Lab3

Cajsa Schöld

2024-09-30

Implementation of the policy functions

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

  # Get a greedy action for state (x,y) from q_table.
  #
  # Args:
  #   x, y: state coordinates.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.

  # Your code here.

  q_values = q_table[x, y, ]

  q_max = max(q_values)

  best = which(q_values == q_max)

  #breaking ties at random
  action = sample(best, 1)

  return(action)
}

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

  # Get an epsilon-greedy action for state (x,y) from q_table.
  #
  # Args:
  #   x, y: state coordinates.
  #   epsilon: probability of acting randomly.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.

  # Your code here.

  random = runif(1)
```

```
  if (random < epsilon) {
    action = sample(1:4, 1)
  } else {
    q_values = q_table[x, y, ]

    q_max = max(q_values)

    best = which(q_values == q_max)

    #breaking ties at random
    action = sample(best, 1)
  }

  return(action)
}
```

Transition function (given)

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

  # Computes the new state after given action is taken. The agent will follow the action
  # with probability (1-beta) and slip to the right or left with probability beta/2 each.
  #
  # Args:
  #   x, y: state coordinates.
  #   action: which action the agent takes (in {1,2,3,4}).
  #   beta: probability of the agent slipping to the side when trying to move.
  #   H, W (global variables): environment dimensions.
  #
  # Returns:
  #   The new state after the action has been taken.

  delta <- sample(-1:1, size = 1, prob = c(0.5*beta,1-beta,0.5*beta))
  final_action <- ((action + delta + 3) %% 4) + 1
  foo <- c(x,y) + unlist(action_deltas[final_action])
  foo <- pmax(c(1,1),pmin(foo,c(H,W)))

  return (foo)
}
```

Implementation of the Q-learning function

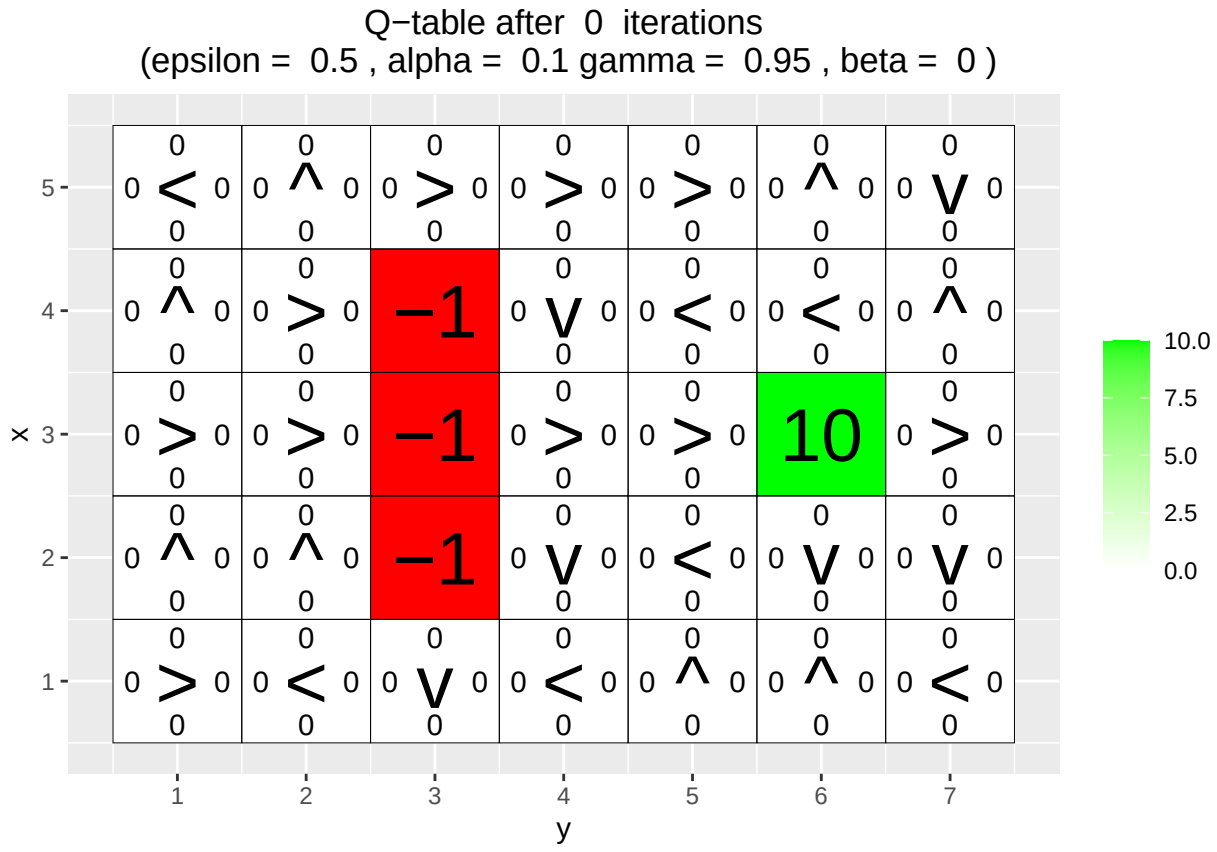
```
q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95,
                       beta = 0){
  episode_correction = 0
  # Perform one episode of Q-learning. The agent should move around in the
  # environment using the given transition model and update the Q-table.
  # The episode ends when the agent reaches a terminal state.
  #
```

```
  # Args:
  #   start_state: array with two entries, describing the starting position of the agent.
  #   epsilon (optional): probability of acting randomly.
  #   alpha (optional): learning rate.
  #   gamma (optional): discount factor.
  #   beta (optional): slipping factor.
  #   reward_map (global variable): a HxW array containing the reward given at each state.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #
  # Returns:
  #   reward: reward received in the episode.
  #   correction: sum of the temporal difference correction terms over the episode.
  #   q_table (global variable): Recall that R passes arguments by value. So, q_table being
  #   a global variable can be modified with the superassigment operator <<-.

  # Your code here.

  repeat{
    # Follow policy, execute action, get reward.

    action = EpsilonGreedyPolicy(start_state[1], start_state[2], epsilon)

    new_state = transition_model(start_state[1], start_state[2], action, beta)

    reward = reward_map[new_state[1], new_state[2]]

    # Q-table update.

    td_correction = reward + gamma * max(q_table[new_state[1], new_state[2], ]) - q_table[start_state[1]

    q_table[start_state[1], start_state[2], action] <<- q_table[start_state[1], start_state[2], action]

    episode_correction = episode_correction + td_correction

    start_state <- new_state

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

}
```

Environment A

```
H <- 5
W <- 7

reward_map <- matrix(0, nrow = H, ncol = W)
reward_map[3,6] <- 10
reward_map[2:4,3] <- -1
```

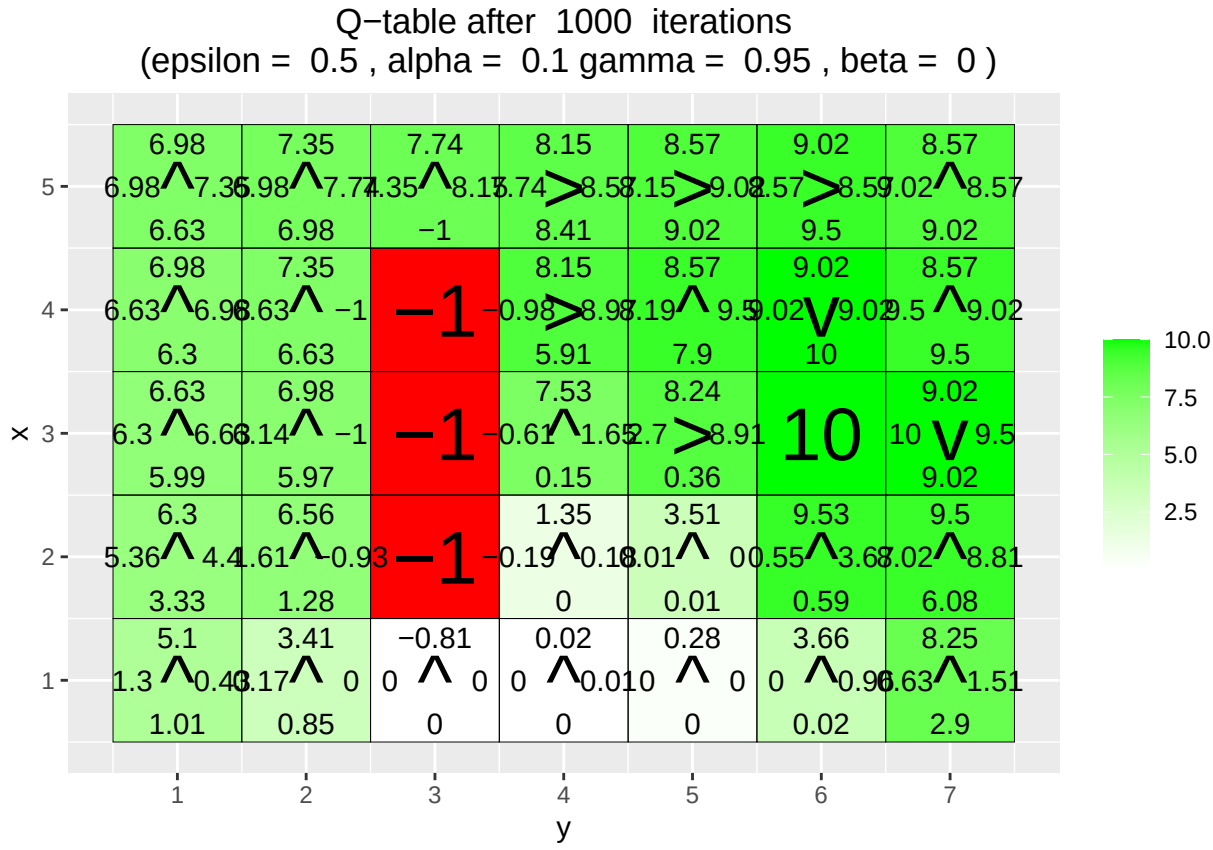
```
q_table <- array(0,dim = c(H,W,4))

vis_environment()
```

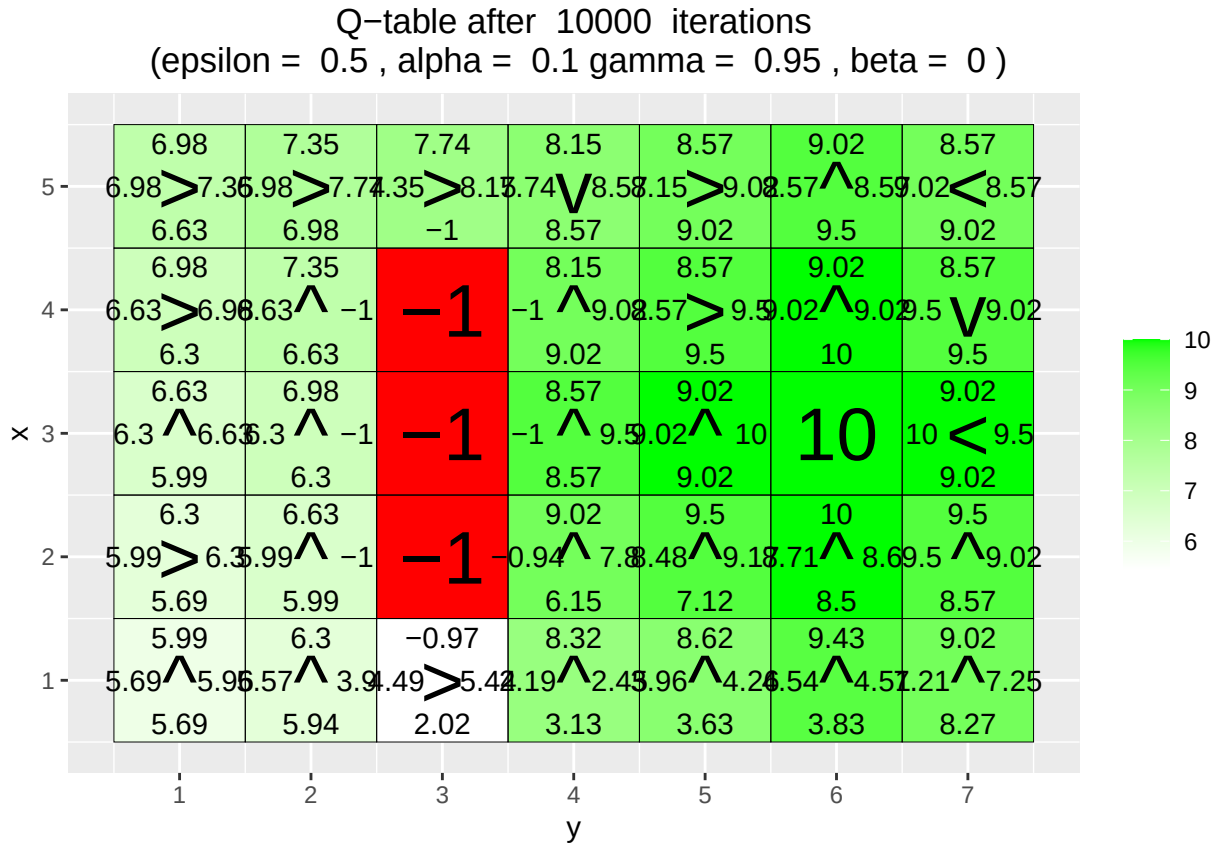



```
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))

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


Q−table after 10 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
1
2
3
4
5
6
7
−0.1
−1
−0.34
−1
−0.27
−1
10
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
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)
6.98 7.35 7.74 8.15 8.57 9.02 8.57
6.98 7.35 −1 8.41 9.02 9.5 9.02
6.98 7.35 −1 8.15 8.57 9.02 8.57
6.63 6.63 −1 5.91 7.9 10 9.5
6.63 6.98 −1 7.53 8.24 10 9.02
6.3 5.97 −1 0.15 0.36 10 9.02
6.3 6.56 −1 1.35 3.51 9.53 9.5
3.33 1.28 −1 0 0.01 0.59 6.08
5.1 3.41 −0.81 0.02 0.28 3.66 8.25
1.01 0.85 0 0 0 0.02 2.9
x
y
1
2
3
4
5
6
7
10.0
7.5
5.0
2.5



What has the agent learned after the first 10 episodes? After the first 10 episodes the agent has learned some information about the environment, but its knowledge is still very limited. Some of the q-values around the negative reward has negative values, implying that the agent has learned that it is not beneficial to move there, and same for some of the q-values around the positive reward which now have positive values after 10 iterations. But the agent is still missing a lot of information.

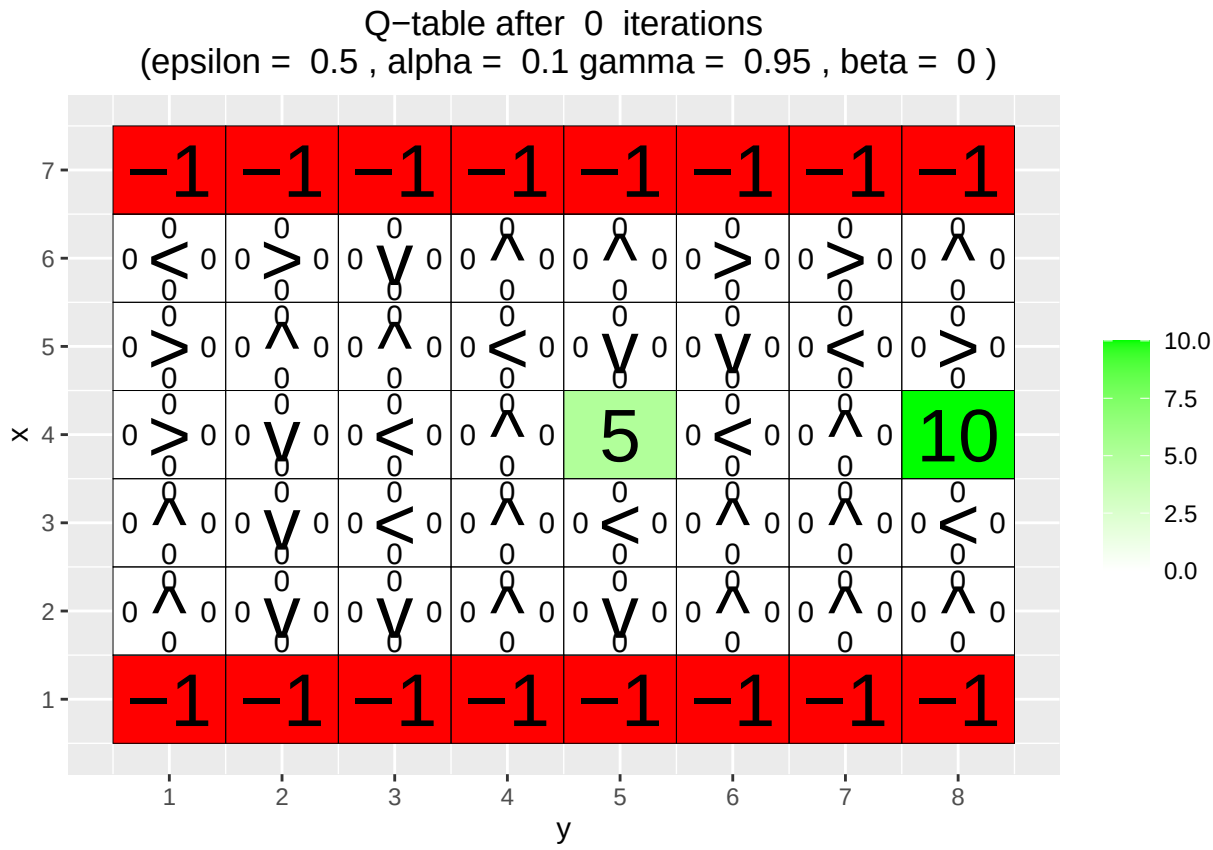
Is the final greedy policy (after 10000 episodes) optimal for all states, i.e. not only for the initial state? Why / Why not? The learned greedy policy after 10000 looks good but not optimal for all states. The agent has learned most of the information about the system but for some states, e.g. (2,4) the q-values are not optimal, since moving to the left should result in a negative reward. For the arrows we can see that they are not always pointing in the direction of the highest q-value, this because we use an epsilon greedy policy.

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? No, the learned Q-values only reflect the path above the negative reward. To get the Q-table to also show the path below the negative reward one could increase the epsilon-value, leading to more exploration of alternative paths.

Environment B

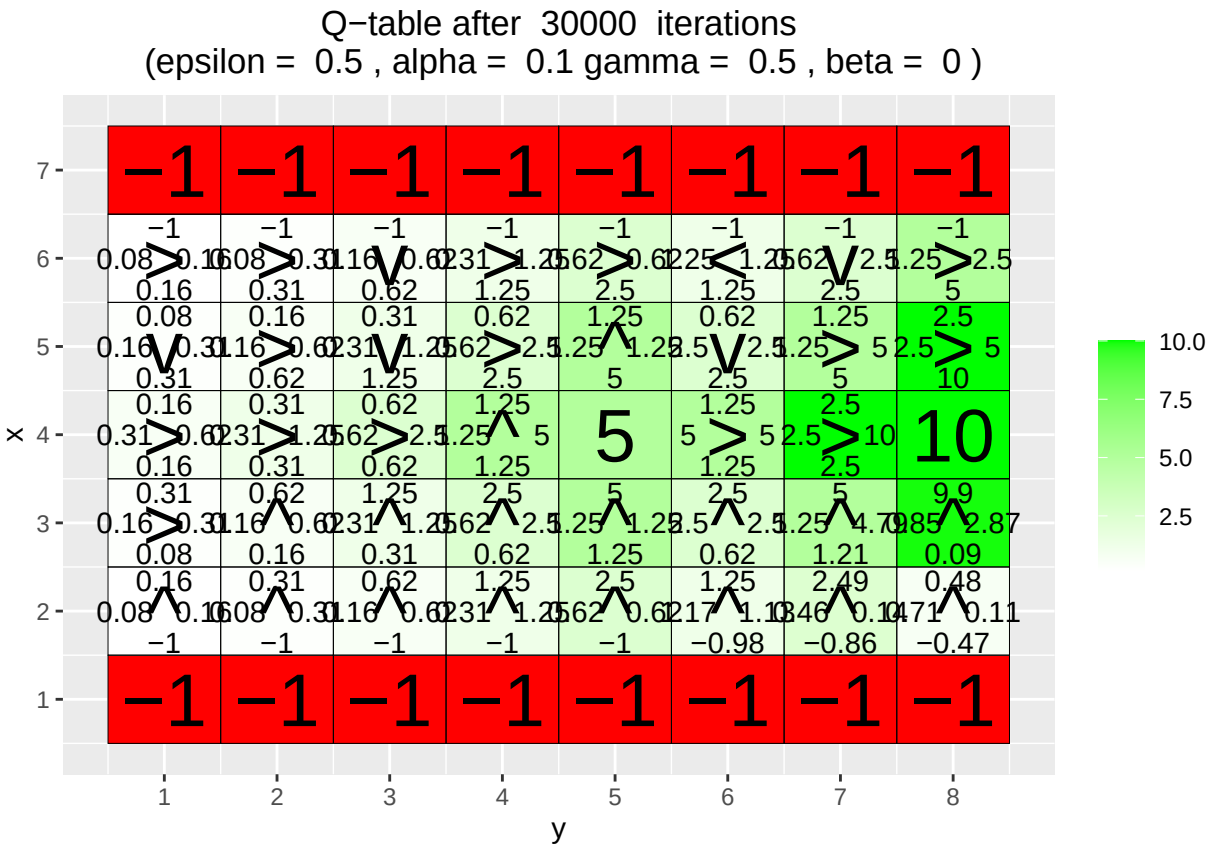
```
H <- 7
W <- 8

reward_map <- matrix(0, nrow = H, ncol = W)
```

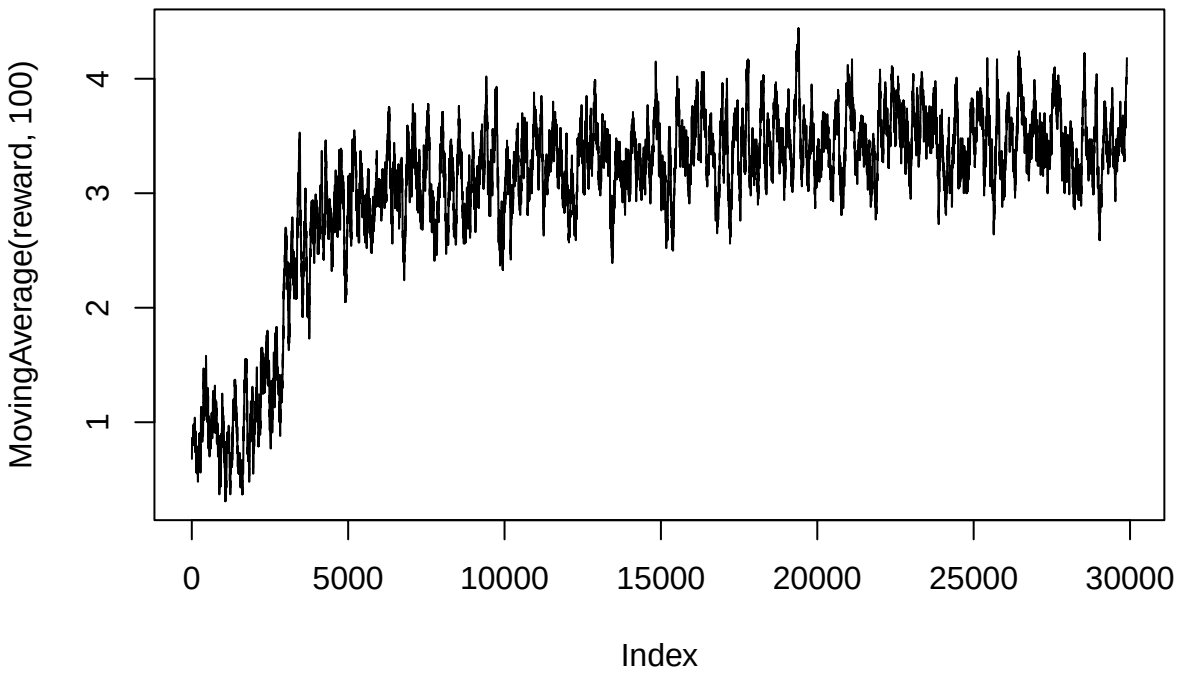
```
reward_map[1,] <- -1
reward_map[7,] <- -1
reward_map[4,5] <- 5
reward_map[4,8] <- 10

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

vis_environment()
```

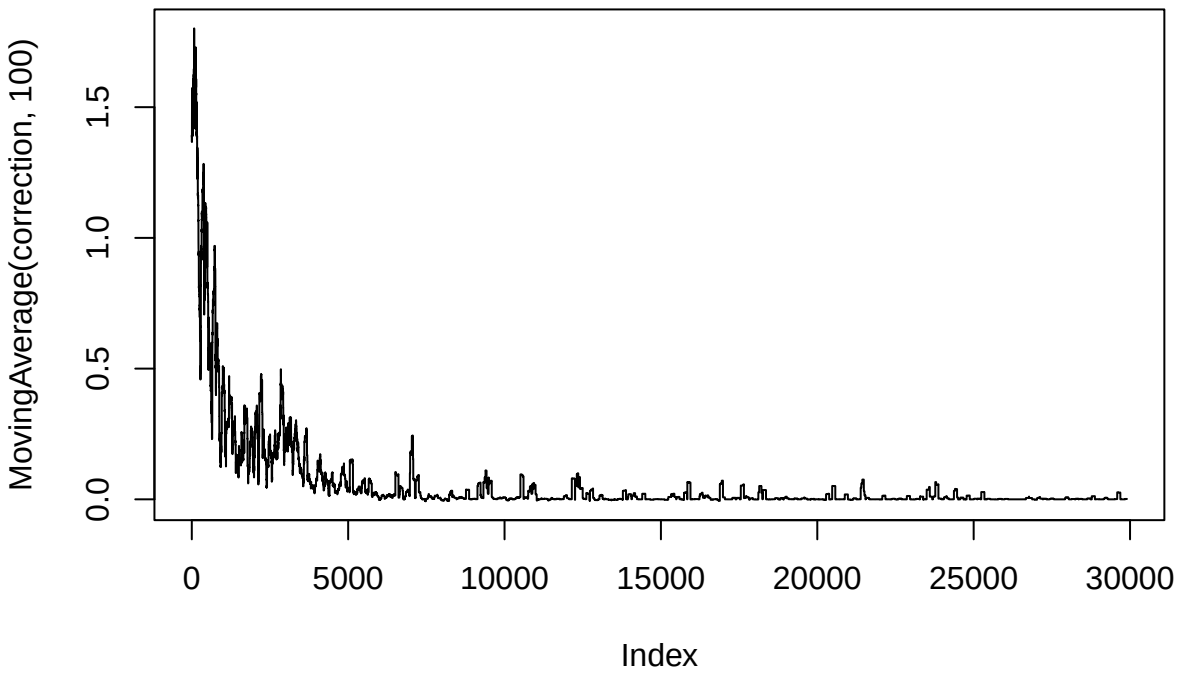


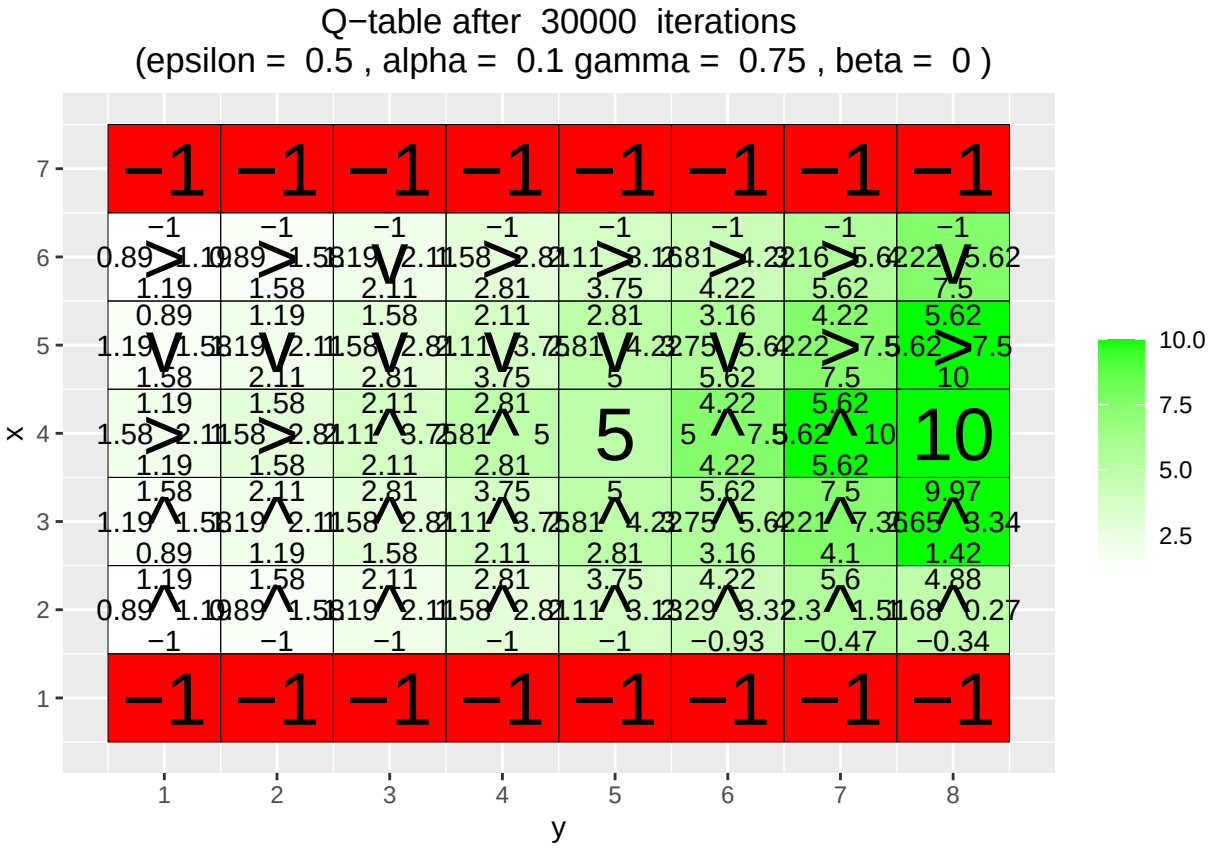

```
MovingAverage <- function(x, n){

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

  return (rsum)
}

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

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

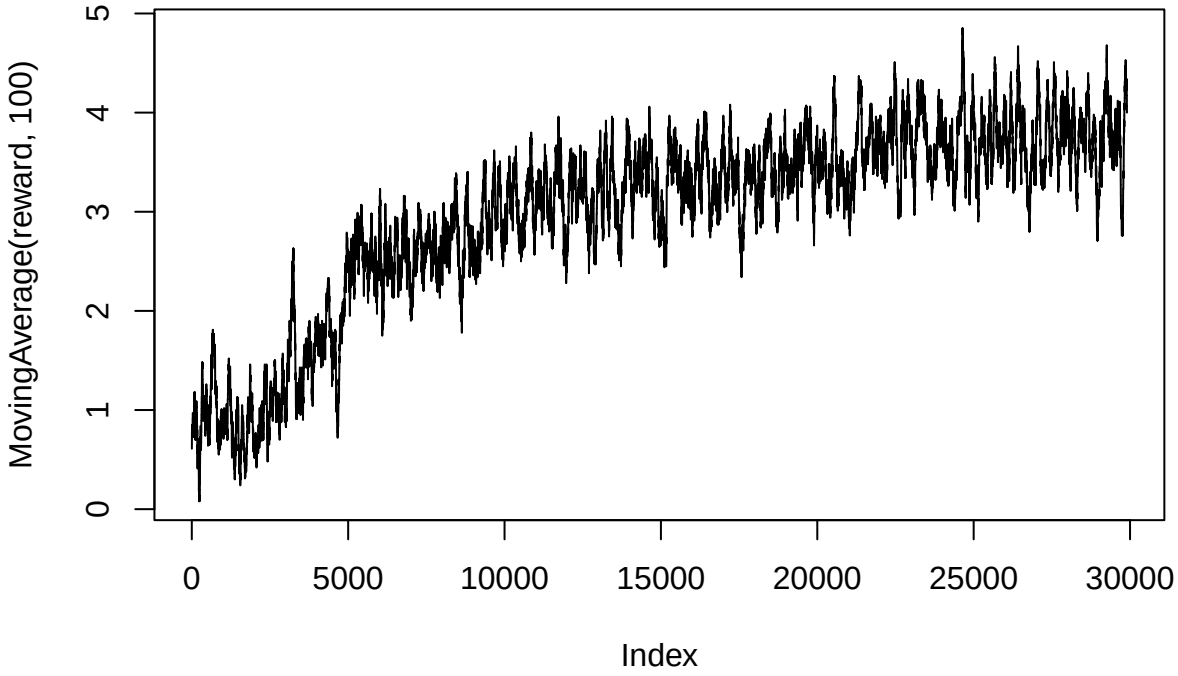
```
    correction <- c(correction,foo[2])
  }

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

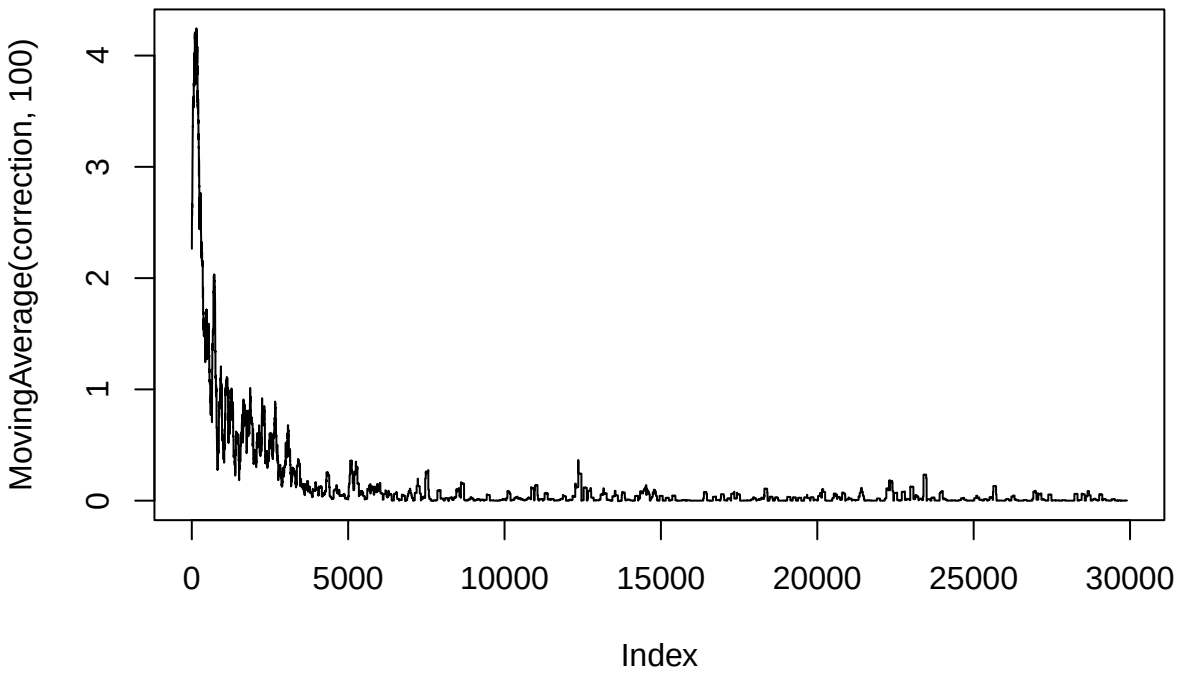


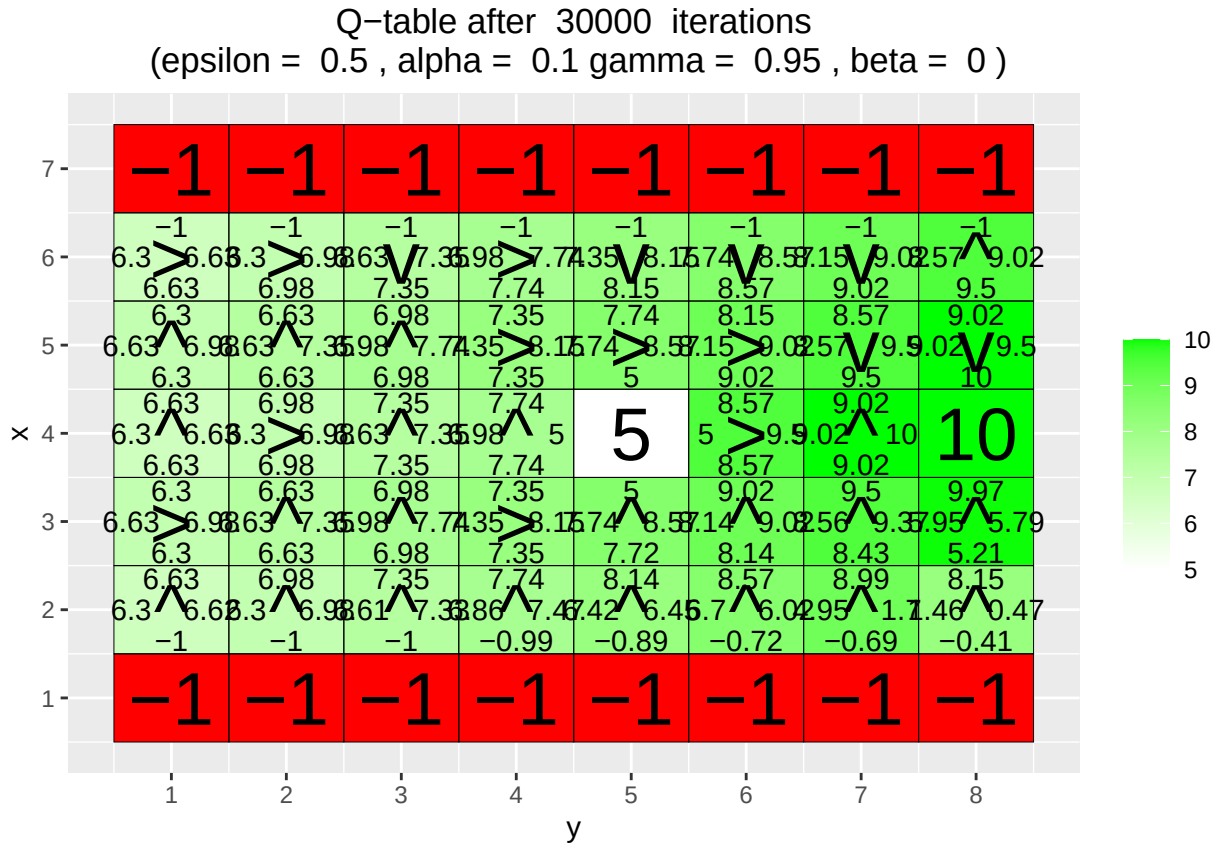

MovingAverage(reward, 100)
1
2
3
4
0
5000
10000
15000
20000
25000
30000
Index

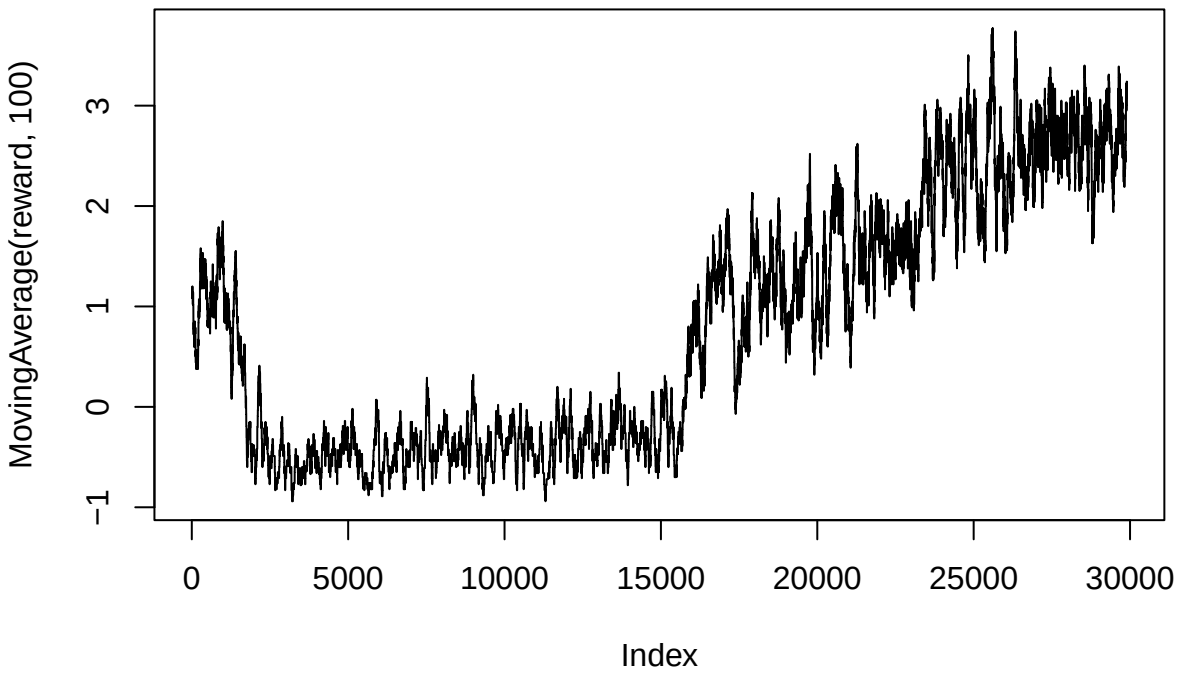

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
0
5000
10000
15000
20000
25000
30000
Index

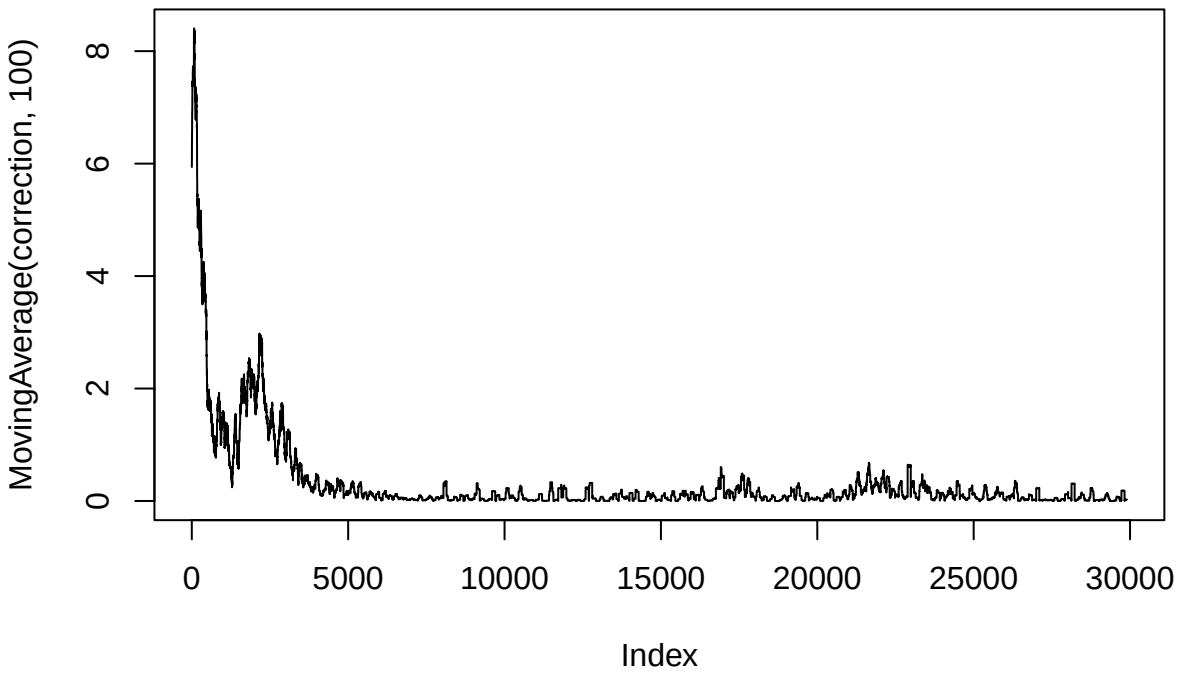

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5


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

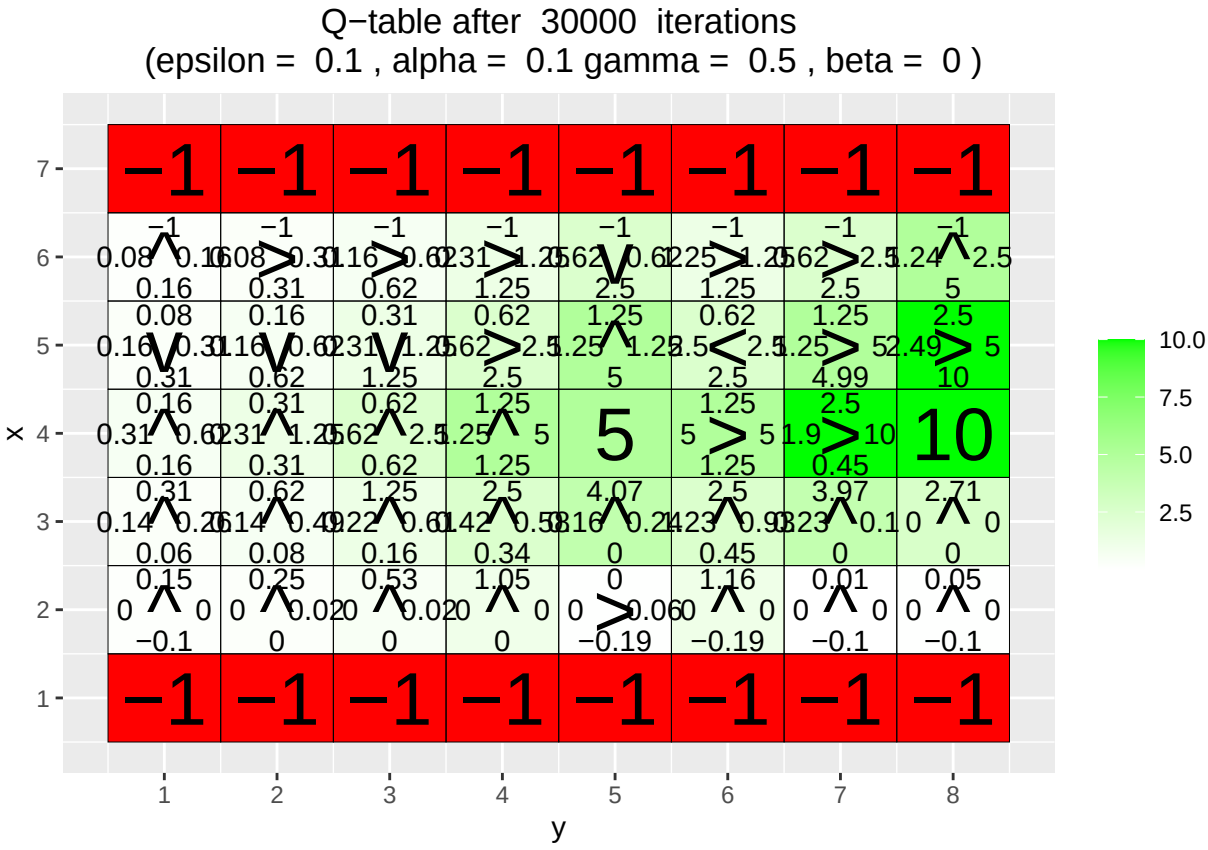

MovingAverage(correction, 100)
0
1
2
3
4
0
5000
10000
15000
20000
25000
30000
Index

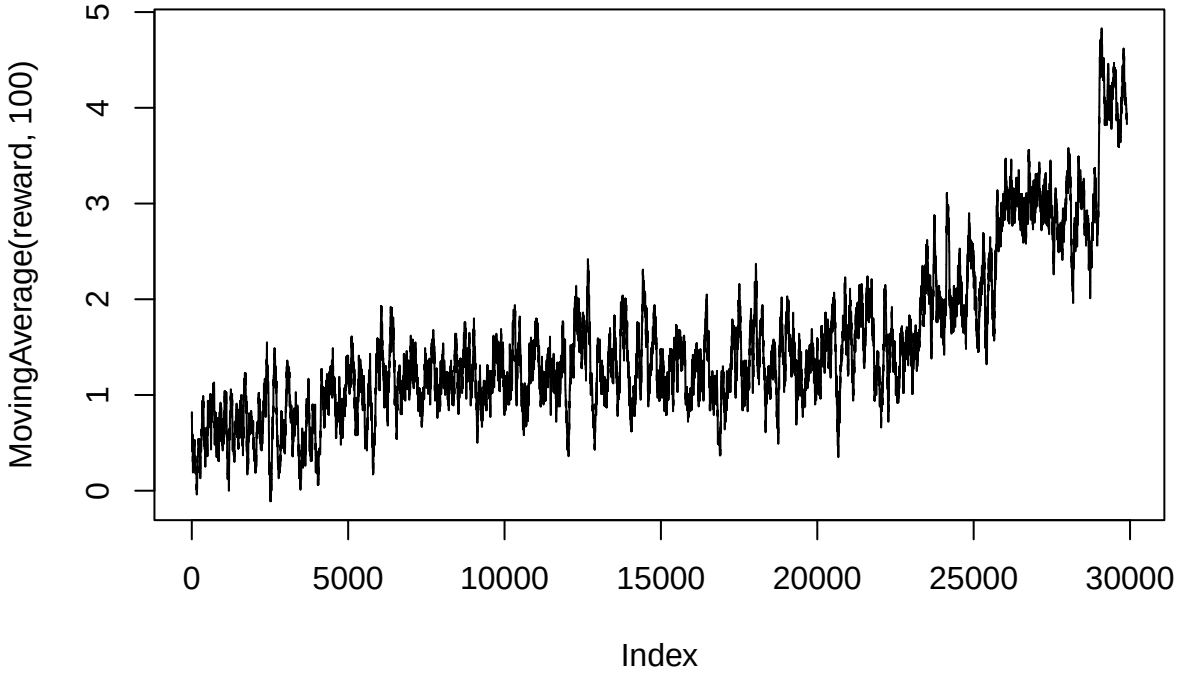

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
5
10

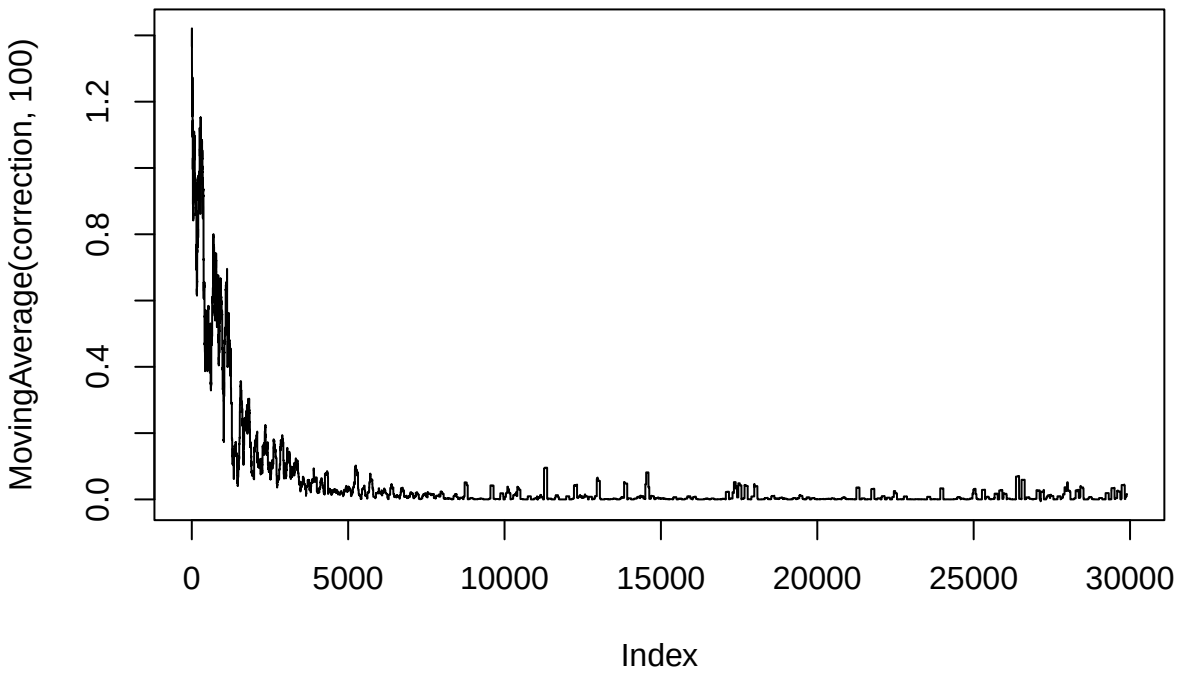

MovingAverage(reward, 100)
−1
0
1
2
3
0
5000
10000
15000
20000
25000
30000
Index

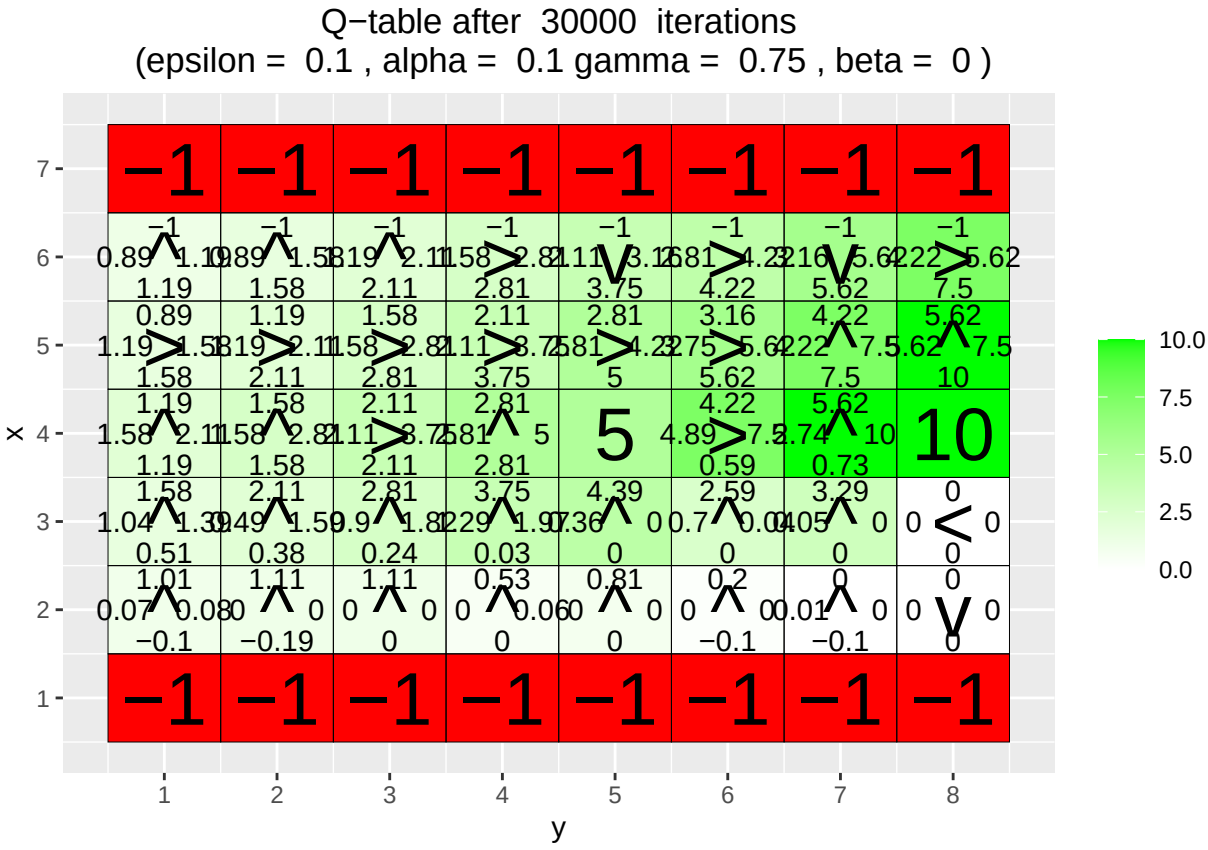


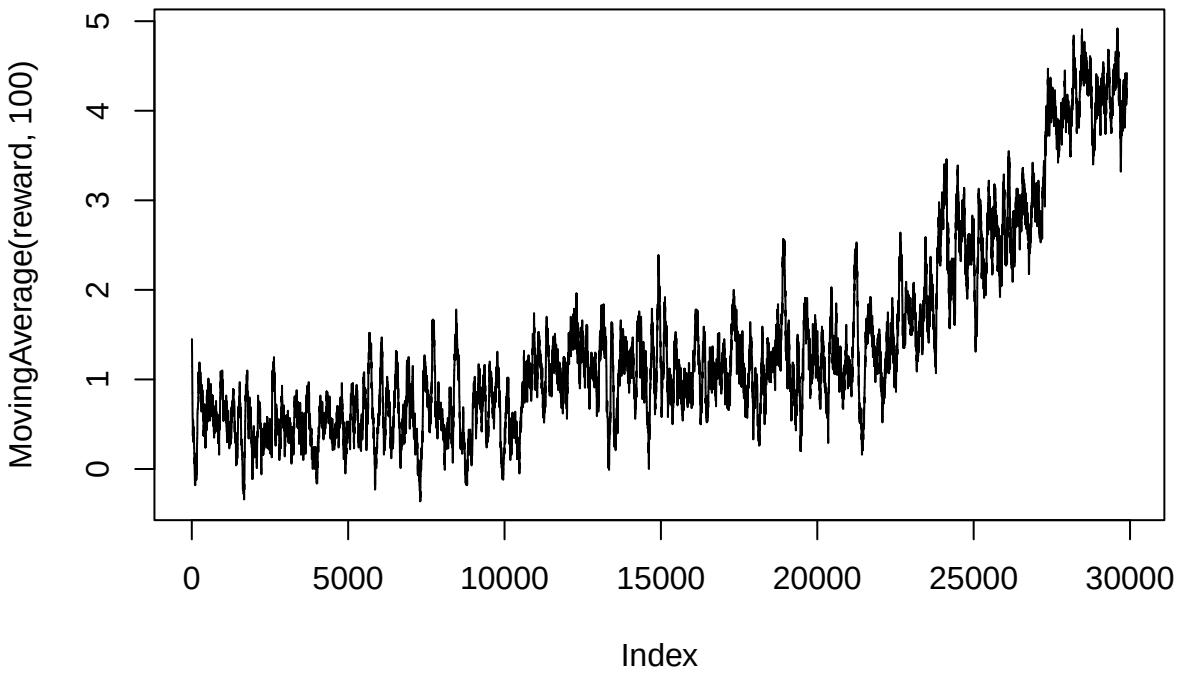

```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  for(i in 1:30000){
    foo <- q_learning(epsilon = 0.1, gamma = j, start_state = c(4,1))
    reward <- c(reward,foo[1])
    correction <- c(correction,foo[2])
  }

  vis_environment(i, epsilon = 0.1, gamma = j)
  plot(MovingAverage(reward,100),type = "l")
  plot(MovingAverage(correction,100),type = "l")
}
```

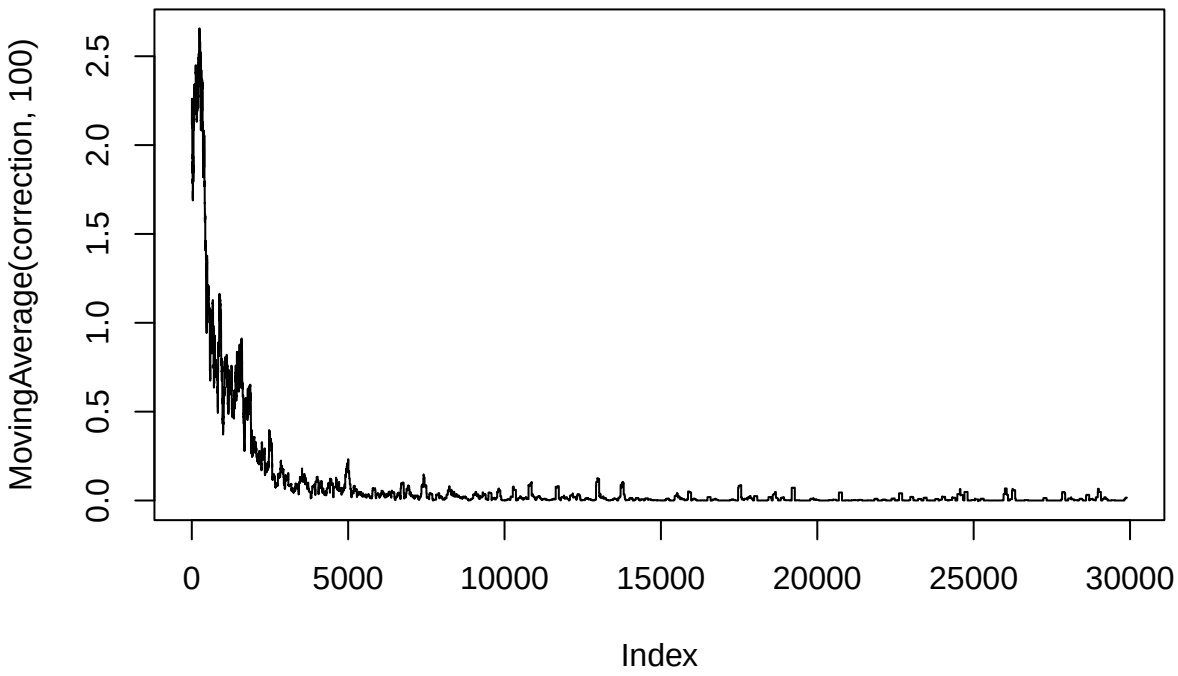

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10.0
7.5
5.0
2.5

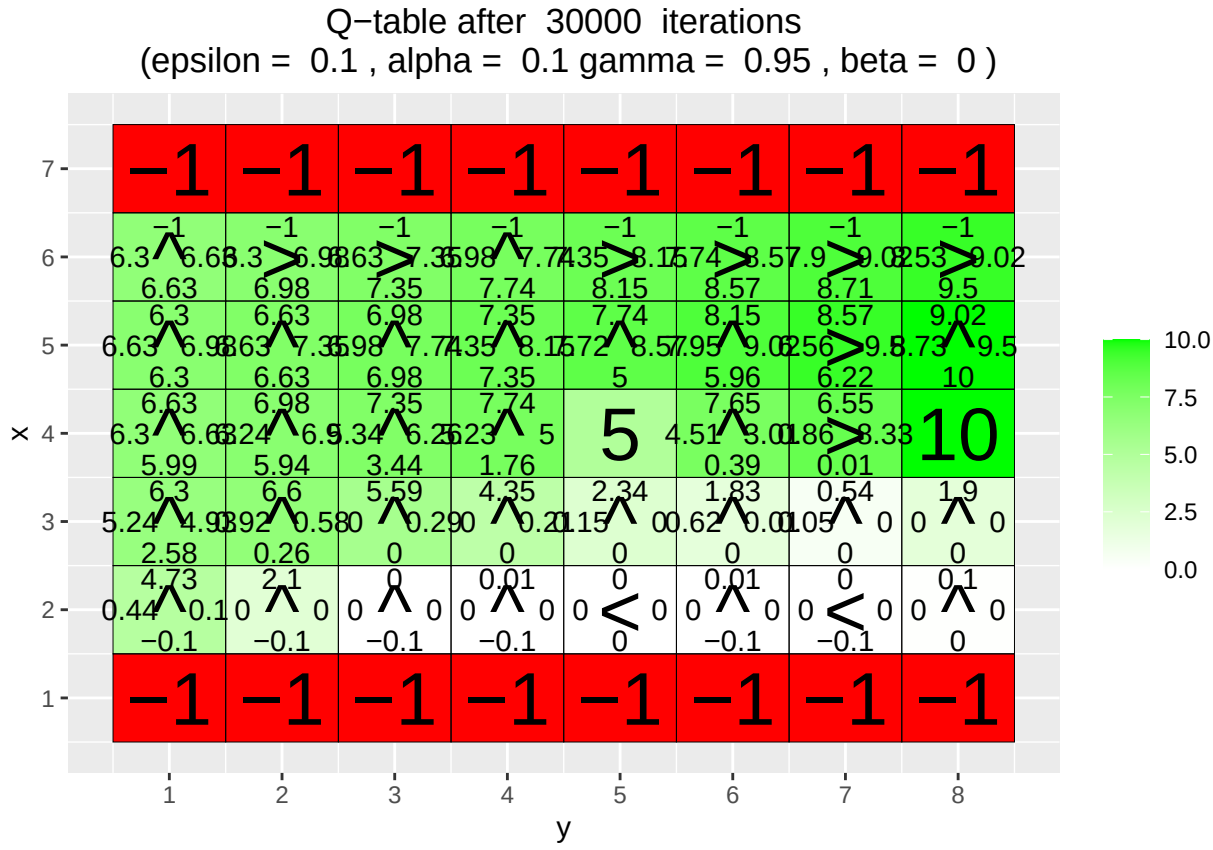

MovingAverage(reward, 100)
0
1
2
3
4
5
0
5000
10000
15000
20000
25000
30000
Index

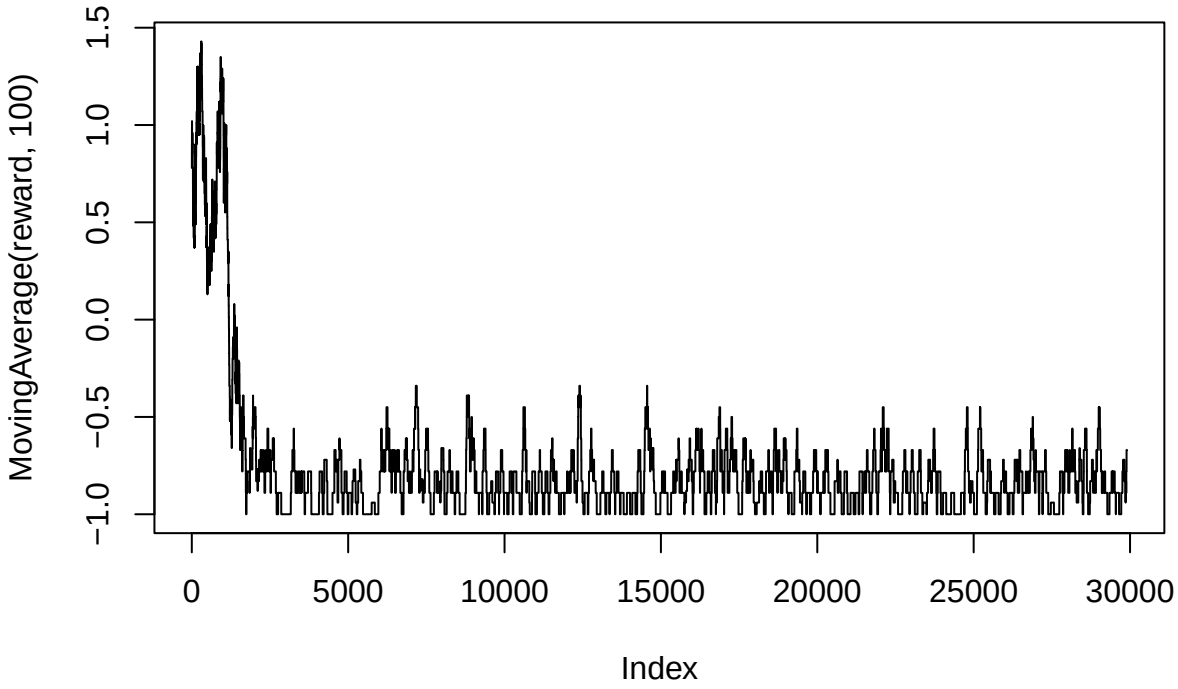

MovingAverage(correction, 100)
0.0
0.4
0.8
1.2
0
5000
10000
15000
20000
25000
30000
Index

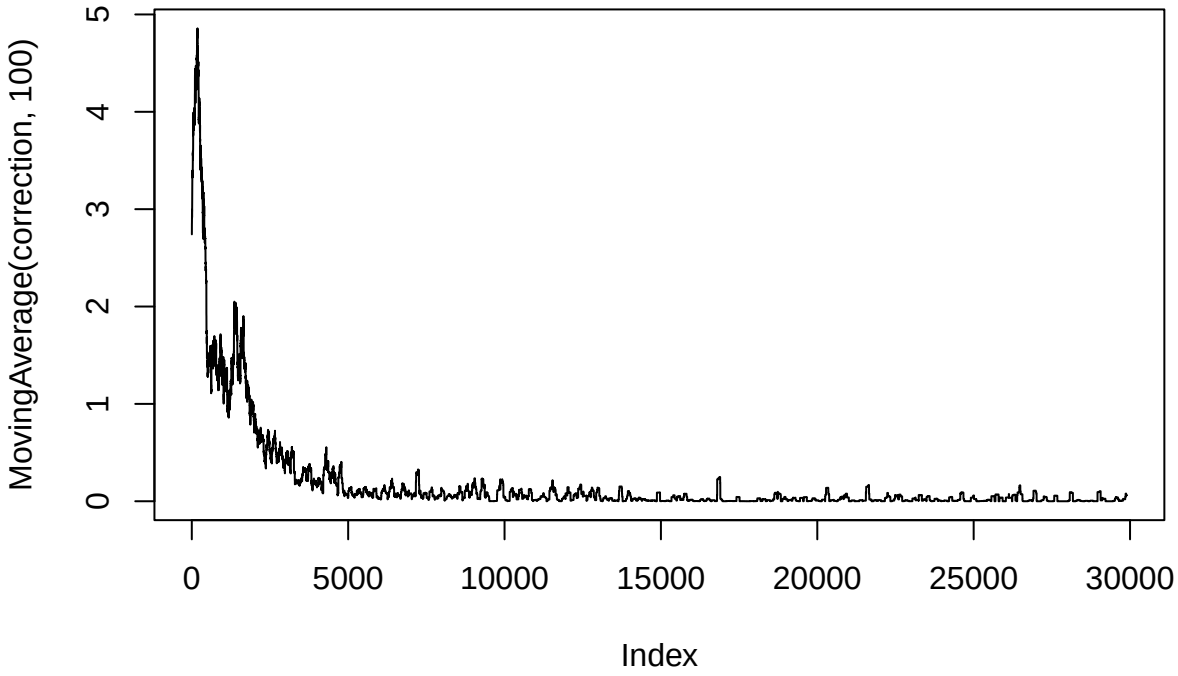

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0


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


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


Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0


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



Your task is to investigate how the epsilon and gamma parameters affect the learned policy by running 30000 episodes of Q-learning with epsilon = 0.1, 0.5, gamma = 0.5, 0.75, 0.95, beta = 0 and alpha = 0.1. Explain your observations.

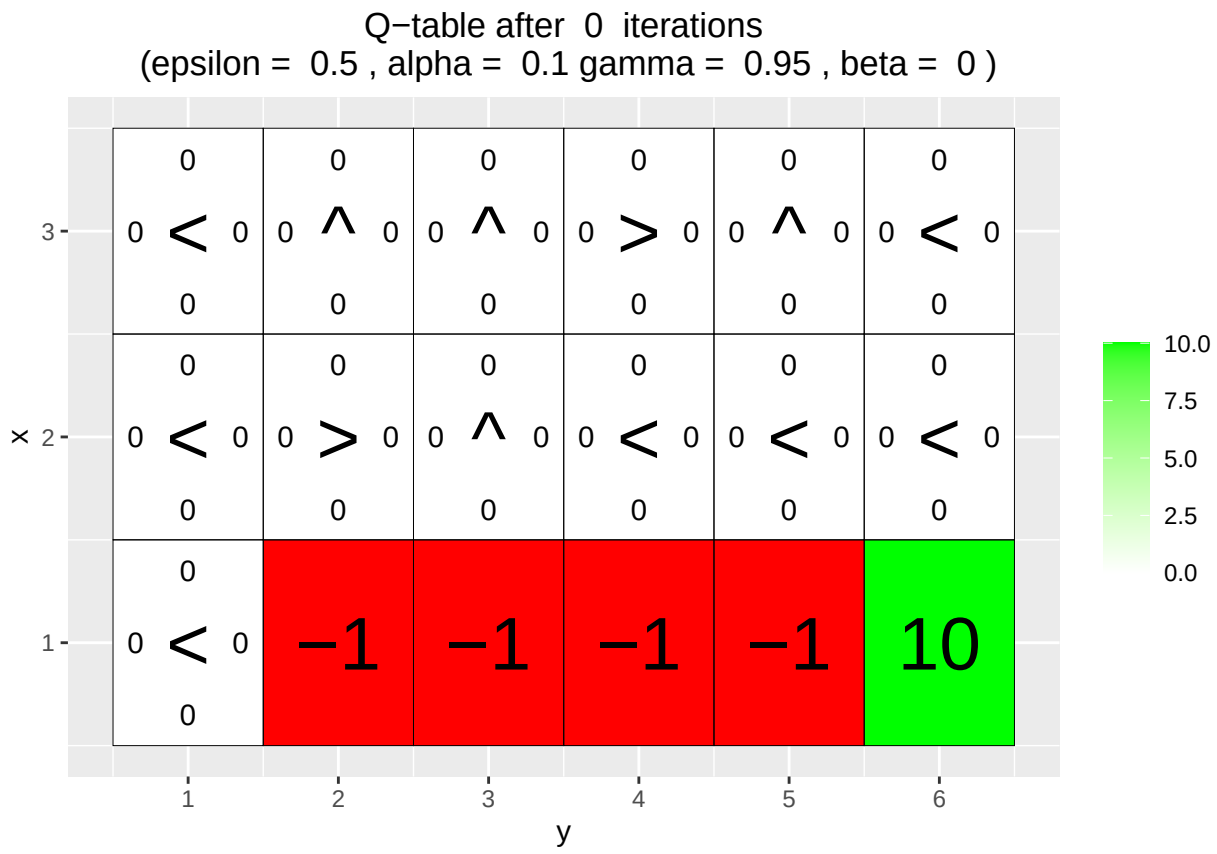
Epsilon affect the learned policy by controlling if the agent focuses more on trying new actions (exploration) or if it focuses more on action that it already known is good (exploitation). A higher epsilon means more exploration, and a lower epsilon means more exploit of the current best known policy. With an epsilon of 0.1 we can see from the reward plots that it takes more iterations for the reward to increase and stabilize. This since the agent focuses more on already known good actions and does not explore new ones that much. When epsilon is 0.5 we can see from the reward plots that they increase faster but are pretty unstable. This since the agent now focuses more on exploration of new paths.

The gamma parameter affect the learned policy by controlling how much the agent will focus on long- and short-time rewards. From the plots we can see that when gamma is 0.95 the agent focus on long time reward, which can be seen in the plot as learned policy does not lead to the reward of 5 in (4,5), instead it leads to the reward of 10 in (4,8). When the gamma is 0.75 the policy focuses on both long time and short time reward and the policy focus both on the reward of 5 and the reward of 10.

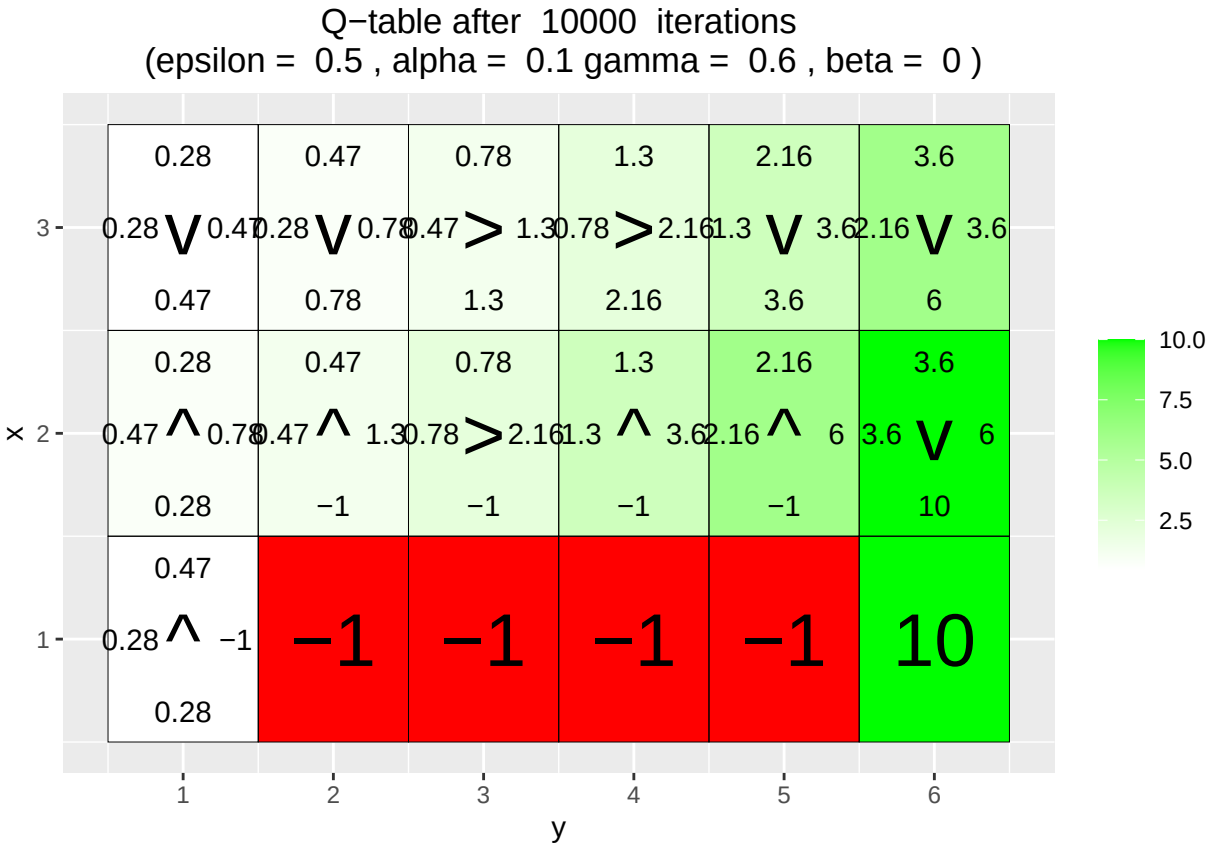
Environment C

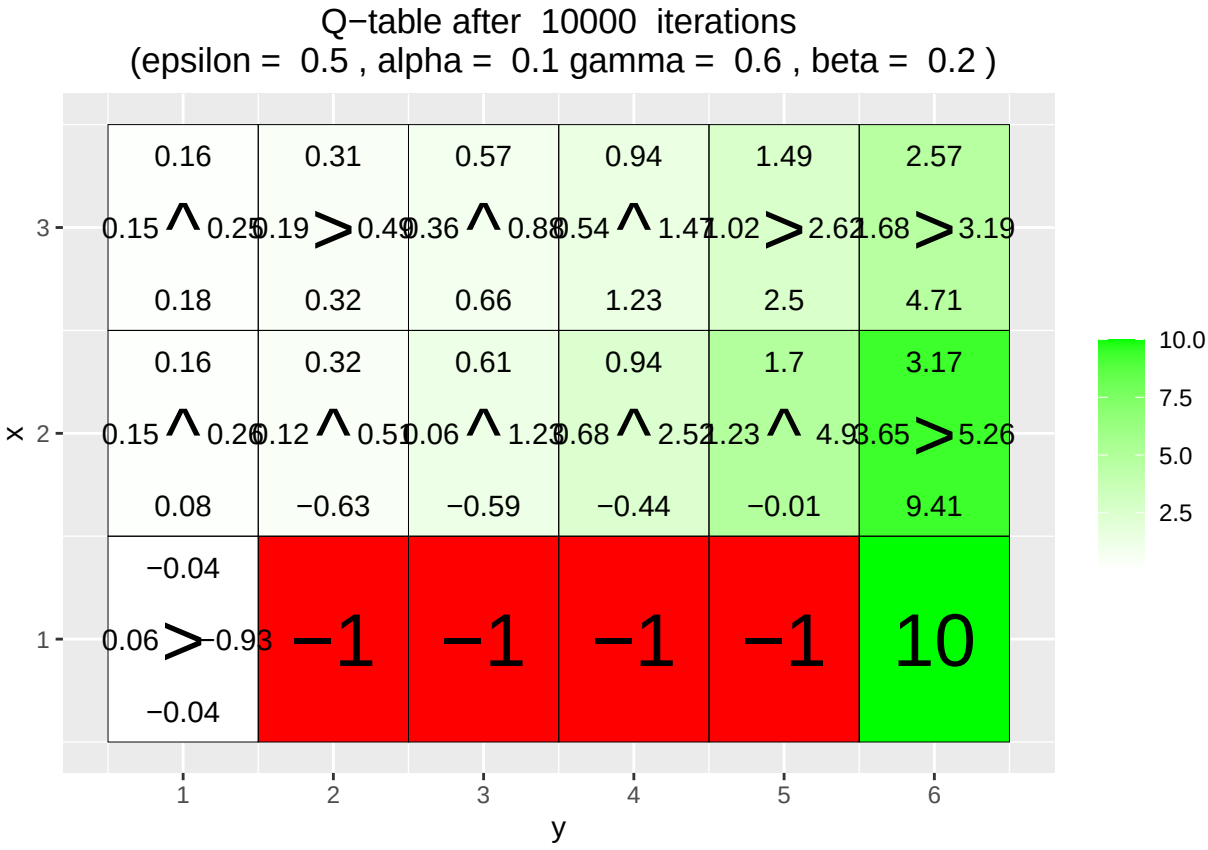
```
H <- 3
W <- 6

reward_map <- matrix(0, nrow = H, ncol = W)
reward_map[1,2:5] <- -1
```

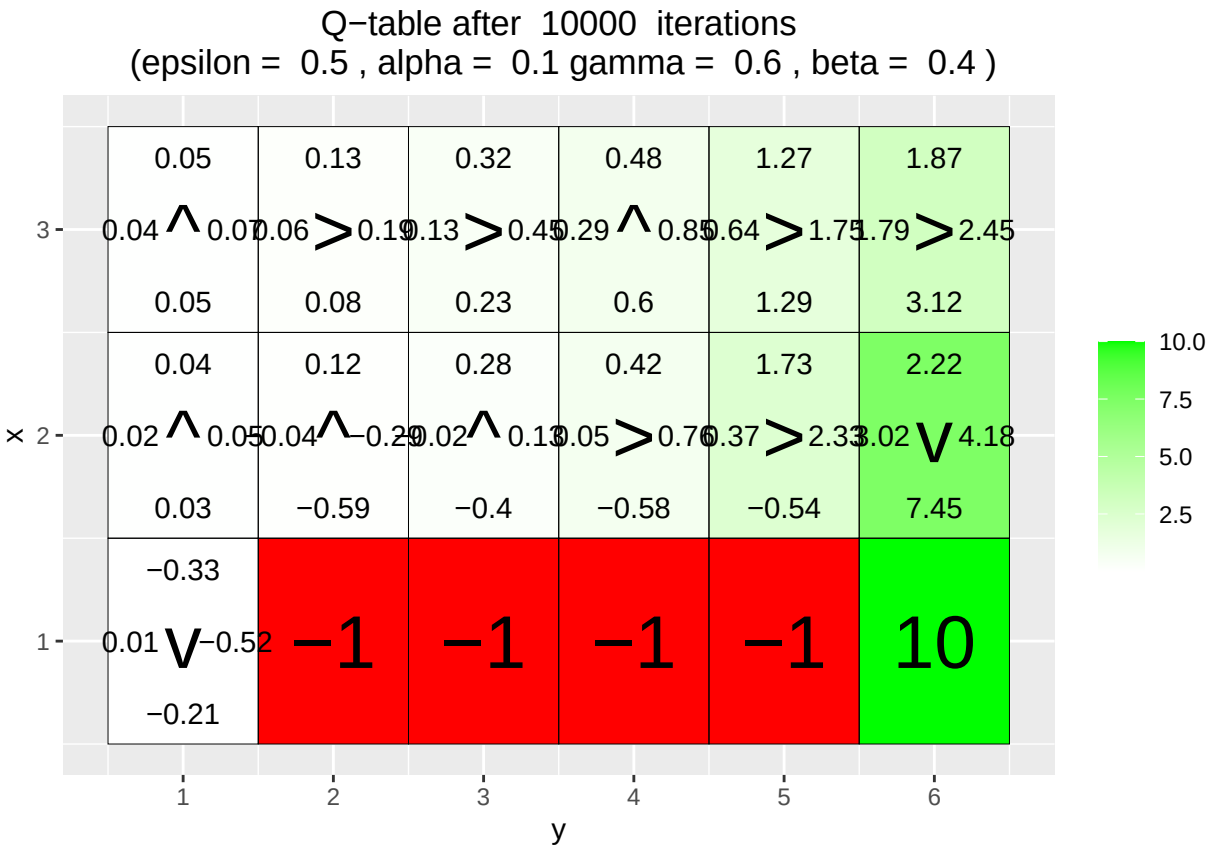
```
reward_map[1,6] <- 10

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

vis_environment()
```

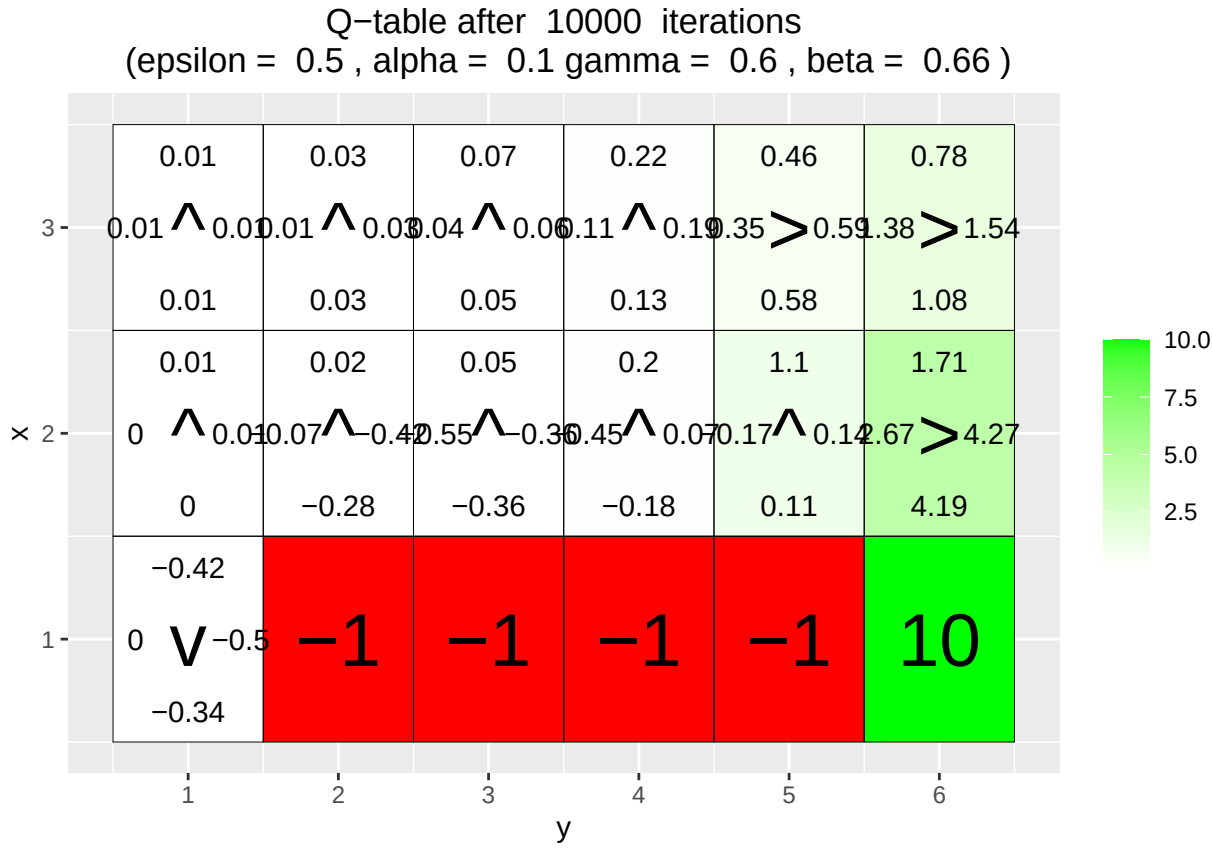



```
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

  for(i in 1:10000)
    foo <- q_learning(gamma = 0.6, beta = j, start_state = c(1,1))

  vis_environment(i, gamma = 0.6, beta = j)
}
```


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0)
0.28
0.28 V 0.47
0.47
0.47
0.28 V 0.78
0.78
0.78
0.47 > 1.3
1.3
1.3
0.78 > 2.16
2.16
2.16
1.3 V 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
3
2
1
x
1
2
3
4
5
6
y
10.0
7.5
5.0
2.5


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)
0.16
0.15 ^ 0.25
0.18
0.31
0.19 > 0.49
0.32
0.57
0.36 ^ 0.88
0.66
0.94
0.54 ^ 1.47
1.23
1.49
1.02 > 2.62
2.5
2.57
1.68 > 3.19
4.71
0.16
0.15 ^ 0.26
0.08
0.32
0.12 ^ 0.51
−0.63
0.61
0.06 ^ 1.23
−0.59
0.94
0.68 ^ 2.52
−0.44
1.7
1.23 ^ 4.93
−0.01
3.17
3.65 > 5.26
9.41
−0.04
0.06 > −0.93
−0.04
−1
−1
−1
−1
10
x
1
2
3
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 = 0.4)
0.05
0.04 ^ 0.07
0.05
0.13
0.06 > 0.19
0.08
0.32
0.13 > 0.45
0.23
0.48
0.29 ^ 0.85
0.6
1.27
0.64 > 1.75
1.29
1.87
1.79 > 2.45
3.12
0.04
0.02 ^ 0.05
0.03
0.12
0.04 ^ −0.29
−0.59
0.28
0.02 ^ 0.13
−0.4
0.42
0.05 > 0.76
−0.58
1.73
0.37 > 2.33
−0.54
2.22
3.02 V 4.18
7.45
−0.33
0.01 V −0.52
−0.21
−1
−1
−1
−1
10
x
y
1
2
3
1
2
3
4
5
6
10.0
7.5
5.0
2.5



investigate how the beta parameter affects the learned policy by running 10000 episodes of Q-learning with beta = 0, 0.2, 0.4, 0.66, epsilon = 0.5, gamma = 0.6 and alpha = 0.1. Explain your observations.

The beta parameter affect the policy by adding randomness to the system, as the agent will follow th intended action with probability (1-beta) and take a step to left or right from the intended action with probability beta/2. This can be seen in the plots as the path towards the reward of 10 is not as clear when increasing the beta parameter.

Environment D

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

Yes, from the plots we can see that the agent has learned a good policy. This is because the both the training and validation goal positions are spread across the entire grid, so the agent can learn a policy that generalize well for the entire grid.

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

No, Q-learning could not been used to solve this task since the goal is moving, which would result in that every update of the Q-table would be useless when the target has moved.

Environment E

Has the agent learned a good policy? Why / Why not? No, the agent has not learned a good policy, this since during the training all goal positions where on the top row and then for the validation all goal

positions where below the top row. This resulted in that the agent has not learned to move downwards in the grid and can not find the goal if the agent is above the goal position.

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

Yes the results from environment D and environment E differ since in D the agent is trained with the goal located anywhere on the grid, while for environment E the agent is only trained with the goal located on the top row of the grid. This results in that the agent does not learn to move downwards to reach the goal.