LAB 3: Reinforcement Learning

xiali125@student.liu.se linfr259@student.liu.se qinzh916@student.liu.se
huali824@student.liu.se qincu578@student.liu.se

2025-10-05

Complete the implementation of Q-Learning

1. `GreedyPolicy` Function

Its goal is to select the action `a` that maximizes the Q value for a given state `s`.

The policy $\pi(s)$ is defined as the action `a` that maximizes the Q value:

$$\pi(s) = \arg\max_{a \in A} Q(s, a)$$

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

  foo <- which(q_table[x,y,] == max(q_table[x,y,]))
  return (ifelse(length(foo)>1,sample(foo, size = 1),foo))

}
```

2. `EpsilonGreedyPolicy` Function

ϵ-Greedy "exploitation" (taking the best action) and "exploration" (trying random actions).

The action a_t chosen at time `t` :

$$a_t = \begin{cases} \text{random action from } A & \text{with prob } \epsilon \\ \arg\max_{a \in A} Q(s_t, a) & \text{with prob } 1 - \epsilon \end{cases}$$

- probability of exploring ϵ is between 0 and 1

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

  if (runif(1) < epsilon) {
    return(sample(1:4, 1))
  } else {
    return(GreedyPolicy(x, y))
  }
}
```

3 Transition Model Function

The agent intends to perform an action, but with probability β, it "slips" to the side.

Given the intended action $a_{intended}$, the probability of the actual action a_{actual} is:

$$P(a_{actual}|a_{intended}) = \begin{cases} 1-\beta & \text{no slip} \\ \beta/2 & \text{left of } a_{intended} \\ \beta/2 & \text{right of } a_{intended} \end{cases}$$

The next state `s'` is then determined by the actual action taken: `s'`.

```
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)
}
```

4 Q learning Function

Bellman equation update rule for Q Learning:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[\underbrace{r_{t+1} + \gamma \max_a Q(s_{t+1},a)}_{\text{TD Target}} - \underbrace{Q(s_t,a_t)}_{\text{Old Value}} \right]$$

```
q_learning <- function(...){
  current_state <- start_state
  correction <- 0

  repeat{
    # Follow policy, execute action, get reward.
    x <- current_state[1]
    y <- current_state[2]

    action <- EpsilonGreedyPolicy(x, y, epsilon)
    next_state <- transition_model(x, y, action, beta)
    reward <- reward_map[next_state[1], next_state[2]]

    # Q-table update.
    old_q_value <- q_table[x, y, action]

    if (reward != 0) {
      max_q_next = 0
    } else {
      max_q_next = max(q_table[next_state[1], next_state[2], ])
    }

    td_error <- reward + gamma * max_q_next - old_q_value
    q_table[x, y, action] <<- old_q_value + alpha * td_error

    correction <- correction + td_error

    current_state <- next_state

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

  }
}
```

Environment A

Grid: `5 x 7` grid

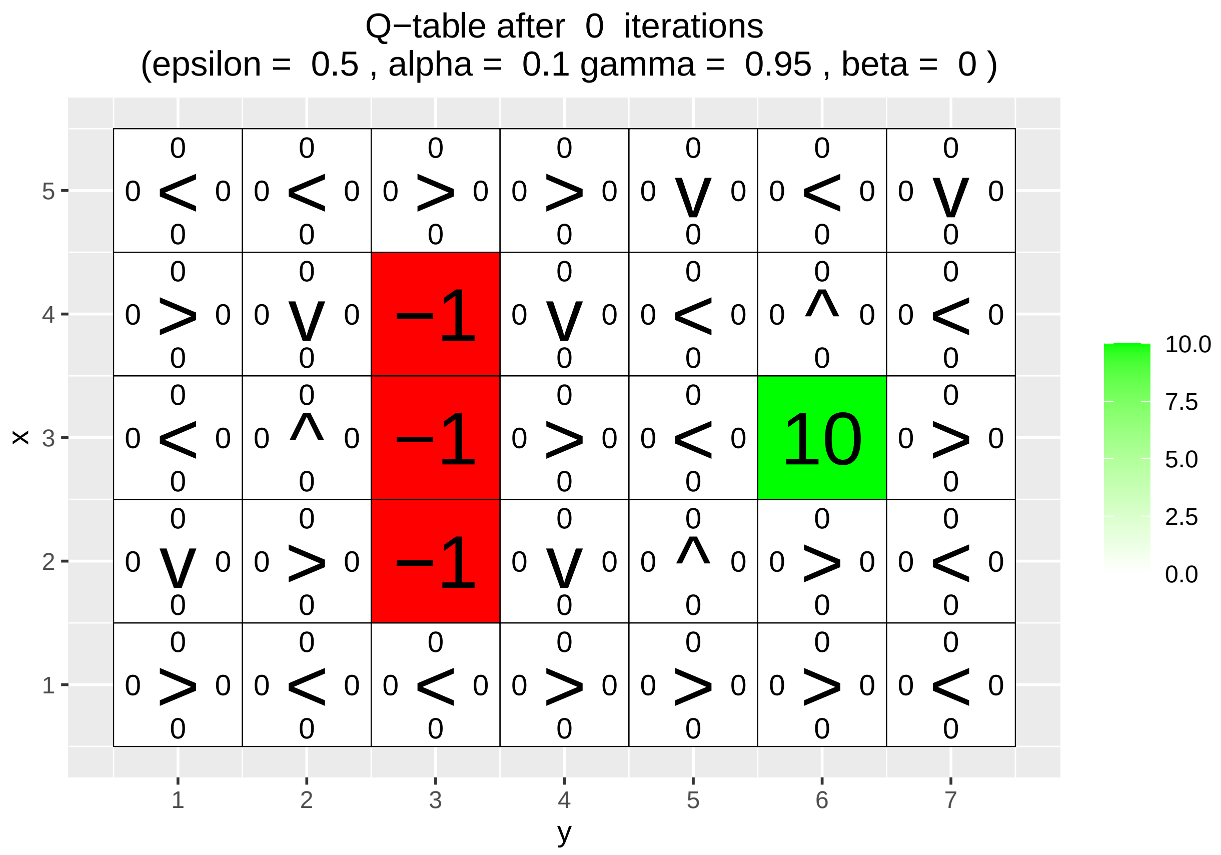
Reward: a reward of `10` at state (3,6), and a penalty of `-1` at states (2,3), (3,3), and (4,3).All other states are `0`.

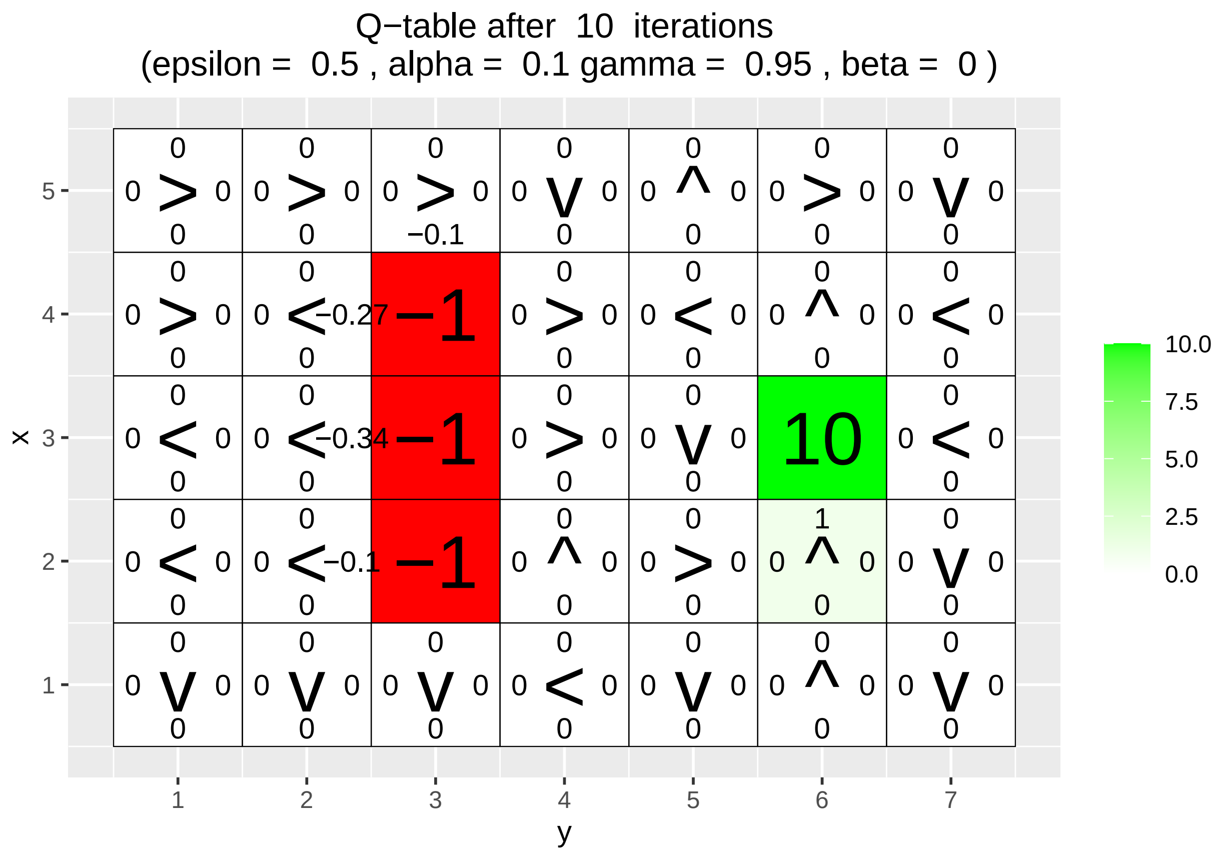
Start State: every episode, initial position is `(3, 1)`.

Q-table dimensions `5 x 7 x 4`.

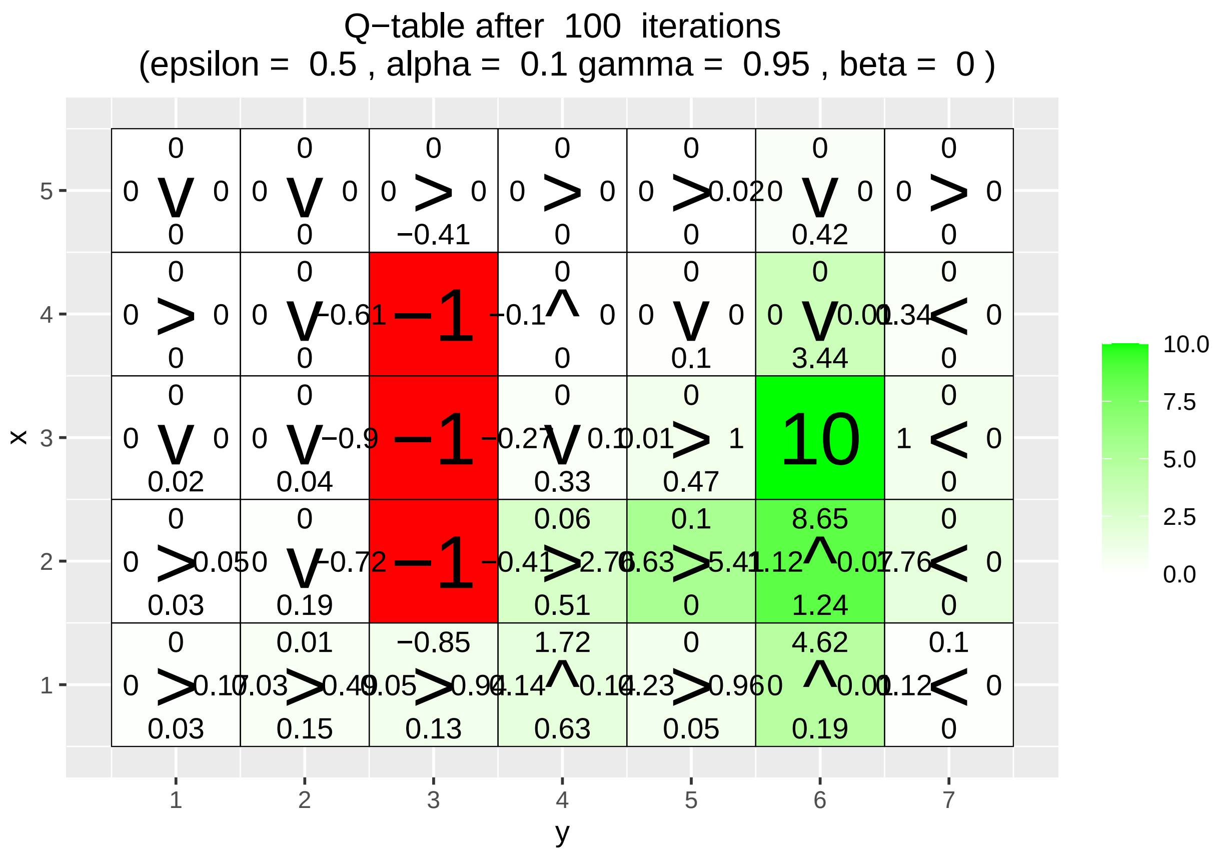
Initialization: The Q-table is initialized with all values set to `0`.

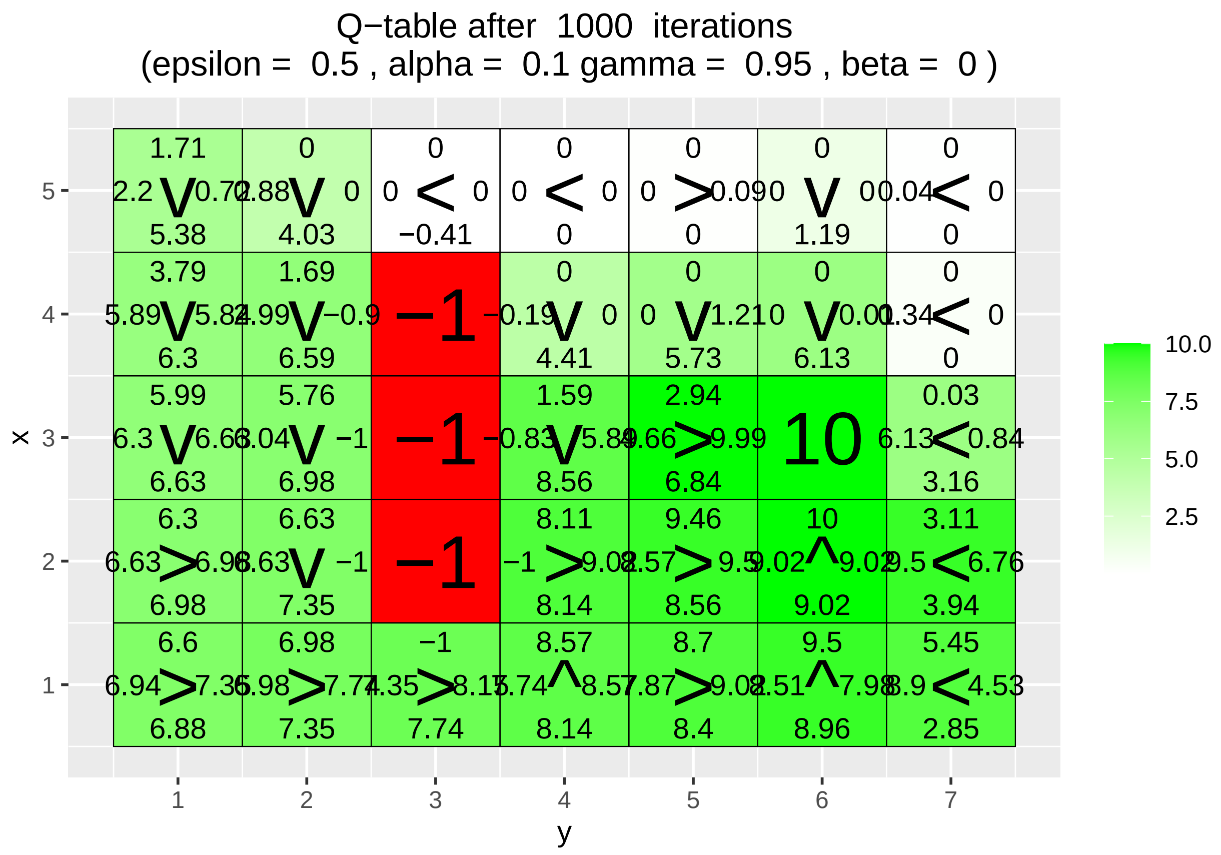
Parameters:The agent is trained over 10,000 episodes using the following: $\epsilon = 0.5$, $\beta = 0$, $\alpha = 0.1$, and $\gamma = 0.95$

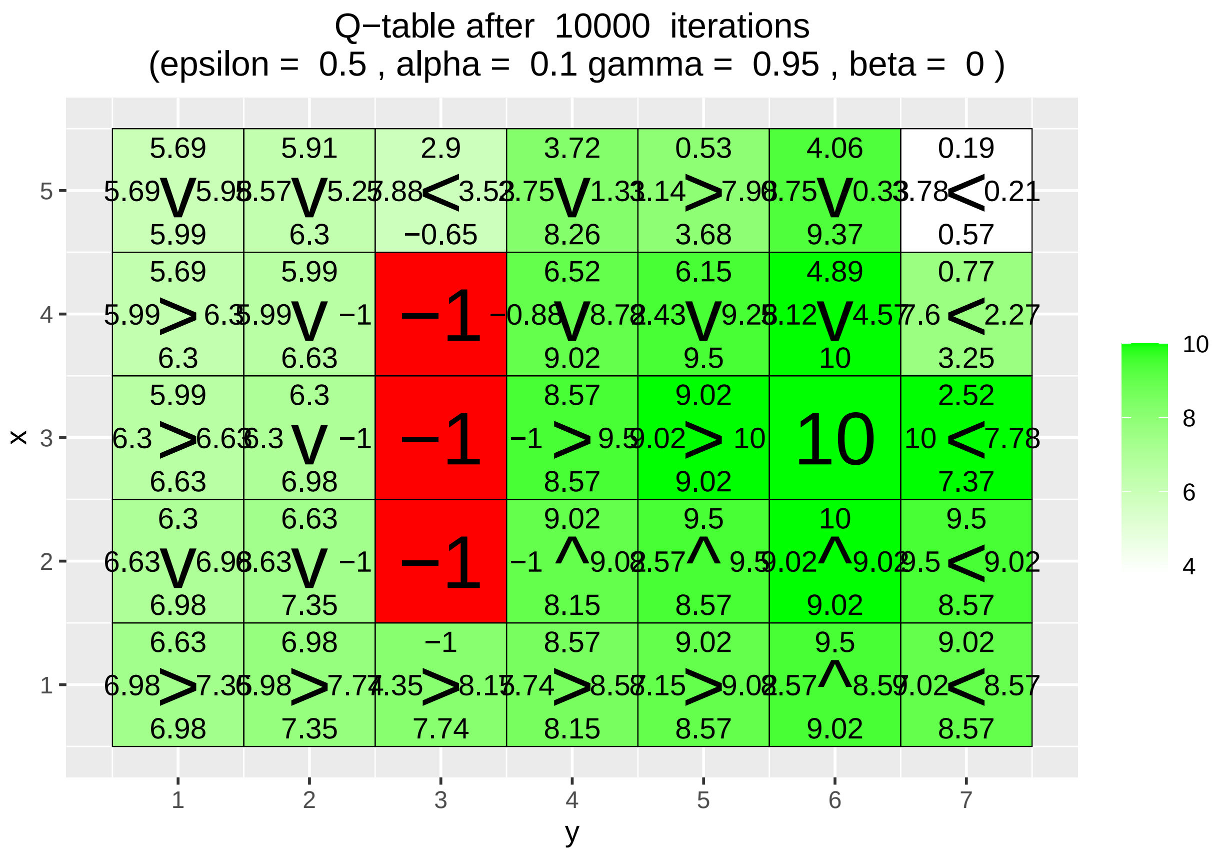

Q−table after 0 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
−1
−1
−1
10
x
y
10.0
7.5
5.0
2.5
0.0



Q−table after 10 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
−0.1
−0.27
−0.34
−0.1
−1
−1
−1
10
1
10.0
7.5
5.0
2.5
0.0


Q−table after 100 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
1
2
3
4
5
6
7
−1
10
10.0
7.5
5.0
2.5
0.0


Q−table after 1000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
−1
10



Question 1. What has the agent learned after the first 10 episodes ?

After 10 iterations, the agent has learned very little. The Q-table remains mostly zeros, indicating minimal exploration of the environment. The only knowledge acquired so far is a rudimentary avoidance of penalty areas . The agent has not yet discovered the +10 reward zone. At this early stage, the greedy policy is unreliable and appears almost random, as the agent has only explored a limited portion of the state space.

Question 2. Is the final greedy policy (after 10000 episodes) optimal for all states, i.e. not only for the initial state ? Why / Why not ?

Although after 10,000 iterations the agent clearly knows how to avoid the red penalty zones and efficiently navigate toward the green reward area, it has found a very good policy but not yet an optimal one.

For example, at positions (5,3), the agent should turn right directly toward the target location.

The primary reasons the agent failed to find the optimal are: Insufficient Training Episodes (10,000): Given the high exploration rate, the limited number of episodes provides insufficient experience for the successful, long-term optimal path to be consistently reinforced.

Question 3. Do the learned values in the Q-table reflect the fact that there are multiple paths(above and below the negative rewards) to get to the positive reward ? If not, what could be done to make it happen ?

The Q-table has captured the lower optimal path, but the path above the negative rewards has not been well captured.

`Lower Path (rows 1-2):` Q-values (max Q per state) increase smoothly: 7.35 $\rightarrow$ 7.74 $\rightarrow$ 8.15 $\rightarrow$ 8.57 $\rightarrow$ 9.02.

`Upper Path (rows 4-5):` No clear gradient due to insufficient exploration and convergence: 5.99 $\rightarrow$ 6.3 $\rightarrow$ 5.88 (drop) $\rightarrow$ 8.26 (sudden jump) $\rightarrow$ 7.98.

The main reasons the agent failed to find the Upper Path are likely:Insufficient Iterations: The limited number of training steps (10,000) is insufficient to effectively reinforce the long, optimal path.

Environment B

Grid: `7 x 8` grid

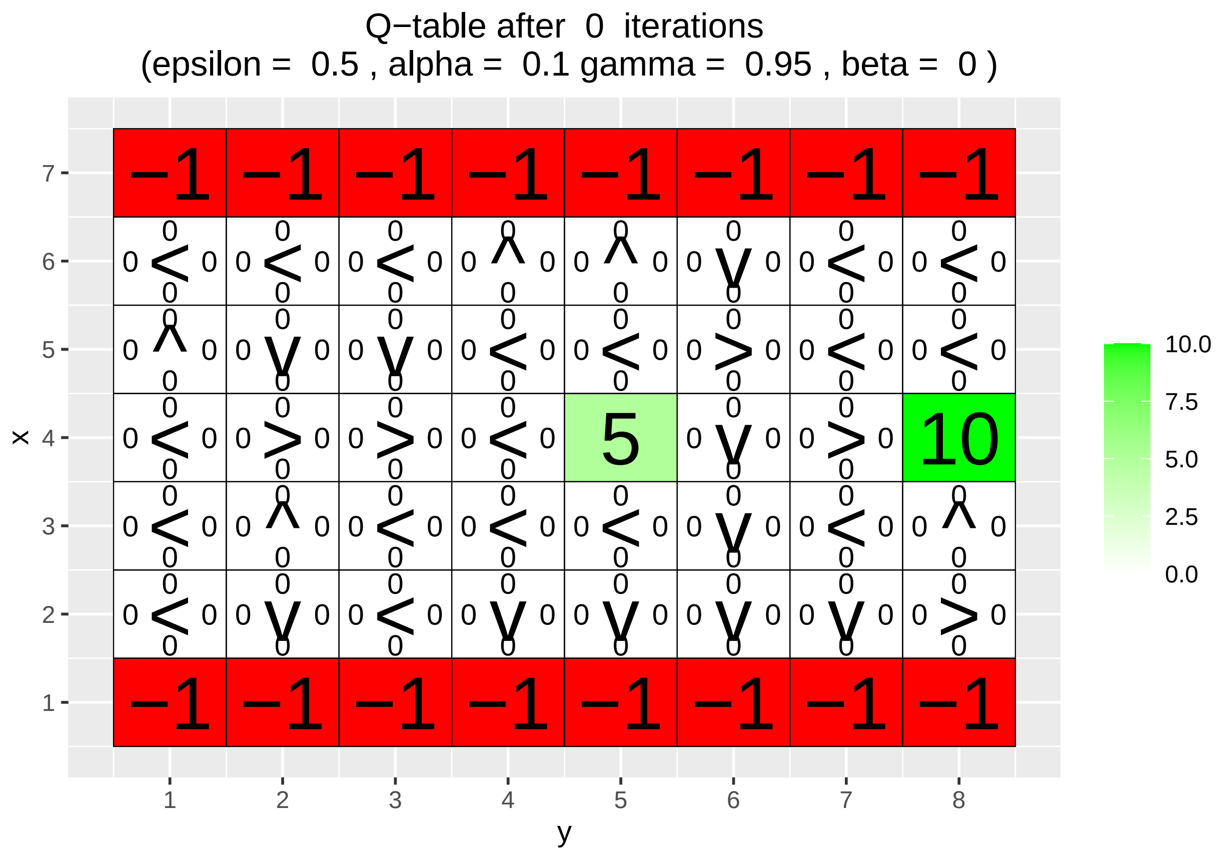
Reward: Negative rewards for states in the top and bottom rows. Two positive rewards: +5 (easily reachable) and +10 (requires navigating around the first reward)

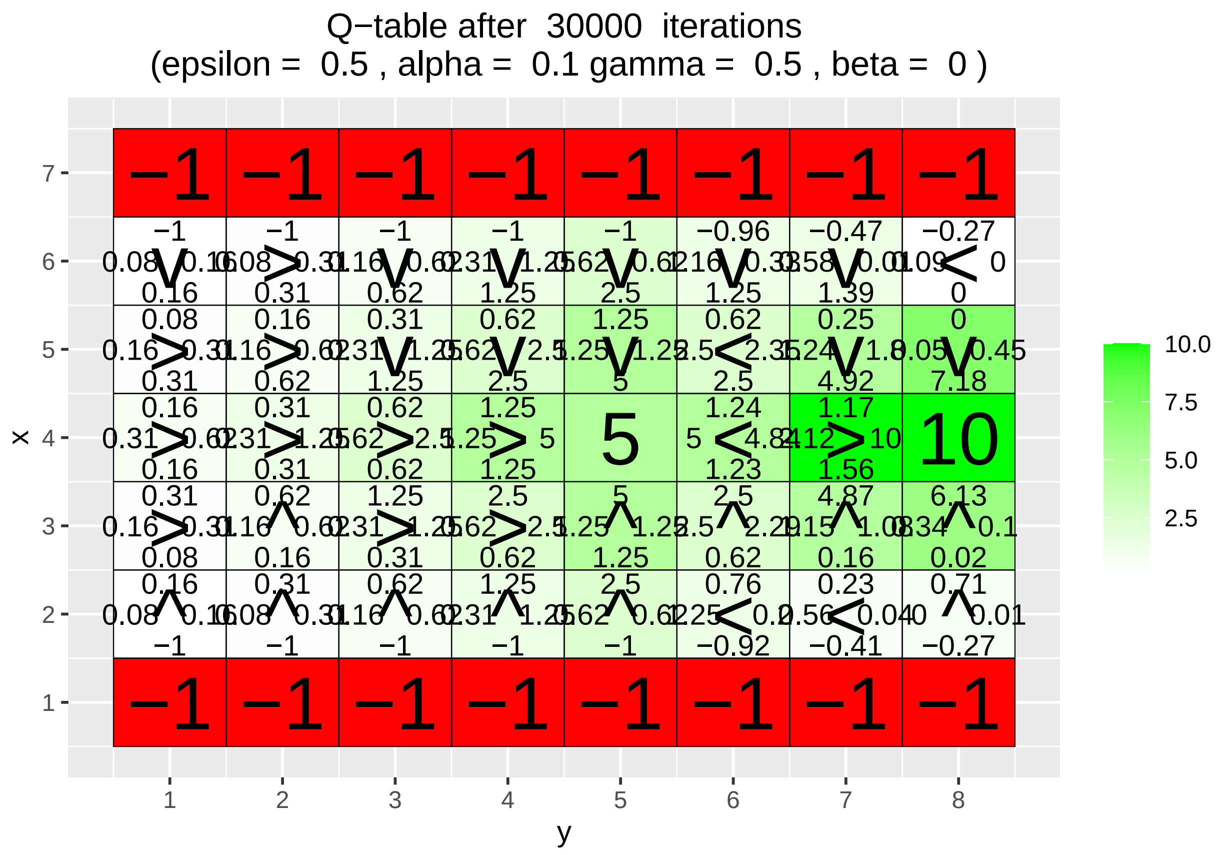
Start State: every episode, initial position is `(4, 1)`.

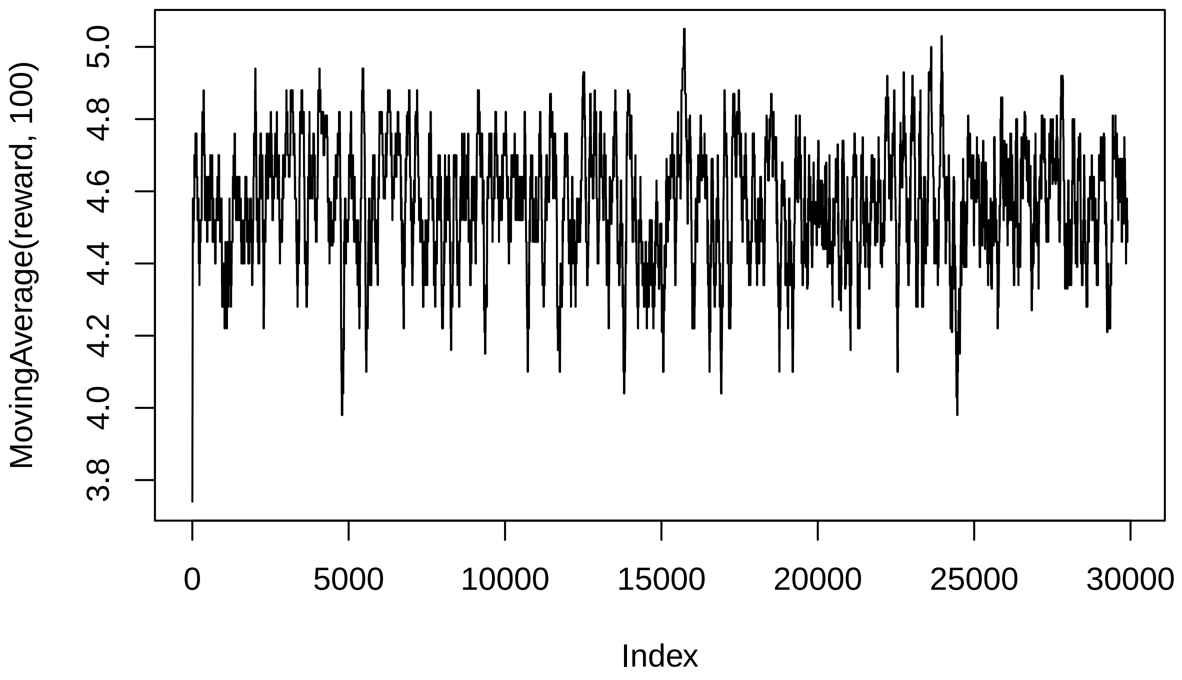
Q-table dimensions `7 x 8 x 4`.

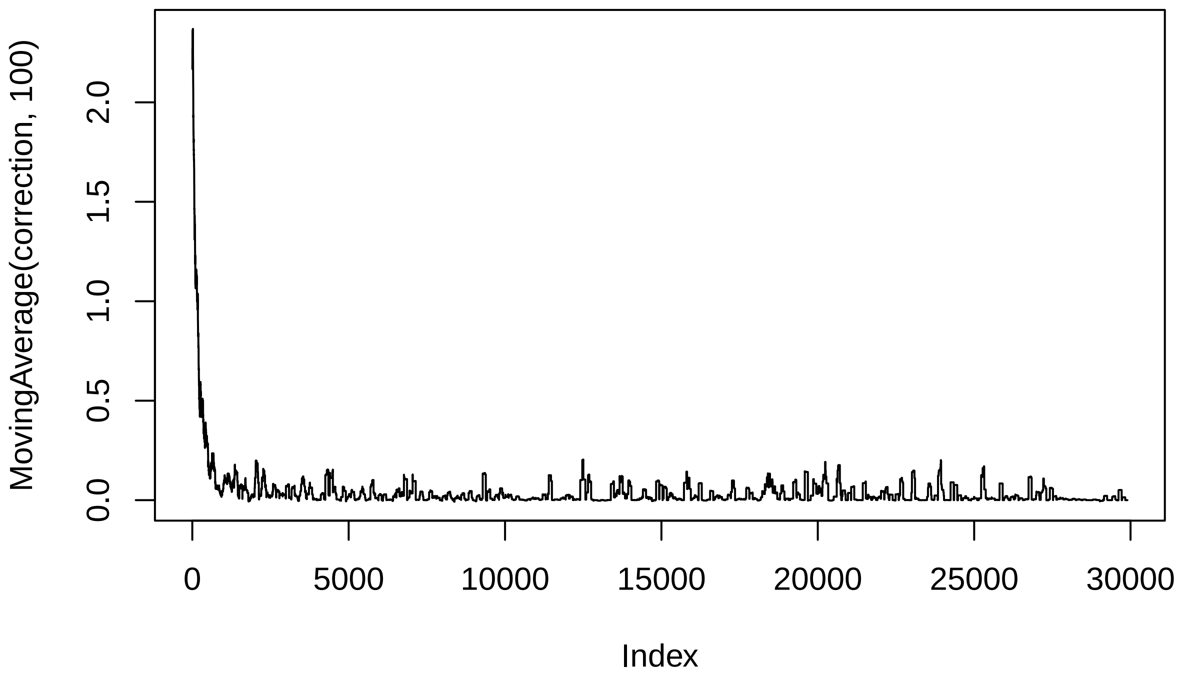
Initialization: The Q-table is initialized with all values set to `0`.

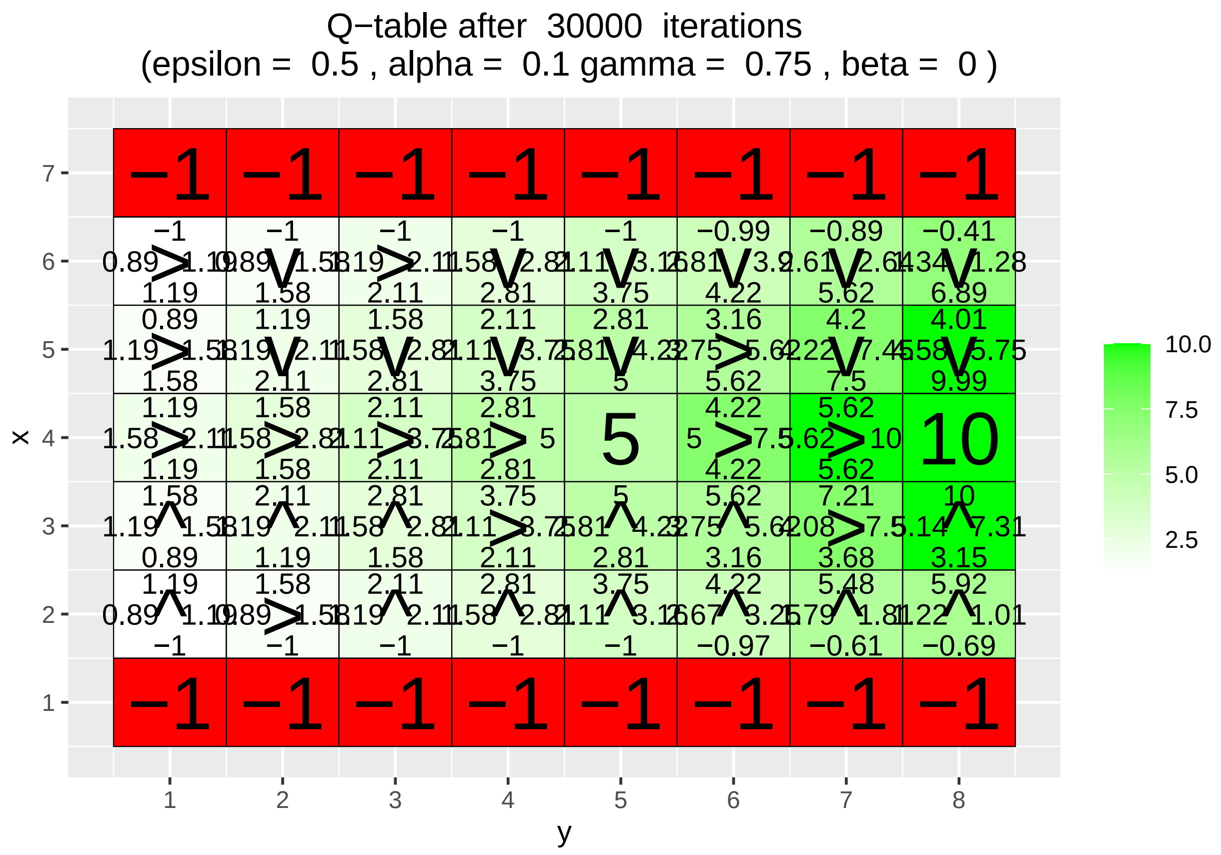
Parameters:The agent is trained over 30,000 episodes using the following: $\epsilon \in \{0.1, 0.5\}$, $\gamma \in \{0.5, 0.75, 0.95\}$, $\beta = 0$, and $\alpha = 0.1$

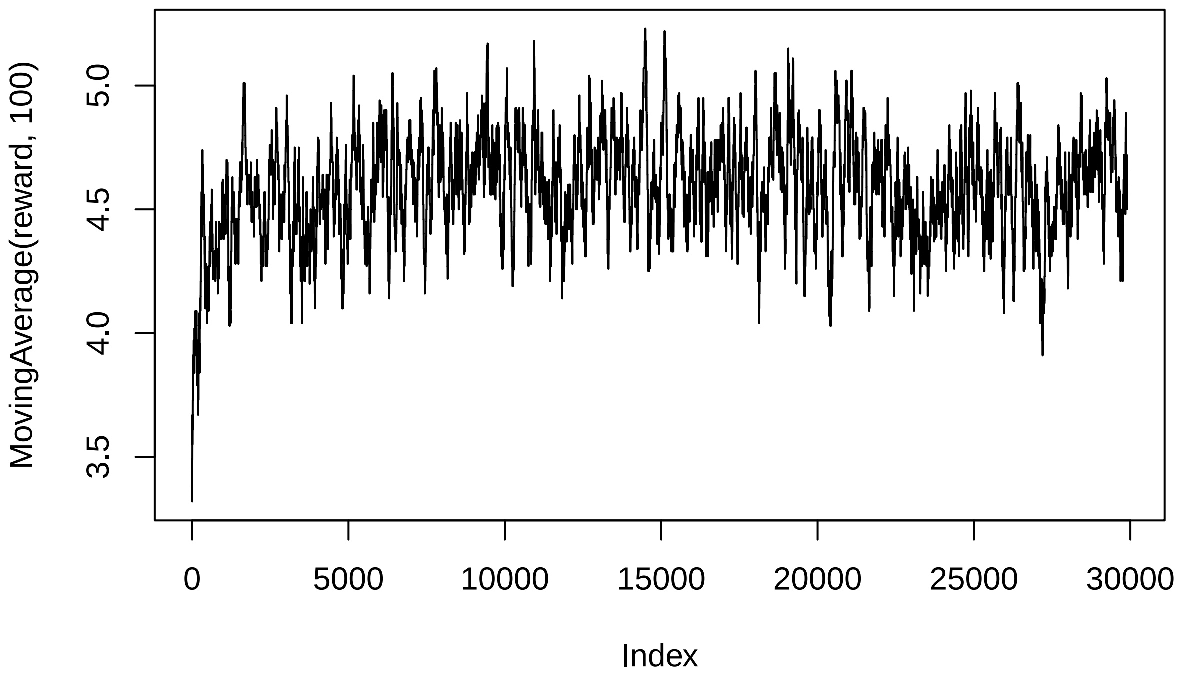

Q−table after 0 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
−1
5
10
x
y
10.0
7.5
5.0
2.5
0.0

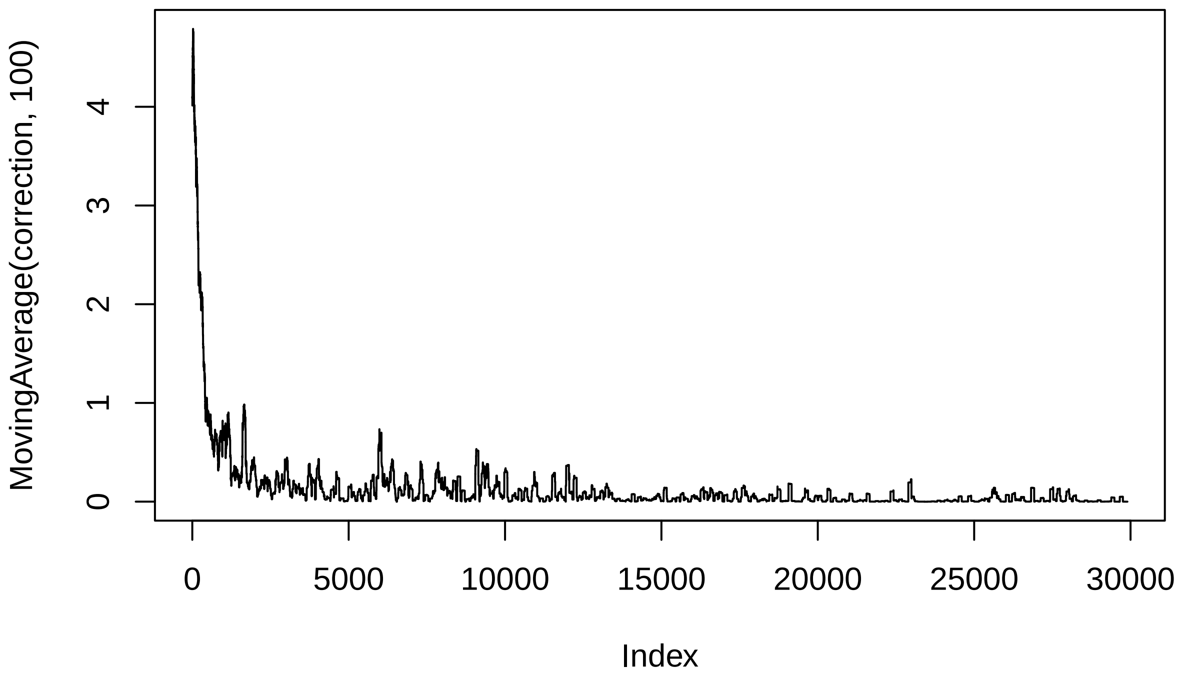

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10.0
7.5
5.0
2.5

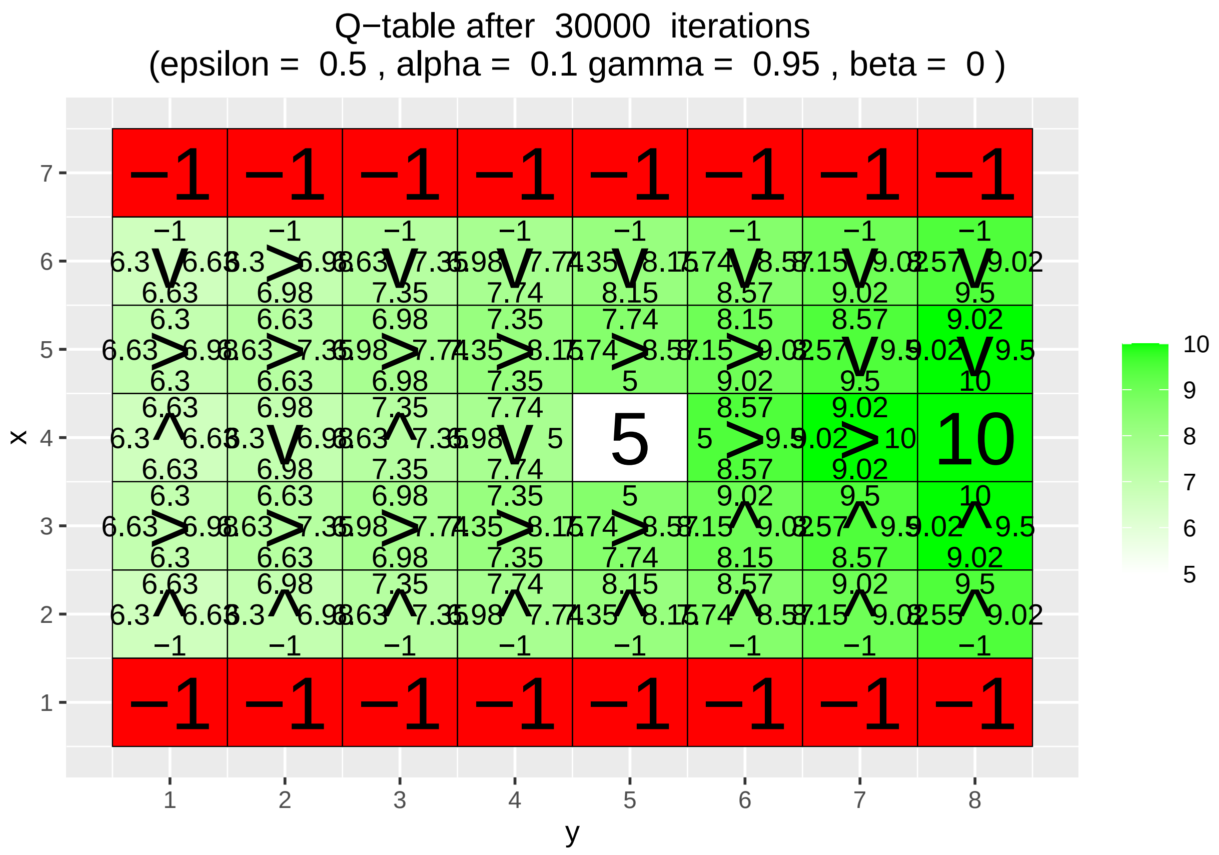

MovingAverage(reward, 100)
3.8
4.0
4.2
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index

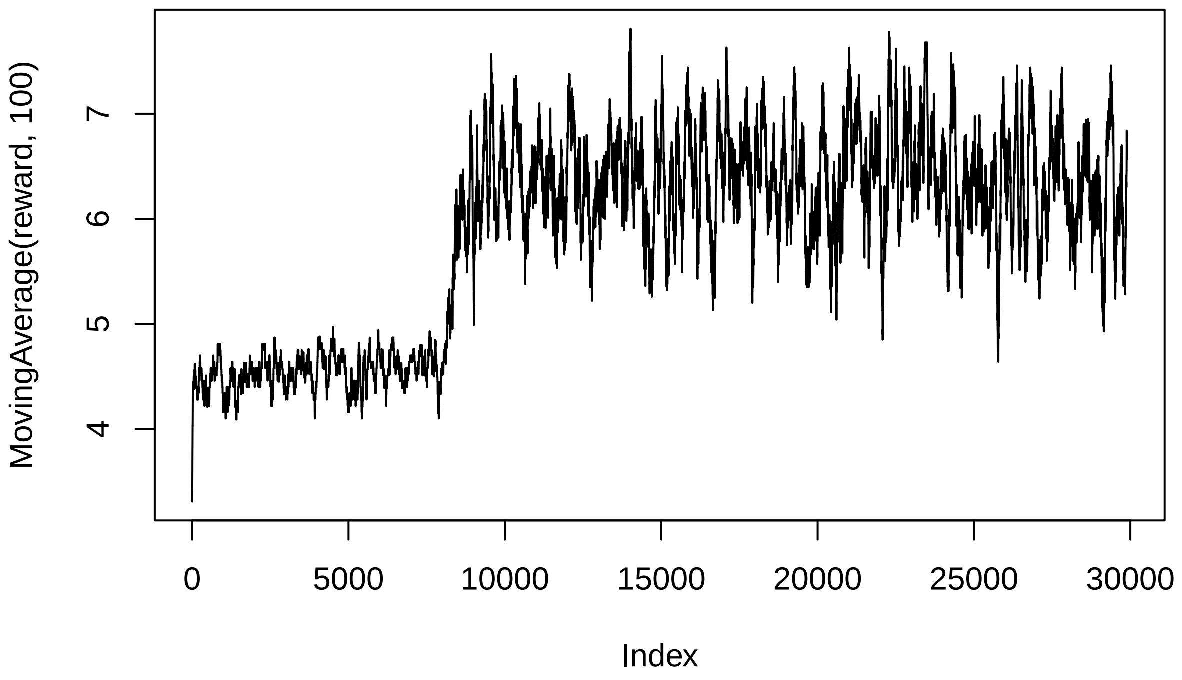

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index

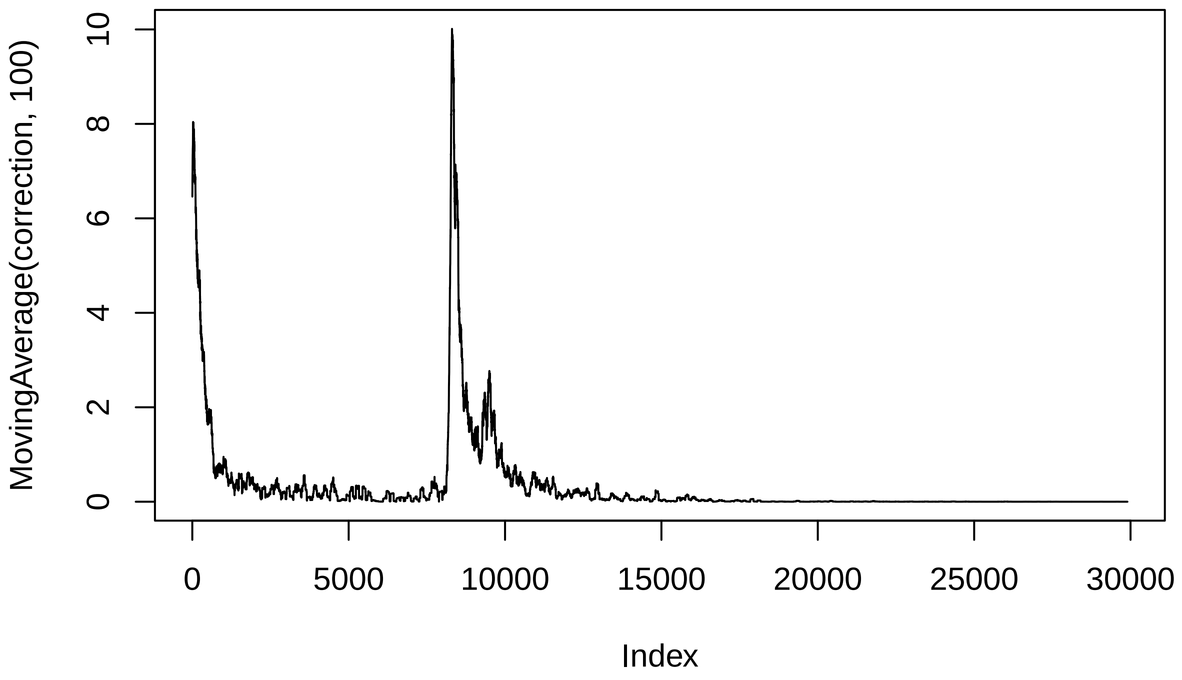

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5
5
10
−1

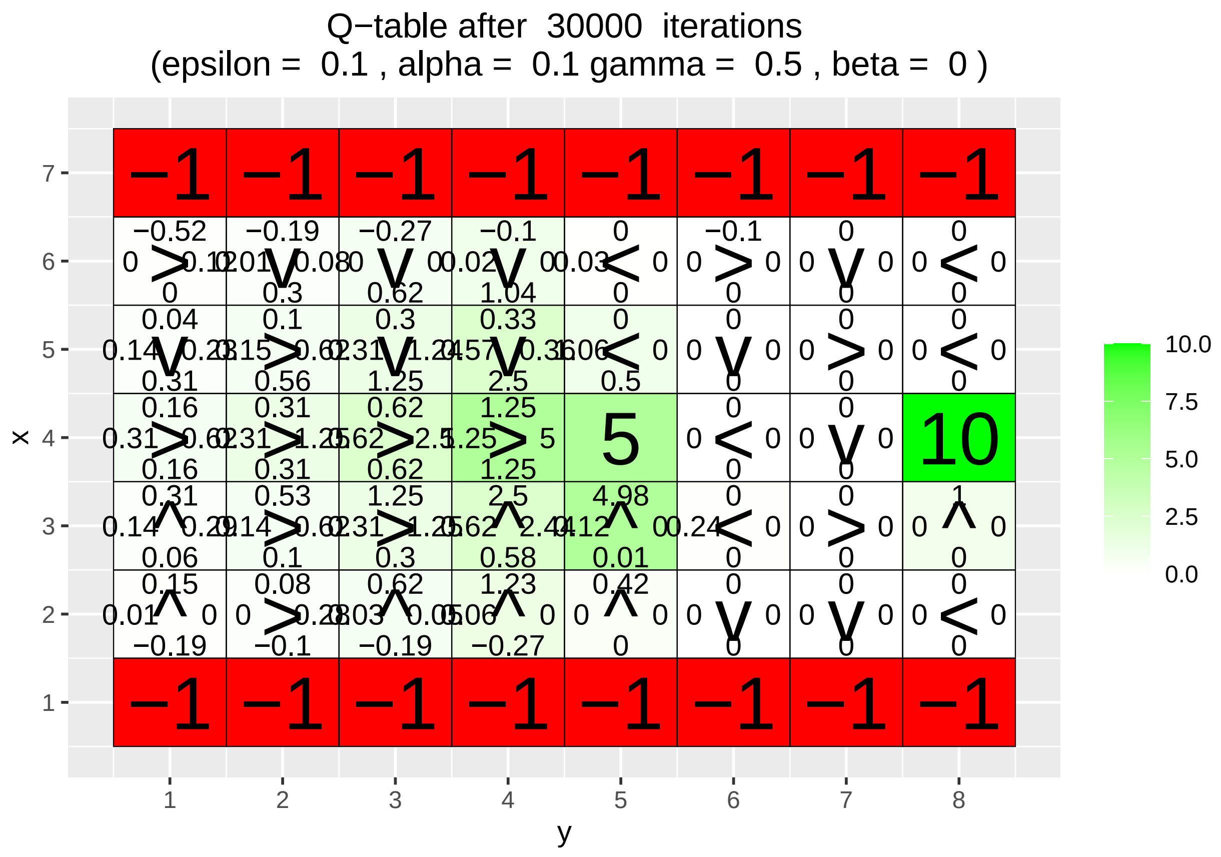

MovingAverage(reward, 100)
3.5
4.0
4.5
5.0
0
5000
10000
15000
20000
25000
30000
Index

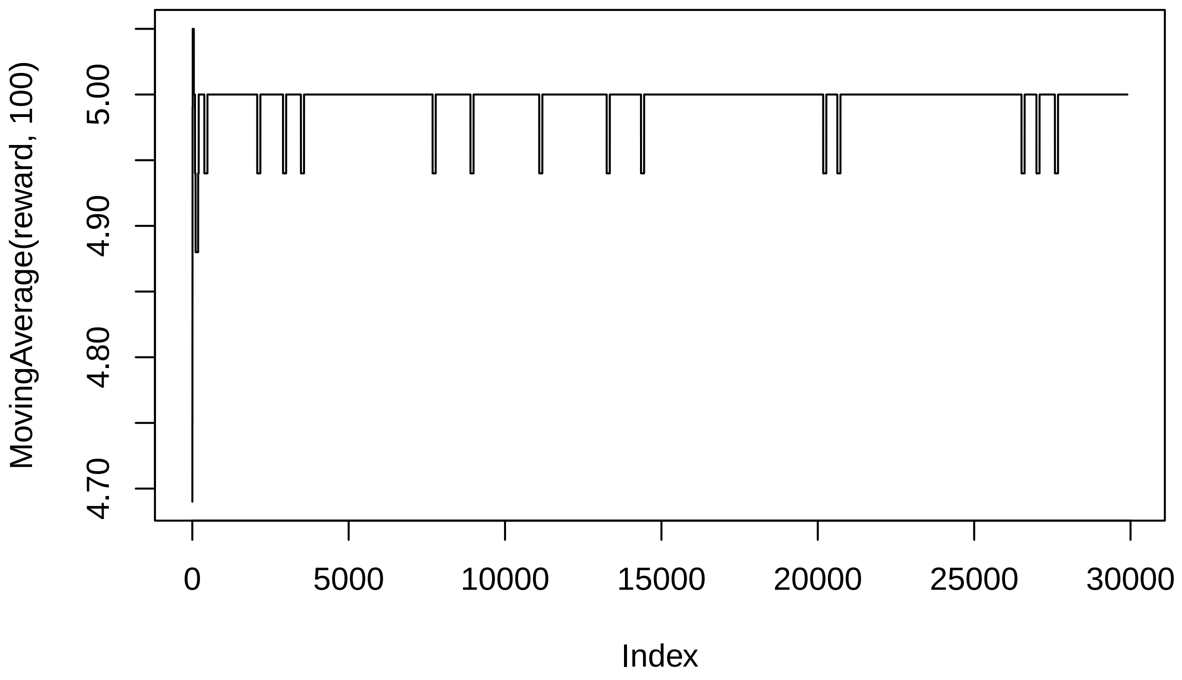

MovingAverage(correction, 100)
0
1
2
3
4
0
5000
10000
15000
20000
25000
30000
Index

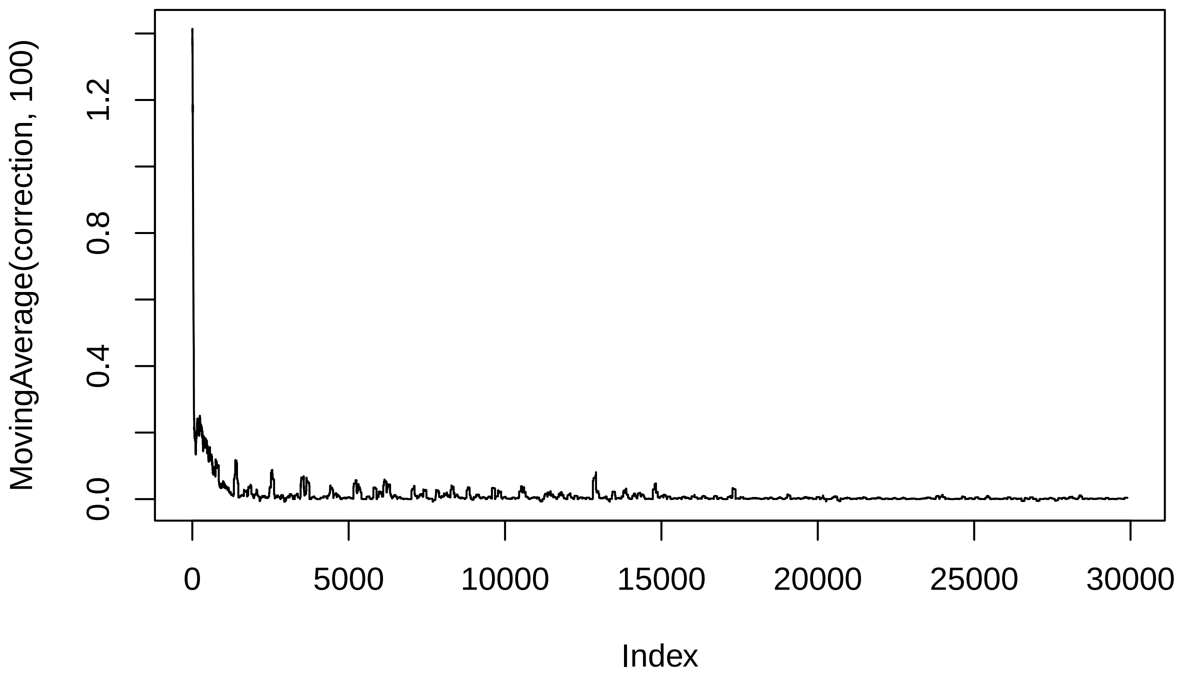

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
−1
5
10

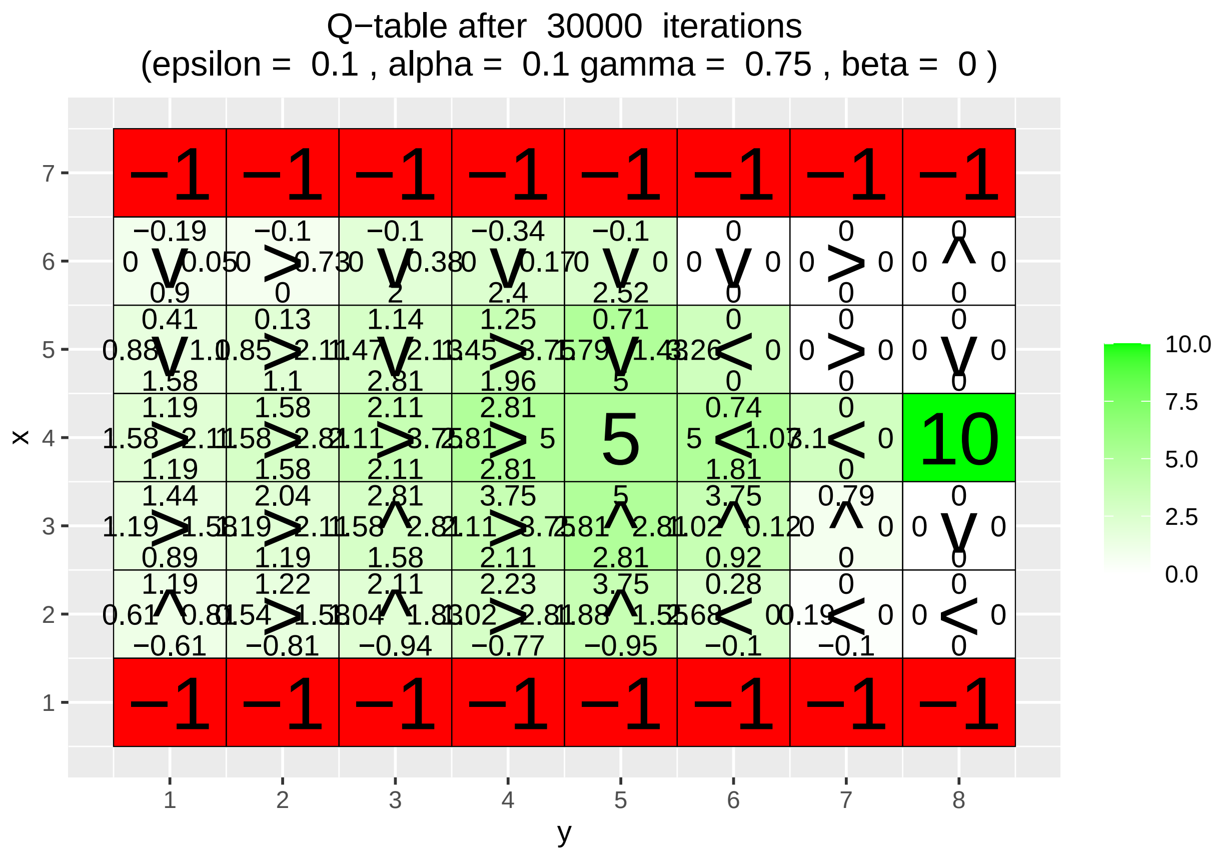

MovingAverage(reward, 100)
4
5
6
7
0
5000
10000
15000
20000
25000
30000
Index


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

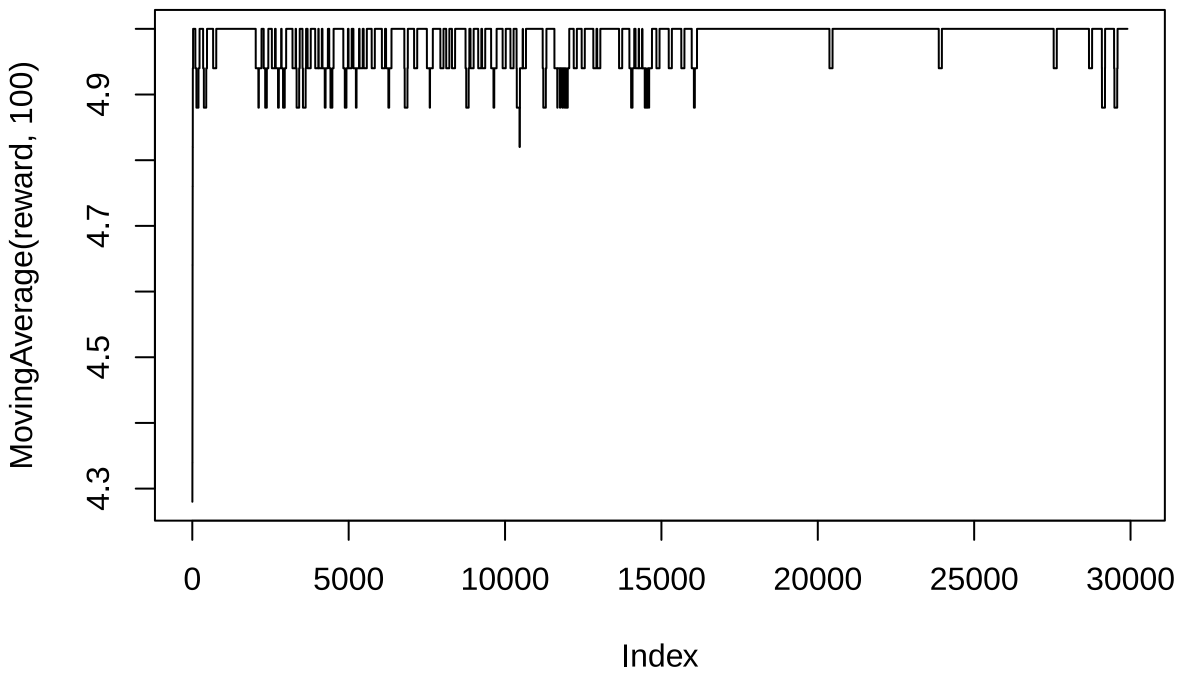

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
1
2
3
4
5
6
7
8
−1
5
10
10.0
7.5
5.0
2.5
0.0


MovingAverage(reward, 100)
4.70
4.80
4.90
5.00
0
5000
10000
15000
20000
25000
30000
Index

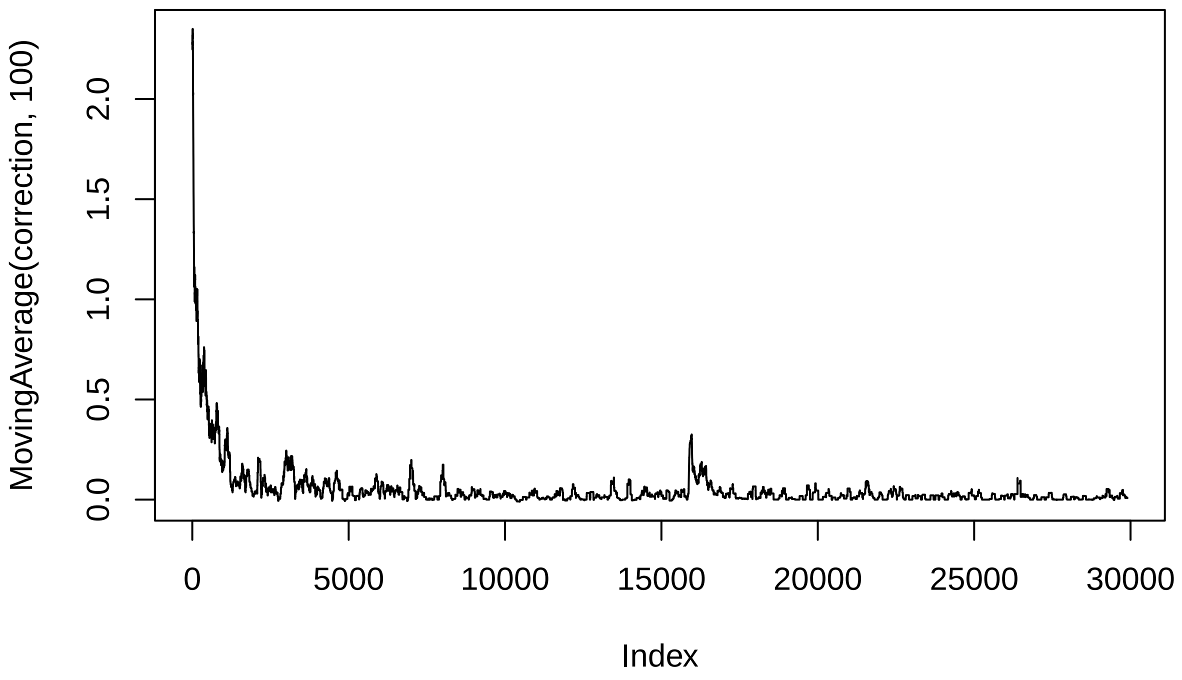

MovingAverage(correction, 100)
0.0
0.4
0.8
1.2
0
5000
10000
15000
20000
25000
30000
Index

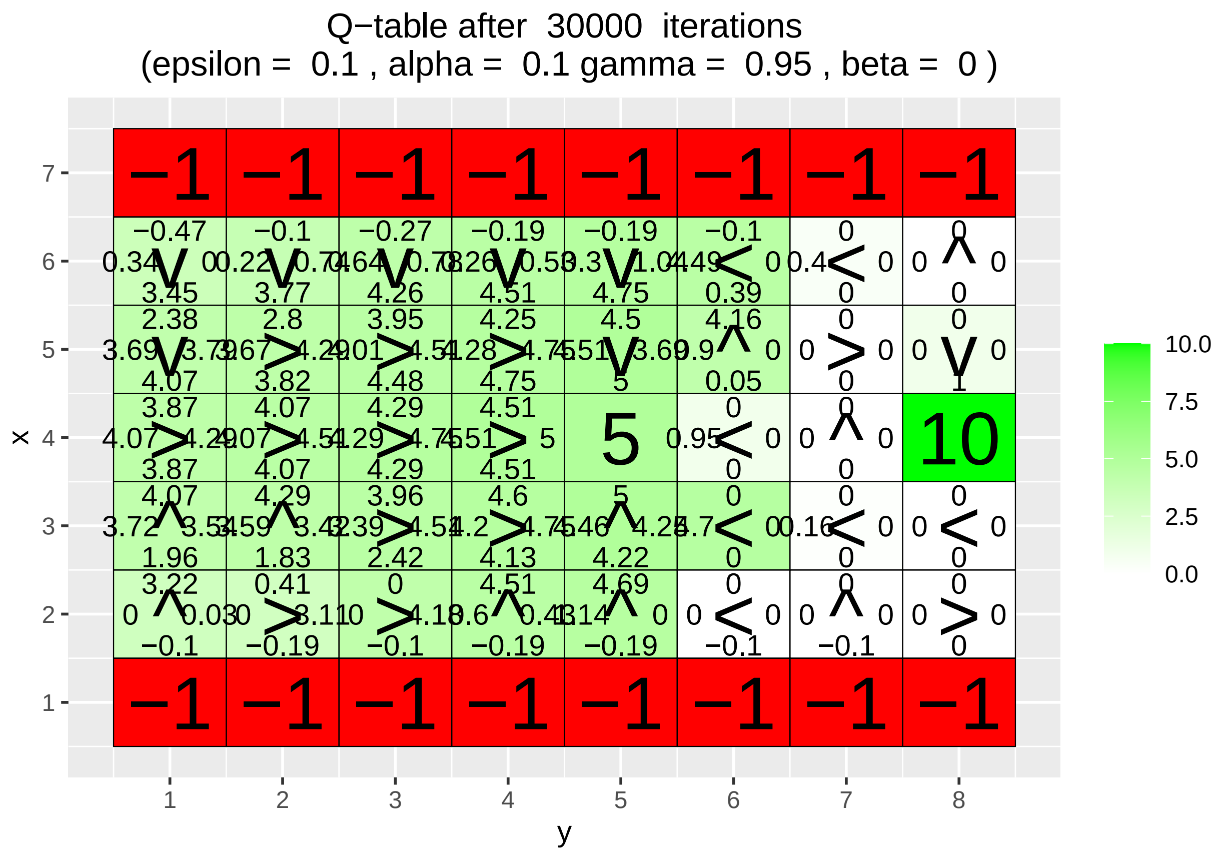


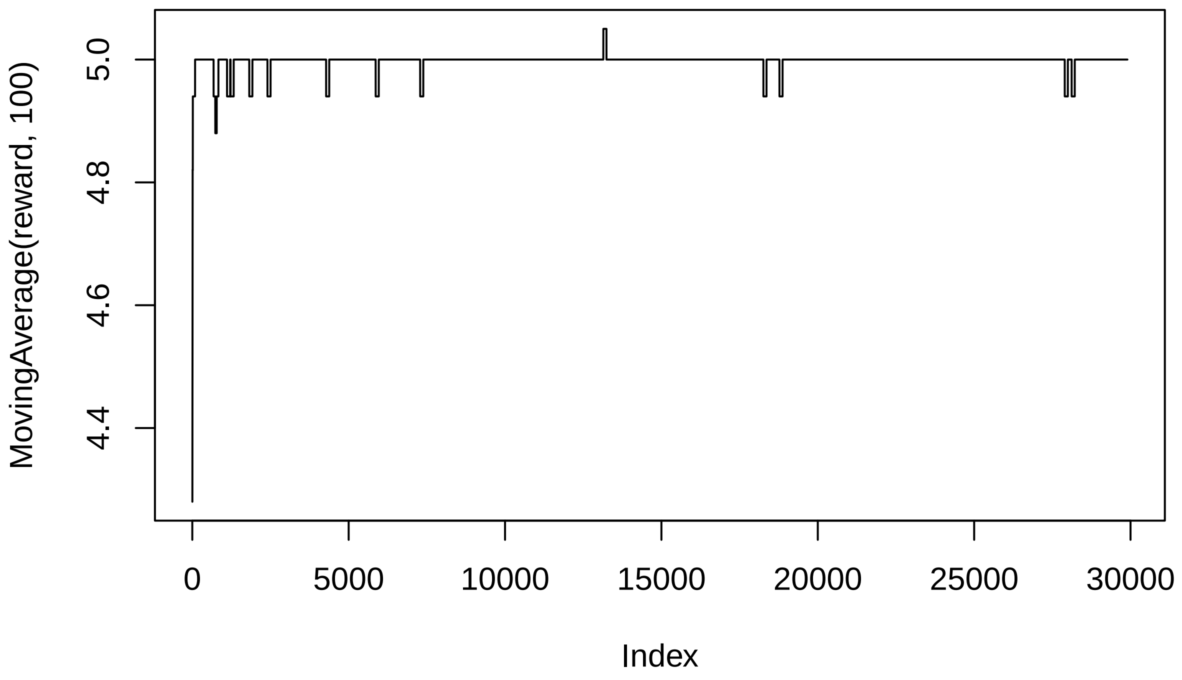
Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0

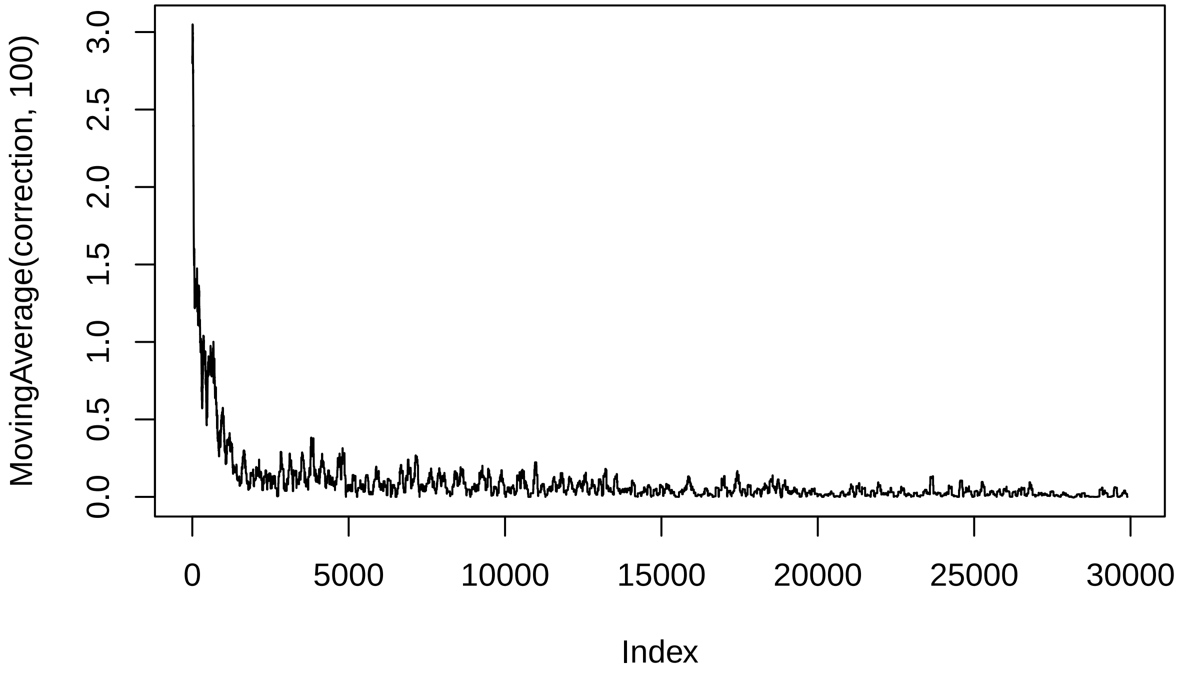


MovingAverage(reward, 100)
4.3
4.5
4.7
4.9
0
5000
10000
15000
20000
25000
30000
Index


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)
x
y
10.0
7.5
5.0
2.5
0.0


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



Investigate how ϵ and γ affect Q-learning policy

$\epsilon \in \{0.1, 0.5\}$, $\gamma \in \{0.5, 0.75, 0.95\}$, $\beta = 0$, and $\alpha = 0.1$

Impact of discount factor: $\gamma = 0.5$: Agent is short-sighted, prefers nearby +5 reward (return = 2.5) over distant +10 (return = 0.625)

$\gamma = 0.75$: Agent begins favoring +10, showing increased foresight

$\gamma = 0.95$: Agent strongly prefers +10, with high Q-values propagating across long distances

Impact of exploration rate: $\epsilon = 0.5$: High exploration causes reward fluctuations (between 4-7), prevents convergence despite learning correct policy

$\epsilon = 0.1$: Low exploration produces stable rewards but risks:

`Large unexplored areas` (Q-values remain zero)

`Local optima traps` ($\gamma = 0.75$case: agent switches from +10 to +5 strategy around iteration 15,000)

Optimal combination: $\epsilon = 0.1$, $\gamma = 0.95$

Agent discovers and maintains optimal +10 strategy throughout training

Reward consistently between 7-9

Q-values form clear gradient toward +10

TD Error patterns: All experiments show large initial correction peaks when discovering rewards, followed by rapid convergence near zero, indicating successful Q-value stabilization.

Environment C

Grid: `3 x 6` grid

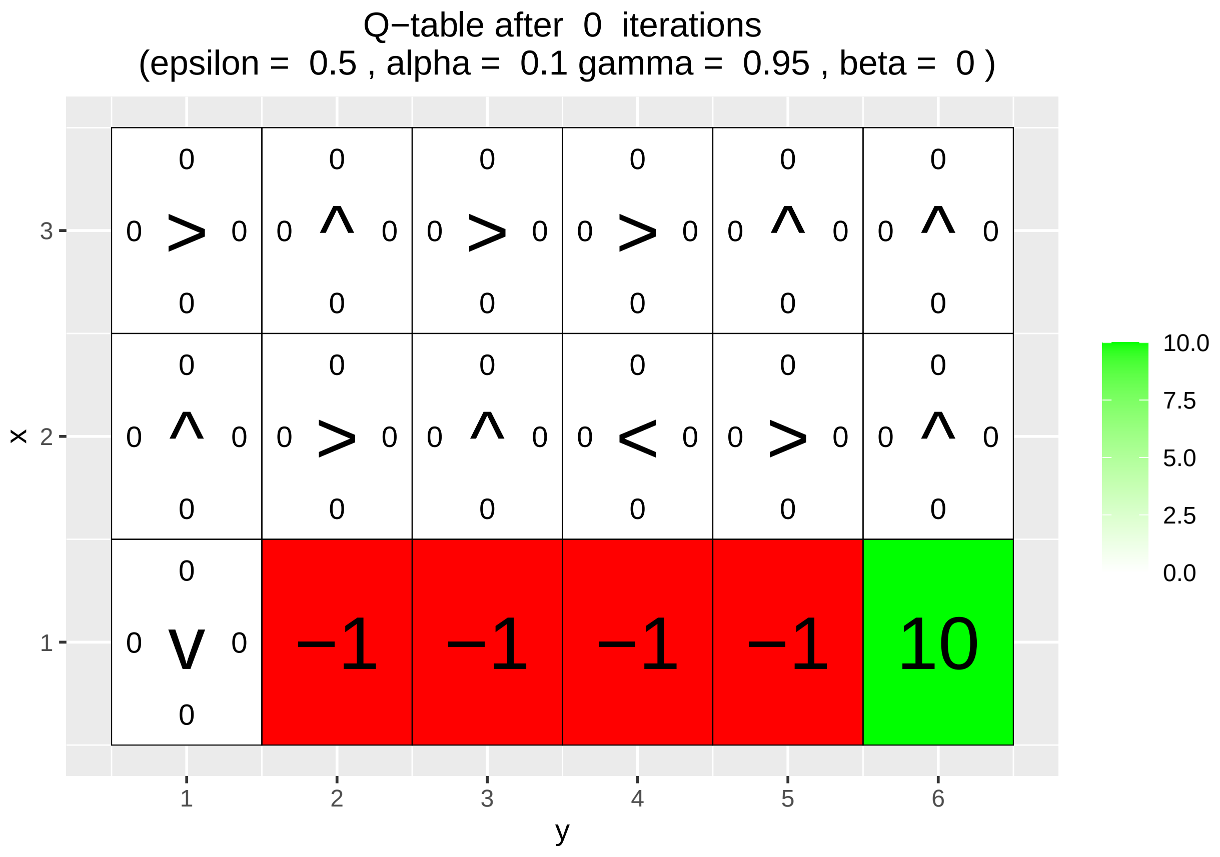
Reward: +10 at state (1, 6)(terminal goal state). -1 at states (1, 2),(1, 3), (1, 4), and (1, 5)(also terminal states).

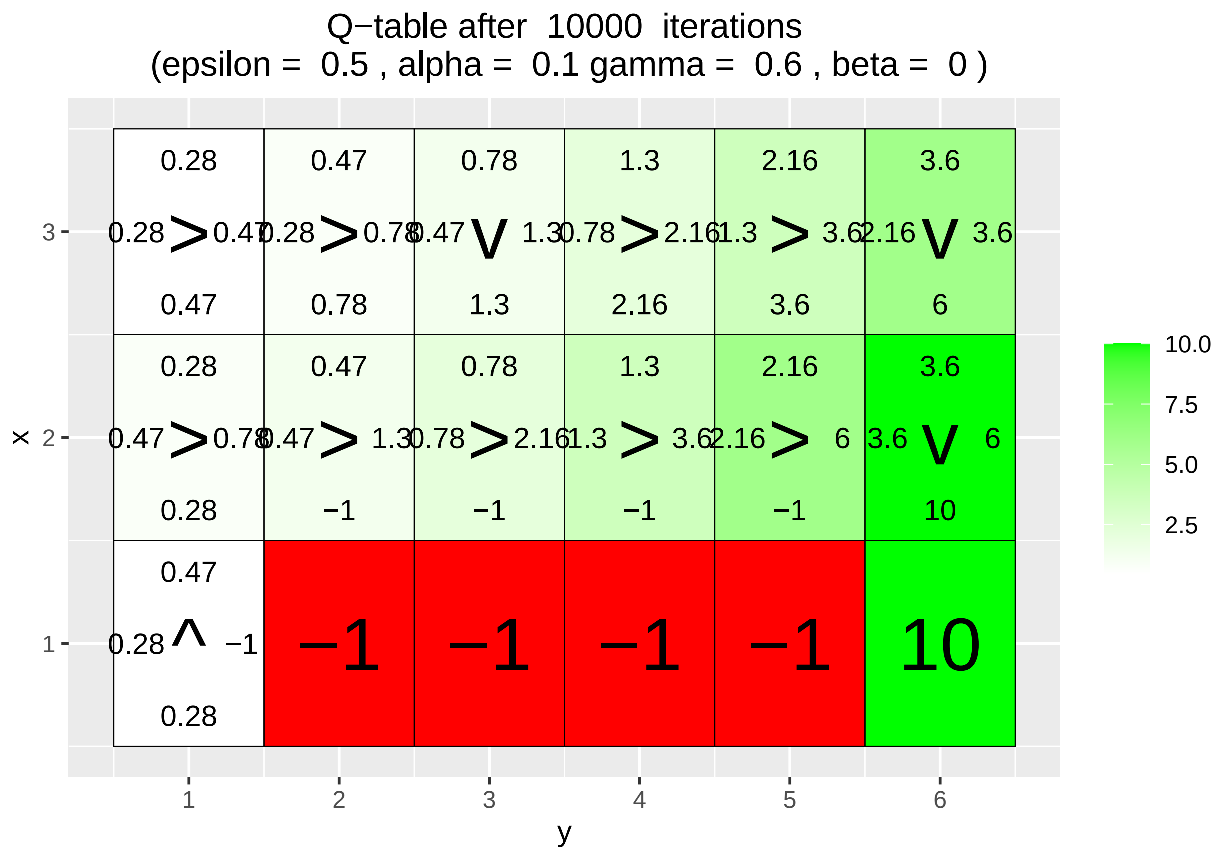
Start State: every episode, initial position is `(1, 1)`.

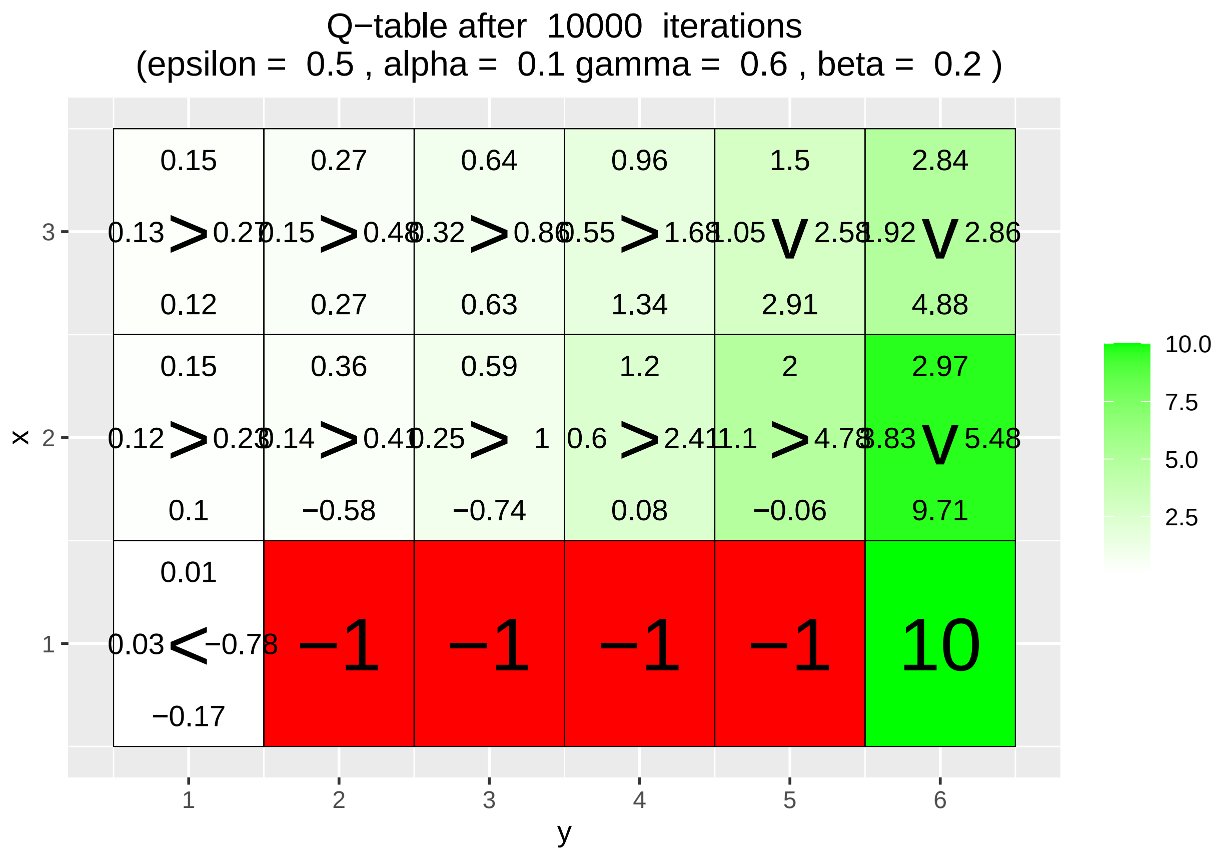
Q-table dimensions `3 x 6 x 4`.

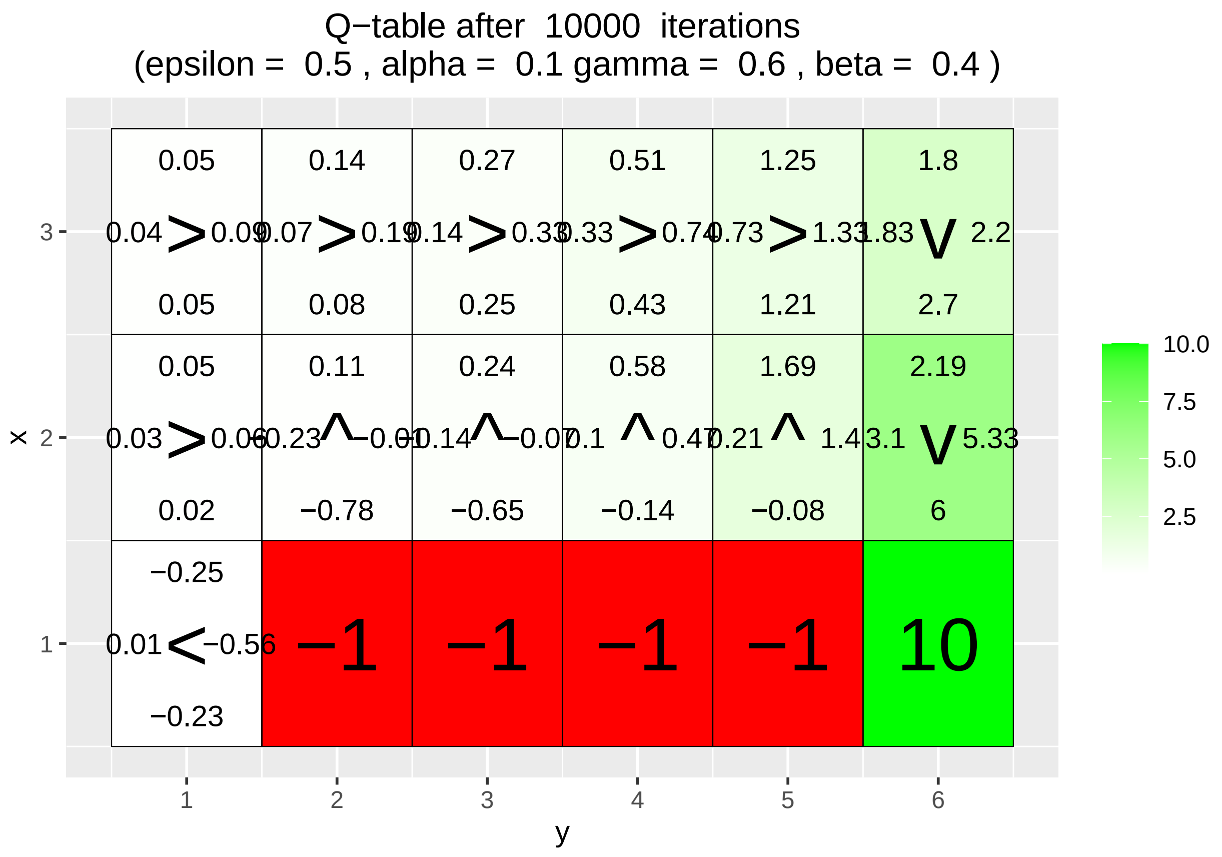
Initialization: The Q-table is initialized with all values set to `0`.

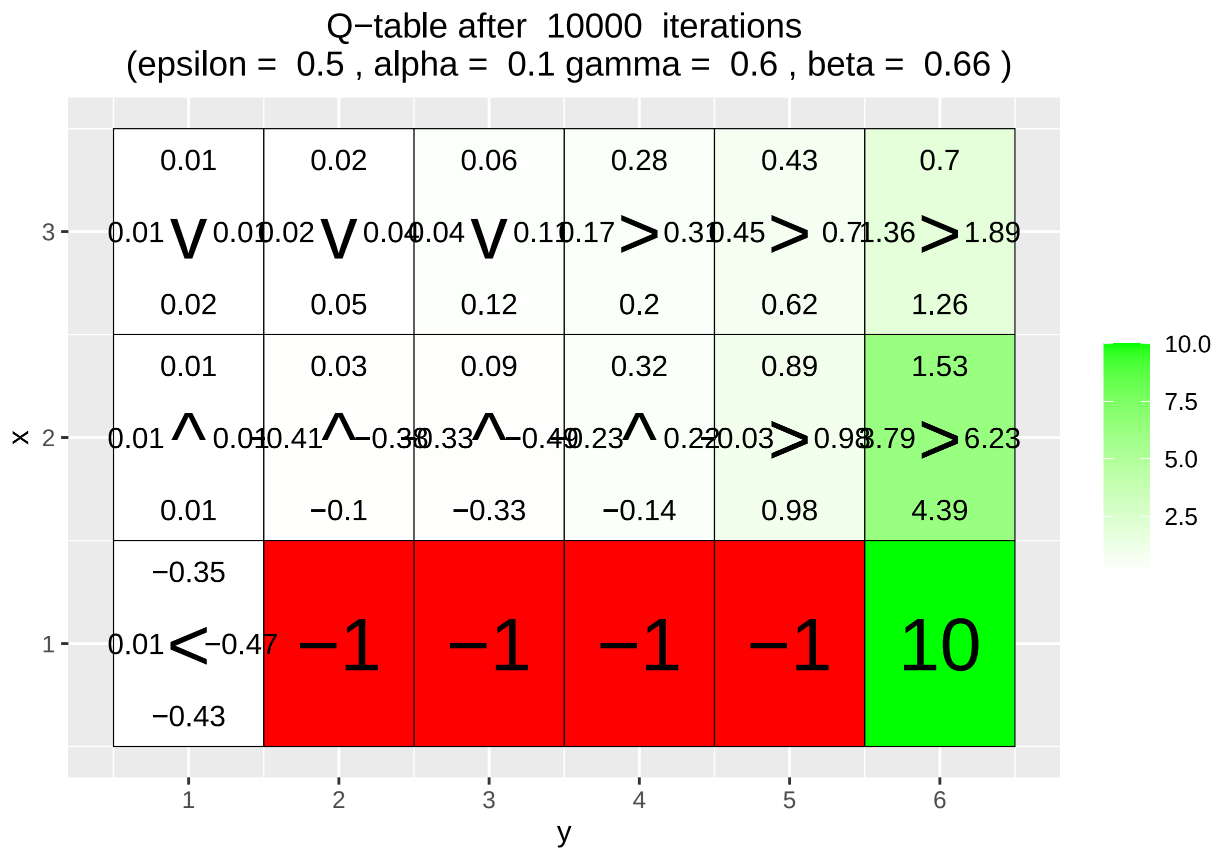
Parameters:The agent is trained over 10,000 episodes using the following: $\epsilon = 0.5$, $\alpha = 0.1$, $\gamma = 0.6$, $\beta = 0, 0.2, 0.4, 0.66$




Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0)
0.28
0.28 > 0.47
0.47
0.47
0.28 > 0.78
0.78
0.78
0.47 V 1.3
1.3
1.3
0.78 > 2.16
2.16
2.16
1.3 > 3.6
3.6
3.6
2.16 V 3.6
6
0.28
0.47 > 0.78
0.28
0.47
0.47 > 1.3
−1
0.78
0.78 > 2.16
−1
1.3
1.3 > 3.6
−1
2.16
2.16 > 6
−1
3.6
3.6 V 6
10
0.47
0.28 ^ −1
0.28
−1
−1
−1
−1
10
x
y
1
2
3
4
5
6
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)
0.15
0.13 > 0.27
0.12
0.27
0.15 > 0.48
0.27
0.64
0.32 > 0.86
0.63
0.96
0.55 > 1.68
1.34
1.5
1.05 V 2.58
2.91
2.84
1.92 V 2.86
4.88
0.15
0.12 > 0.23
0.1
0.36
0.14 > 0.41
−0.58
0.59
0.25 > 1
−0.74
1.2
0.6 > 2.41
0.08
2
1.1 > 4.78
−0.06
2.97
3.83 V 5.48
9.71
0.01
0.03 < −0.78
−0.17
−1
−1
−1
−1
10
x
y
10.0
7.5
5.0
2.5


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.4)
0.05
0.04 > 0.09
0.05
0.14
0.07 > 0.19
0.08
0.27
0.14 > 0.33
0.25
0.51
0.33 > 0.74
0.43
1.25
0.73 > 1.33
1.21
1.8
1.83 V 2.2
2.7
0.05
0.03 > 0.06
0.02
0.11
−0.23 ^ −0.01
−0.78
0.24
0.14 ^ −0.07
−0.65
0.58
0.1 ^ 0.47
−0.14
1.69
0.21 ^ 1.4
−0.08
2.19
3.1 V 5.33
6
−0.25
0.01 < −0.56
−0.23
−1
−1
−1
−1
10
10.0
7.5
5.0
2.5
x
3
2
1
y
1
2
3
4
5
6



The β controls the randomness of the environment: $\beta = 0$ means the agent's movement always matches expectation, while higher β values increase the probability that the agent will slip to the side when moving.

Investigate how β affect Q-learning policy

$\epsilon = 0.5$, $\alpha = 0.1$, $\gamma = 0.6$, $\beta = 0, 0.2, 0.4, 0.66$

$\beta = 0$ (Deterministic environment): Agent learns the shortest path along row 2 (edge of penalty zone) No slipping risk, so moving close to penalties is safe

$\beta = 0.2$ and $\beta = 0.4$(Increasing stochasticity): Policy becomes more conservative

At $\beta = 0.2$: Agent begins preferring upward movement away from penalties.

At $\beta = 0.4$: Agent moves to safest row 3 early, then travels right

$\beta = 0.66$ (High stochasticity): Agent moves to row 3 immediately.

Agent prioritizes safety over path length, choosing the longest but safest route. Equal slip probability in all directions makes row 2 too risky

Conclusion: The Q-learning agent adapts its policy to environmental uncertainty. As uncertainty increases, the learned policy shifts from optimal shortest-path to conservative strategies.

Environment D

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

The results for 8 different training goals after 5,000 episodes are presented in Figure 7. It successfully reaches validation goals that were not encountered during training, demonstrating that it has acquired a generalizable strategy for reaching targets rather than merely memorizing specific paths.

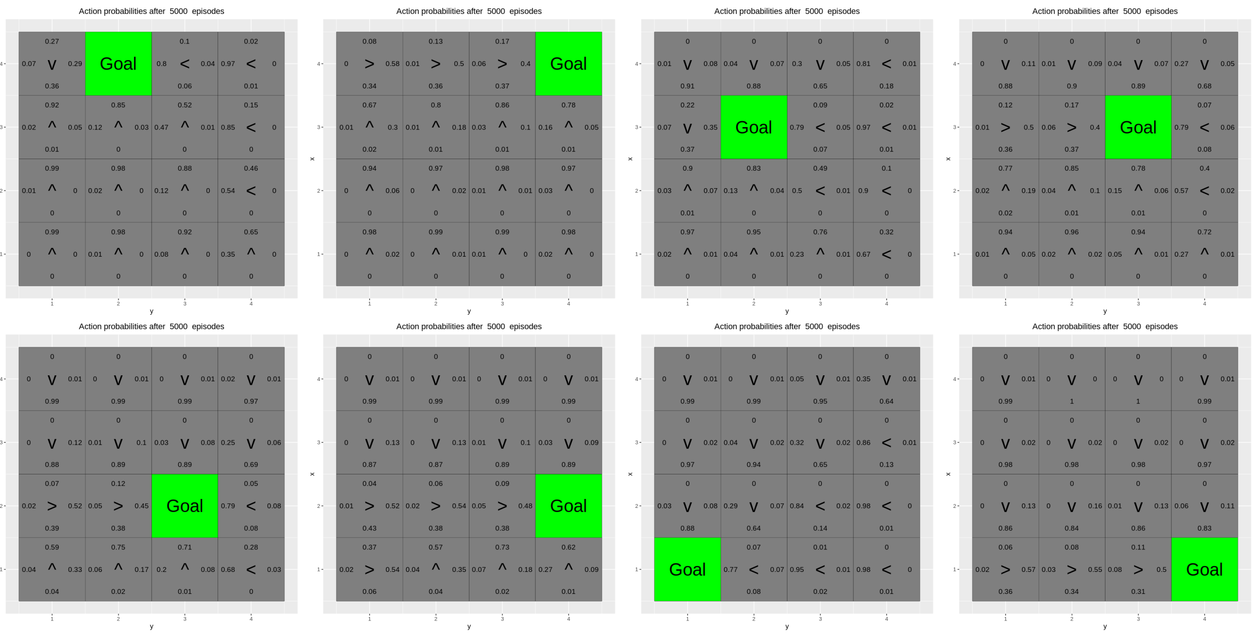


Figure 8. Train results after 5000 episodes with 8 goals

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

Not really. The Q-learning algorithm can learn a policy for a fixed goal location, but it cannot generalize to new, unseen goals. Each agent–goal pair is treated as a separate state with its own Q-values, so knowledge cannot transfer between goals. In contrast, the REINFORCE approach parameterizes the policy with a neural network, which learns a general strategy for reaching targets rather than memorizing actions for one specific goal.

Environment E

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

The results for three different training goals after 5,000 episodes are shown in Figure 8. The learned policy remains clearly suboptimal, with most arrows pointing in the same direction. This outcome is mainly due to the biased training setup: all training goals were located in the top row. As a result, the policy learned an oversimplified pattern that fails to generalize to validation goals, which are more uniformly distributed across the grid.

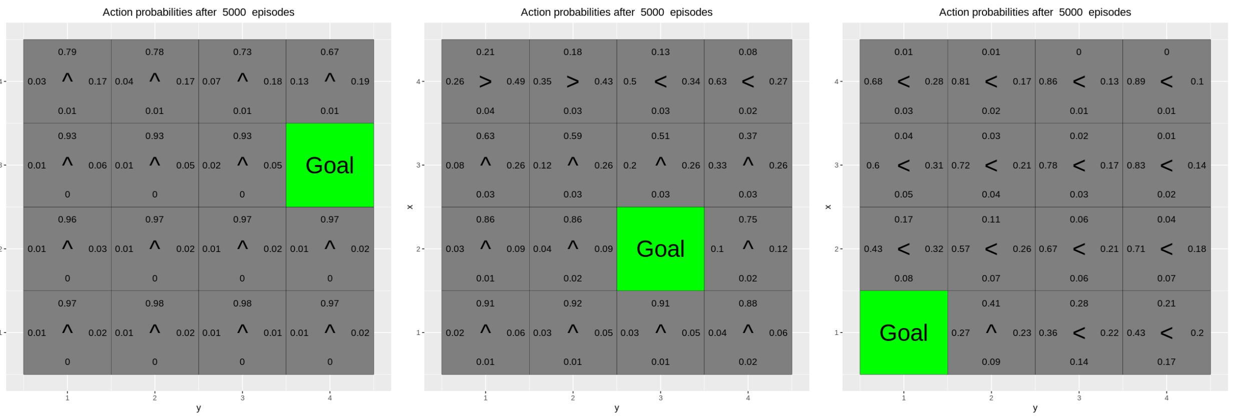


Figure 9. Train Results After 5000 Episodes With 6 goals

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

The difference between environments D and E arises primarily from the diversity of training goals. In Environment D, the agent was exposed to goals distributed across the entire grid, enabling the policy network to learn a general mapping from state and goal coordinates to actions. This diversity supports robust generalization to unseen validation goals.

In contrast, Environment E restricted all training goals to the top row, which biased the data and led the agent to learn an oversimplified rule. As a result, the policy failed to transfer effectively to validation goals located elsewhere in the grid.

Contribution

This report was completed through the joint efforts of team members: Xiaochen, Linn, Qinxia, Qingxuan and Huaide. We collectively tackled the core concepts of Hidden Markov Models through group discussions, coded and debugged the R script as a team, and analyzed the experimental results together. The writing and final review of this document were also a result of our collaborative work, ensuring the consistency and quality of the final report.