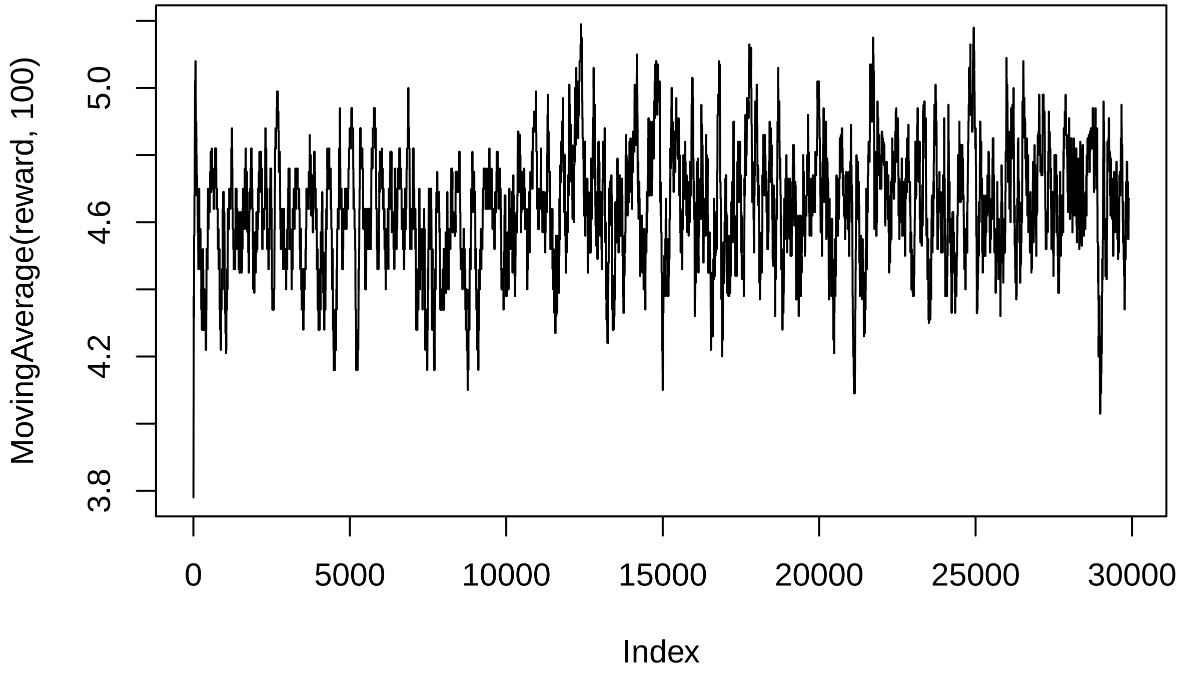
Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.5 , beta = 0)

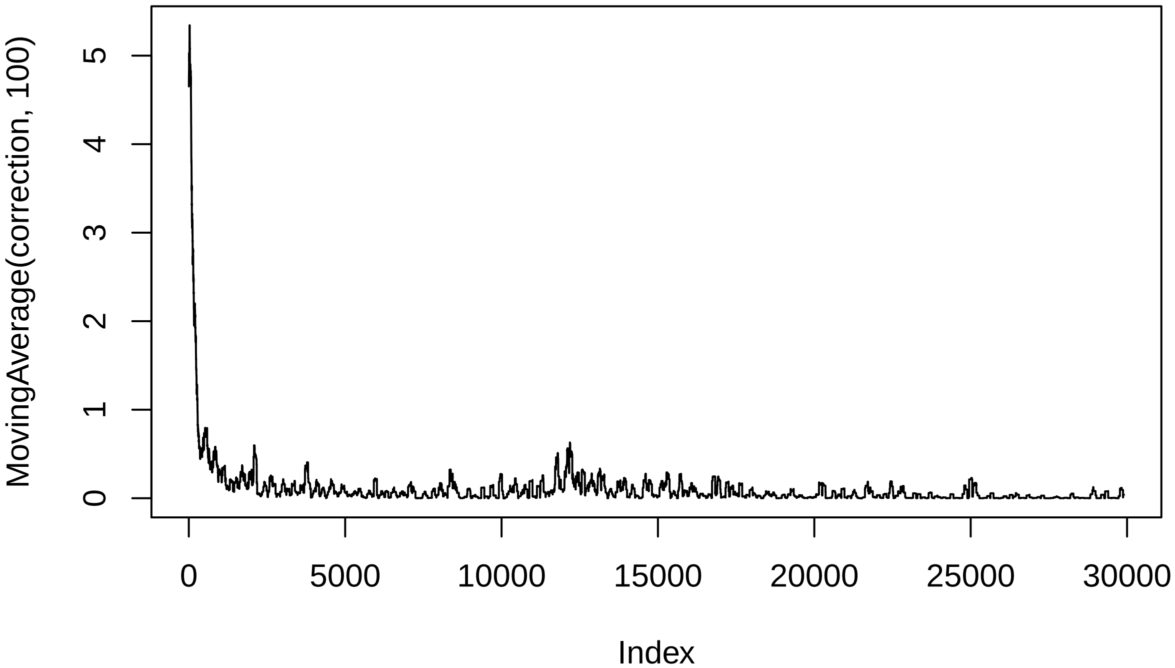



MovingAverage(reward, 100)
4.0
4.2
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index

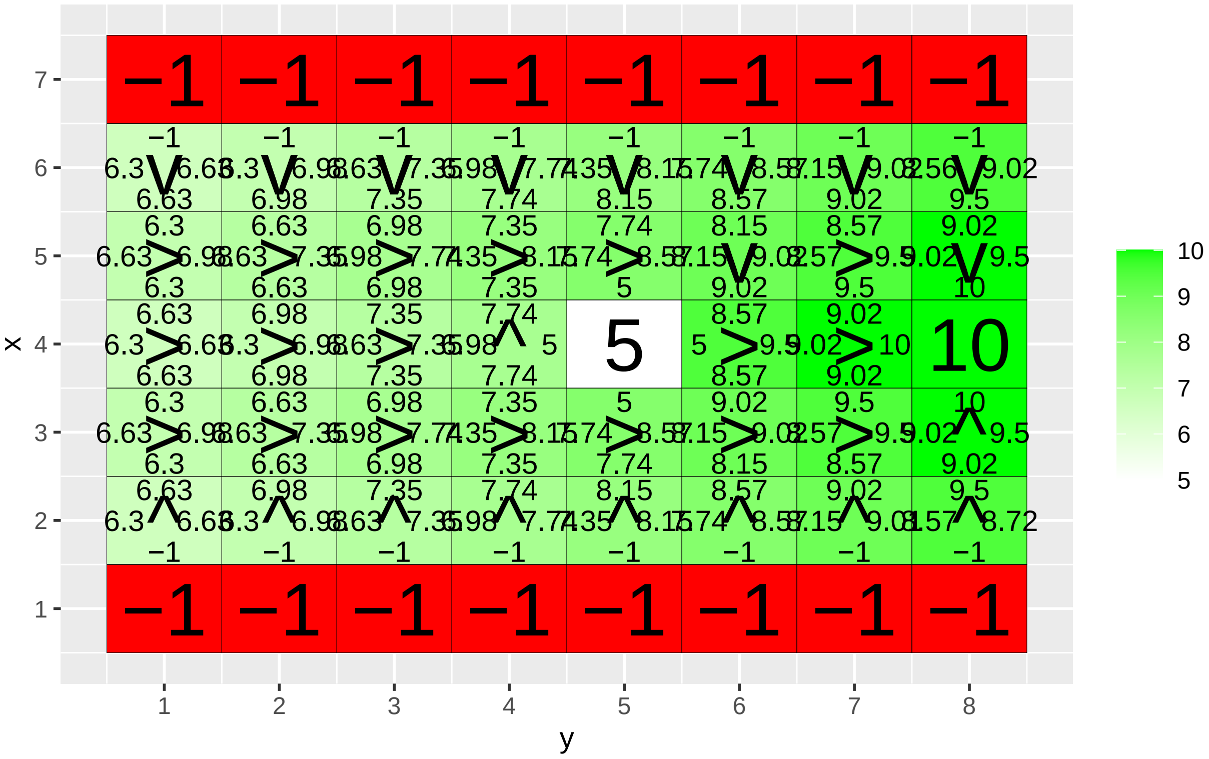

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
2.5
0
5000
10000
15000
20000
25000
30000
Index

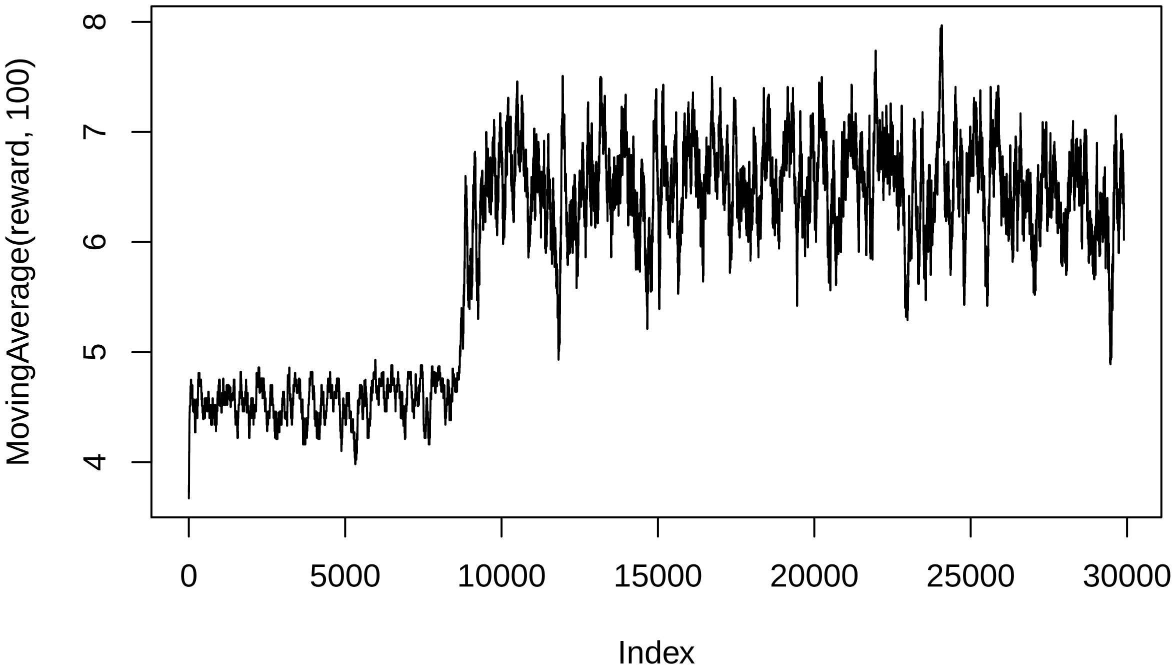

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
1
2
3
4
5
6
7
8
−1
5
10
10.0
7.5
5.0
2.5

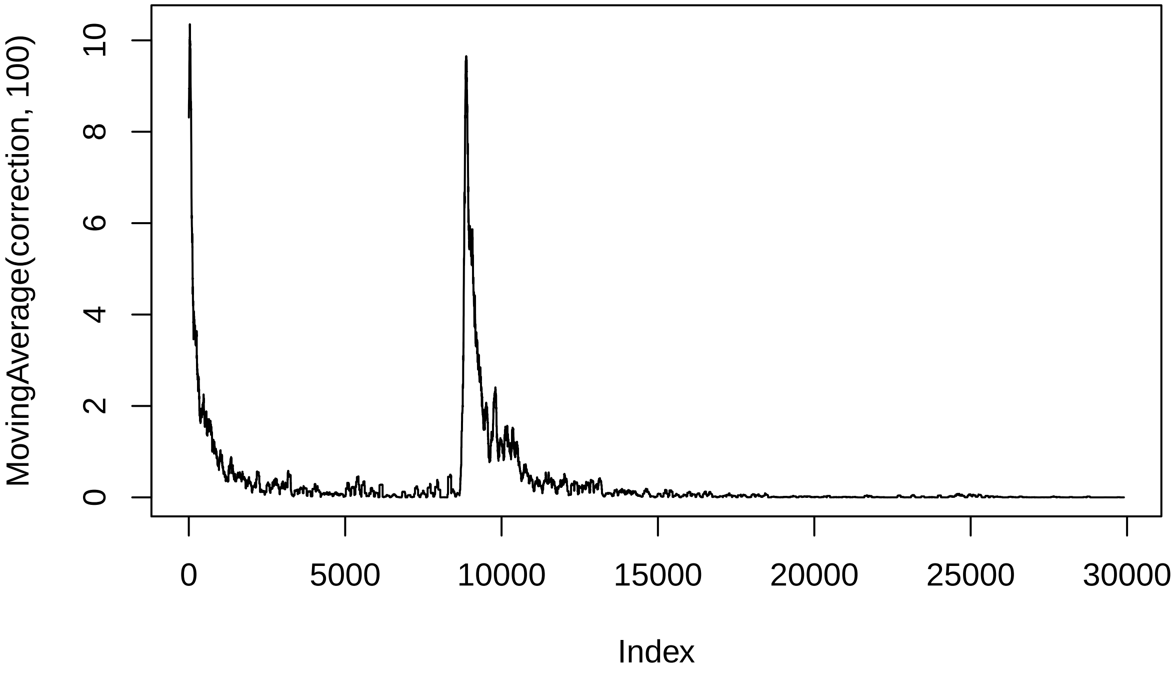

MovingAverage(reward, 100)
Index
3.8
4.2
4.6
5.0
0
5000
10000
15000
20000
25000
30000

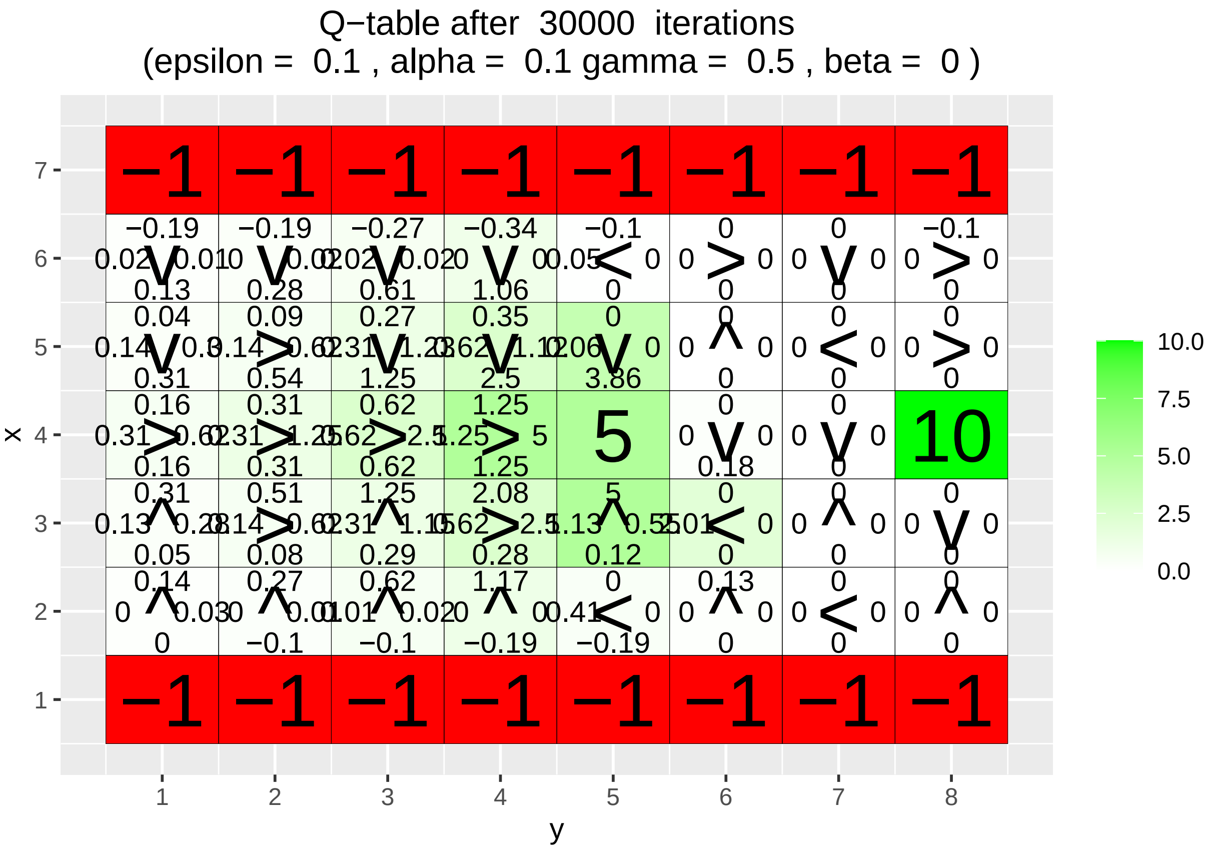


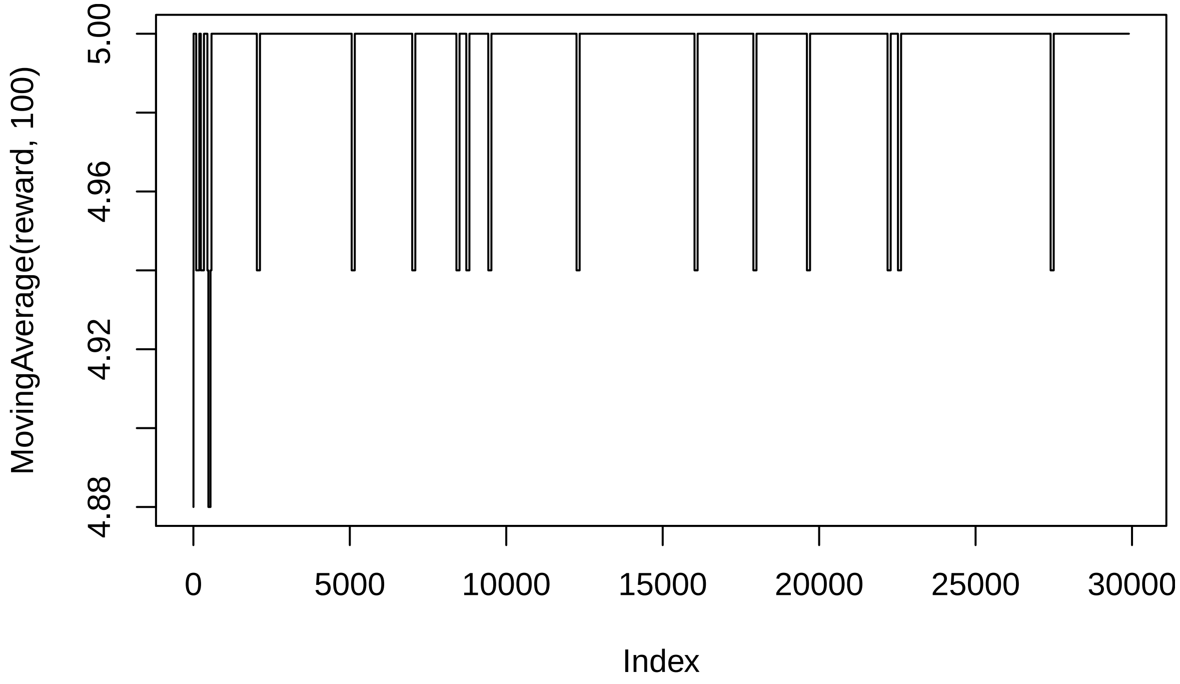
Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)

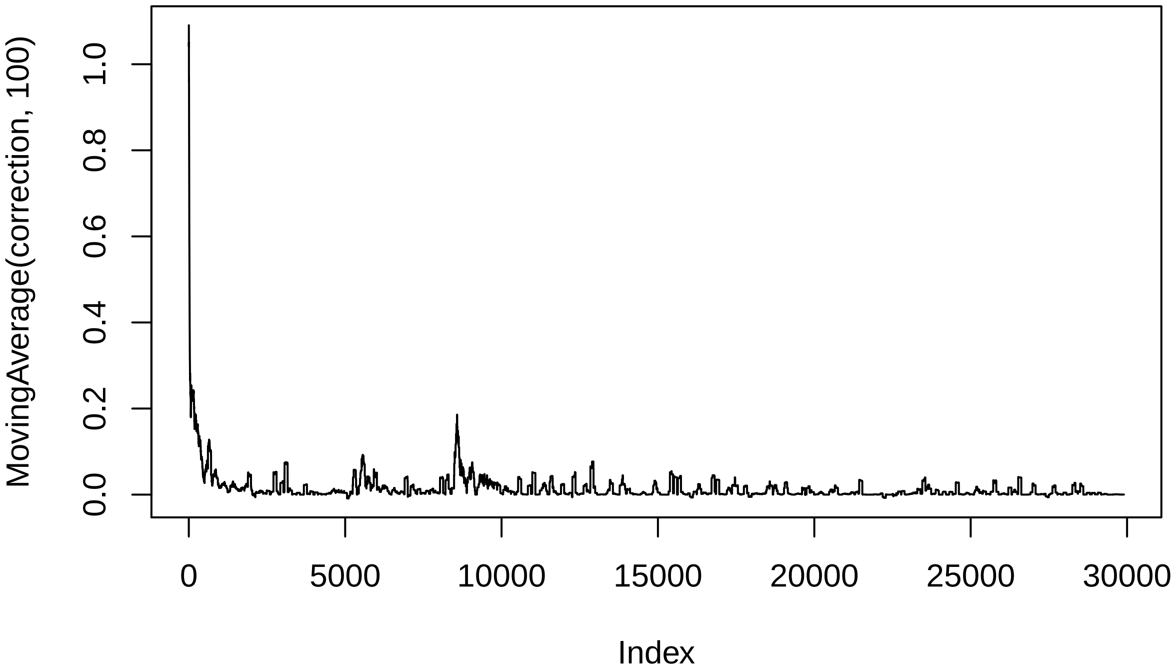



MovingAverage(reward, 100)
8
7
6
5
4
0
5000
10000
15000
20000
25000
30000
Index

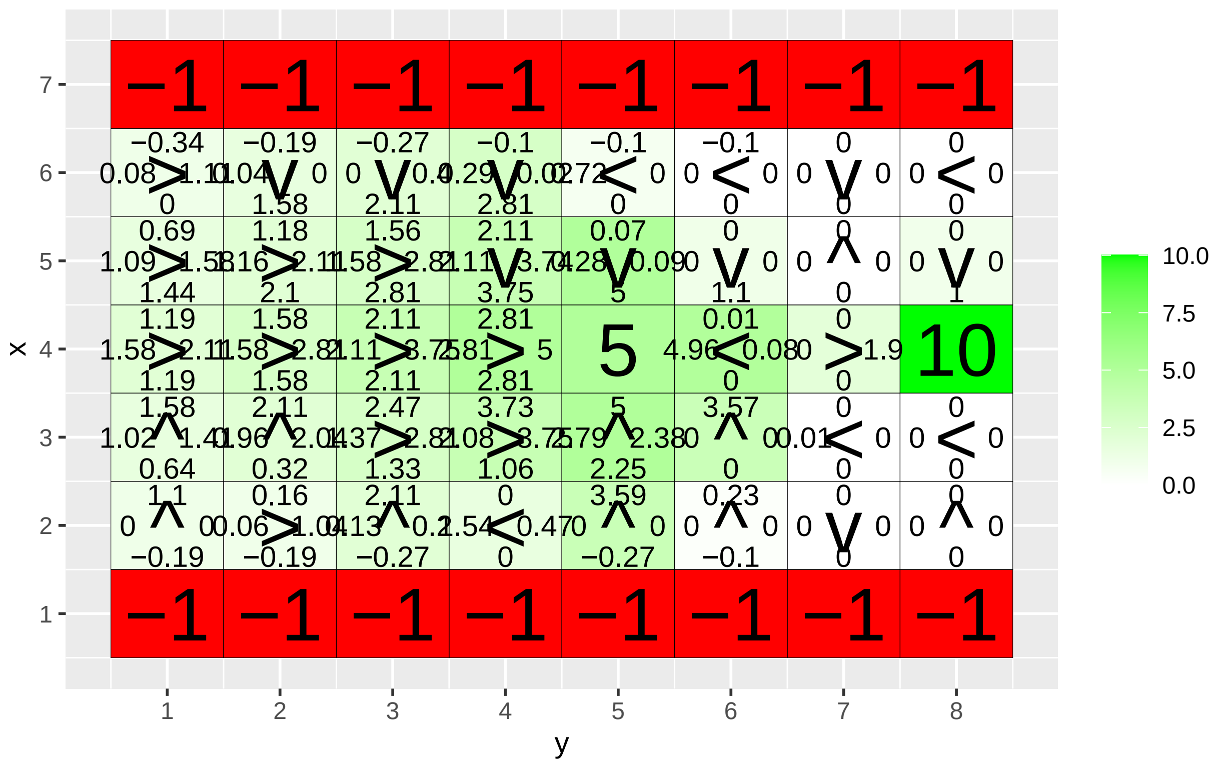

MovingAverage(correction, 100)
10
8
6
4
2
0
0
5000
10000
15000
20000
25000
30000
Index

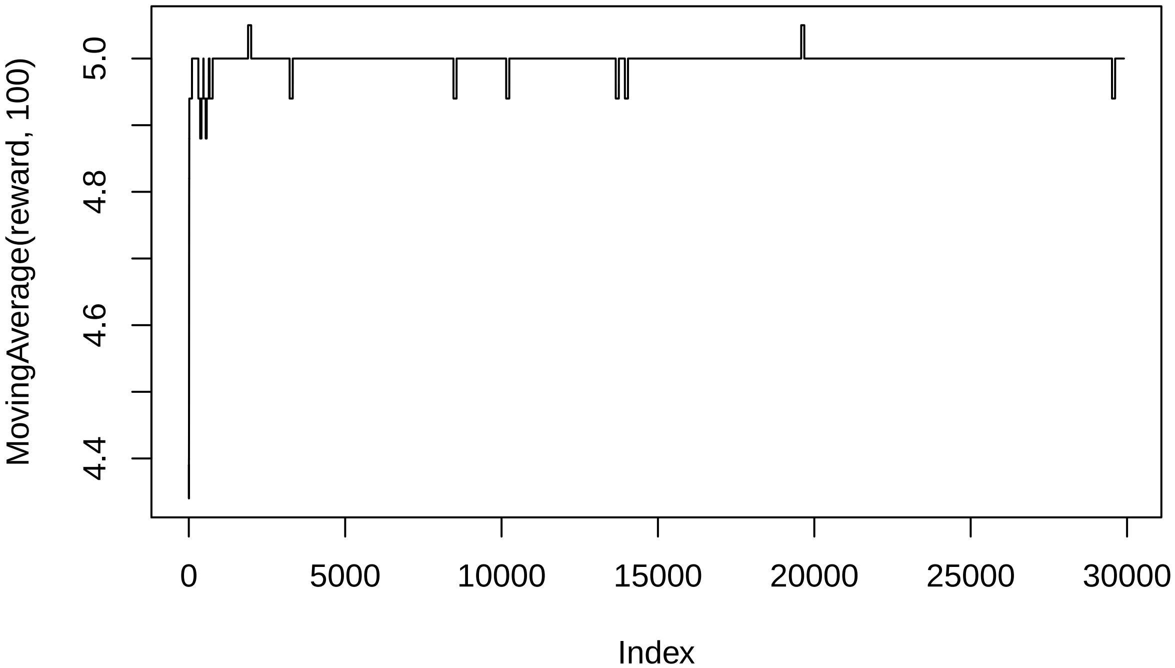

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
−1 −1 −1 −1 −1 −1 −1 −1
−0.19 −0.19 −0.27 −0.34 −0.1 0 0 −0.1
0.02 V 0.01 0 V 0.02 0.02 V 0.02 0 V 0 0.05 < 0 0 > 0 0 V 0 0 > 0
0.13 0.28 0.61 1.06 0 0 0 0
0.04 0.09 0.27 0.35 0 0 0 0
0.14 V 0.3 0.14 > 0.62 0.31 V 1.2 0.62 V 1.1 0.06 V 0 0 ^ 0 0 < 0 0 > 0
0.31 0.54 1.25 2.5 3.86 0 0 0
0.16 0.31 0.62 1.25 0 0
0.31 > 0.62 0.31 > 1.25 0.62 > 2.5 1.25 > 5 5 0 V 0 0 V 0 10
0.16 0.31 0.62 1.25 0.18 0
0.31 0.51 1.25 2.08 5 0 0 0
0.13 ^ 0.28 0.14 > 0.62 0.31 ^ 1.1 0.62 > 2.5 1.13 ^ 0.5 0.01 < 0 0 ^ 0 0 V 0
0.05 0.08 0.29 0.28 0.12 0 0 0
0.14 0.27 0.62 1.17 0 0.13 0 0
0 ^ 0.03 0 ^ 0.01 0.01 ^ 0.02 0 ^ 0 0.41 < 0 0 ^ 0 0 < 0 0 ^ 0
0 −0.1 −0.1 −0.19 −0.19 0 0 0
−1 −1 −1 −1 −1 −1 −1 −1
x
7
6
5
4
3
2
1
y
1 2 3 4 5 6 7 8
10.0
7.5
5.0
2.5
0.0

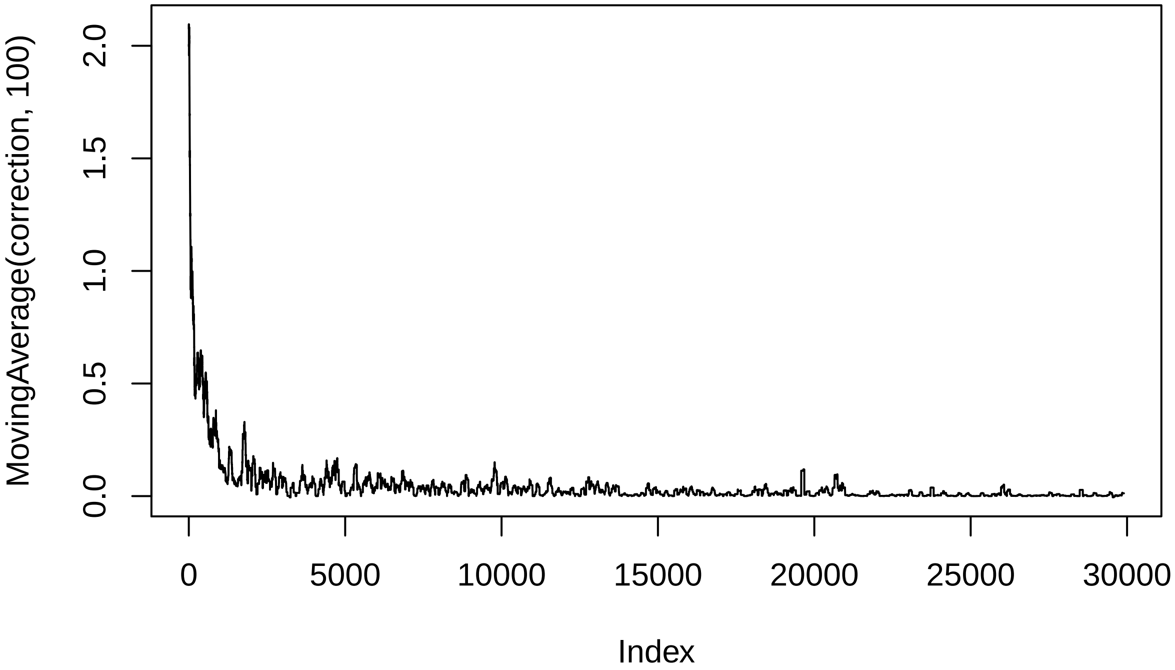

MovingAverage(reward, 100)
5.00
4.96
4.92
4.88
0 5000 10000 15000 20000 25000 30000
Index

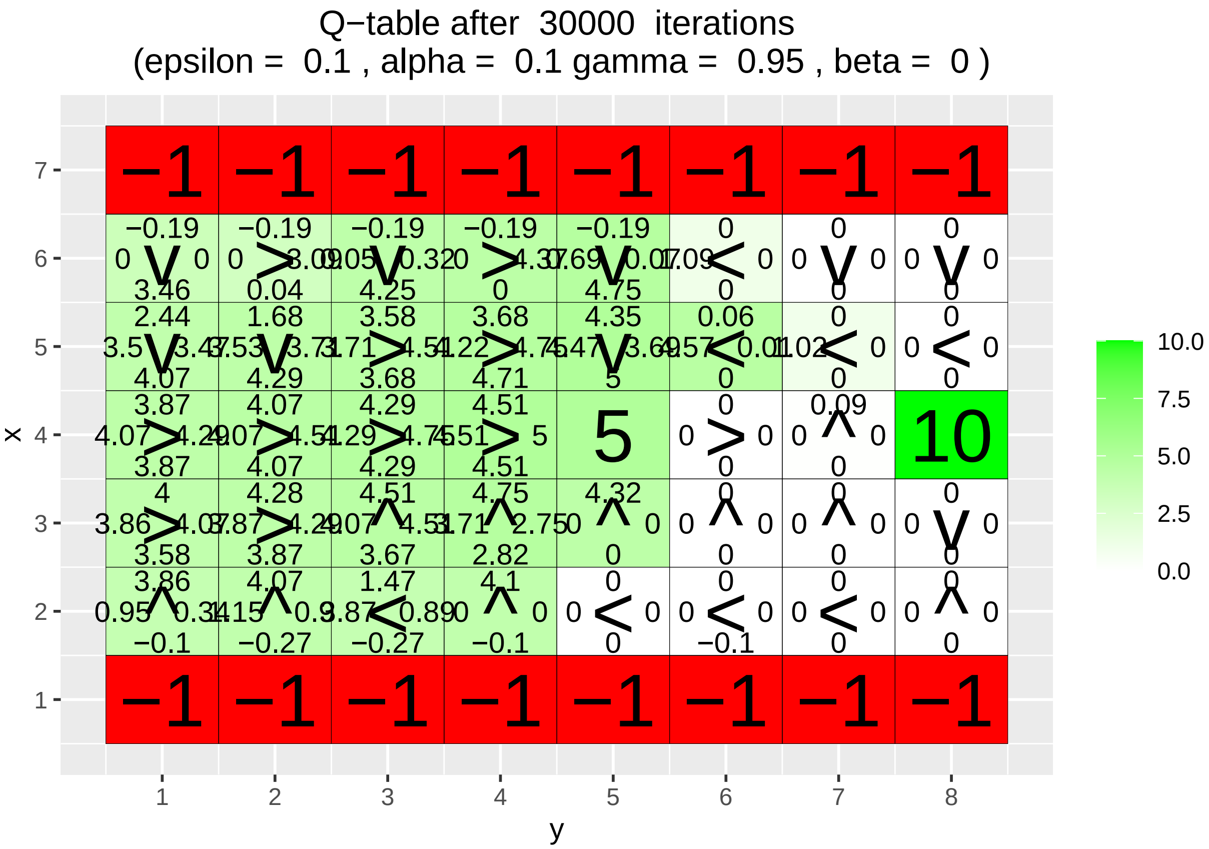


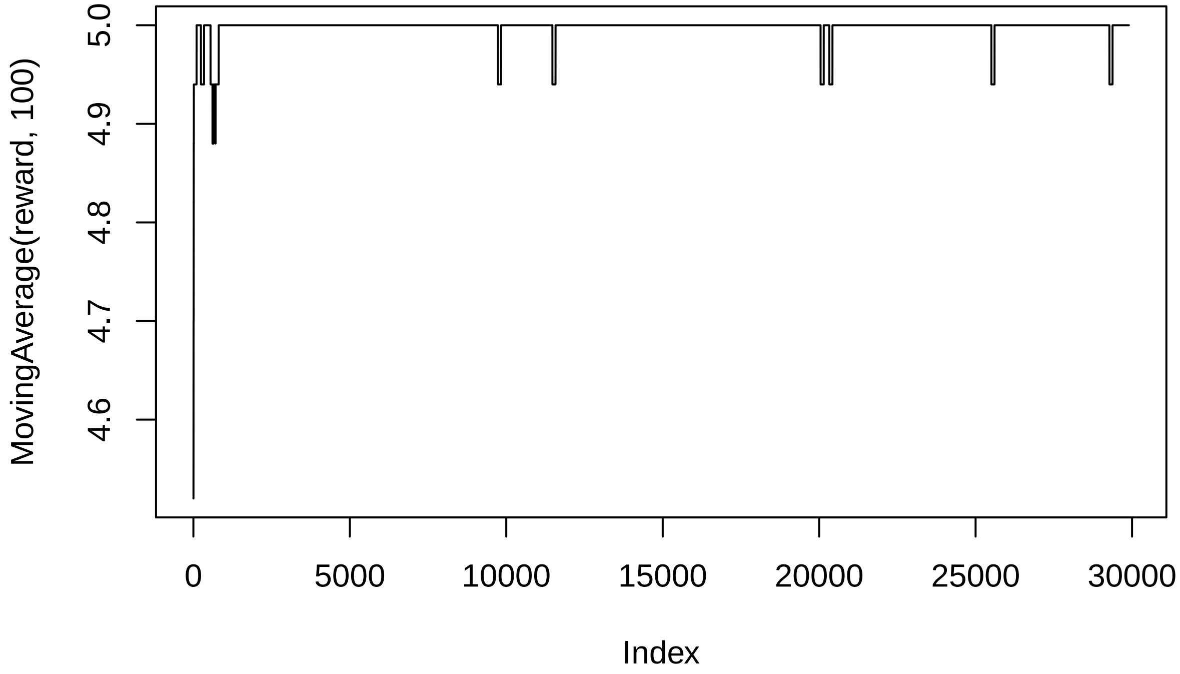
Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)

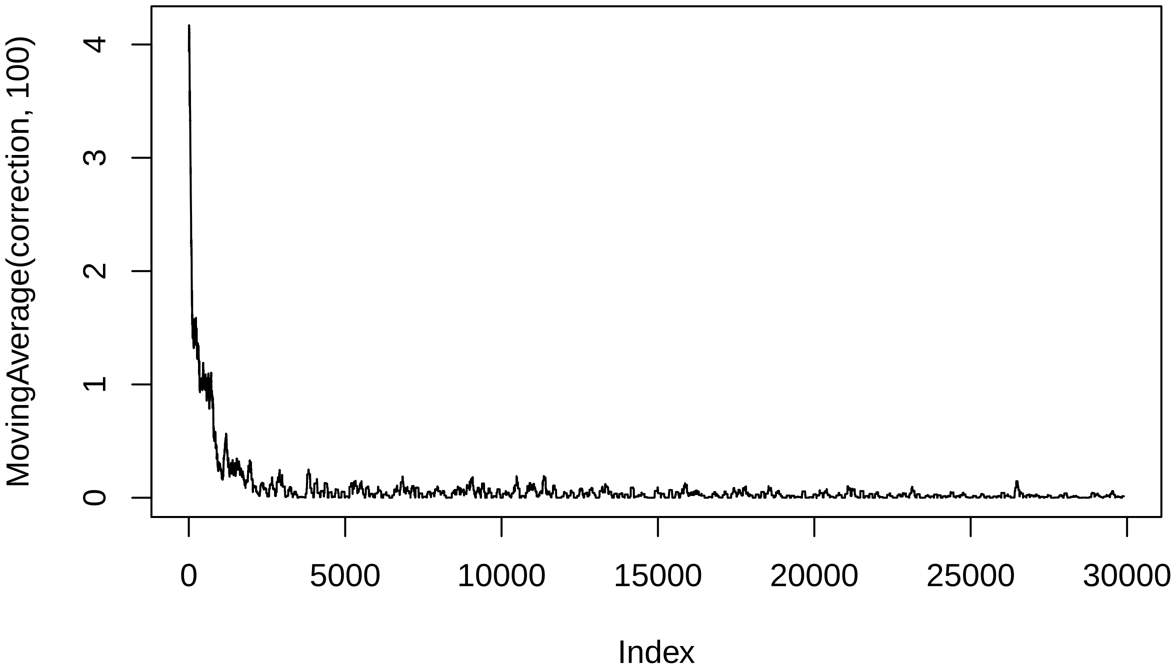



MovingAverage(reward, 100)
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index


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


Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.95 , beta = 0)
−1 −1 −1 −1 −1 −1 −1 −1
−0.19 −0.19 −0.19 −0.19 −0.19 0 0 0
0 V 0 0 > 3.09 0.05 V 0.32 0 > 4.37 0.69 V 0.07 1.09 < 0 0 V 0 0 V 0
3.46 0.04 4.25 0 4.75 0 0 0
2.44 1.68 3.58 3.68 4.35 0.06 0 0
3.5 V 3.47 3.53 V 3.73 3.71 > 4.54 4.22 > 4.74 4.47 V 3.69 4.57 < 0.01 1.02 < 0 0 < 0
4.07 4.29 3.68 4.71 5 0 0 0
3.87 4.07 4.29 4.51 0 0.09
4.07 > 4.29 4.07 > 4.54 4.29 > 4.74 4.51 > 5 5 0 > 0 0 ^ 0 10
3.87 4.07 4.29 4.51 0 0
4 4.28 4.51 4.75 4.32 0 0 0
3.86 > 4.07 3.87 > 4.29 4.07 ^ 4.53 3.71 ^ 2.75 0 ^ 0 0 ^ 0 0 ^ 0 0 V 0
3.58 3.87 3.67 2.82 0 0 0 0
3.86 4.07 1.47 4.1 0 0 0 0
0.95 ^ 0.31 4.15 ^ 0.9 3.87 < 0.89 0 ^ 0 0 < 0 0 < 0 0 < 0 0 ^ 0
−0.1 −0.27 −0.27 −0.1 0 −0.1 0 0
−1 −1 −1 −1 −1 −1 −1 −1
7
6
5
4
3
2
1
x
1
2
3
4
5
6
7
8
y
10.0
7.5
5.0
2.5
0.0


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



Comment

Investigate how the parameters affect the learned policy by running 30000 episodes of Q-learning

(1) Gamma affection

$$q_\pi(s,a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|s,a]$$

Gamma affects the amount of information that the agent can learn. Hence, with the same length of path, large gamma can make the agent learn more information compared with the small one.

1. epsilon, alpha, gamma, beta is 0.5, 0.1, 0.5, 0 respectively

In the Q-table graph, it can be seen that the agent can learn few information of 10 reward and large information of 5 reward. Hence, all the states before $y = 7$ will move to 5 reward and others will move to 10 reward. Most of the q values are small.

In the Movingaverage reward graph, most of the values are below 5.

The initial Movingaverage correction value is quite small.

2. epsilon, alpha, gamma, beta is 0.5, 0.1, 0.75, 0 respectively

In the Q-table graph, it can be seen that the agent can learn more information of 10 reward and also large information of 5 reward. Hence, all the states before $y = 6$ will move to 5 reward and others will move to 10 reward. Most of the q values are larger than previous one.

In the Movingaverage reward graph, many values are larger than 5.

The initial Movingaverage correction value is larger.

3. epsilon, alpha, gamma, beta is 0.5, 0.1, 0.95, 0 respectively

In the Q-table graph, it can be seen that the agent can learn adequate information of 10 reward. The agent learns that 10 reward is a better goal than 5 reward. Hence, all the states navigate around 5 reward to 10 reward. Most of the q values are larger than the other two.

In the Movingaverage reward graph, the values are larger than 5 after about 8000 iterations.

The initial Movingaverage correction value is larger than the other two. There is another peak at about 8000 iterations.

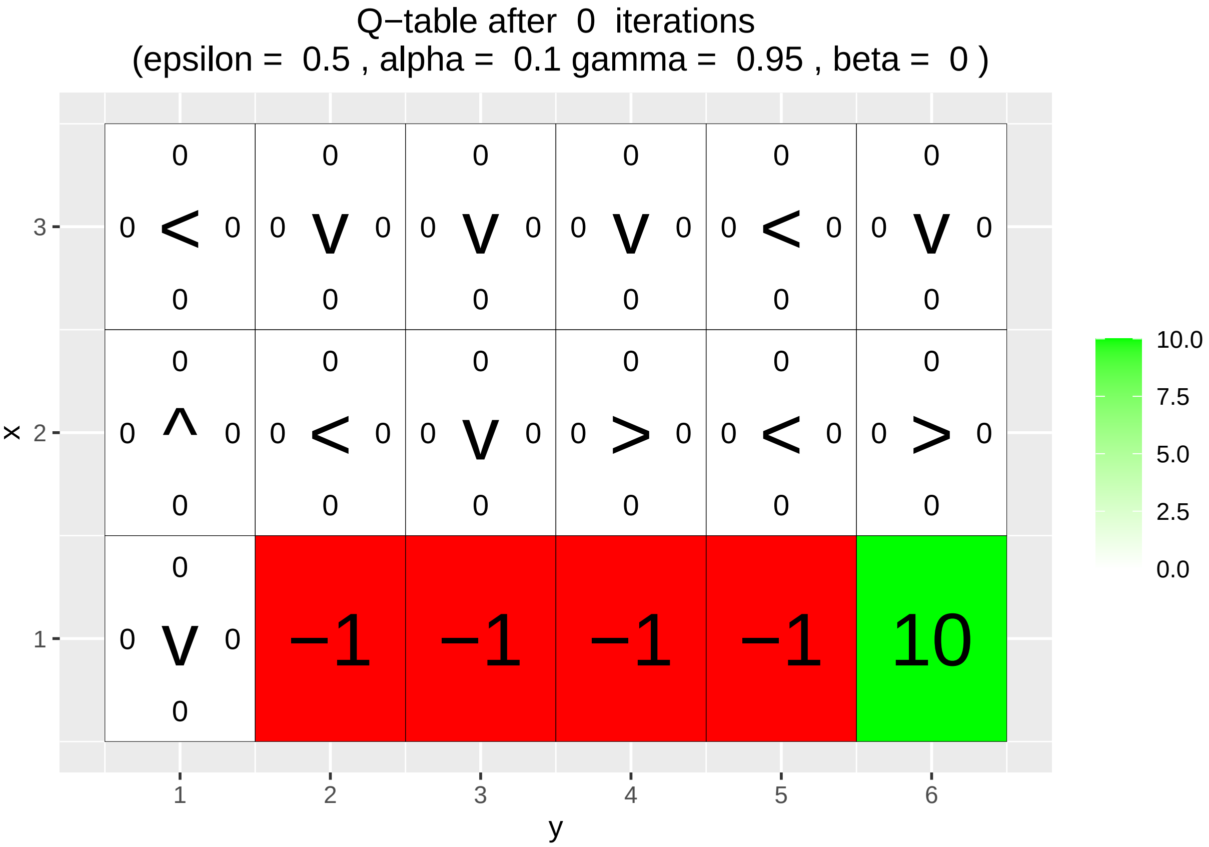
In general, the large gamma can make the agent avoid the local optimal and explore the global optimal.

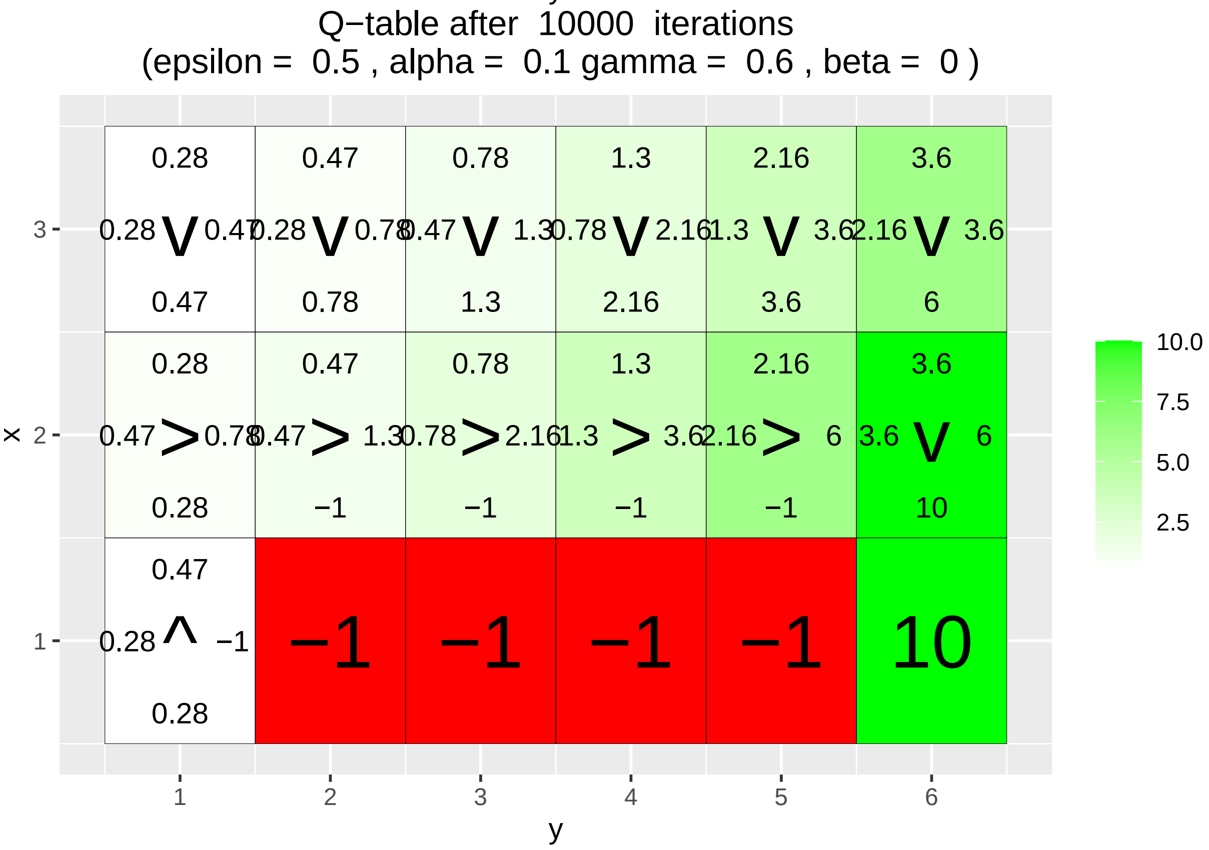
(2) Epsilon affection

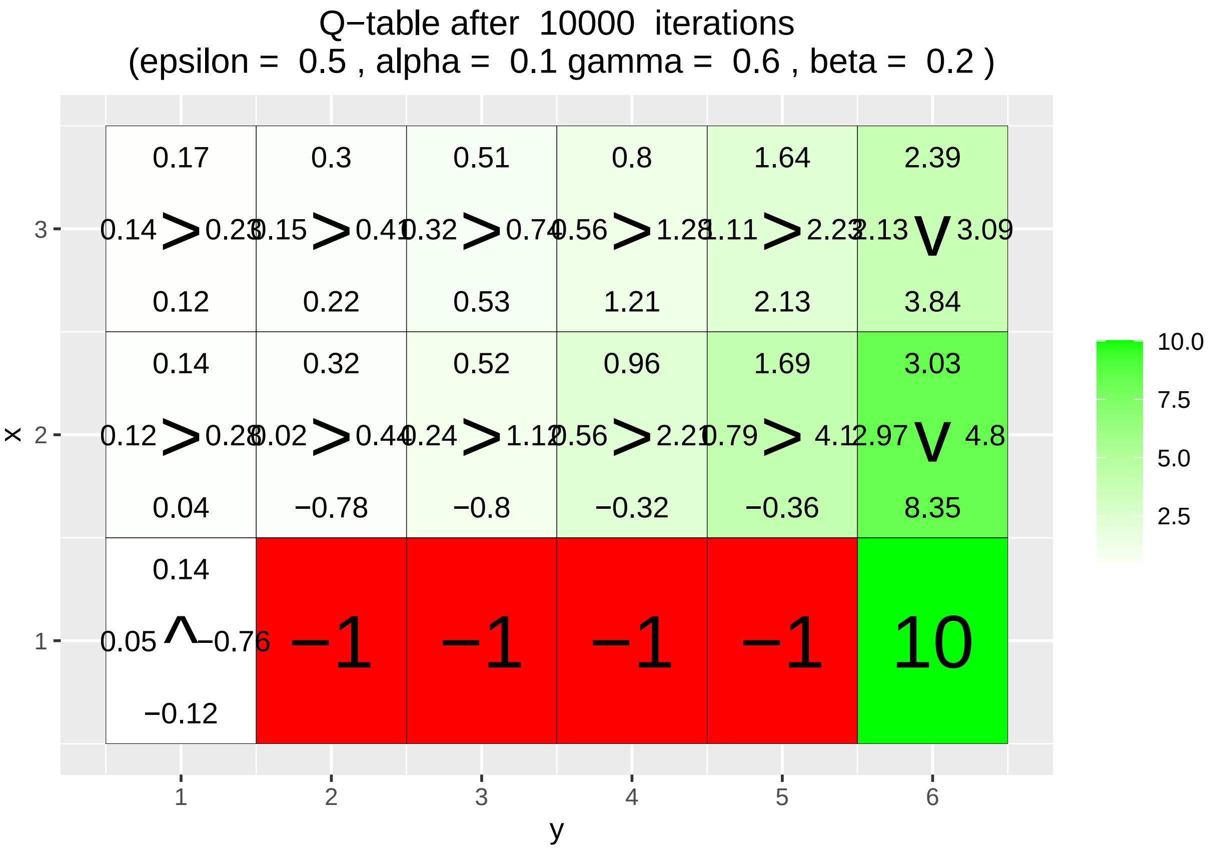
Epsilon can affect the conflict between using the current information to maximize the reward and trying new policies to find even better reward.

When $\epsilon = 0.1$, the agent have few chance to explore the new policies to find the global optimal. Most of the moving average reward is exact 5. The reason is that the agent learn the information at 5 reward firstly because it has a shorter distance than 10 reward, then it just uses this current information to maximize the reward. Hence, the agent can not have access to get the information at 10 reward. Apart from the large iterations(30000) and gamma(0.95), the small epsilon (0.1)can result in the local optimal. The agent has little ability to explore the global optimal.

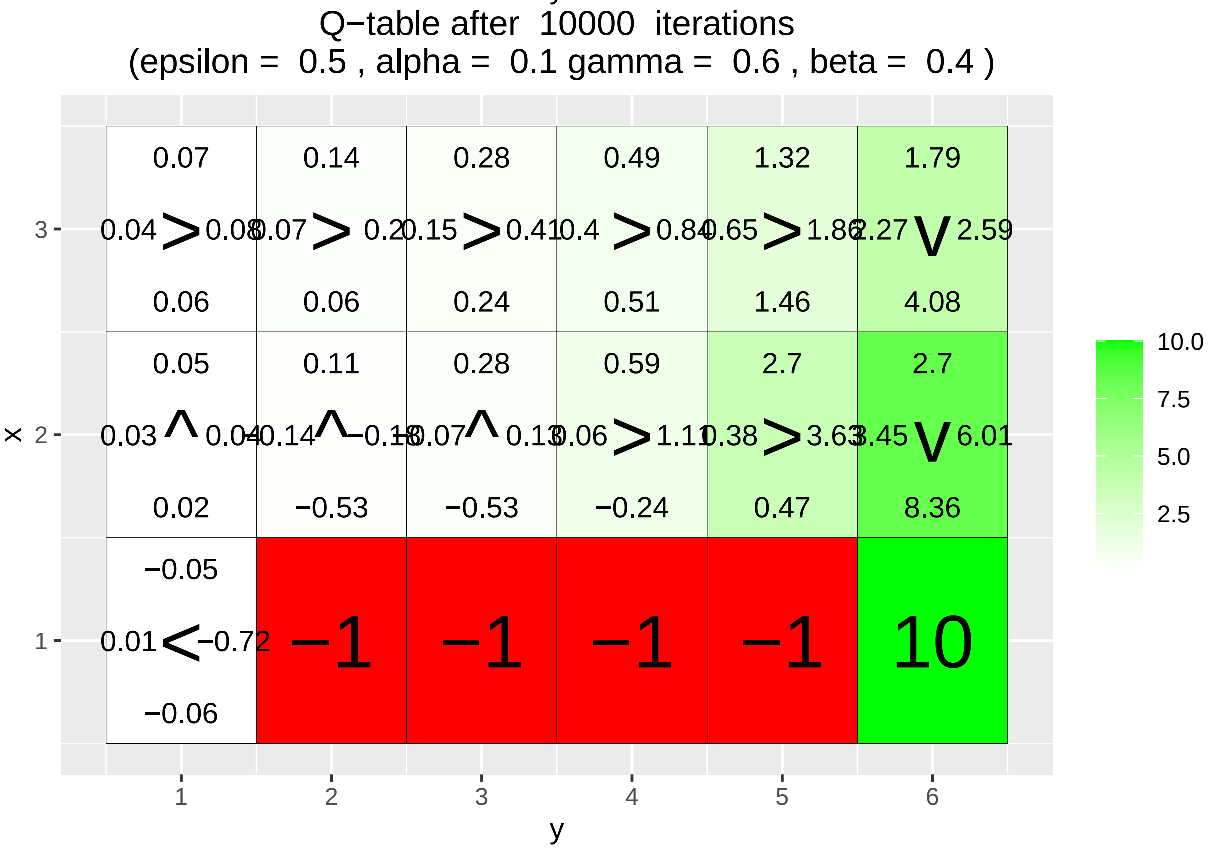
4. Environment C


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

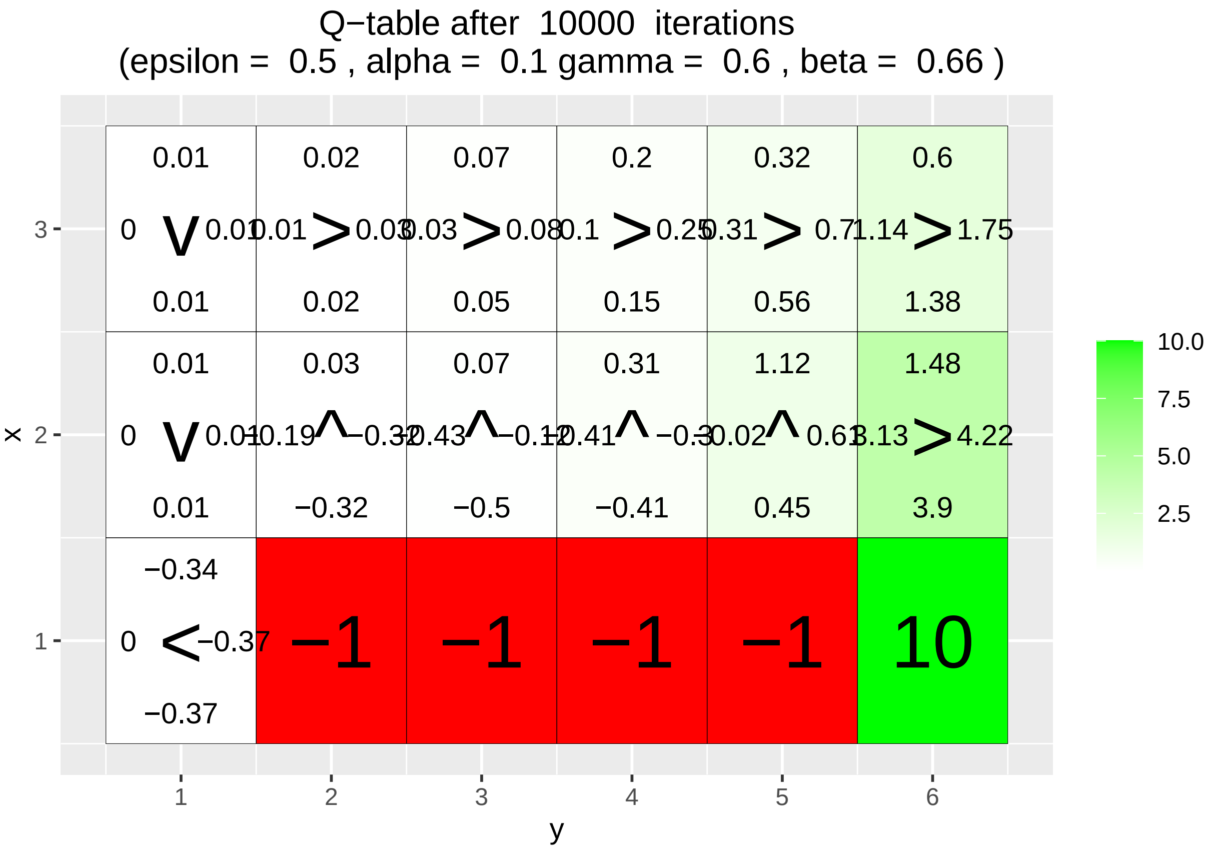

Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0)



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)



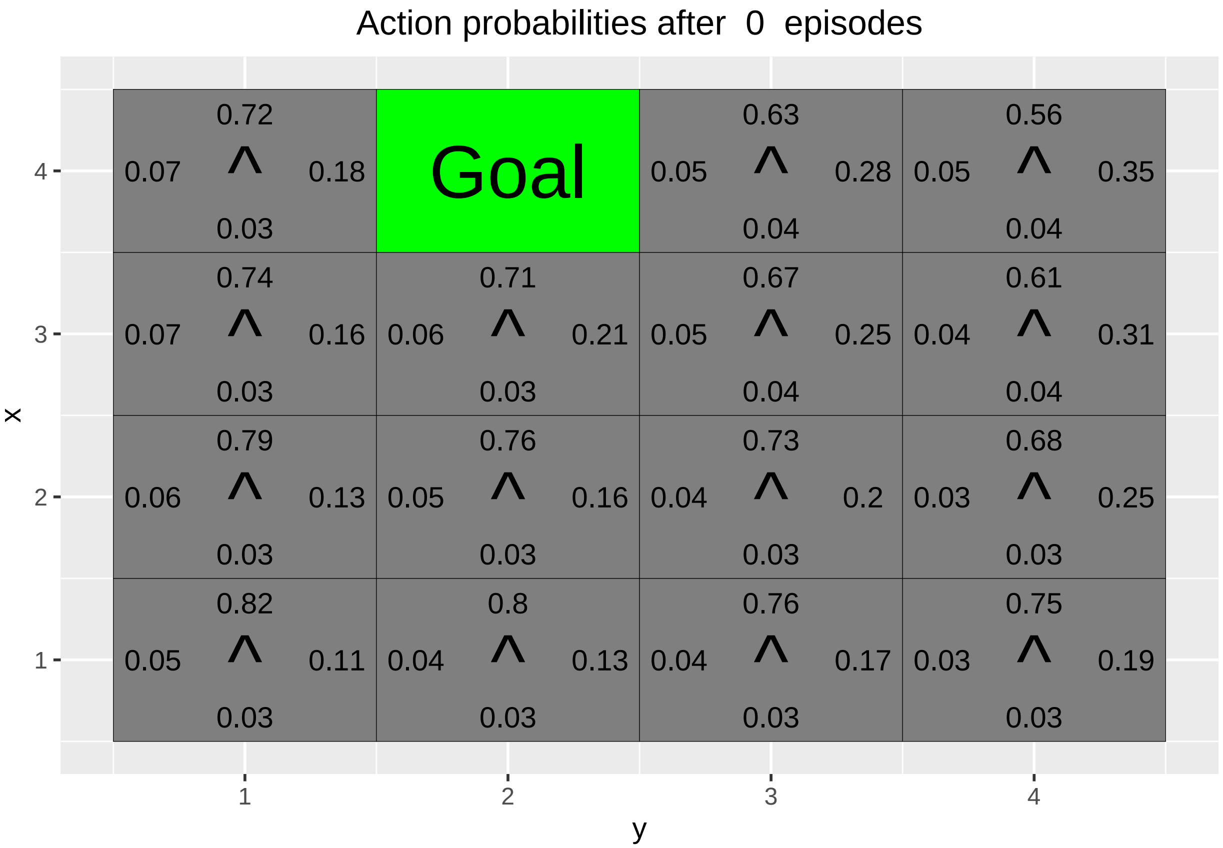
Comment

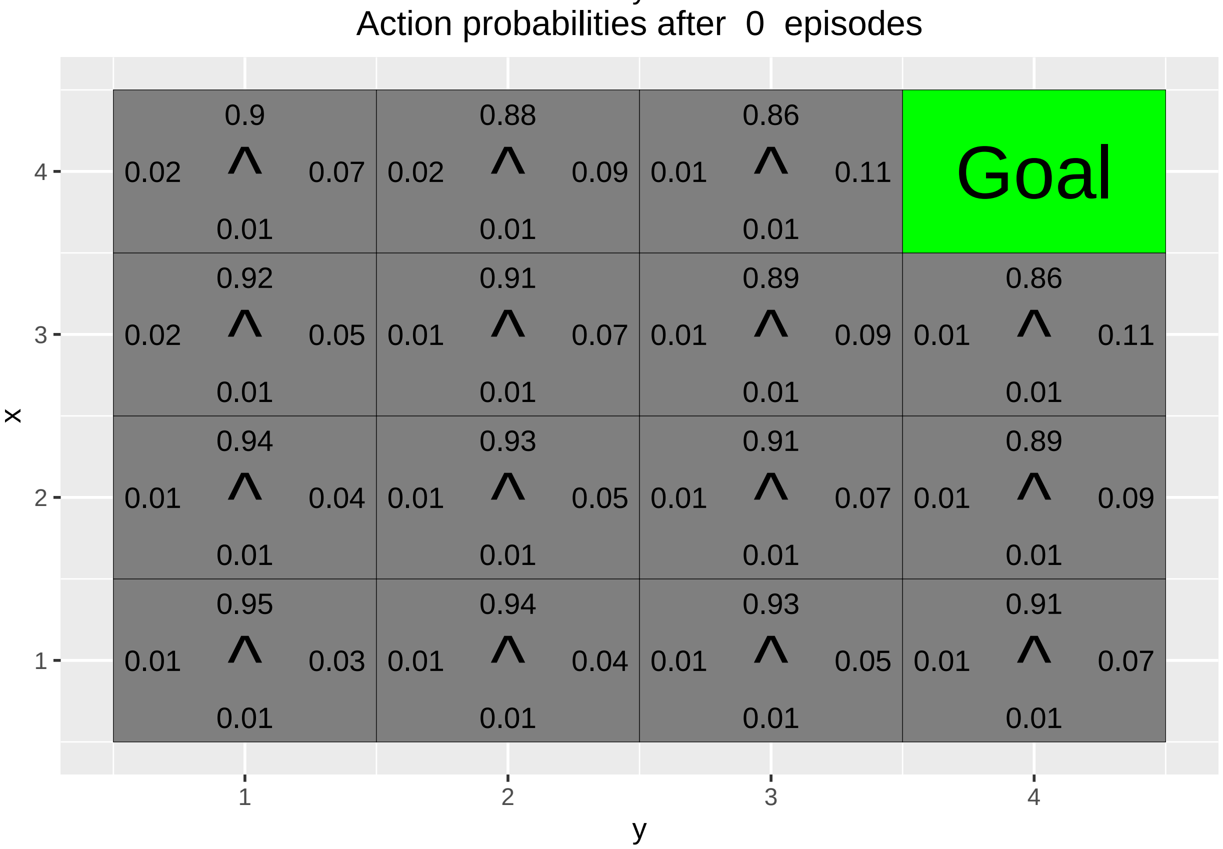
Investigate how the parameters affect the learned policy by running 30000 episodes of Q-learning

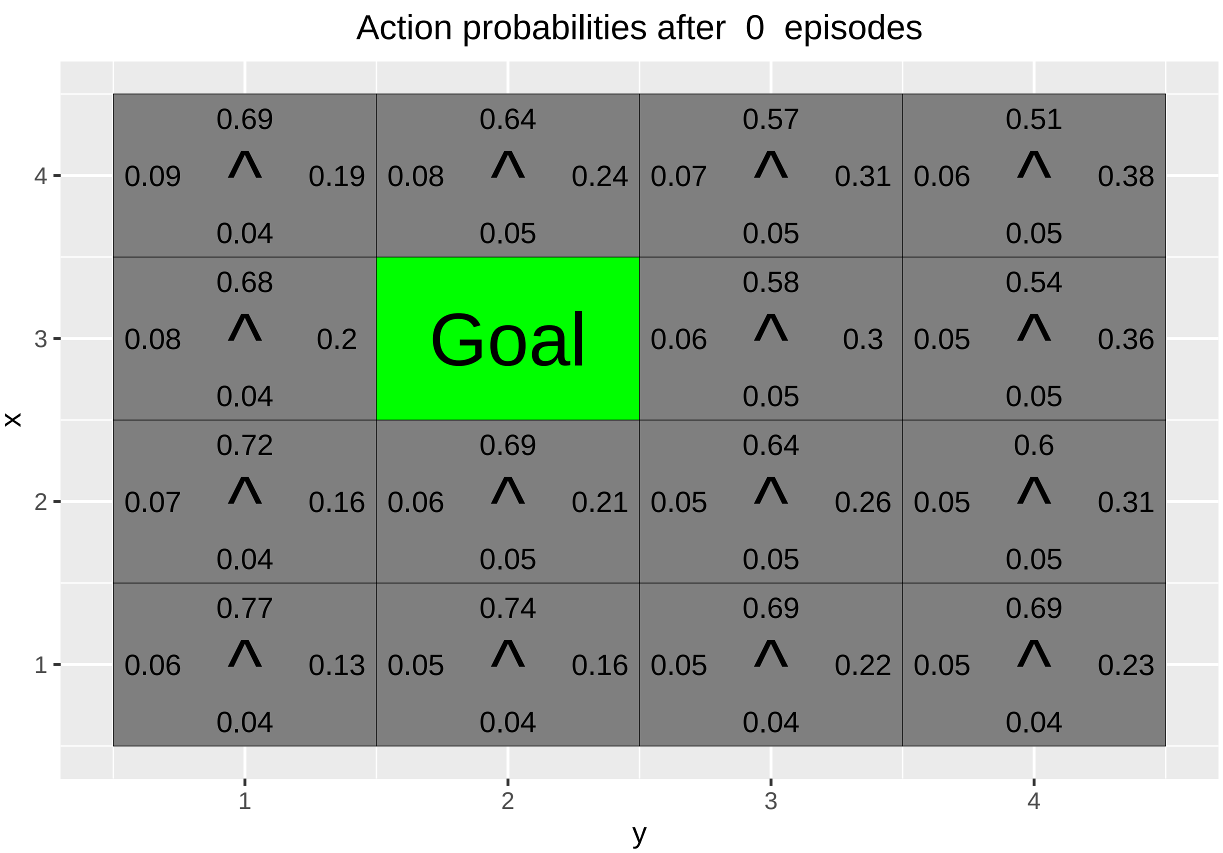
Beta can affect the uncertainty of the environment system.

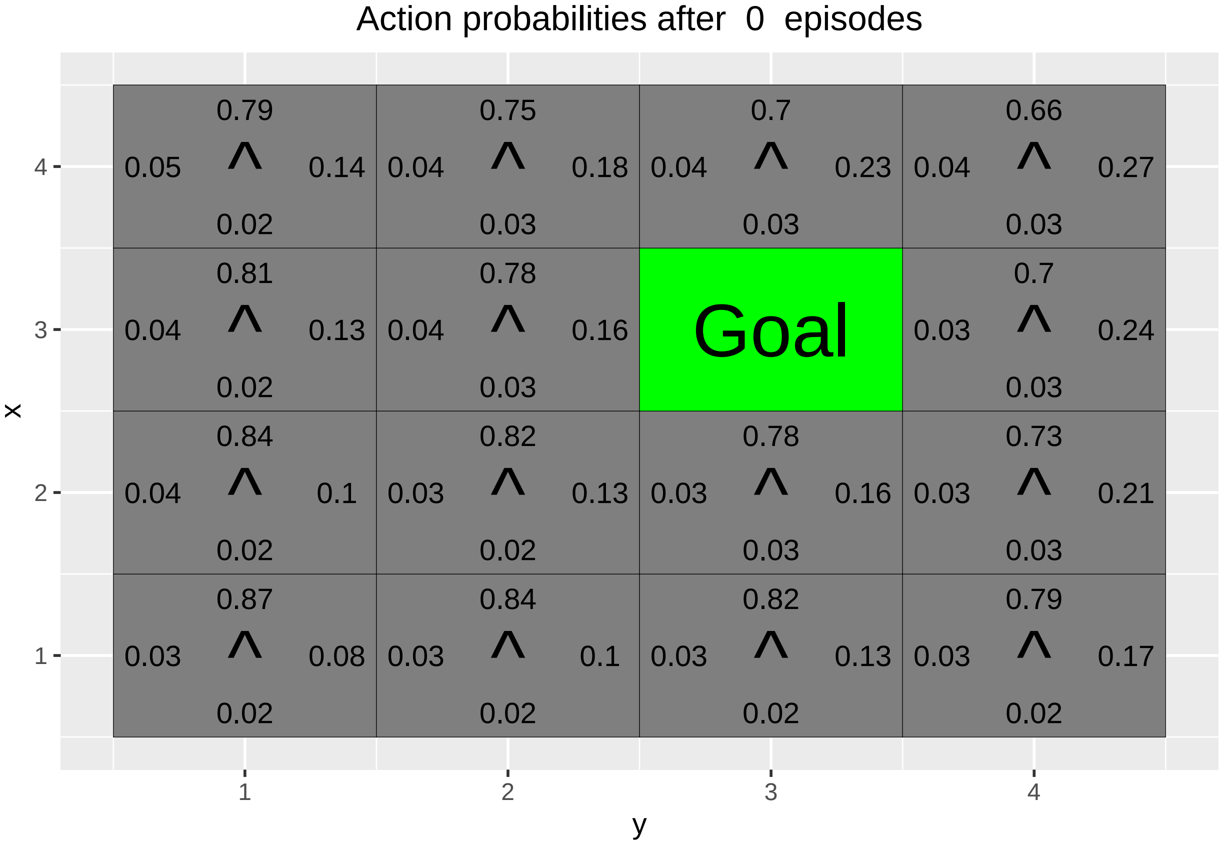
When $\beta = 0$, this means that the transition model is deterministic, the actual action is the given action. With β increasing, we can see the Q-table changes especially in the second row, from right arrows to up arrows. Because of the uncertainty, agent might move to -1 areas (slip right) thus get a negative reward. So agent choose a safety way to go (go up and right then go down to avoid to hit -1 areas). When $\beta >= 0.4$, the policy in the start state (1, 1) changes from up to left. Because of the uncertainty, agent might move to up and avoid to hit -1 areas.

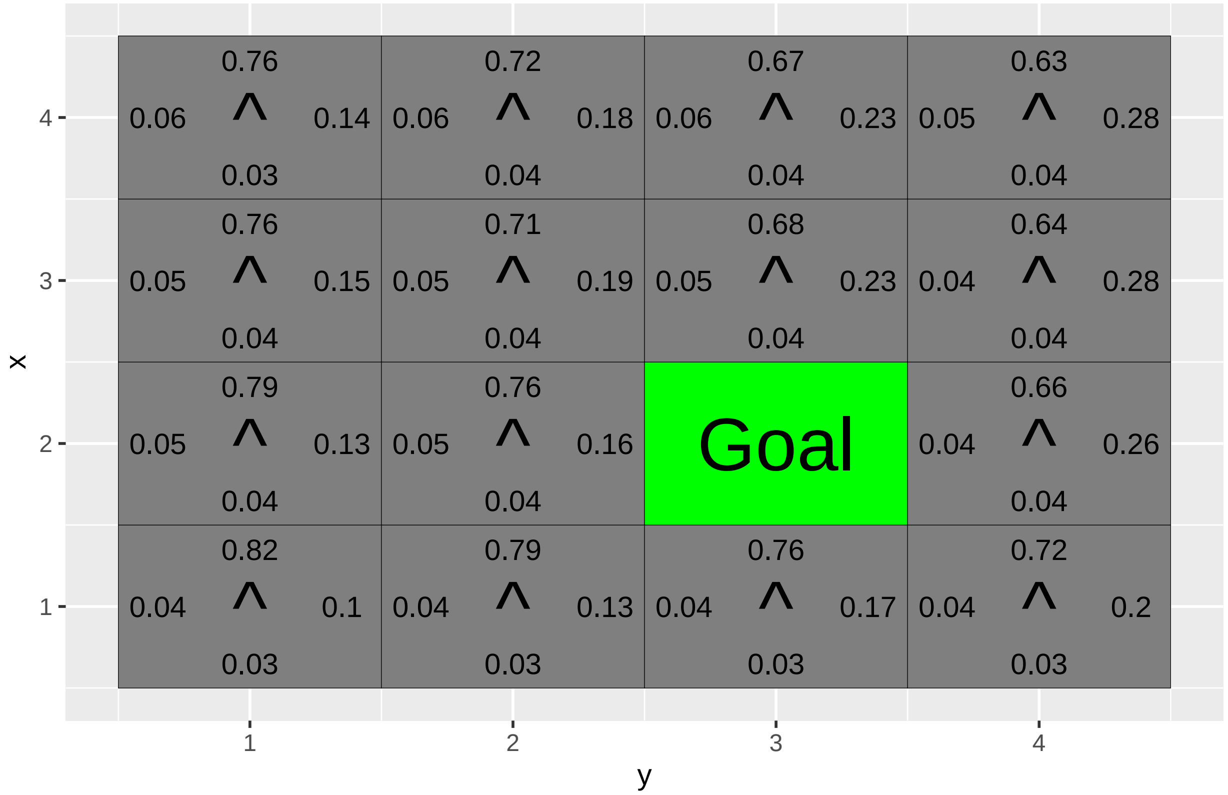
5. Enviroment D


Action probabilities after 0 episodes

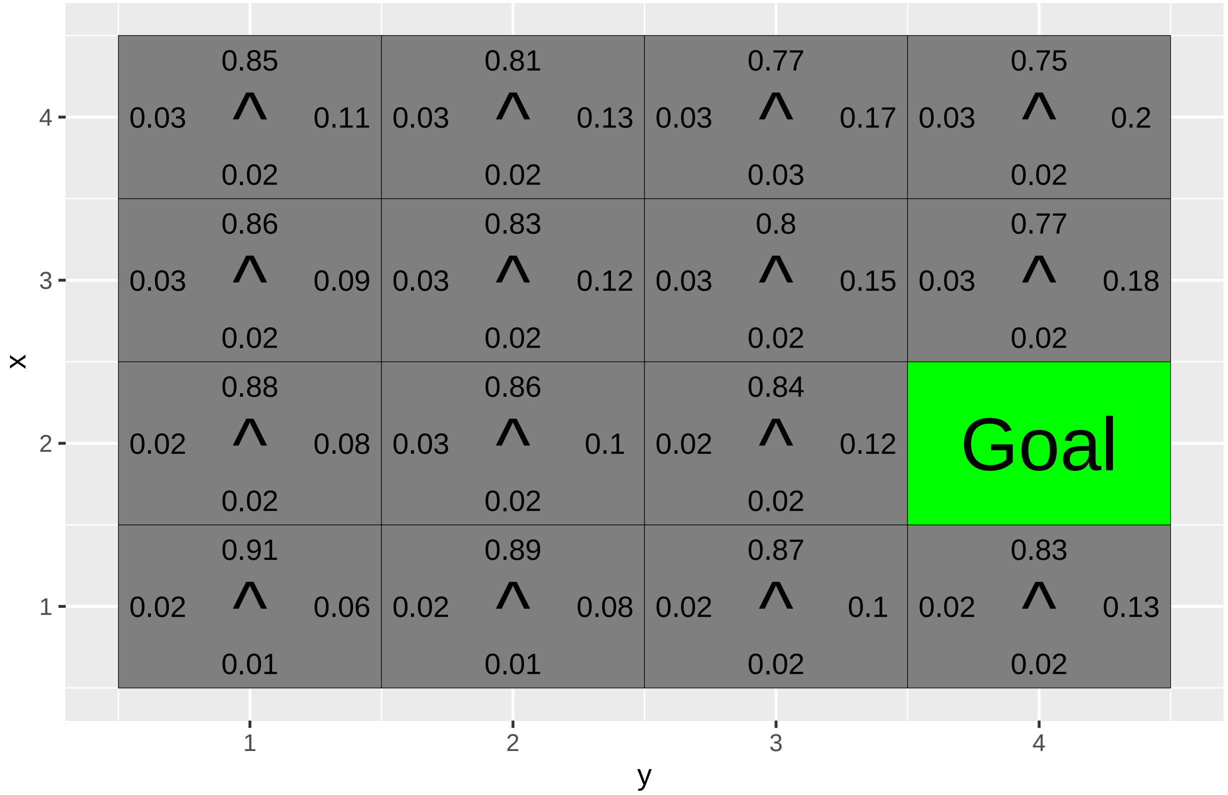

Action probabilities after 0 episodes


Action probabilities after 0 episodes
0.69 0.64 0.57 0.51
4 0.09 ^ 0.19 0.08 ^ 0.24 0.07 ^ 0.31 0.06 ^ 0.38
0.04 0.05 0.05 0.05
0.68 0.58 0.54
3 0.08 ^ 0.2 Goal 0.06 ^ 0.3 0.05 ^ 0.36
0.04 0.05 0.05
0.72 0.69 0.64 0.6
2 0.07 ^ 0.16 0.06 ^ 0.21 0.05 ^ 0.26 0.05 ^ 0.31
0.04 0.05 0.05 0.05
0.77 0.74 0.69 0.69
1 0.06 ^ 0.13 0.05 ^ 0.16 0.05 ^ 0.22 0.05 ^ 0.23
0.04 0.04 0.04 0.04
x
1 2 3 4
y

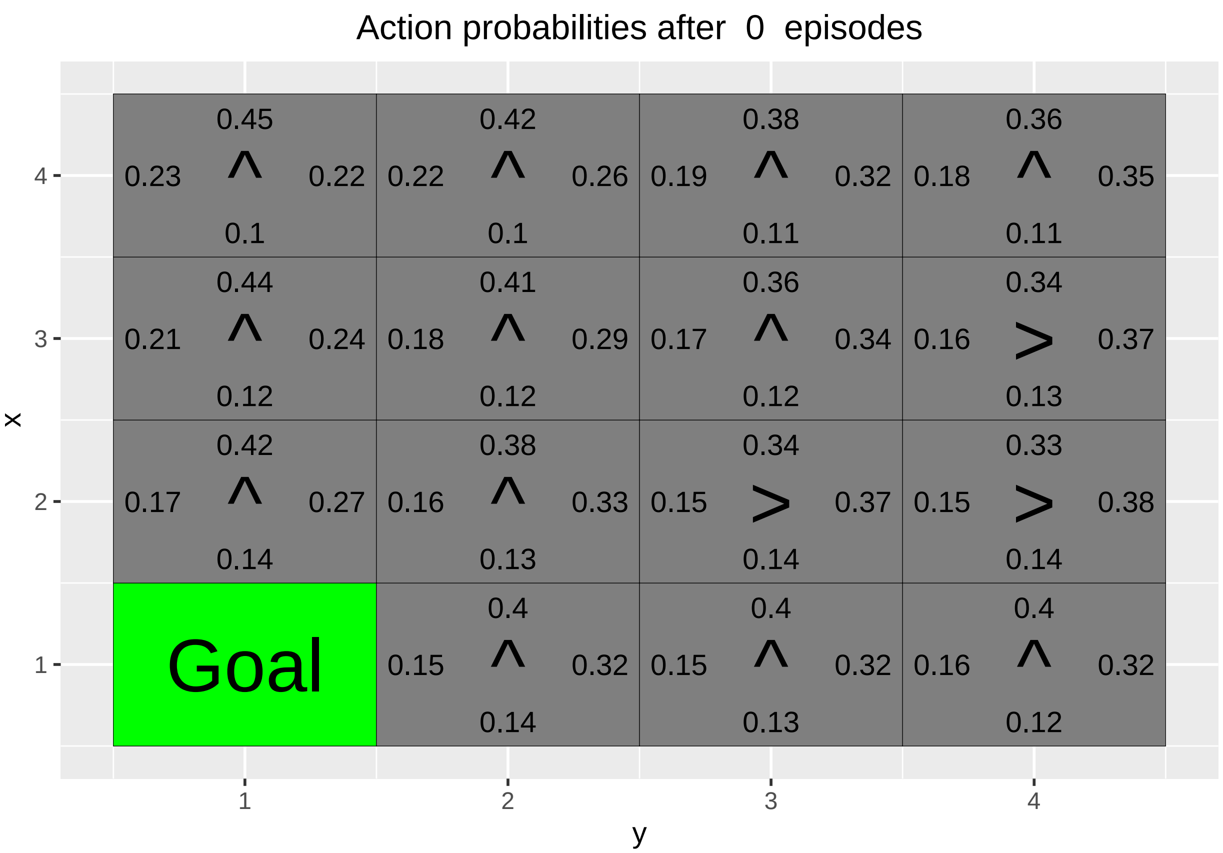

Action probabilities after 0 episodes
0.79 0.75 0.7 0.66
4 0.05 ^ 0.14 0.04 ^ 0.18 0.04 ^ 0.23 0.04 ^ 0.27
0.02 0.03 0.03 0.03
0.81 0.78 0.7
3 0.04 ^ 0.13 0.04 ^ 0.16 Goal 0.03 ^ 0.24
0.02 0.03 0.03
0.84 0.82 0.78 0.73
2 0.04 ^ 0.1 0.03 ^ 0.13 0.03 ^ 0.16 0.03 ^ 0.21
0.02 0.02 0.03 0.03
0.87 0.84 0.82 0.79
1 0.03 ^ 0.08 0.03 ^ 0.1 0.03 ^ 0.13 0.03 ^ 0.17
0.02 0.02 0.02 0.02
x
1 2 3 4
y

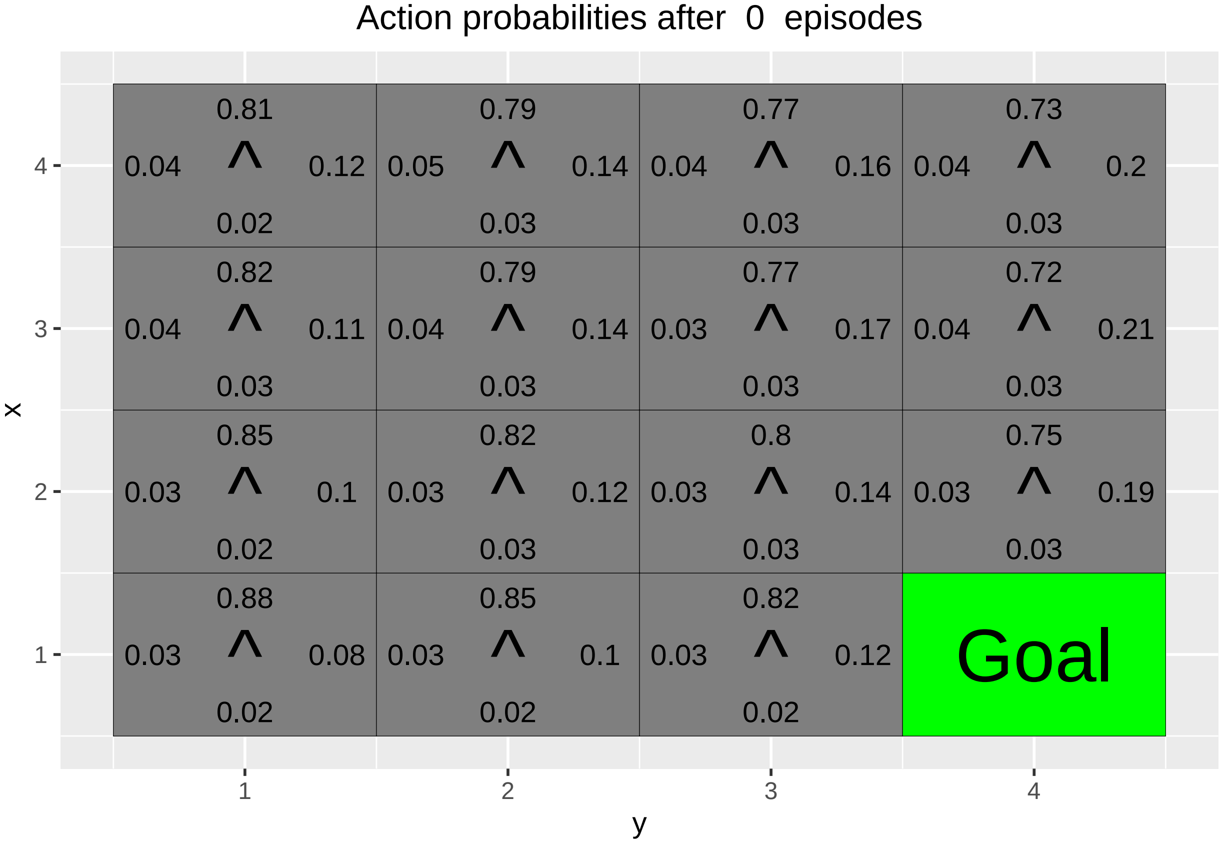


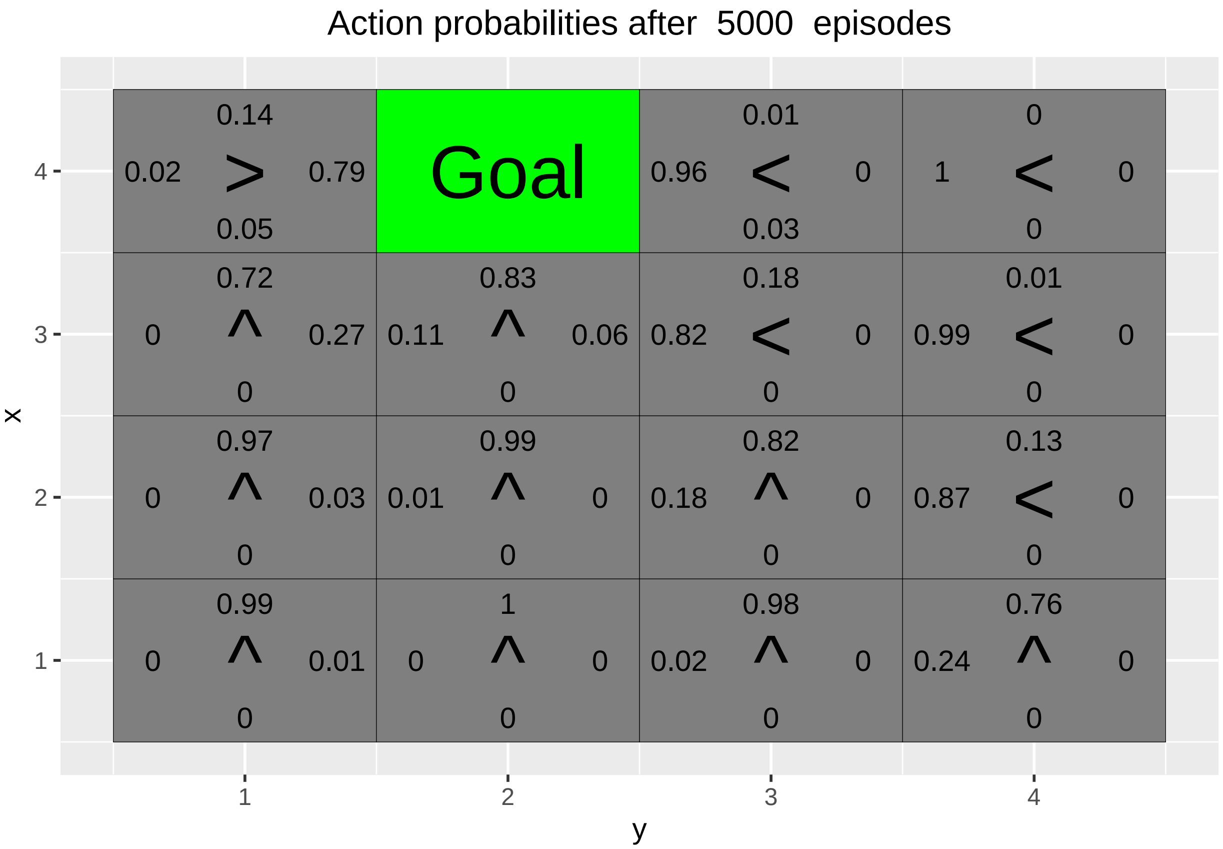
Action probabilities after 0 episodes

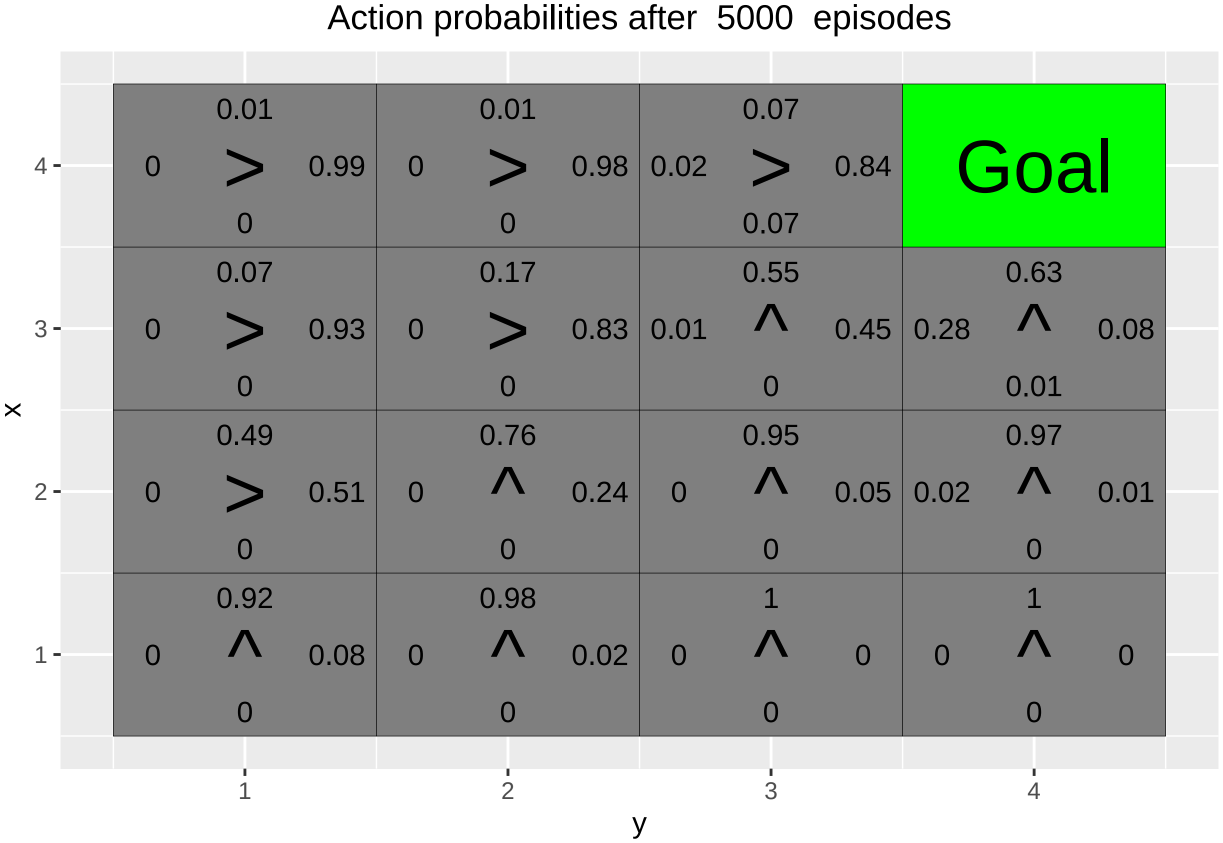


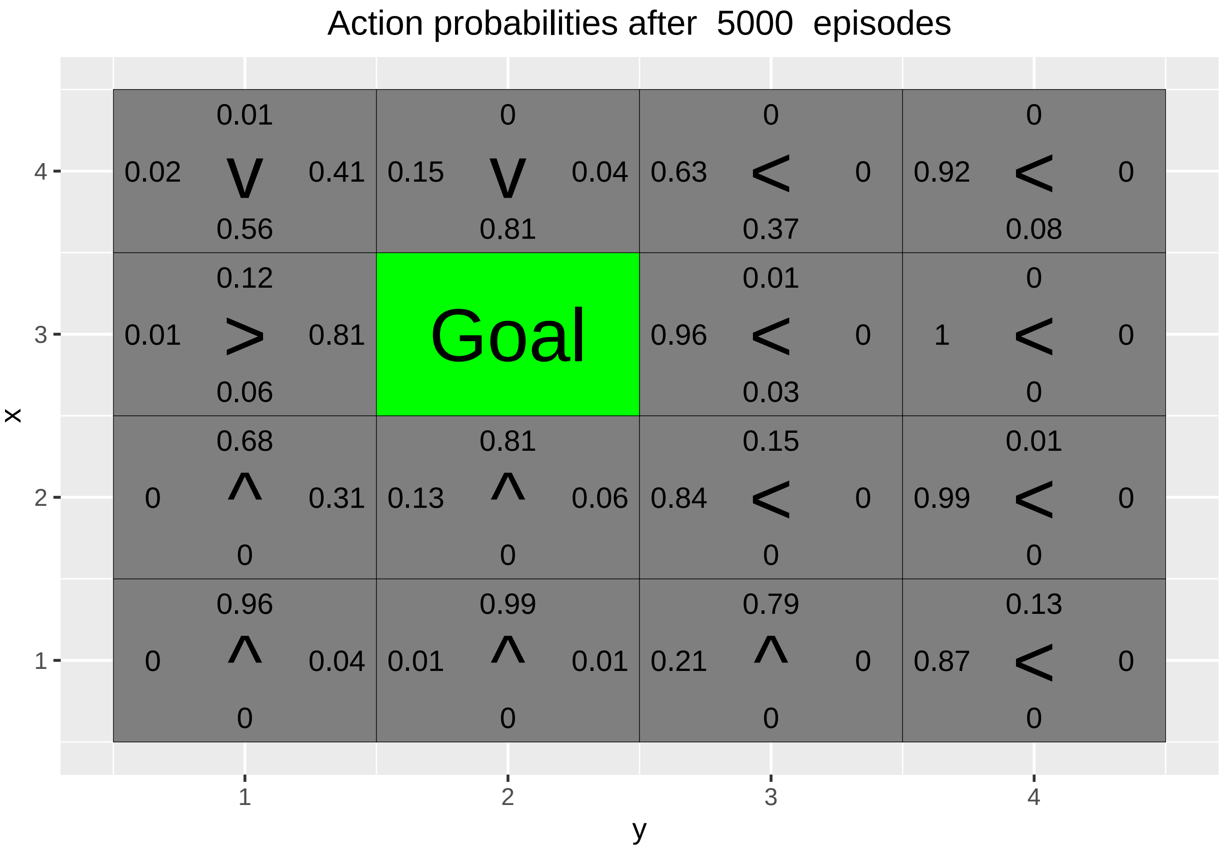
Action probabilities after 0 episodes

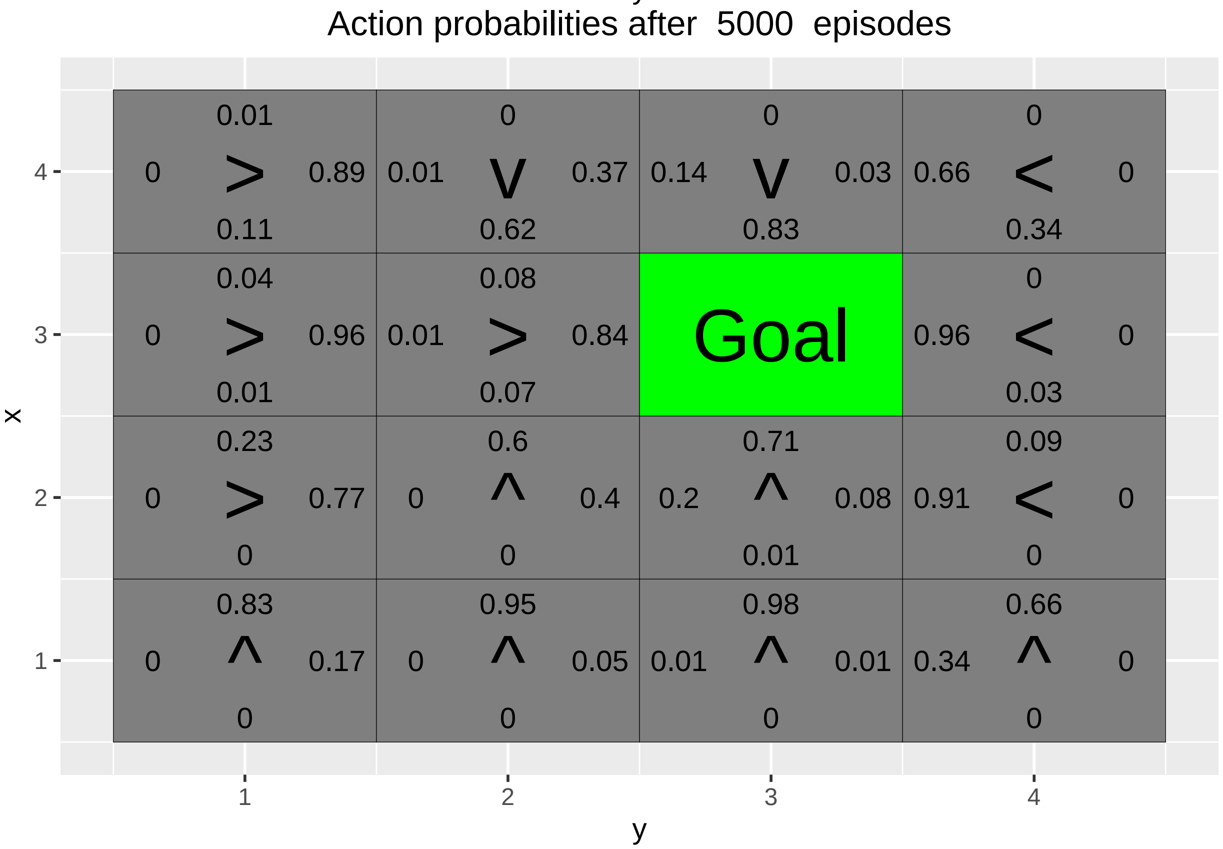

Action probabilities after 0 episodes
0.45
0.23 ^ 0.22
0.1
0.42
0.22 ^ 0.26
0.1
0.38
0.19 ^ 0.32
0.11
0.36
0.18 ^ 0.35
0.11
0.44
0.21 ^ 0.24
0.12
0.41
0.18 ^ 0.29
0.12
0.36
0.17 ^ 0.34
0.12
0.34
0.16 > 0.37
0.13
0.42
0.17 ^ 0.27
0.14
0.38
0.16 ^ 0.33
0.13
0.34
0.15 > 0.37
0.14
0.33
0.15 > 0.38
0.14
Goal
0.4
0.15 ^ 0.32
0.14
0.4
0.15 ^ 0.32
0.13
0.4
0.16 ^ 0.32
0.12
x
4
3
2
1
y
1
2
3
4


Action probabilities after 0 episodes
0.81
0.04 ^ 0.12
0.02
0.79
0.05 ^ 0.14
0.03
0.77
0.04 ^ 0.16
0.03
0.73
0.04 ^ 0.2
0.03
0.82
0.04 ^ 0.11
0.03
0.79
0.04 ^ 0.14
0.03
0.77
0.03 ^ 0.17
0.03
0.72
0.04 ^ 0.21
0.03
0.85
0.03 ^ 0.1
0.02
0.82
0.03 ^ 0.12
0.03
0.8
0.03 ^ 0.14
0.03
0.75
0.03 ^ 0.19
0.03
0.88
0.03 ^ 0.08
0.02
0.85
0.03 ^ 0.1
0.02
0.82
0.03 ^ 0.12
0.02
Goal
x
4
3
2
1
y
1
2
3
4

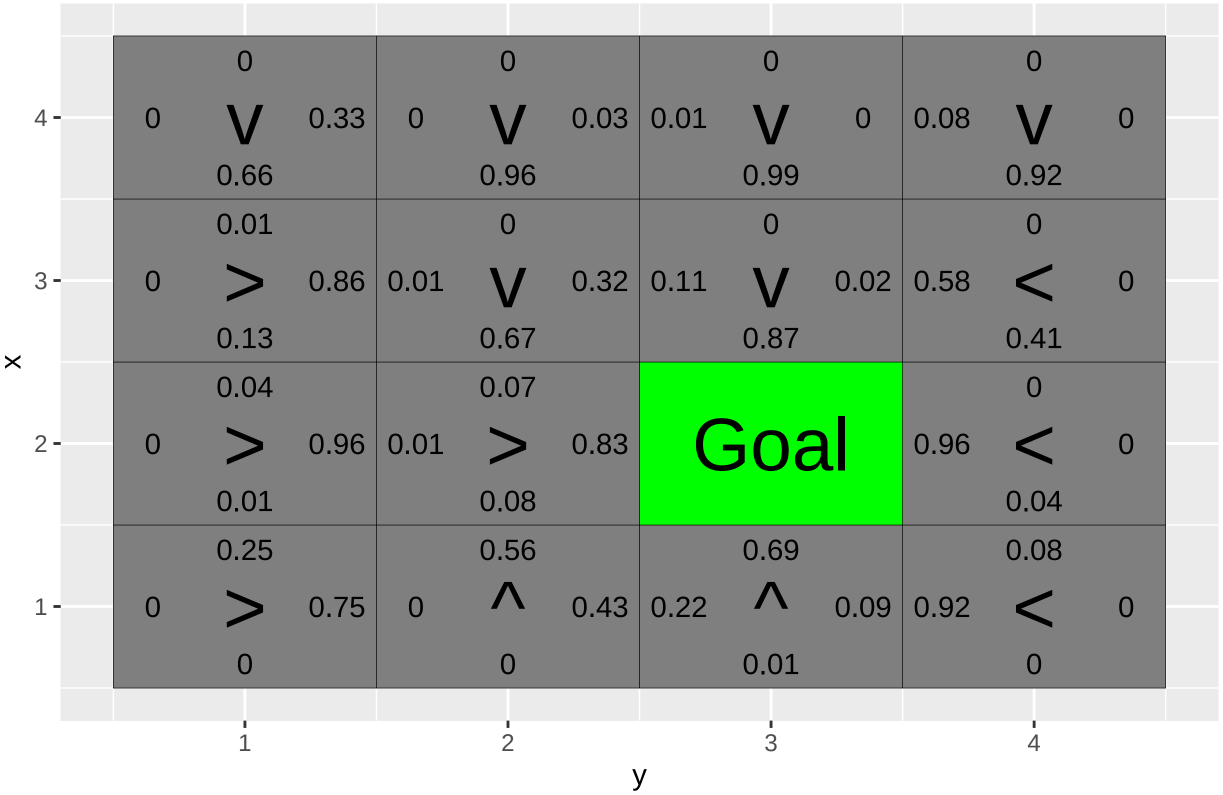

Action probabilities after 5000 episodes
0.14
0.02 > 0.79
0.05
Goal
0.01
0.96 < 0
0.03
0
1 < 0
0
0.72
0 ^ 0.27
0
0.83
0.11 ^ 0.06
0
0.18
0.82 < 0
0
0.01
0.99 < 0
0
0.97
0 ^ 0.03
0
0.99
0.01 ^ 0
0
0.82
0.18 ^ 0
0
0.13
0.87 < 0
0
0.99
0 ^ 0.01
0
1
0 ^ 0
0
0.98
0.02 ^ 0
0
0.76
0.24 ^ 0
0
4
3
2
1
x
1
2
3
4
y


Action probabilities after 5000 episodes
0.01
0 > 0.99
0
0.01
0 > 0.98
0
0.07
0.02 > 0.84
0.07
Goal
0.07
0 > 0.93
0
0.17
0 > 0.83
0
0.55
0.01 ^ 0.45
0
0.63
0.28 ^ 0.08
0.01
0.49
0 > 0.51
0
0.76
0 ^ 0.24
0
0.95
0 ^ 0.05
0
0.97
0.02 ^ 0.01
0
0.92
0 ^ 0.08
0
0.98
0 ^ 0.02
0
1
0 ^ 0
0
1
0 ^ 0
0
4
3
2
1
x
1
2
3
4
y

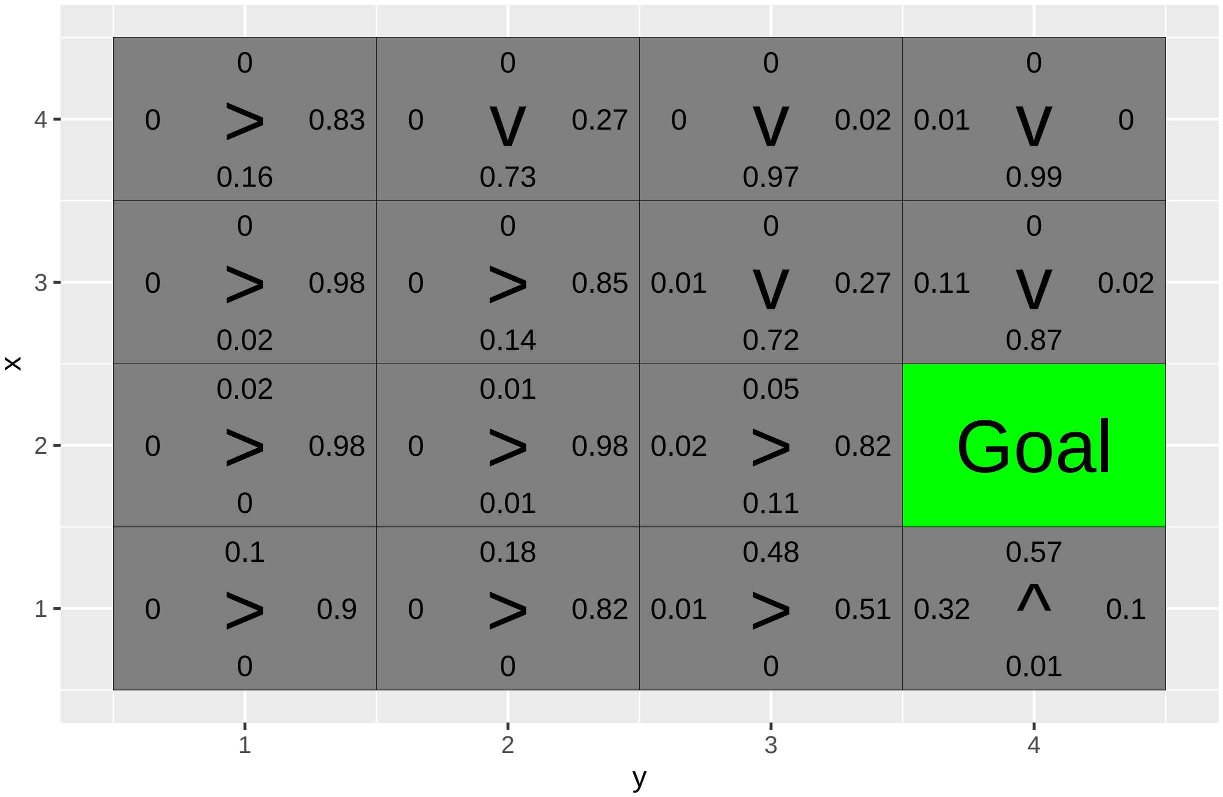

Action probabilities after 5000 episodes
0.01 0 0 0
4 0.02 V 0.41 0.15 V 0.04 0.63 < 0 0.92 < 0
0.56 0.81 0.37 0.08
0.12 0.01 0
3 0.01 > 0.81 Goal 0.96 < 0 1 < 0
0.06 0.03 0
0.68 0.81 0.15 0.01
2 0 ^ 0.31 0.13 ^ 0.06 0.84 < 0 0.99 < 0
0 0 0 0
0.96 0.99 0.79 0.13
1 0 ^ 0.04 0.01 ^ 0.01 0.21 ^ 0 0.87 < 0
0 0 0 0
x
1 2 3 4
y

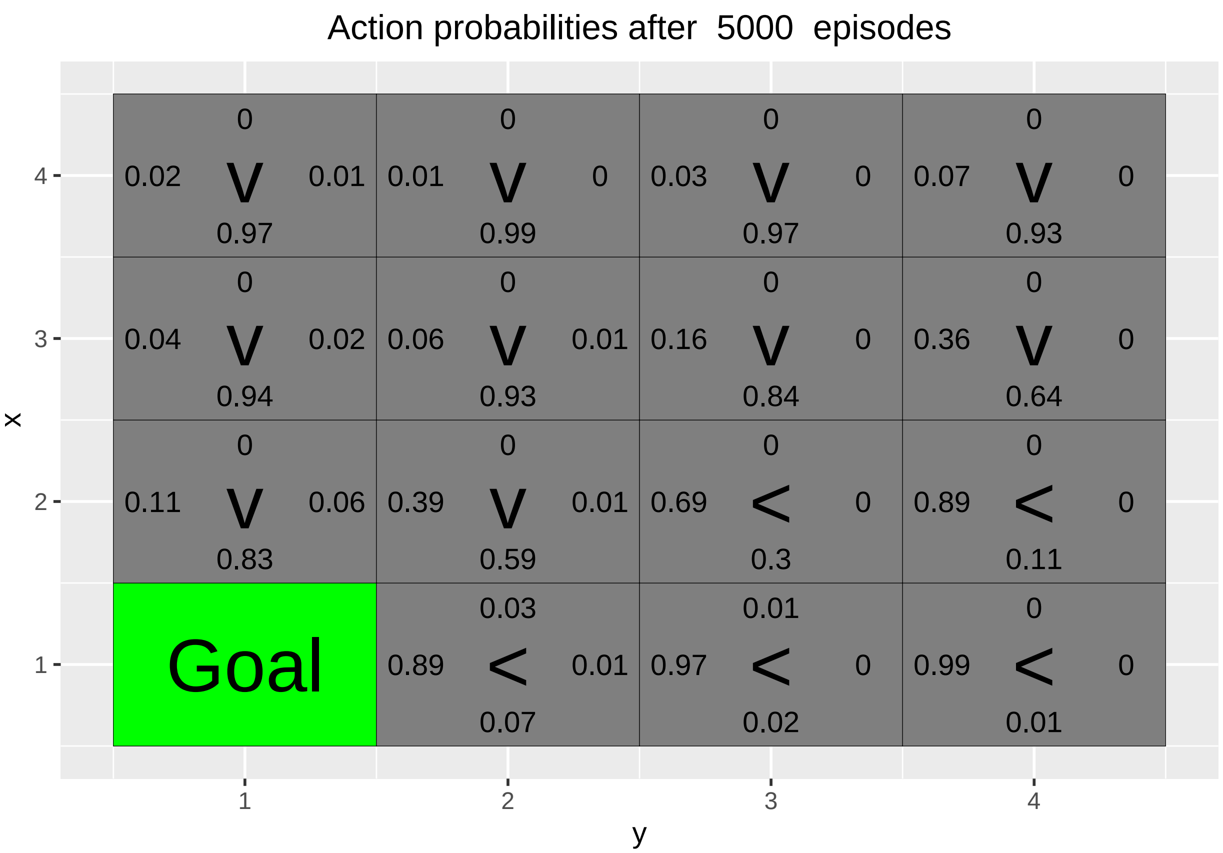

Action probabilities after 5000 episodes
0.01 0 0 0
4 0 > 0.89 0.01 V 0.37 0.14 V 0.03 0.66 < 0
0.11 0.62 0.83 0.34
0.04 0.08 0
3 0 > 0.96 0.01 > 0.84 Goal 0.96 < 0
0.01 0.07 0.03
0.23 0.6 0.71 0.09
2 0 > 0.77 0 ^ 0.4 0.2 ^ 0.08 0.91 < 0
0 0 0.01 0
0.83 0.95 0.98 0.66
1 0 ^ 0.17 0 ^ 0.05 0.01 ^ 0.01 0.34 ^ 0
0 0 0 0
x
1 2 3 4
y

Action probabilities after 5000 episodes

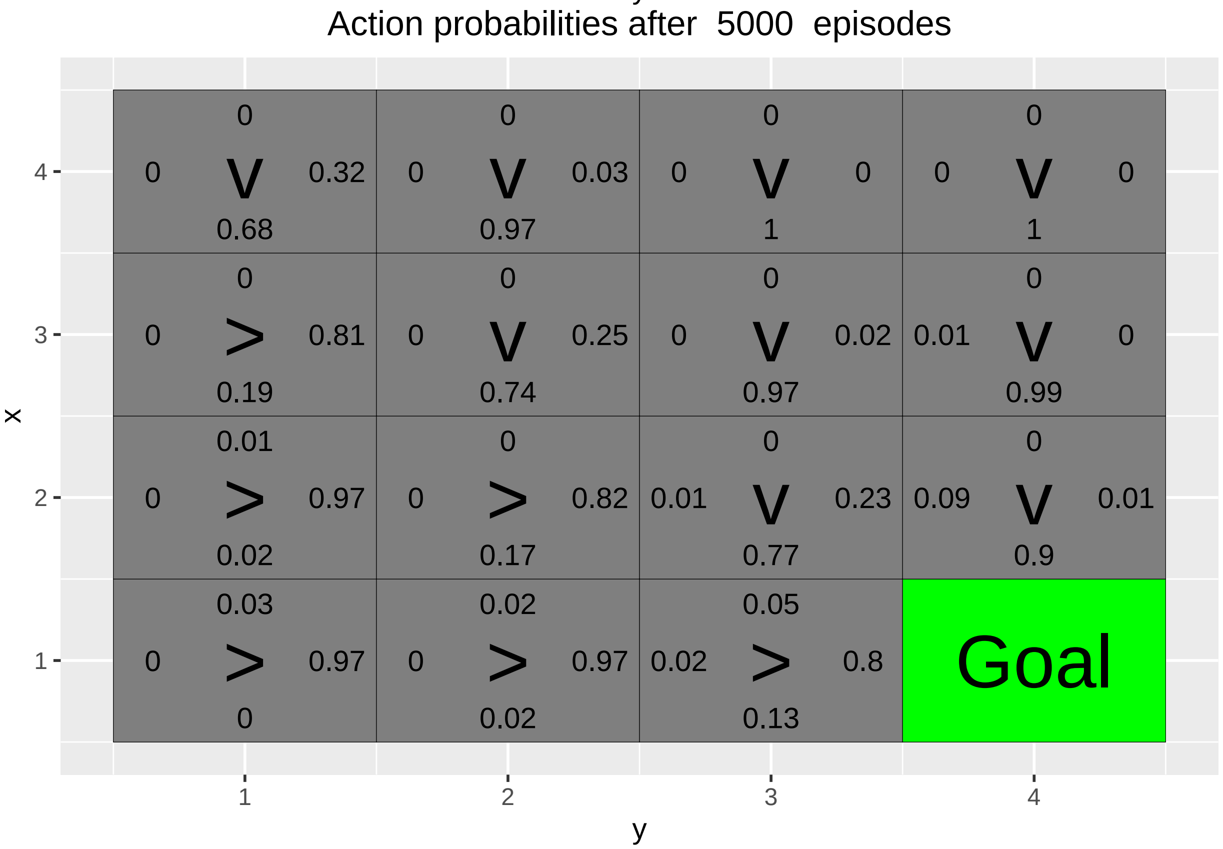


Action probabilities after 5000 episodes





Action probabilities after 5000 episodes



Action probabilities after 5000 episodes

Comment

(1) Has the agent learned a good policy? Why / Why not ?

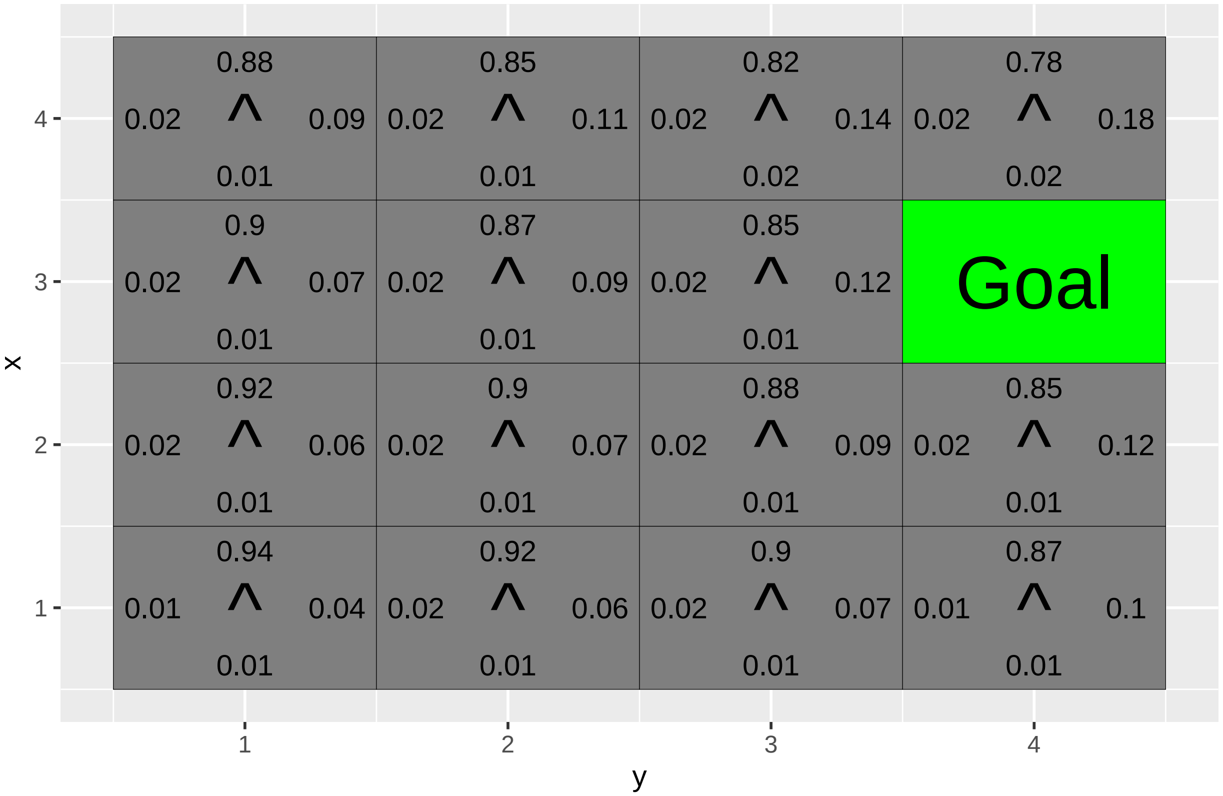
The agent has learned a good policy. In each validation goal's graph, all the states have the optimal path to the goal position.

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

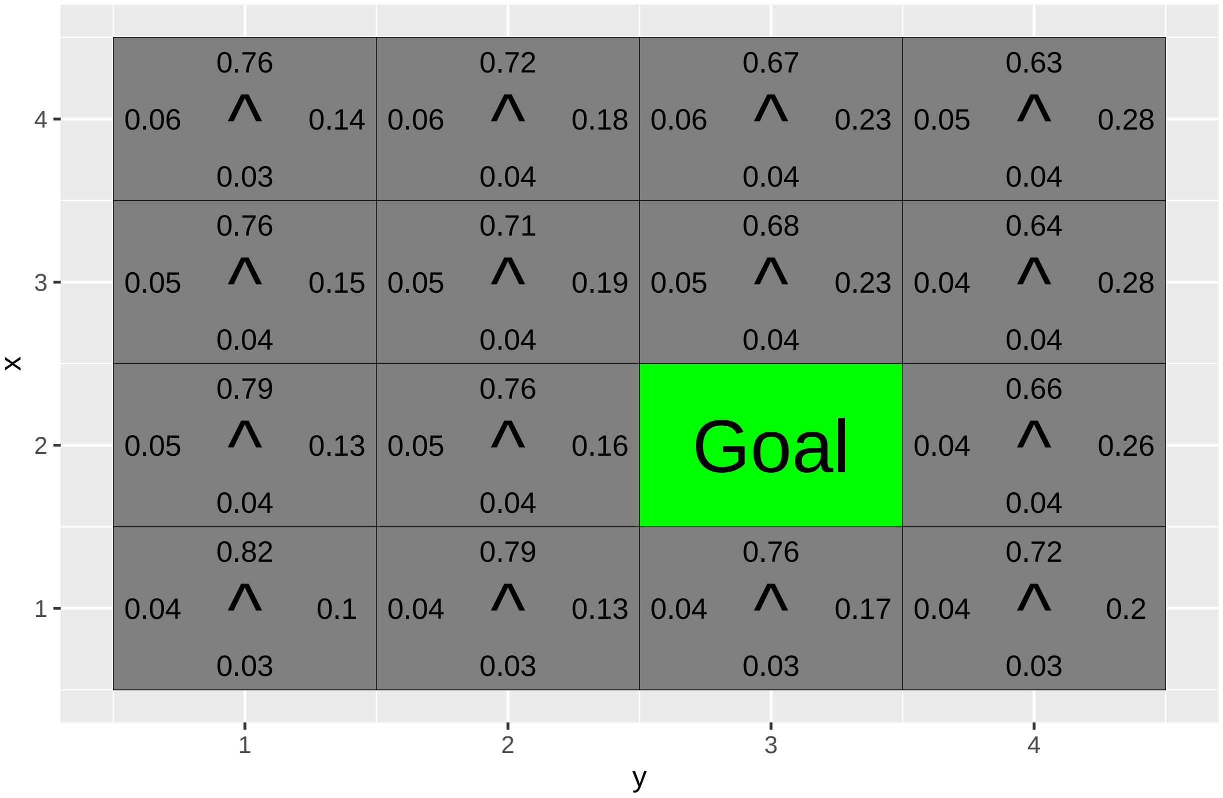
The Q-learning algorithm can not solve this task. It can learn the optimal policy for fixed initial and goal position. The Q-table is also fixed. This policy can not match for another different goal position which also means that Q-learning does not have generalization.

6. Environment E

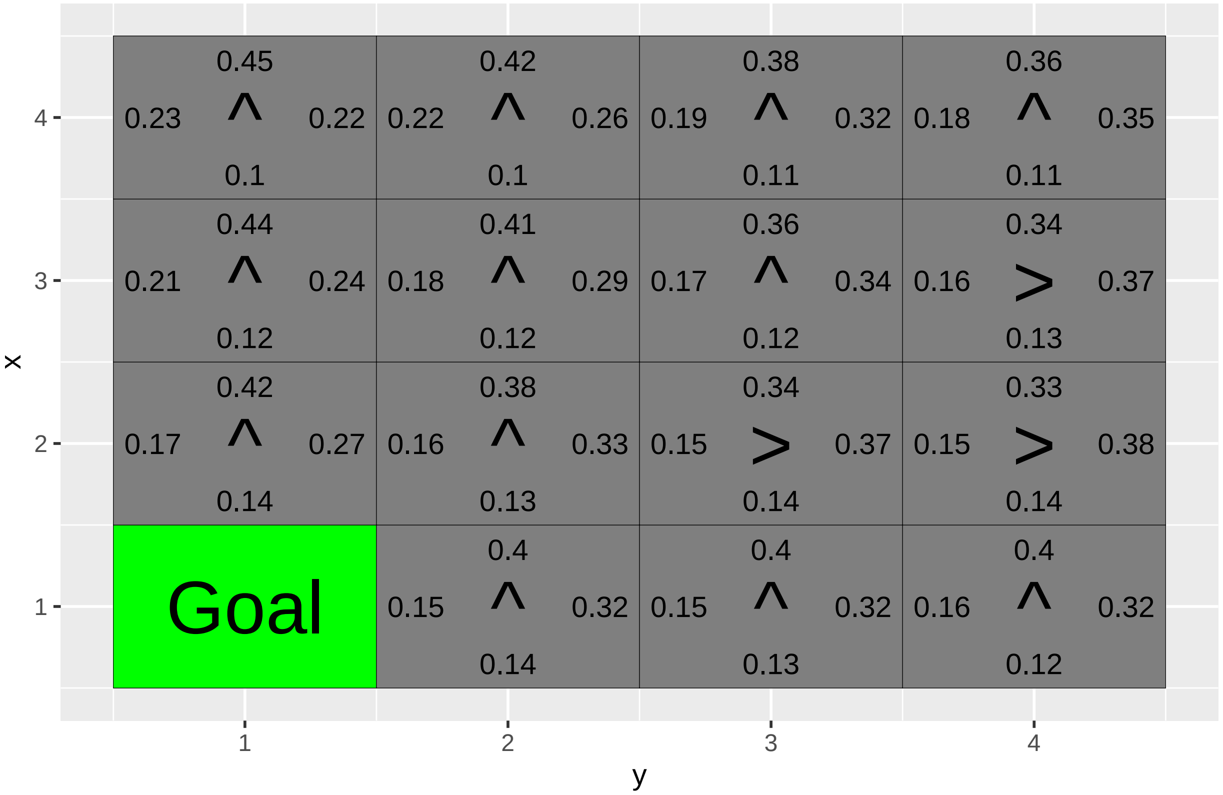
Action probabilities after 0 episodes



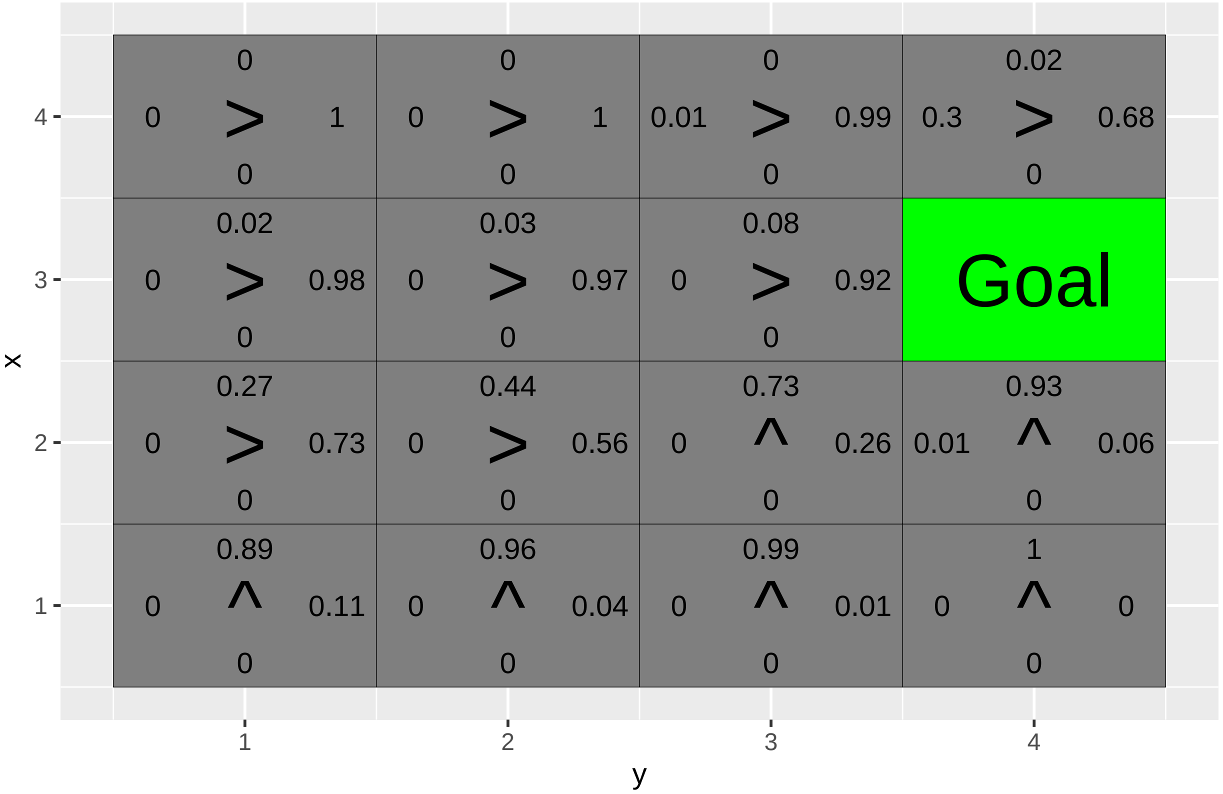
Action probabilities after 0 episodes

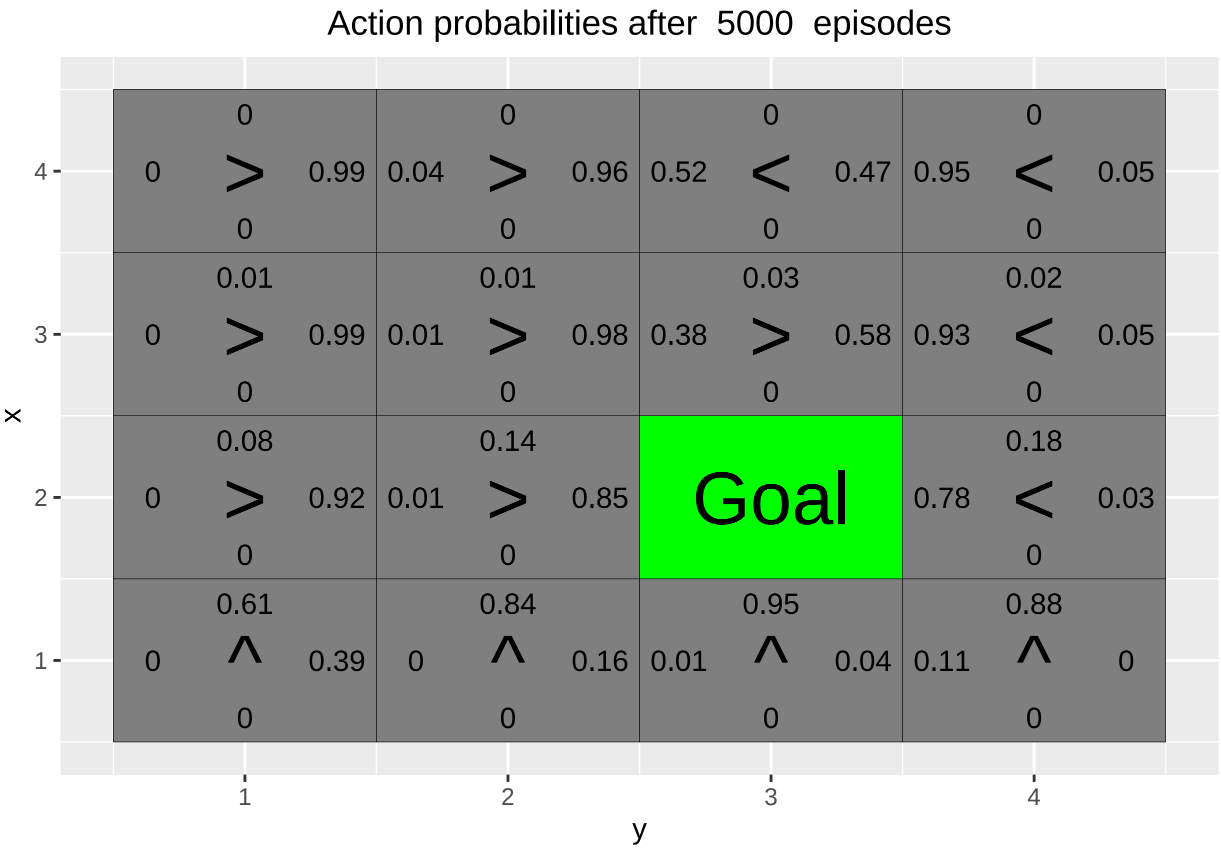


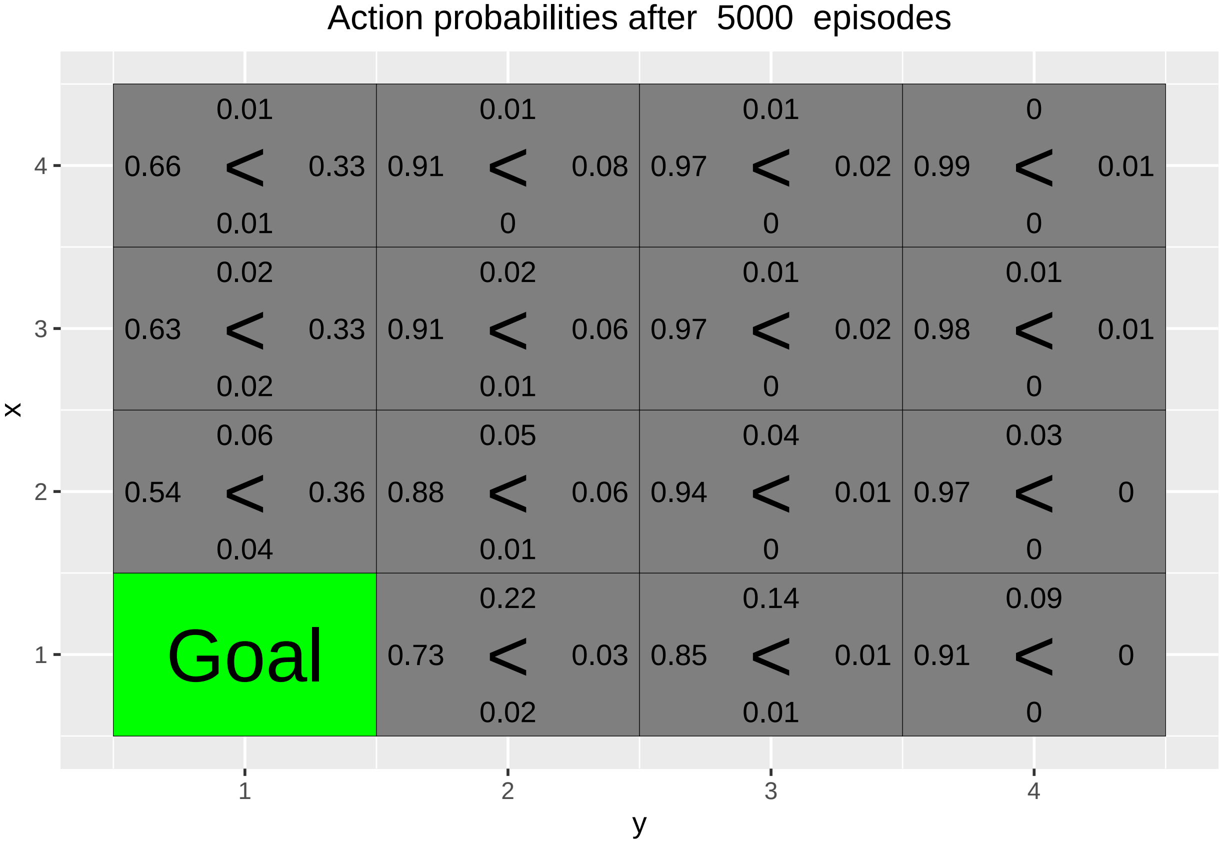
Action probabilities after 0 episodes



Action probabilities after 5000 episodes




Action probabilities after 5000 episodes
0 0 0 0
4 0 > 0.99 0.04 > 0.96 0.52 < 0.47 0.95 < 0.05
0 0 0 0
0.01 0.01 0.03 0.02
3 0 > 0.99 0.01 > 0.98 0.38 > 0.58 0.93 < 0.05
0 0 0 0
0.08 0.14 0.18
2 0 > 0.92 0.01 > 0.85 Goal 0.78 < 0.03
0 0 0
0.61 0.84 0.95 0.88
1 0 ^ 0.39 0 ^ 0.16 0.01 ^ 0.04 0.11 ^ 0
0 0 0 0
x
1 2 3 4
y


Action probabilities after 5000 episodes
0.01 0.01 0.01 0
4 0.66 < 0.33 0.91 < 0.08 0.97 < 0.02 0.99 < 0.01
0.01 0 0 0
0.02 0.02 0.01 0.01
3 0.63 < 0.33 0.91 < 0.06 0.97 < 0.02 0.98 < 0.01
0.02 0.01 0 0
0.06 0.05 0.04 0.03
2 0.54 < 0.36 0.88 < 0.06 0.94 < 0.01 0.97 < 0
0.04 0.01 0 0
0.22 0.14 0.09
1 Goal 0.73 < 0.03 0.85 < 0.01 0.91 < 0
0.02 0.01 0
x
1 2 3 4
y

Comment

(1) Has the agent learned a good policy? Why / Why not ?

The agent has not learned a good policy. In each validation goal's graph, most of the states do not have optimal path to the goal position.

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

In environment D, the goal positions in training data contain all the rows and columns. The agent have access to learn the whole information of the grid. The distribution of the training and validation data are same. Hence, it has a better generalization for the whole grid.

In environment E, the goal positions in training data only contain the first row and all columns. The agent can not have access to learn the whole information of the grid except the first row. The distribution of the training and validation data are not same. Hence, it has a worse generalization for the whole grid.

Appendix

```r
knitr::opts_chunk$set(echo = TRUE)
library(vctrs)
library(ggplot2)
library(tensorflow)
library(keras)
# Implement Greedy algorithm
GreedyPolicy <- function(x, y){
  max_q <- max(q_table[x, y, ])
  index <- which(q_table[x, y, ] == max_q)
  a <- ifelse(length(index) > 1, sample(index, 1), index)
  return(a)
}

# Implement Epsilon Greedy algorithm
EpsilonGreedyPolicy <- function(x, y, epsilon){
  a <- ifelse(runif(1) >= epsilon, GreedyPolicy(x, y), sample(1 : 4, 1))
  return(a)
}

# Implement Q-Learning algorithm
q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95,
                       beta = 0){
  p <- start_state
  episode_correction <- 0
  repeat{
    # Follow policy, execute action, get reward.
    a_1 <- EpsilonGreedyPolicy(p[1], p[2], epsilon)
    p1 <- transition_model(p[1], p[2], a_1, beta)
    reward <- reward_map[p1[1], p1[2]]
    # Q-table update.
    temporal_diff <- reward + gamma * max(q_table[p1[1], p1[2], ]) - q_table[p[1], p[2], a_1]
    q_table[p[1], p[2], a_1] <<- q_table[p[1], p[2], a_1] + alpha * temporal_diff
    p <- p1
    episode_correction <- episode_correction + temporal_diff
    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))
  }
}
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){
  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){
  max_q <- max(q_table[x, y, ])
  index <- which(q_table[x, y, ] == max_q)
  a <- ifelse(length(index) > 1, sample(index, 1), index)
  return(a)
}

EpsilonGreedyPolicy <- function(x, y, epsilon){
  a <- ifelse(runif(1) >= epsilon, GreedyPolicy(x, y), sample(1 : 4, 1))
  return(a)
}
```

```
transition_model <- function(x, y, action, beta){
  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){
  p <- start_state
  episode_correction <- 0
  repeat{
    # Follow policy, execute action, get reward.
    a_1 <- EpsilonGreedyPolicy(p[1], p[2], epsilon)
    p1 <- transition_model(p[1], p[2], a_1, beta)
    reward <- reward_map[p1[1], p1[2]]
    # Q-table update.
    temporal_diff <- reward + gamma * max(q_table[p1[1], p1[2], ]) - q_table[p[1], p[2], a_1]
    q_table[p[1], p[2], a_1] <<- q_table[p[1], p[2], a_1] + alpha * temporal_diff
    p = p1
    episode_correction = episode_correction + temporal_diff
    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))
  }

}
set.seed(1)
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)
}

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

  if(any(i==c(10000)))
    vis_environment(i)
}
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")
}

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")
}
```

```r
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)
}

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

vis_prob <- function(goal, episodes = 0){
  df <- expand.grid(x=1:H,y=1:W)
  dist <- array(data = NA, dim = c(H,W,4))
  class <- array(data = NA, dim = c(H,W))
  for(i in 1:H)
    for(j in 1:W){
      dist[i,j,] <- DeepPolicy_dist(i,j,goal[1],goal[2])
      foo <- which(dist[i,j,]==max(dist[i,j,]))
      class[i,j] <- ifelse(length(foo)>1,sample(foo, size = 1),foo)
    }

  foo <- mapply(function(x,y) ifelse(all(c(x,y) == goal),NA,dist[x,y,1]),df$x,df$y)
  df$val1 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(all(c(x,y) == goal),NA,dist[x,y,2]),df$x,df$y)
  df$val2 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(all(c(x,y) == goal),NA,dist[x,y,3]),df$x,df$y)
  df$val3 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(all(c(x,y) == goal),NA,dist[x,y,4]),df$x,df$y)
  df$val4 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(all(c(x,y) == goal),NA,class[x,y]),df$x,df$y)
  df$val5 <- as.vector(arrows[foo])
  foo <- mapply(function(x,y) ifelse(all(c(x,y) == goal),"Goal",NA),df$x,df$y)
  df$val6 <- as.vector(foo)

  print(ggplot(df,aes(x = y,y = x)) +
          geom_tile(fill = 'white', colour = 'black') +
          scale_fill_manual(values = c('green')) +
          geom_tile(aes(fill=val6), show.legend = FALSE, colour = 'black') +
          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,na.rm = TRUE) +
          geom_text(aes(label = val6),size = 10,na.rm = TRUE) +
          ggtitle(paste("Action probabilities after ",episodes," episodes")) +
          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)))

}

transition_model <- function(x, y, action, beta){
  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)
}

DeepPolicy_dist <- function(x, y, goal_x, goal_y){
  foo <- matrix(data = c(x,y,goal_x,goal_y), nrow = 1)
  return (predict_on_batch(model, x = foo)) # Faster.

}

DeepPolicy <- function(x, y, goal_x, goal_y){
  foo <- DeepPolicy_dist(x,y,goal_x,goal_y)
  return (sample(1:4, size = 1, prob = foo))

}

DeepPolicy_train <- function(states, actions, goal, gamma){

  inputs <- matrix(data = states, ncol = 2, byrow = TRUE)
  inputs <- cbind(inputs,rep(goal[1],nrow(inputs)))
  inputs <- cbind(inputs,rep(goal[2],nrow(inputs)))

  targets <- array(data = actions, dim = nrow(inputs))
  targets <- to_categorical(targets-1, num_classes = 4)

  # Sample weights. Reward of 5 for reaching the goal.
  weights <- array(data = 5*(gamma^(nrow(inputs)-1)), dim = nrow(inputs))

  # Train on batch. Note that this runs a SINGLE gradient update.
  train_on_batch(model, x = inputs, y = targets, sample_weight = weights)

}

reinforce_episode <- function(goal, gamma = 0.95, beta = 0){
  cur_pos <- goal
  while(all(cur_pos == goal))
    cur_pos <- c(sample(1:H, size = 1),sample(1:W, size = 1))

  states <- NULL
  actions <- NULL

  steps <- 0 # To avoid getting stuck and/or training on unnecessarily long episodes.
  while(steps < 20){
    steps <- steps+1

    # Follow policy and execute action.
    action <- DeepPolicy(cur_pos[1], cur_pos[2], goal[1], goal[2])
    new_pos <- transition_model(cur_pos[1], cur_pos[2], action, beta)

    # Store states and actions.
    states <- c(states,cur_pos)
    actions <- c(actions,action)
    cur_pos <- new_pos

    if(all(new_pos == goal)){
      # Train network.
      DeepPolicy_train(states,actions,goal,gamma)
      break
    }
  }

}

# 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)

# for(i in 1:5000){
#   # if(i%%10==0) cat("episode",i,"\n")
#   goal <- sample(train_goals, size = 1)
#   reinforce_episode(unlist(goal))
# }

fault <- lapply(1 : 5000, function(x) reinforce_episode(unlist(sample(train_goals, size = 1))))

show_validation(5000)

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)

# for(i in 1:5000){
#   # if(i%%10==0) cat("episode", i,"\n")
#   goal <- sample(train_goals, size = 1)
#   reinforce_episode(unlist(goal))
# }

fault <- lapply(1 : 5000, function(x) reinforce_episode(unlist(sample(train_goals, size = 1))))

show_validation(5000)
```