Advanced Machine Learning - Reinforcement Learning

Ali Etminan

10/8/2020

Q Learning

Policy and Q-learning functions

As a prior step, we define three functions to complete the Q-learning algorithm.

Greedy Policy

The optimal action is chosen at all times.

```
GreedyPolicy <- function(x, y){
  r <- q_table[x,y,]
  best <- which(r == max(r))
  return(sample(best, 1))
}
```

Epsilon Greedy Policy

The optimal action is chosen with probability $1 - \epsilon$ and a random action with probability ϵ.

```
EpsilonGreedyPolicy <- function(x, y, epsilon){
  r <- q_table[x,y,]
  best <- which(r == max(r))
  if(runif(1) < epsilon) {
    actions <- c(1,2,3,4)
    return(sample(actions, 1))
  }
  else {
    return(sample(best, 1))
  }
}
```

Q-Learning

Now if we assume a deterministic transition model where s and a are always followed by s' and r then the optimal policy is

$$q_\star(s,a) = r + \gamma(\max_{a'} q_\star(s',a'))$$

which is only the correction term in the **Bellman Equation** and therefore

$$0 = r + \gamma(\max_{a'} q_\star(s',a')) - q_\star(s,a)$$

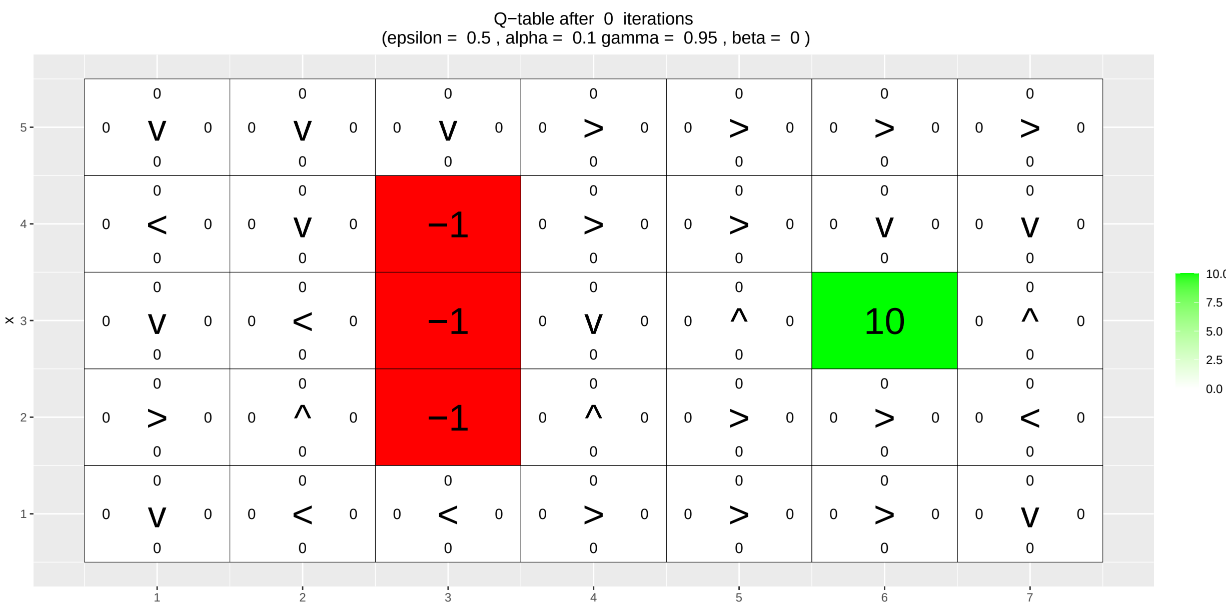
Through iteration we aim to enforce the above equation to hold or in other words, we iterate until there is no correction and we reach the optimal policy and update the q_table as follows

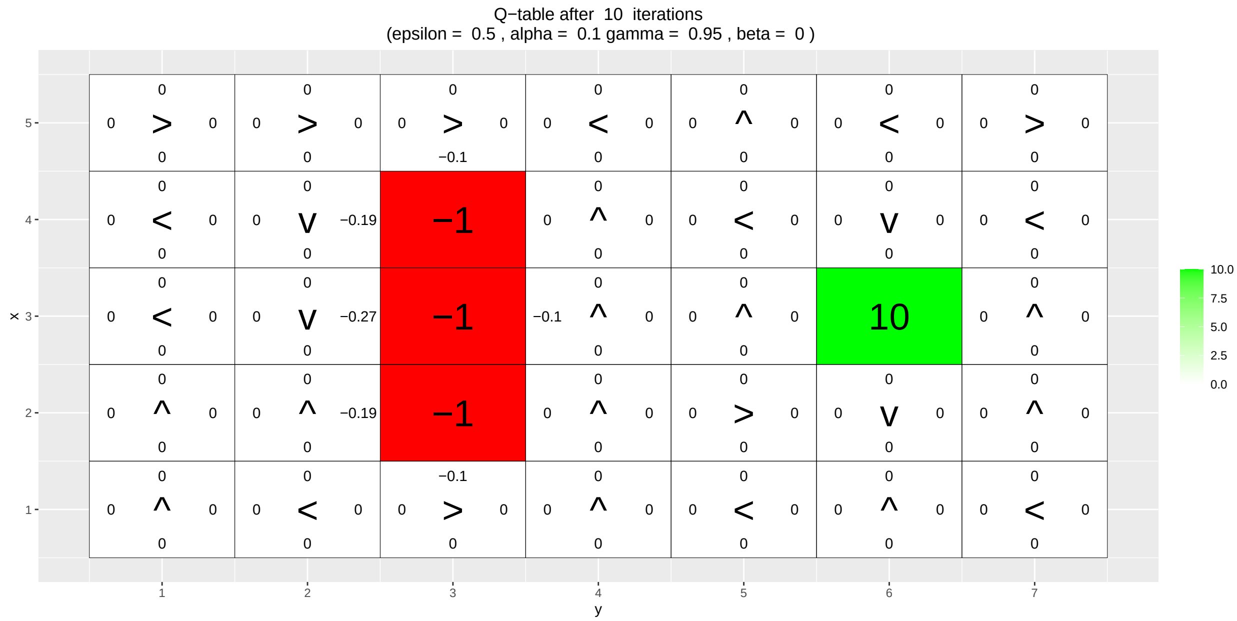
$$q_{\star}(s,a) = q_{\star}(s,a) + \alpha(r + \gamma(\max_{a'} q_{\star}(s',a')) - q_{\star}(s,a))$$

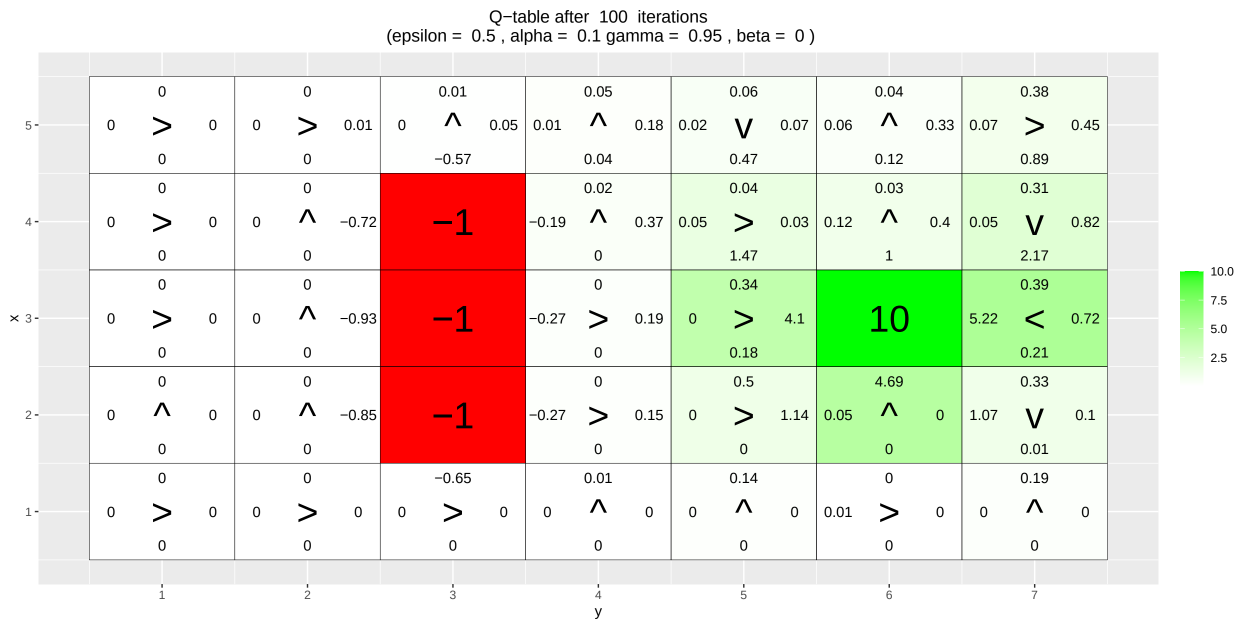
where α is the learning rate.

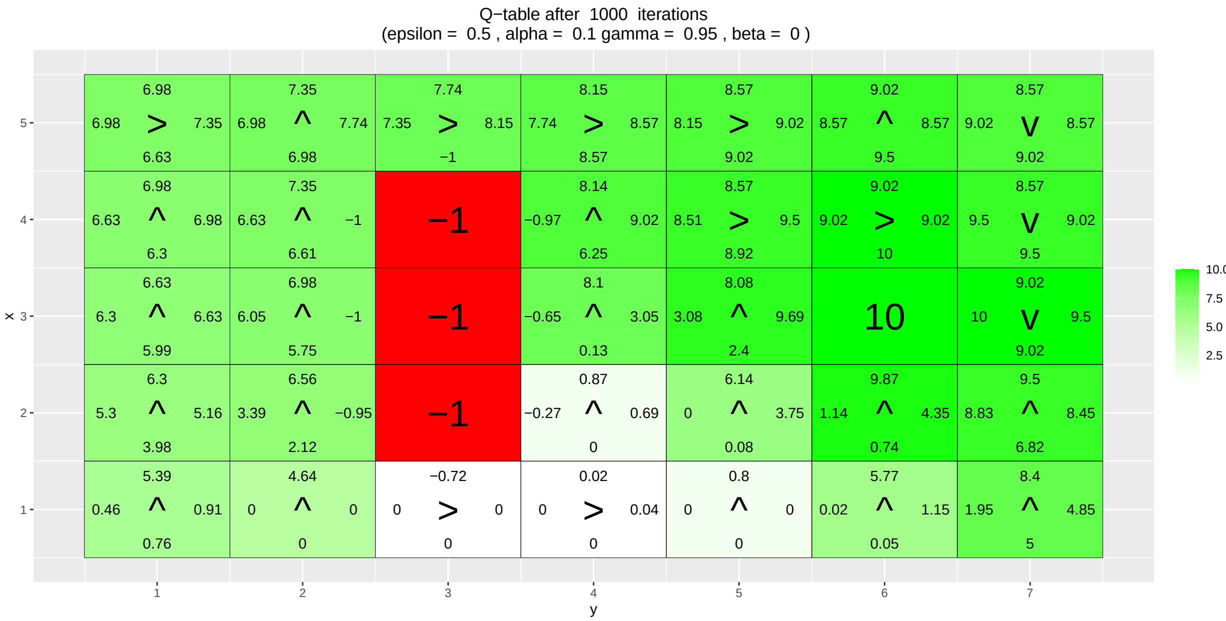
```
q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95, beta = 0){
  x <- start_state[1]
  y <- start_state[2]
  episode_correction <- 0

  repeat{
    # Follow policy, execute action, get reward.
    a <- GreedyPolicy(x,y)
    A <- EpsilonGreedyPolicy(x,y,epsilon)
    trstate <- transition_model(x,y,A,beta)
    reward <- reward_map[trstate[1],trstate[2]]

    # Q-table update.
    correction <- alpha*(reward+gamma*max(q_table[trstate[1],trstate[2],]) - q_table[x,y,A])
    episode_correction <- episode_correction + correction
    q_table[x,y,A] <<- q_table[x,y,A] + correction
    x <- trstate[1]
    y <- trstate[2]
    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))
  }
}
```

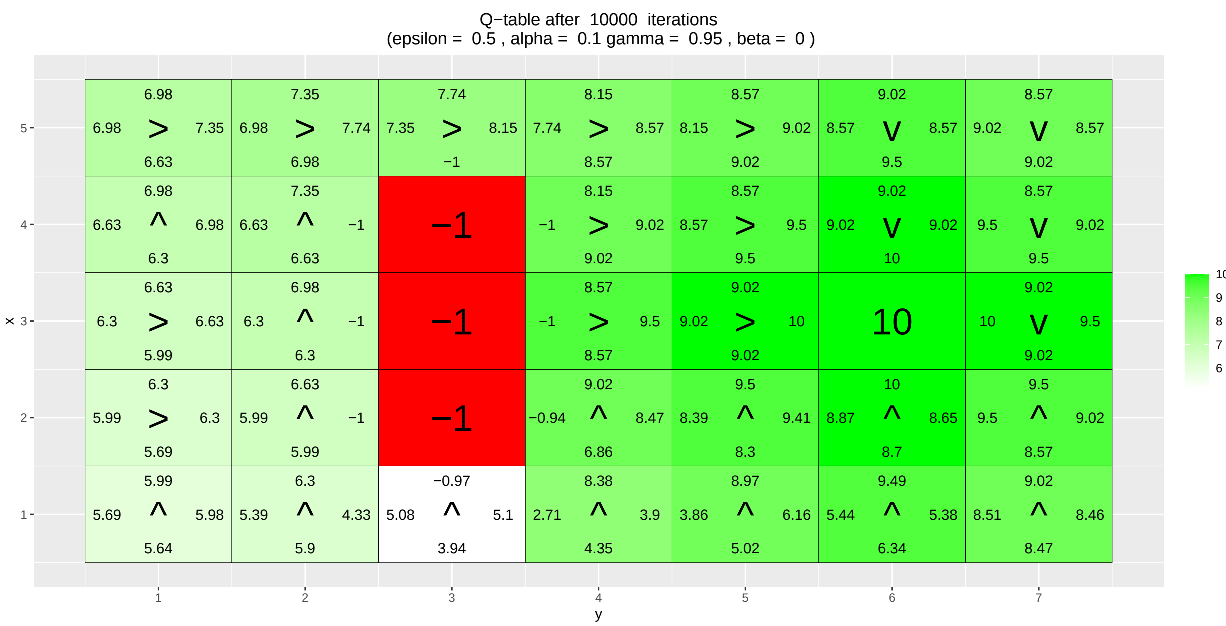
Environment A




Q−table after 10 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 100 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5





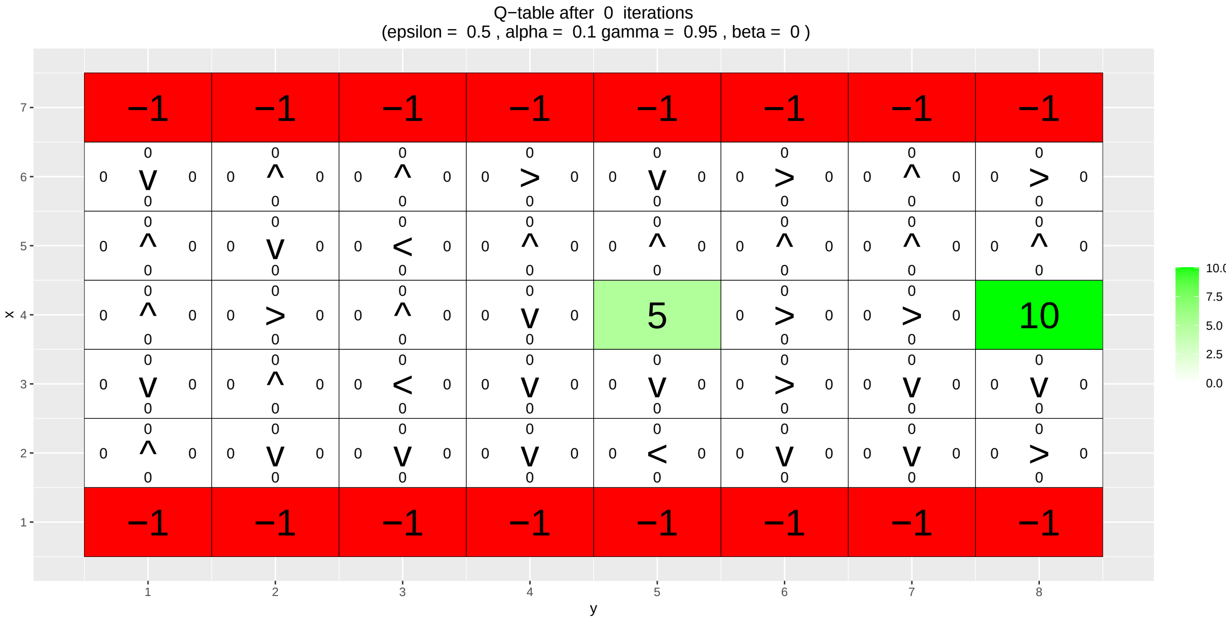
After only 10 episodes, the agent does not learn much and practically has not had the chance to explore the environment. However, it has slightly learned to avoid negative reward states.

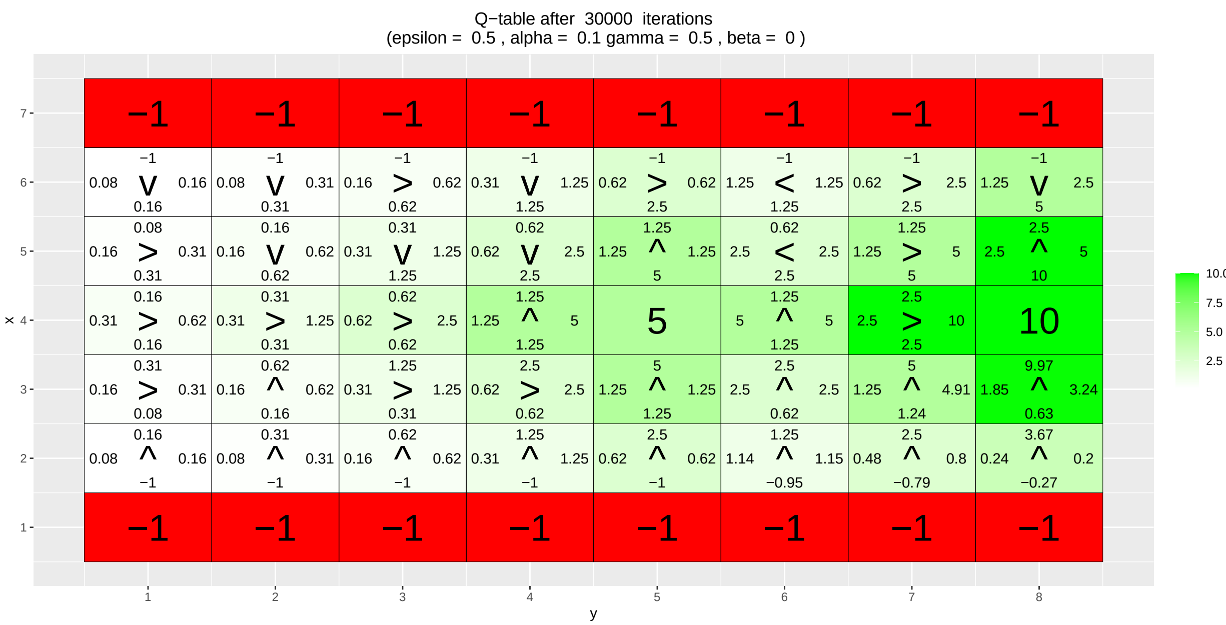
After 100 episodes, the q values for the negative states almost converge to -1 and the agent learns to ignore them. In addition, the agent has started to discover the target state.

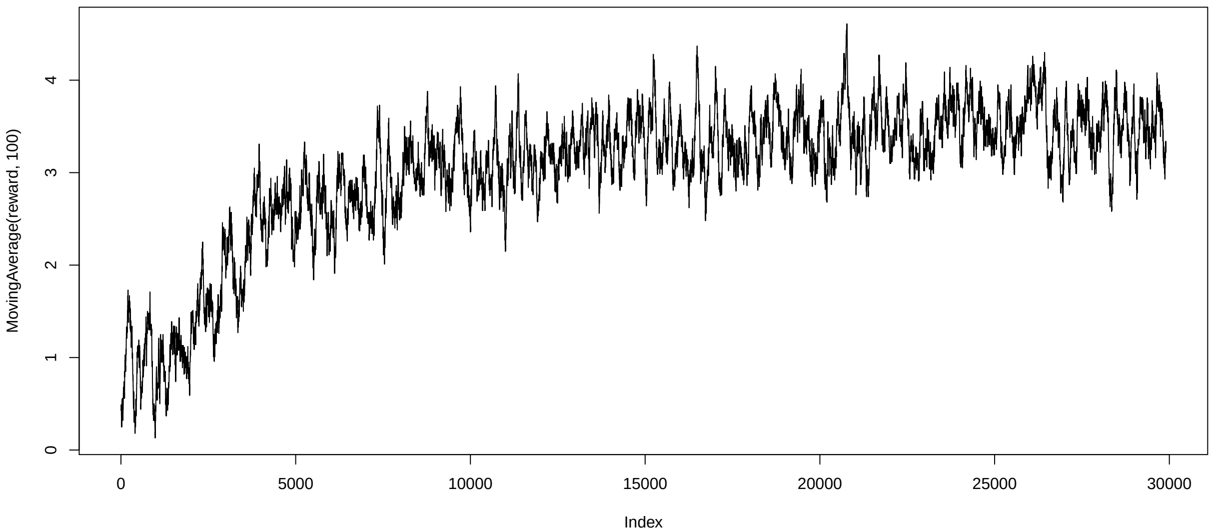
With 1000 and 10000 episodes, explores "almost" the entire map. It has tried several paths and the q values for the negative and target states have converged to the true reward value. In each episode, there is a 50% chance for the agent to prefer a random over the optimum action ($\epsilon = 0.5$). Therefore with more episodes the agent gets a bigger chance of exploring the environment.

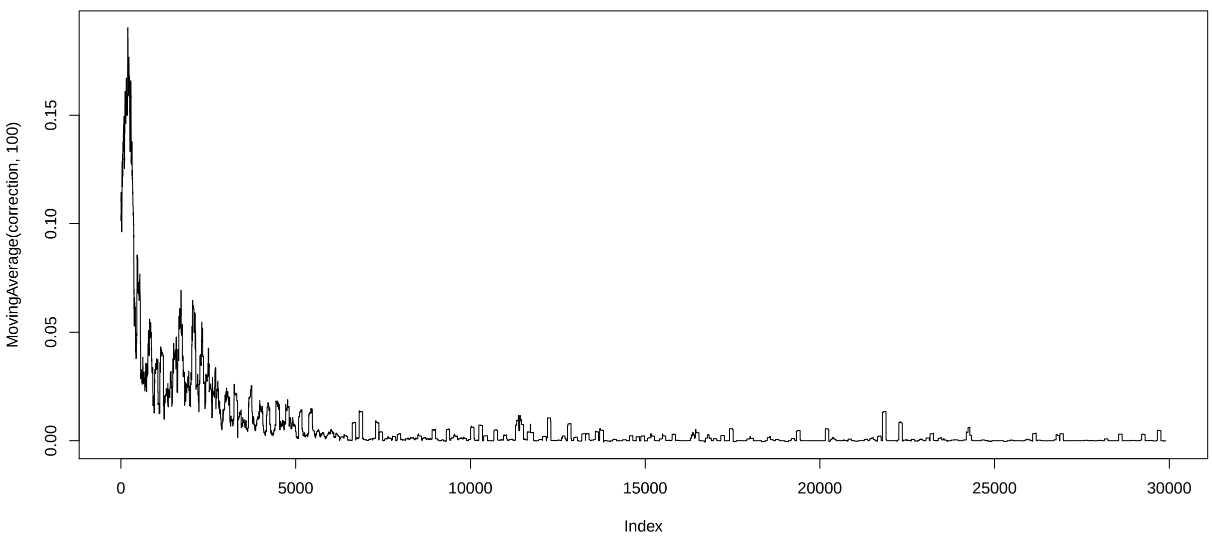
However, there is room for improvement and the obtained policy cannot be deemed optimal. In an optimal policy, the agent learns more paths and better explores the environment by increasing ϵ value. As an instance, when our agent is at any of the ([1,1],[1,2],[2,1],[2,2]) states, it takes the upper path and goes around the red states to reach the target which is not optimum.

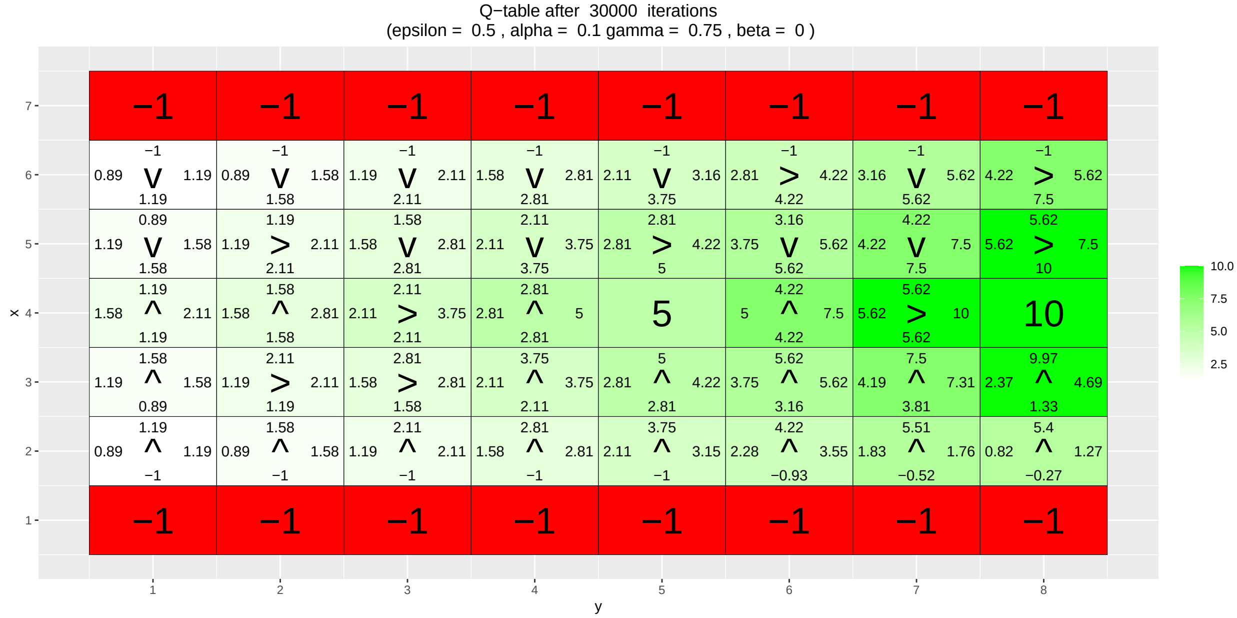
Environment B

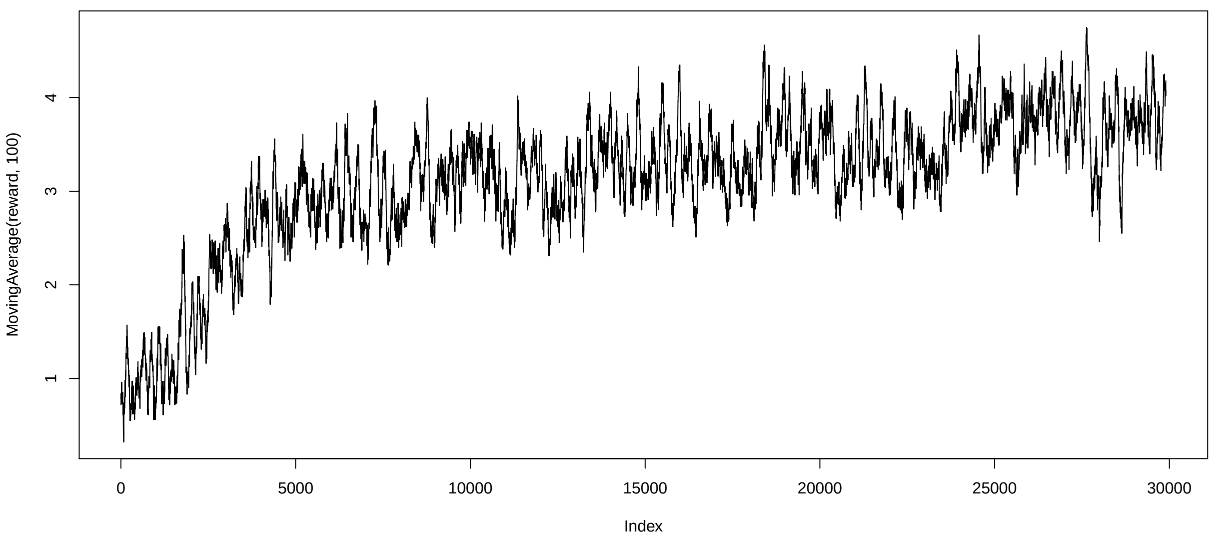


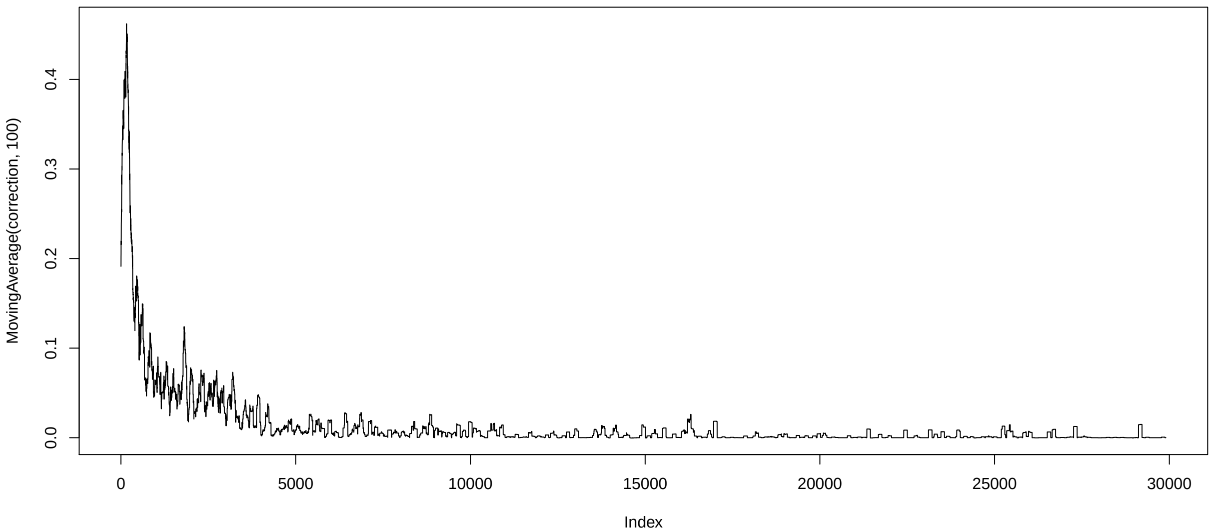


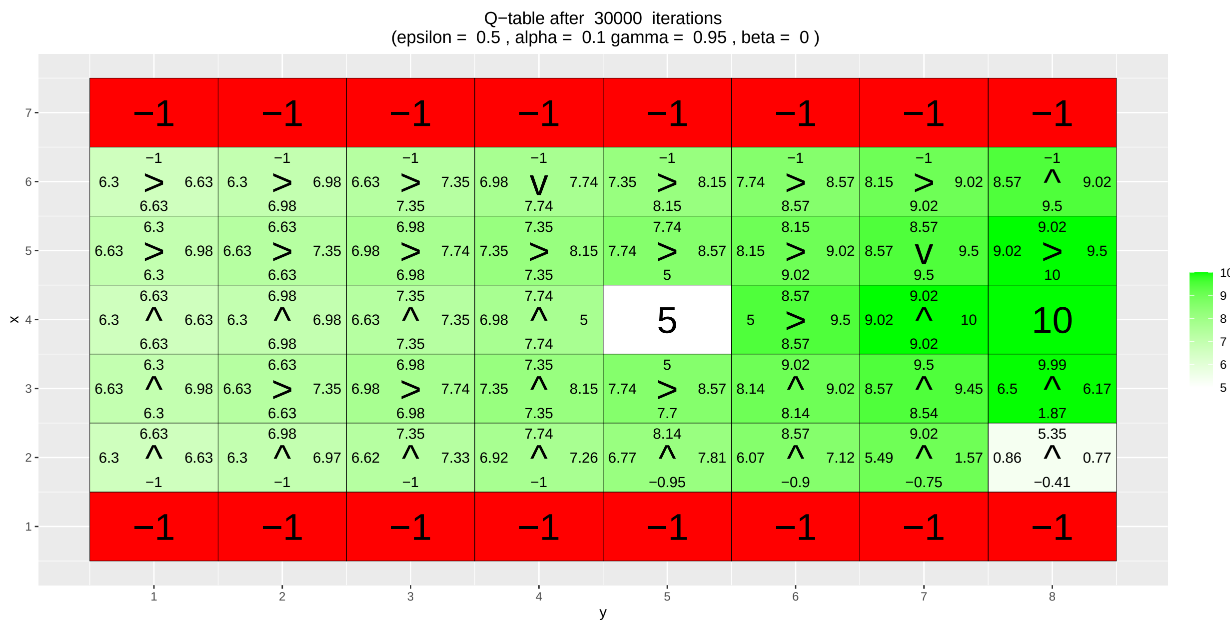

MovingAverage(reward, 100)
0
1
2
3
4
0
5000
10000
15000
20000
25000
30000
Index

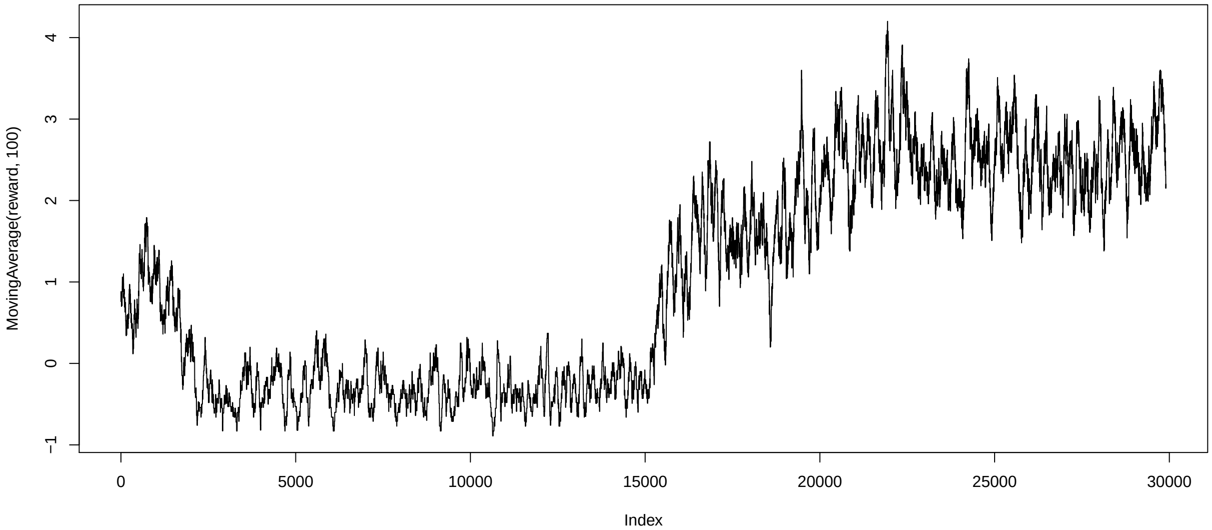

MovingAverage(correction, 100)
0.00
0.05
0.10
0.15
0
5000
10000
15000
20000
25000
30000
Index

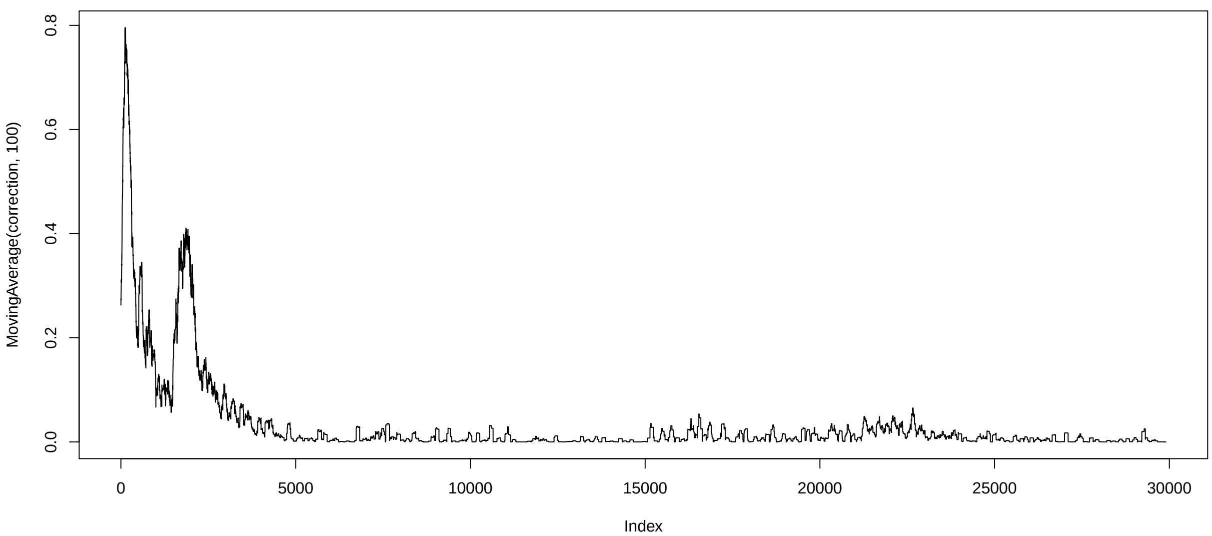

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5

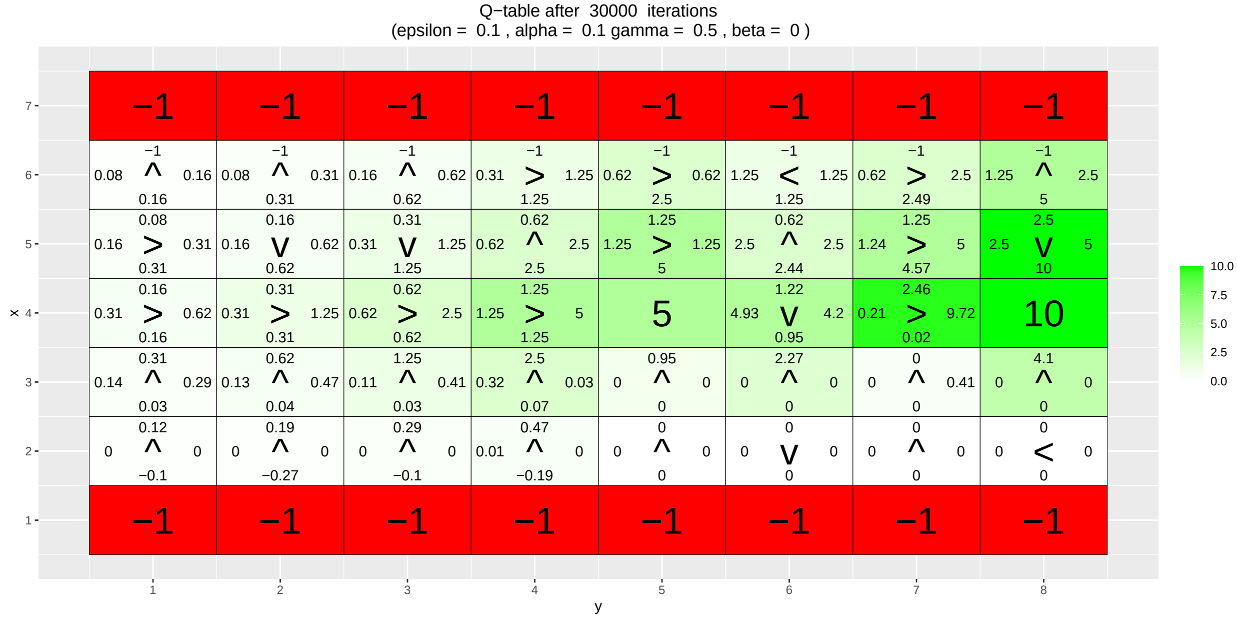

MovingAverage(reward, 100)
Index

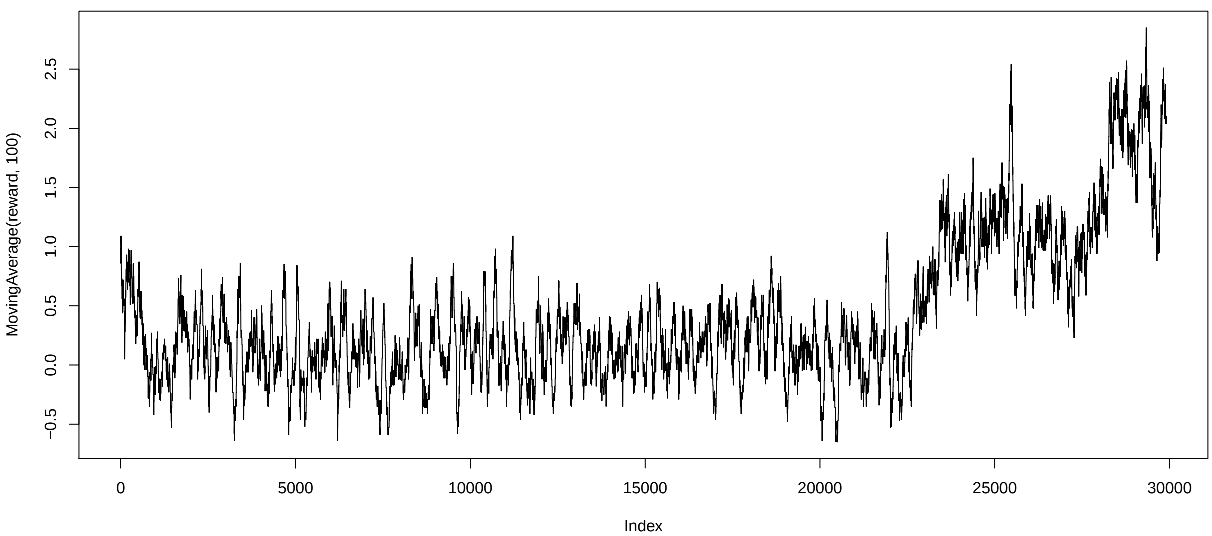

MovingAverage(correction, 100)
Index

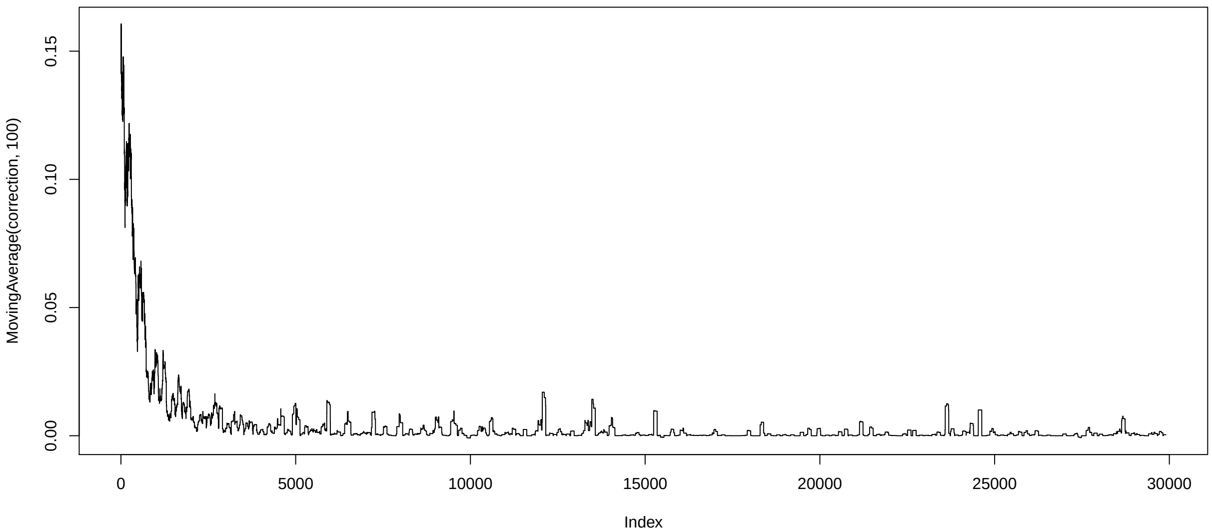

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y


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

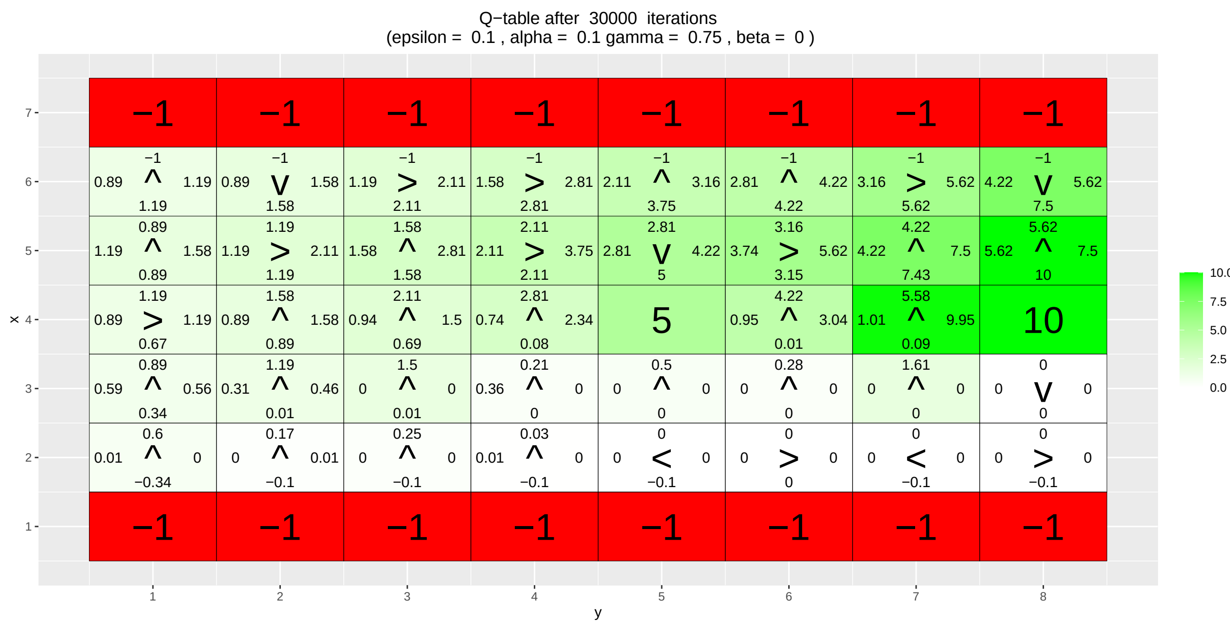

MovingAverage(correction, 100)
Index
0
5000
10000
15000
20000
25000
30000
0.0
0.2
0.4
0.6
0.8


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

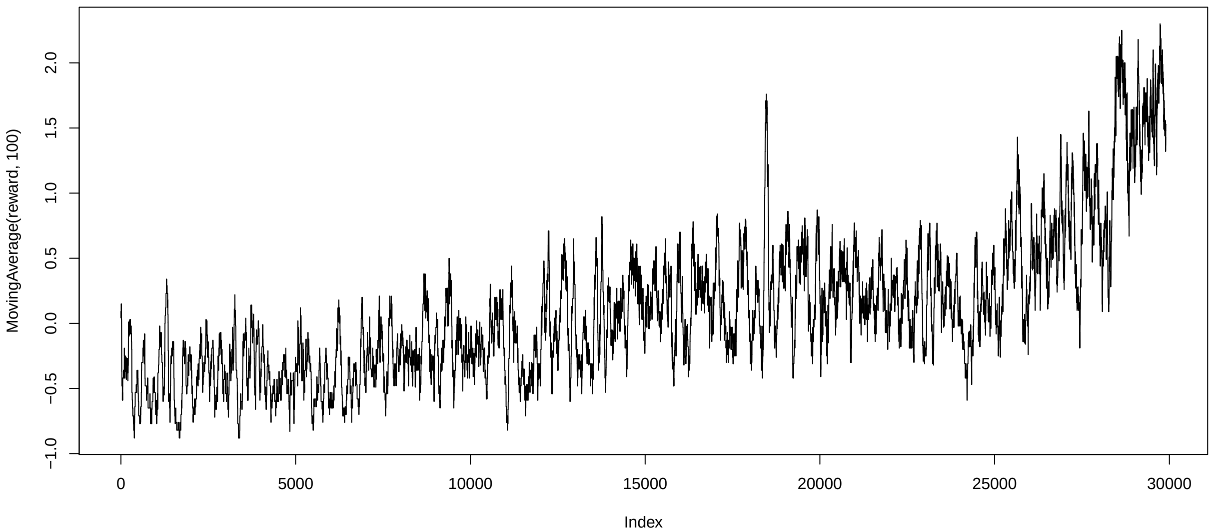

MovingAverage(reward, 100)
Index

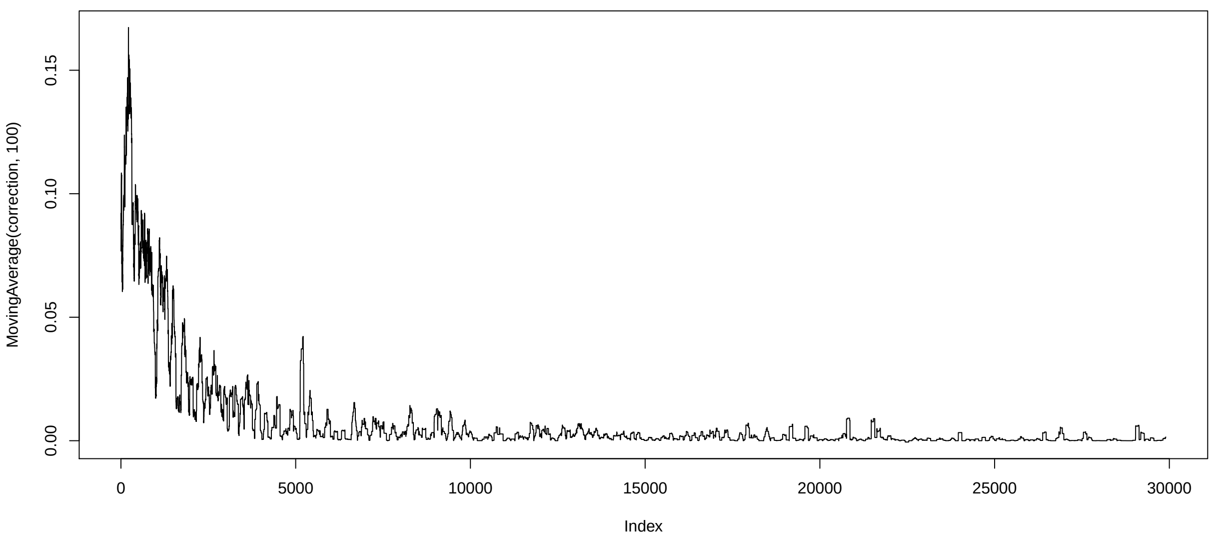


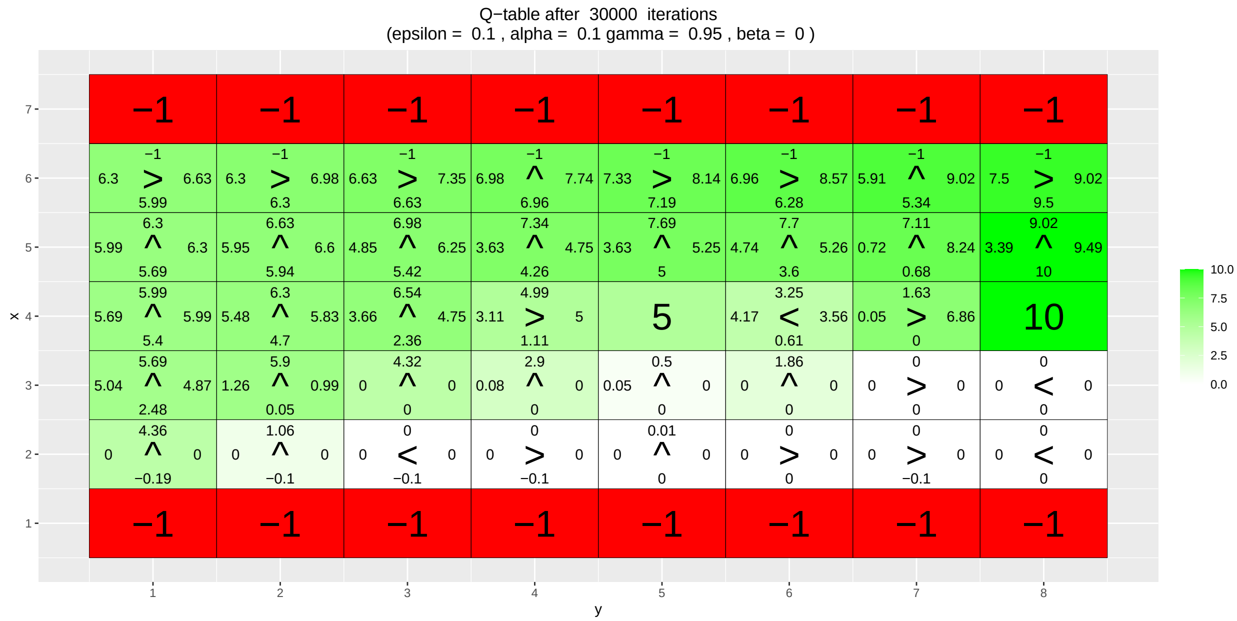
MovingAverage(correction, 100)
Index

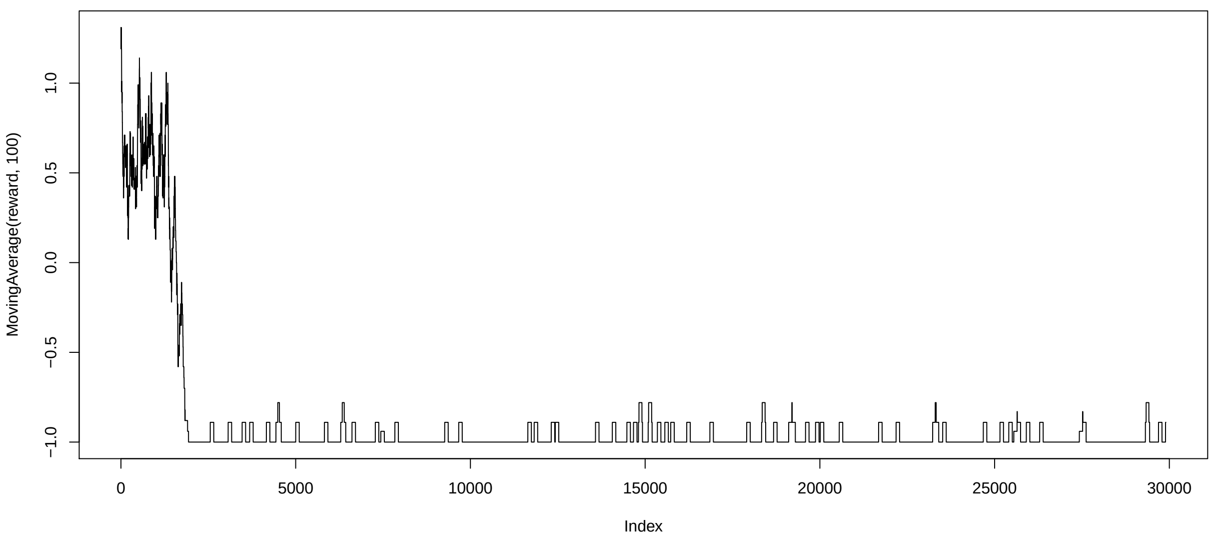


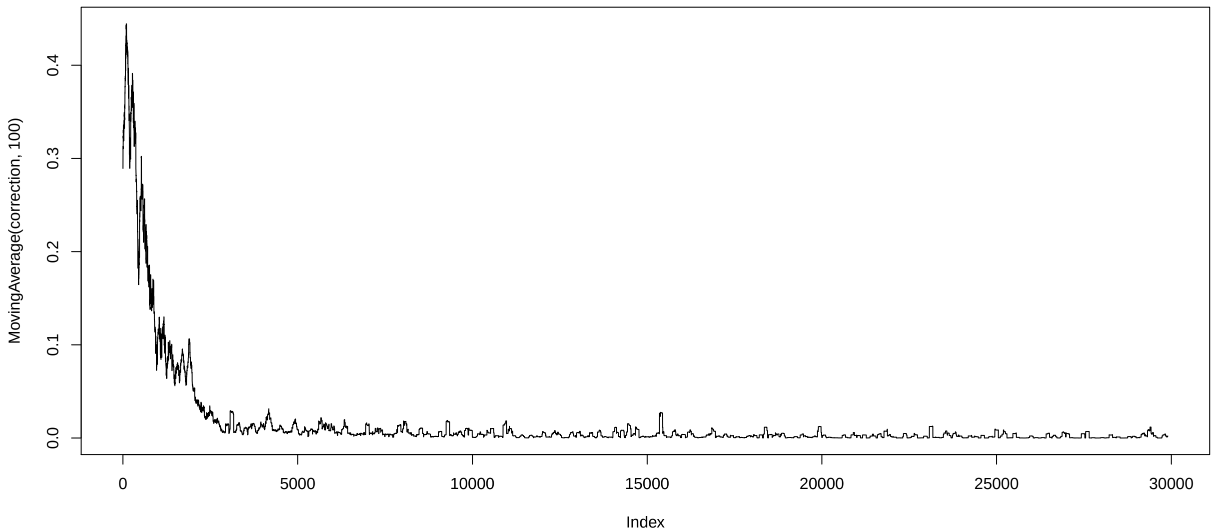
Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y


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


MovingAverage(correction, 100)
0.00
0.05
0.10
0.15
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


MovingAverage(reward, 100)
Index



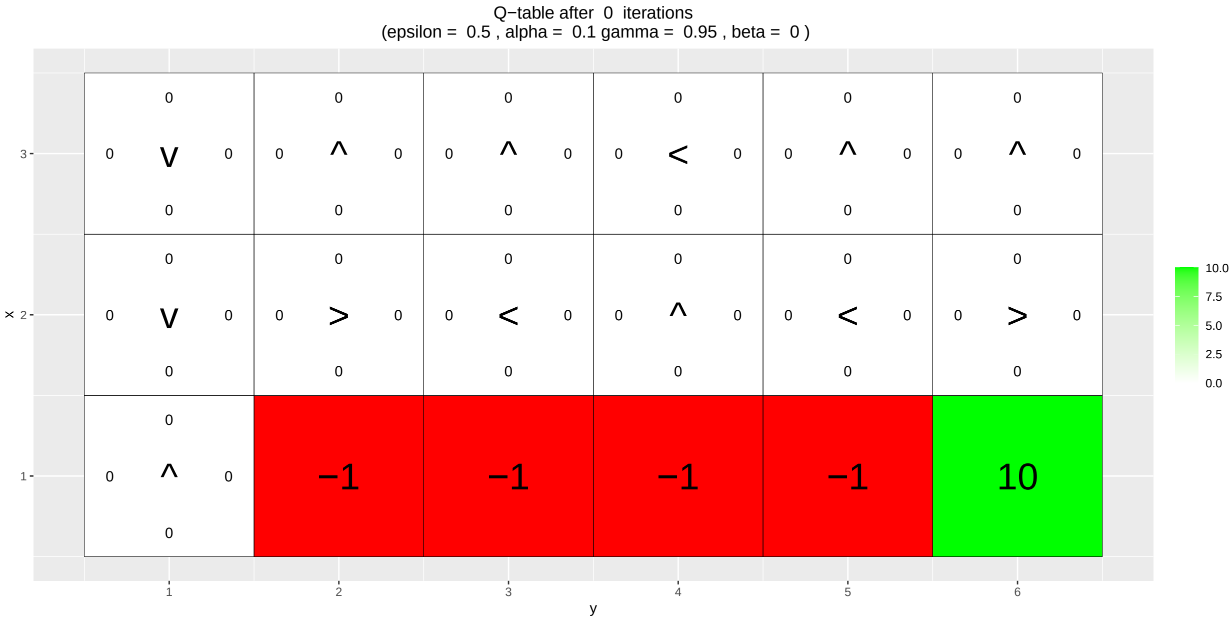
First we observe the impact of γ:

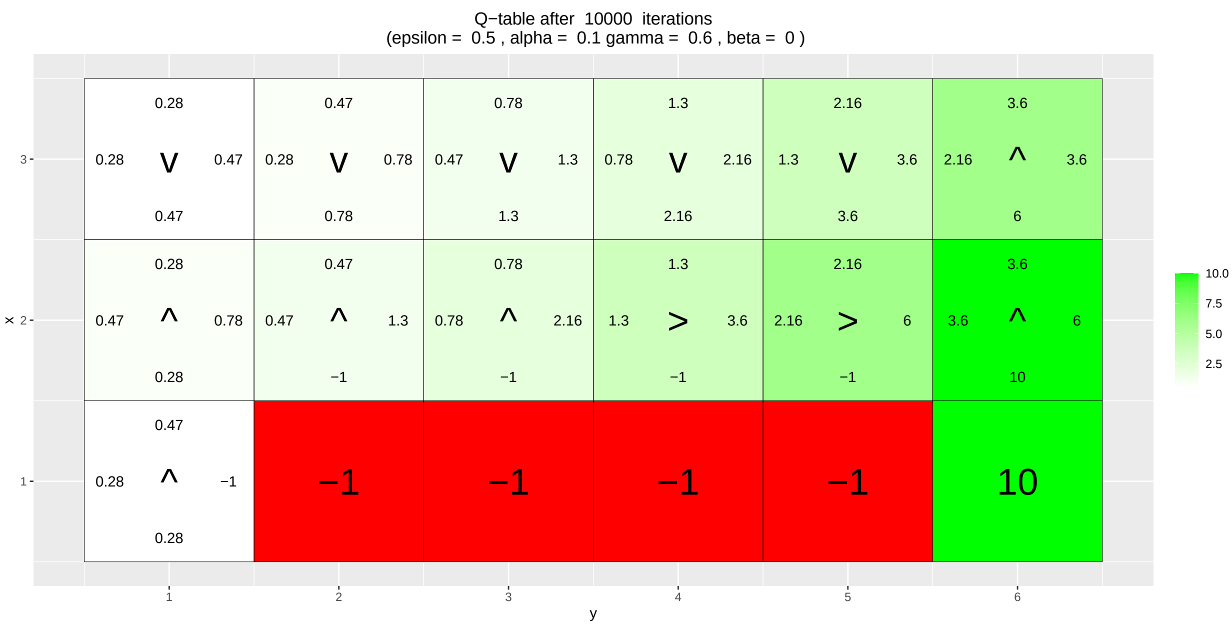
Since γ controls whether the agent prefers to get the reward immediately or later on, we can observe that when $\gamma = 0.5$ the agent tends to move near the target state with the smaller reward whereas for higher values $\gamma = 0.75$ or $\gamma = 0.95$ the agent ignores the smaller reward to get to the target state with the higher reward. In other words, when γ is close to 1, there is almost no discount on the total return and the agent acts less greedily for immediate rewards.

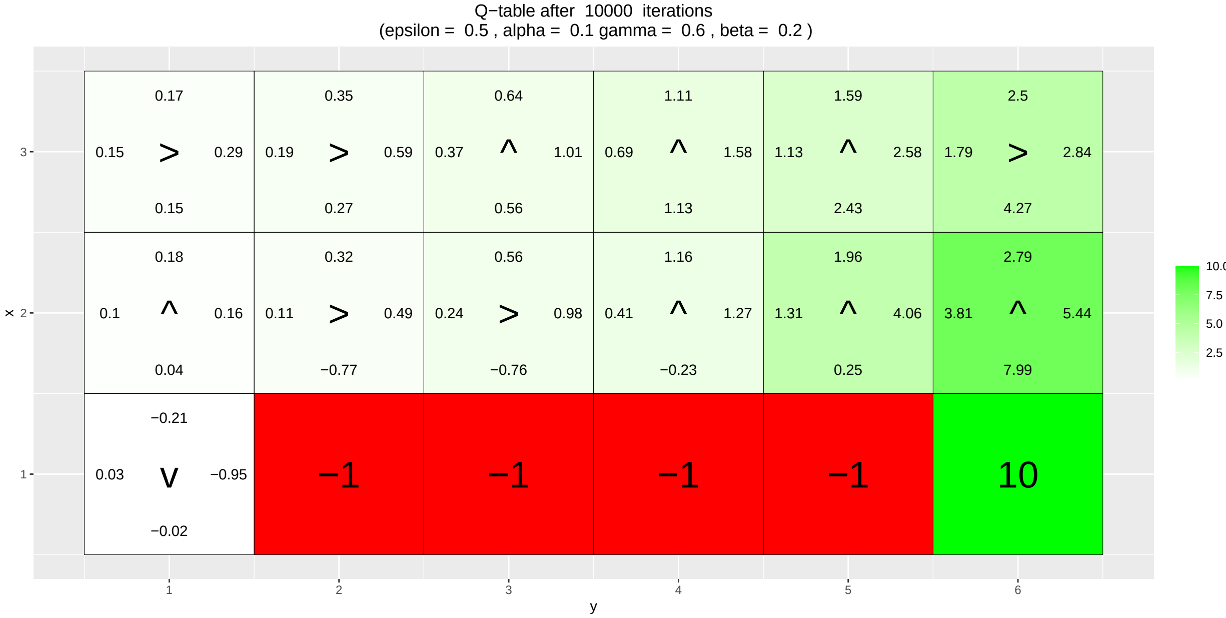
Now we observe the impact of ϵ:

As previously discussed, smaller values for ϵ means smaller probability of the agent taking a random rather than the optimal action. Here we can see that the agent explores more states when $\epsilon = 0.5$ with any given γ, compared to when $\epsilon = 0.1$. Comparing $\epsilon = 0.1$ with $\epsilon = .5$ when $\gamma = 0.95$, the agent avoids the lower states when $\epsilon = 0.1$ since there is only 10% chance for taking a random action while in both cases it reaches out for the long-term reward.

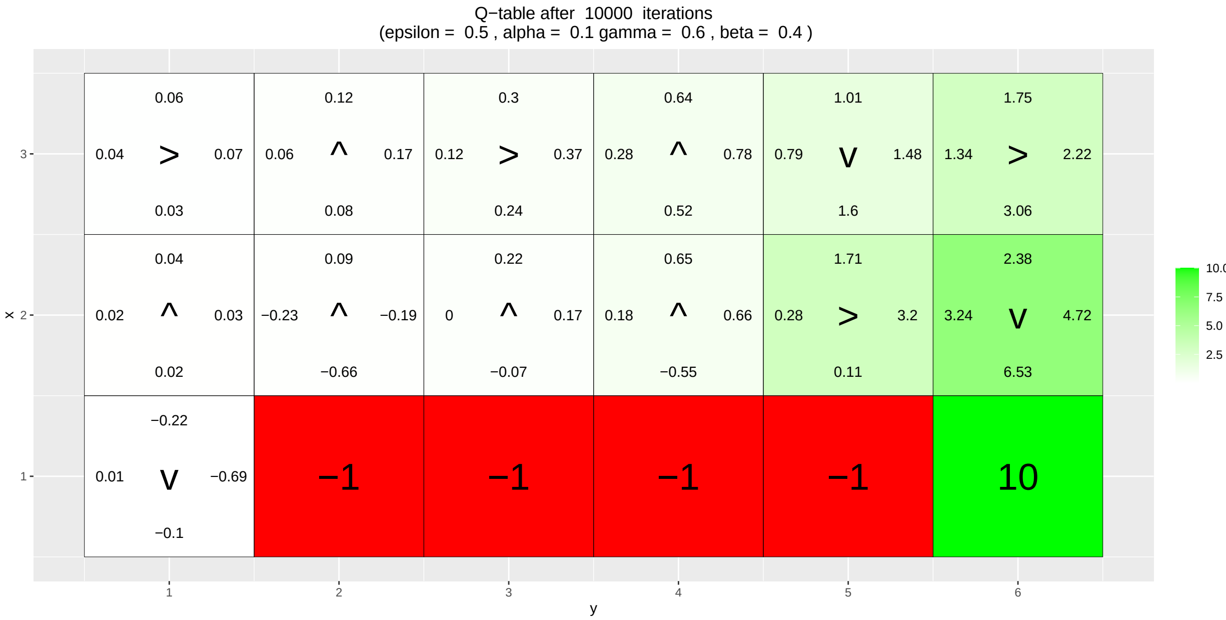
Environment C



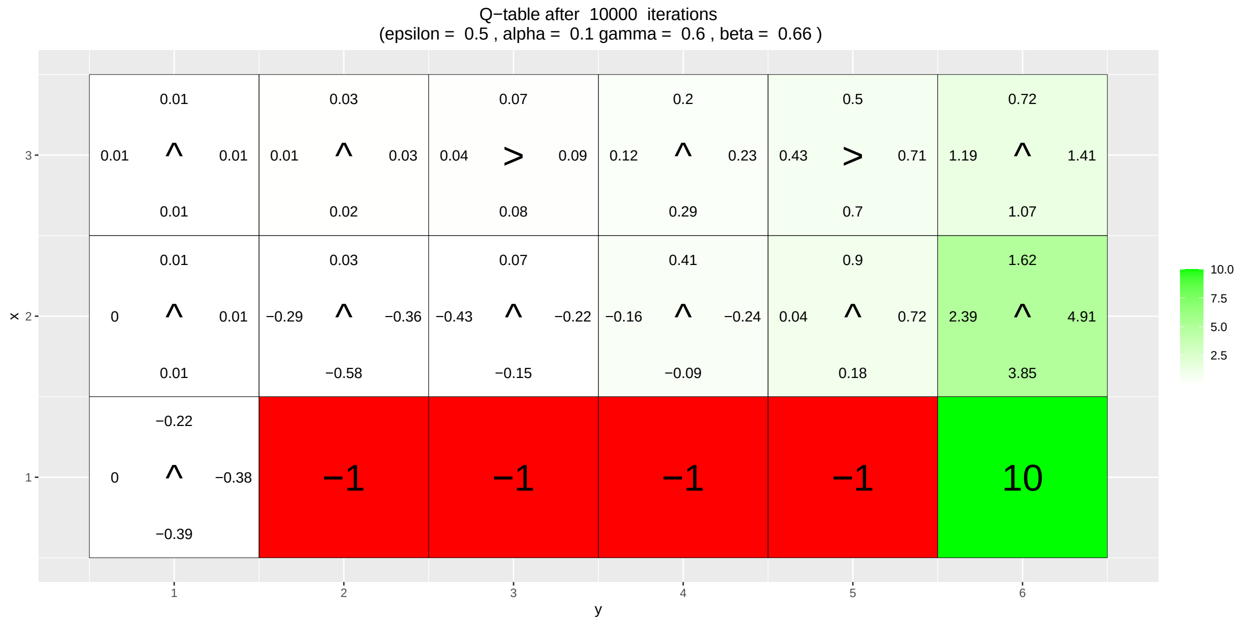




Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)
0.17
0.15 > 0.29
0.15
0.35
0.19 > 0.59
0.27
0.64
0.37 ^ 1.01
0.56
1.11
0.69 ^ 1.58
1.13
1.59
1.13 ^ 2.58
2.43
2.5
1.79 > 2.84
4.27
0.18
0.1 ^ 0.16
0.04
0.32
0.11 > 0.49
−0.77
0.56
0.24 > 0.98
−0.76
1.16
0.41 ^ 1.27
−0.23
1.96
1.31 ^ 4.06
0.25
2.79
3.81 ^ 5.44
7.99
−0.21
0.03 V −0.95
−0.02
−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)
0.06
0.04 > 0.07
0.03
0.12
0.06 ^ 0.17
0.08
0.3
0.12 > 0.37
0.24
0.64
0.28 ^ 0.78
0.52
1.01
0.79 V 1.48
1.6
1.75
1.34 > 2.22
3.06
0.04
0.02 ^ 0.03
0.02
0.09
−0.23 ^ −0.19
−0.66
0.22
0 ^ 0.17
−0.07
0.65
0.18 ^ 0.66
−0.55
1.71
0.28 > 3.2
0.11
2.38
3.24 V 4.72
6.53
−0.22
0.01 V −0.69
−0.1
−1
−1
−1
−1
10
x
y
10.0
7.5
5.0
2.5

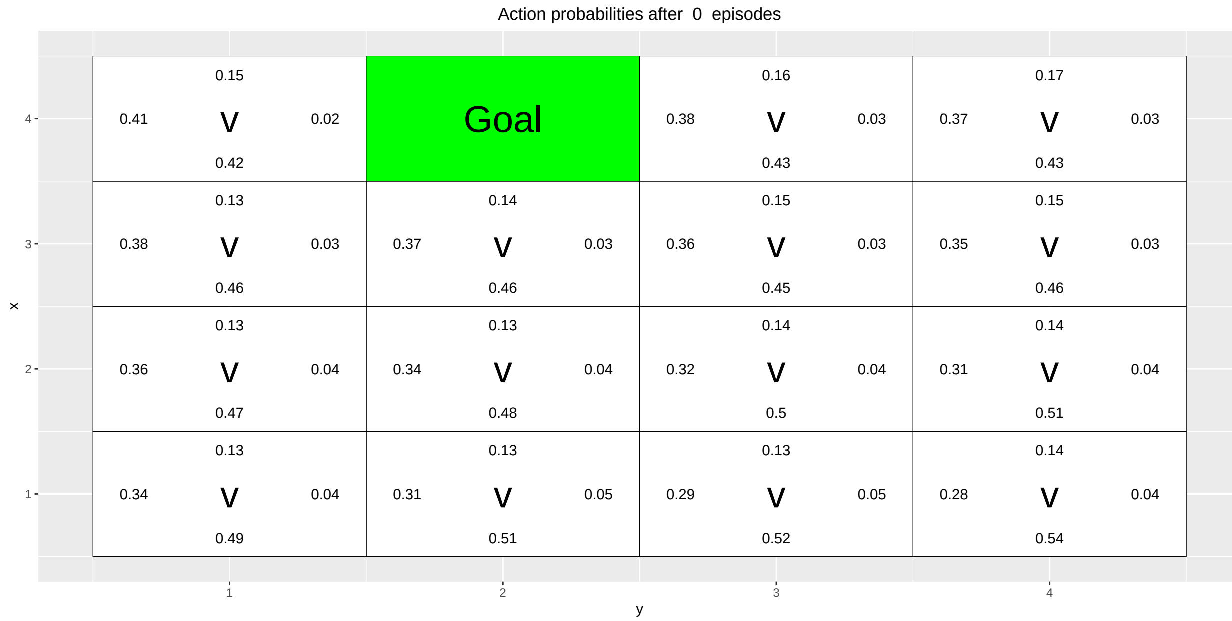


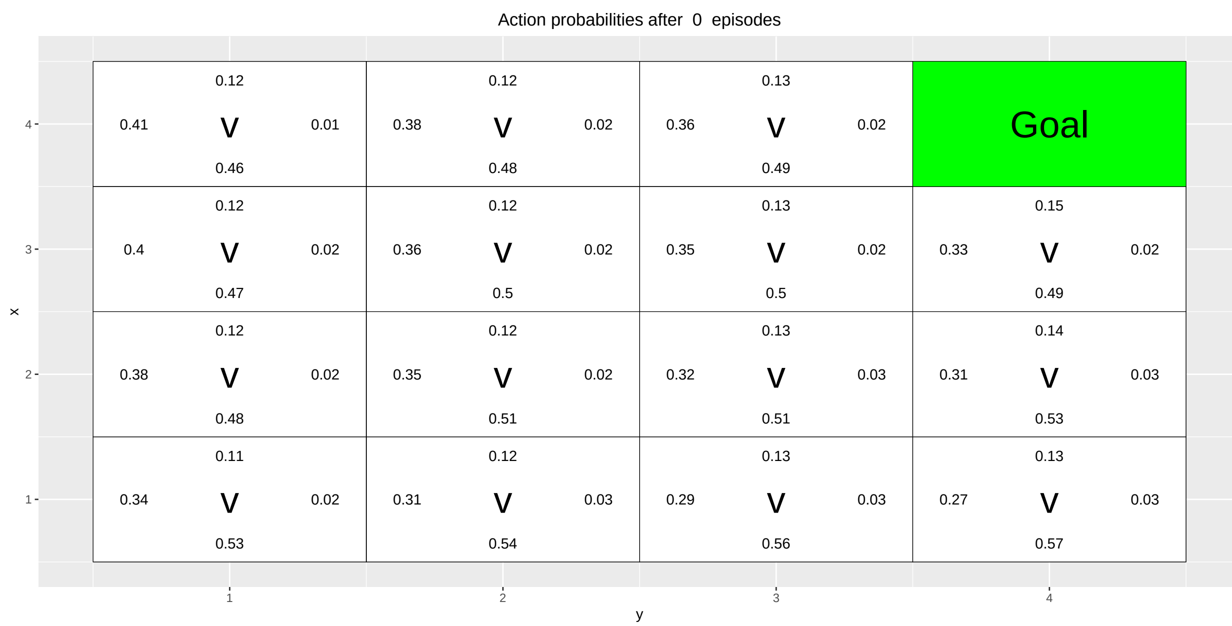
The β parameter determines the certainty of the agent moving to the intended direction. Larger β means more chance for the agent to end up in states adjacent to the intended state. When $\beta = 0$ the agent takes the shortest path to the goal state since there is no chance for it to slip into states with negative reward ([1,2],[1,3],[1,4],[1,5]). It can also be observed that the q values for all these states have converged to -1 which is the true reward.

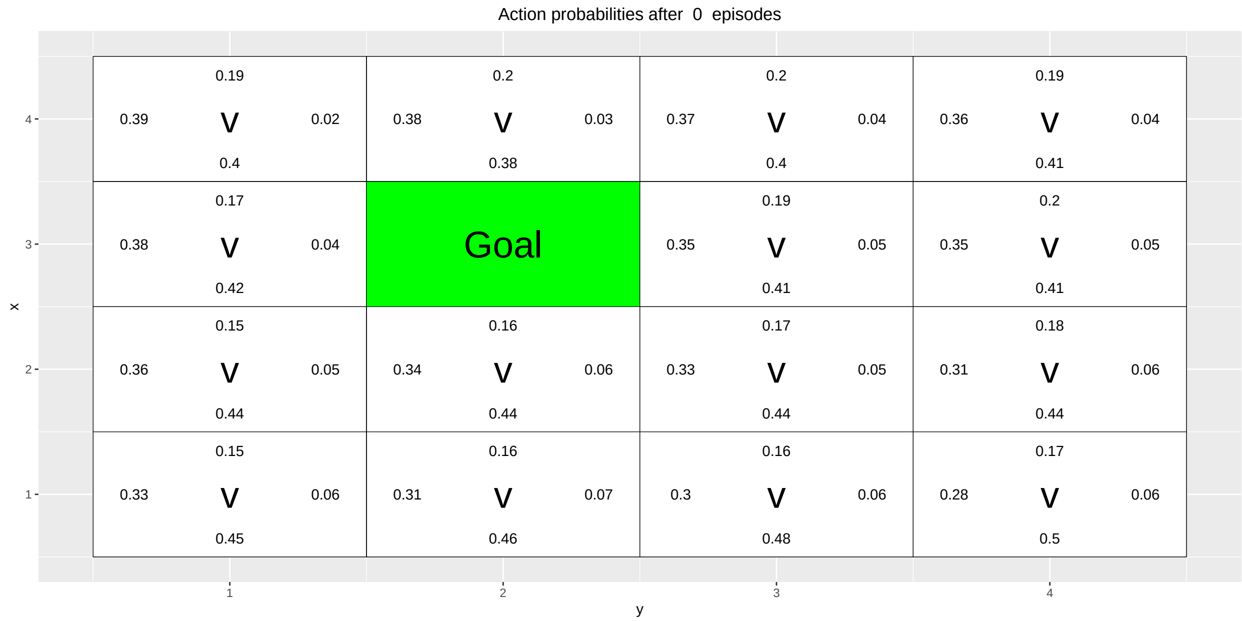
However, as β is increased, the agent learns through iterations, that there is a chance for it to end up in a state with negative reward if it takes the shortest path and therefore prefers to take the top row instead. When $\beta = 0.66$ the agent completely avoids the middle row and consecutively, the q values for the negative states grow larger (do not converge to true rewards).

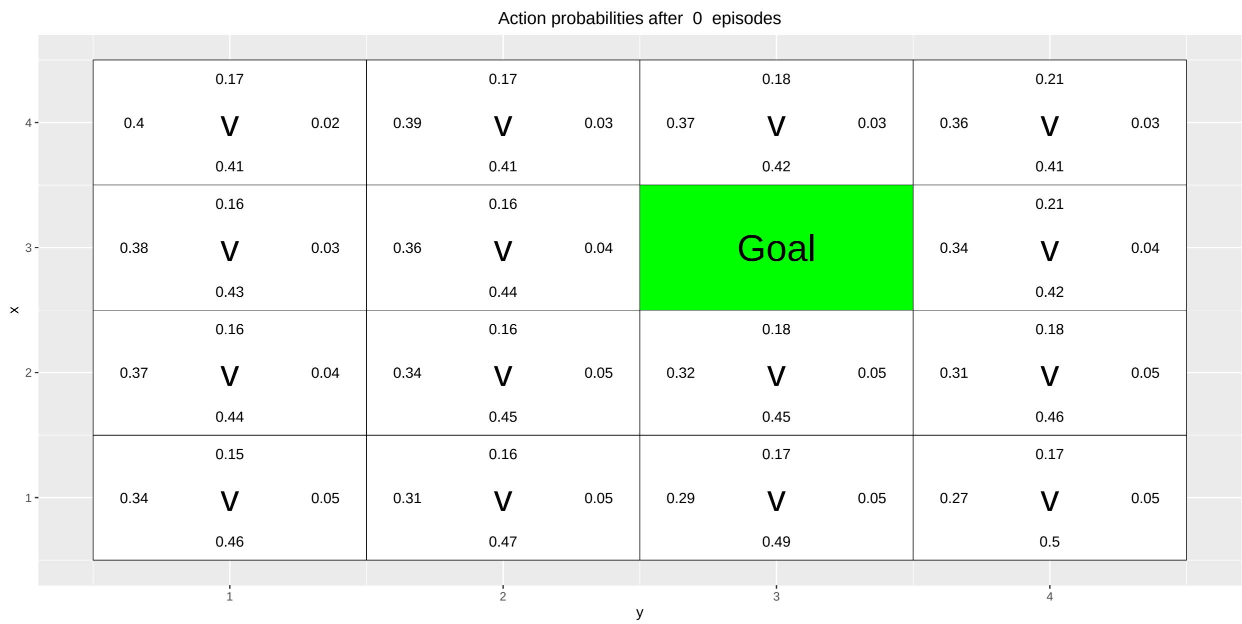
Reinforcement Learning

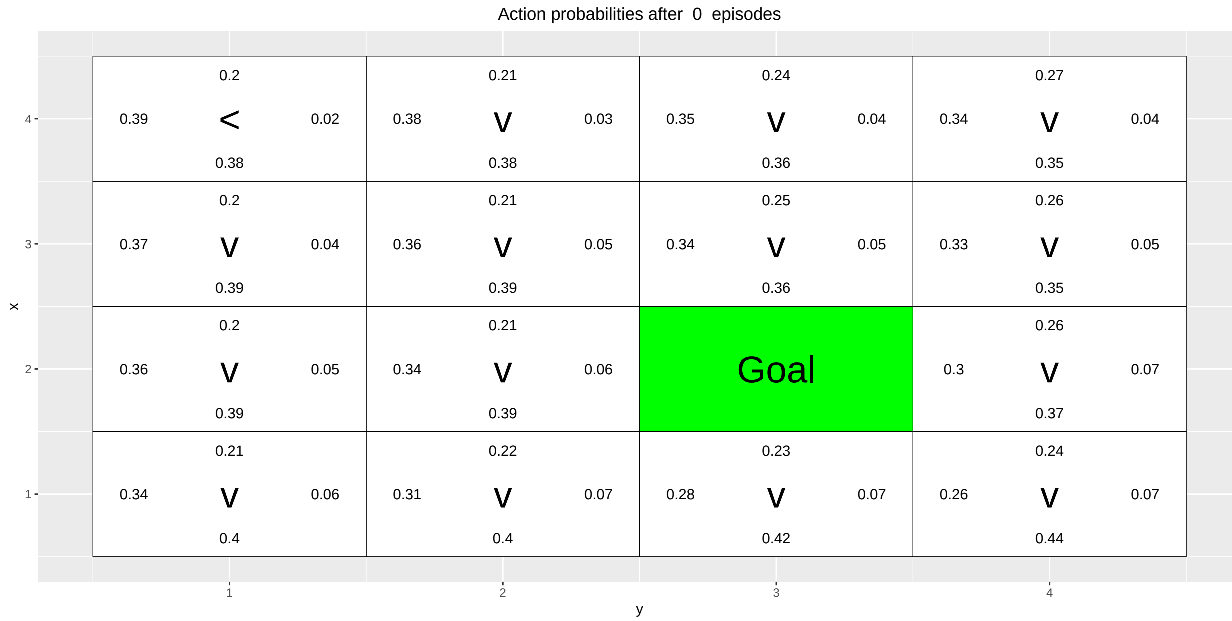
Environment D

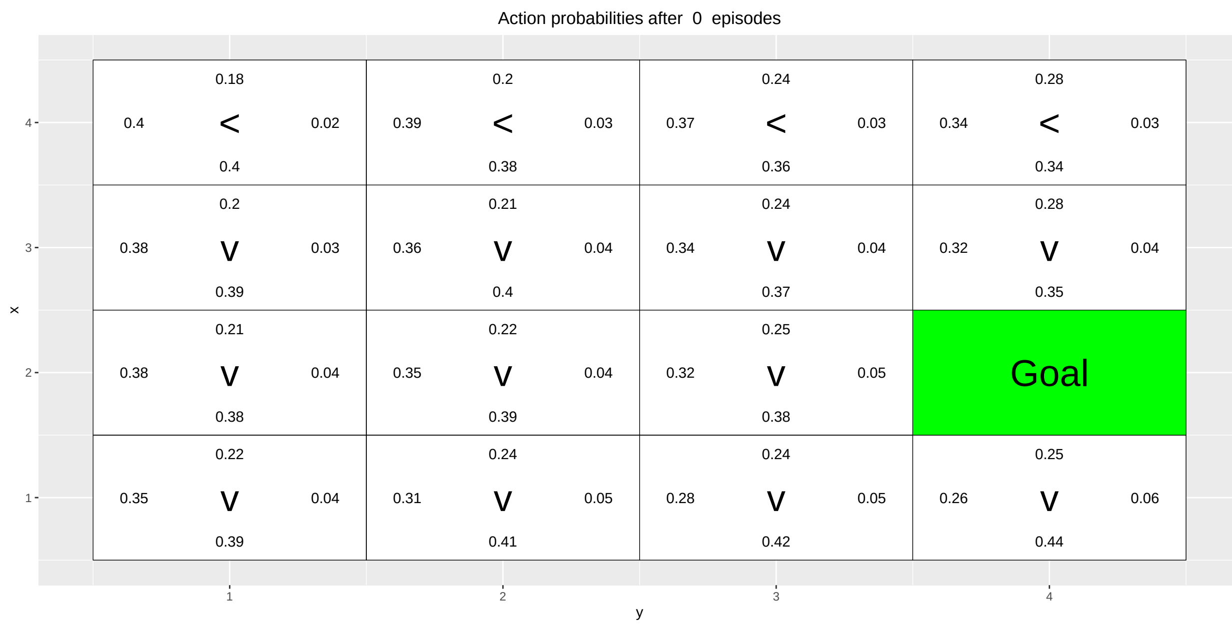


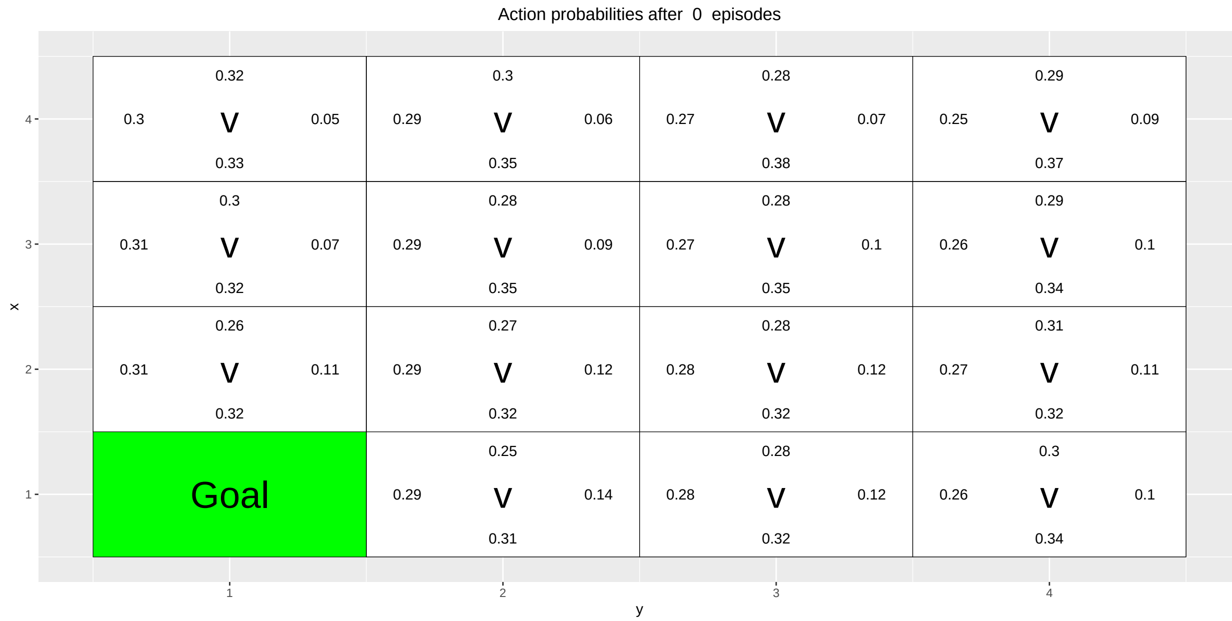


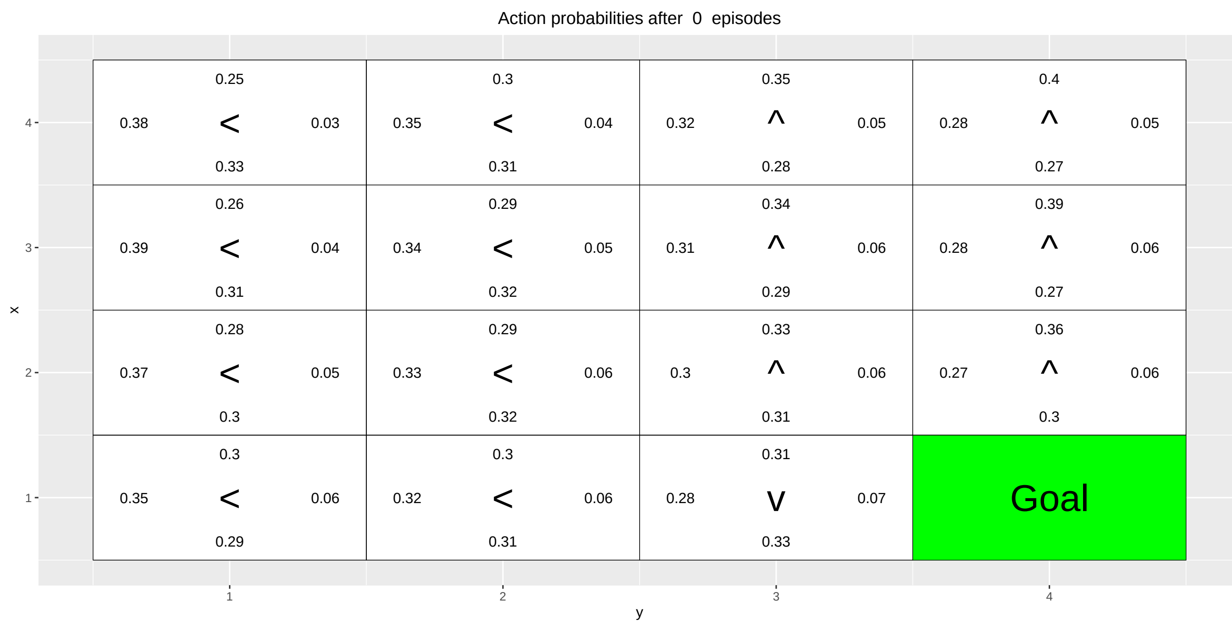

Action probabilities after 0 episodes
Goal
x
y


Action probabilities after 0 episodes
Goal
x
y


Action probabilities after 0 episodes
Goal
x
y


Action probabilities after 0 episodes
Goal
x
y






```
## episode 10
## episode 20
## episode 30
## episode 40
## episode 50
## episode 60
## episode 70
## episode 80
## episode 90
## episode 100
## episode 110
## episode 120
## episode 130
## episode 140
```

```
## episode 150
## episode 160
## episode 170
## episode 180
## episode 190
## episode 200
## episode 210
## episode 220
## episode 230
## episode 240
## episode 250
## episode 260
## episode 270
## episode 280
## episode 290
## episode 300
## episode 310
## episode 320
## episode 330
## episode 340
## episode 350
## episode 360
## episode 370
## episode 380
## episode 390
## episode 400
## episode 410
## episode 420
## episode 430
## episode 440
## episode 450
## episode 460
## episode 470
## episode 480
## episode 490
## episode 500
## episode 510
## episode 520
## episode 530
## episode 540
## episode 550
## episode 560
## episode 570
## episode 580
## episode 590
## episode 600
## episode 610
## episode 620
## episode 630
## episode 640
## episode 650
## episode 660
## episode 670
## episode 680
```

```
## episode 690
## episode 700
## episode 710
## episode 720
## episode 730
## episode 740
## episode 750
## episode 760
## episode 770
## episode 780
## episode 790
## episode 800
## episode 810
## episode 820
## episode 830
## episode 840
## episode 850
## episode 860
## episode 870
## episode 880
## episode 890
## episode 900
## episode 910
## episode 920
## episode 930
## episode 940
## episode 950
## episode 960
## episode 970
## episode 980
## episode 990
## episode 1000
## episode 1010
## episode 1020
## episode 1030
## episode 1040
## episode 1050
## episode 1060
## episode 1070
## episode 1080
## episode 1090
## episode 1100
## episode 1110
## episode 1120
## episode 1130
## episode 1140
## episode 1150
## episode 1160
## episode 1170
## episode 1180
## episode 1190
## episode 1200
## episode 1210
## episode 1220
```

```
## episode 1230
## episode 1240
## episode 1250
## episode 1260
## episode 1270
## episode 1280
## episode 1290
## episode 1300
## episode 1310
## episode 1320
## episode 1330
## episode 1340
## episode 1350
## episode 1360
## episode 1370
## episode 1380
## episode 1390
## episode 1400
## episode 1410
## episode 1420
## episode 1430
## episode 1440
## episode 1450
## episode 1460
## episode 1470
## episode 1480
## episode 1490
## episode 1500
## episode 1510
## episode 1520
## episode 1530
## episode 1540
## episode 1550
## episode 1560
## episode 1570
## episode 1580
## episode 1590
## episode 1600
## episode 1610
## episode 1620
## episode 1630
## episode 1640
## episode 1650
## episode 1660
## episode 1670
## episode 1680
## episode 1690
## episode 1700
## episode 1710
## episode 1720
## episode 1730
## episode 1740
## episode 1750
## episode 1760
```

```
## episode 1770
## episode 1780
## episode 1790
## episode 1800
## episode 1810
## episode 1820
## episode 1830
## episode 1840
## episode 1850
## episode 1860
## episode 1870
## episode 1880
## episode 1890
## episode 1900
## episode 1910
## episode 1920
## episode 1930
## episode 1940
## episode 1950
## episode 1960
## episode 1970
## episode 1980
## episode 1990
## episode 2000
## episode 2010
## episode 2020
## episode 2030
## episode 2040
## episode 2050
## episode 2060
## episode 2070
## episode 2080
## episode 2090
## episode 2100
## episode 2110
## episode 2120
## episode 2130
## episode 2140
## episode 2150
## episode 2160
## episode 2170
## episode 2180
## episode 2190
## episode 2200
## episode 2210
## episode 2220
## episode 2230
## episode 2240
## episode 2250
## episode 2260
## episode 2270
## episode 2280
## episode 2290
## episode 2300
```

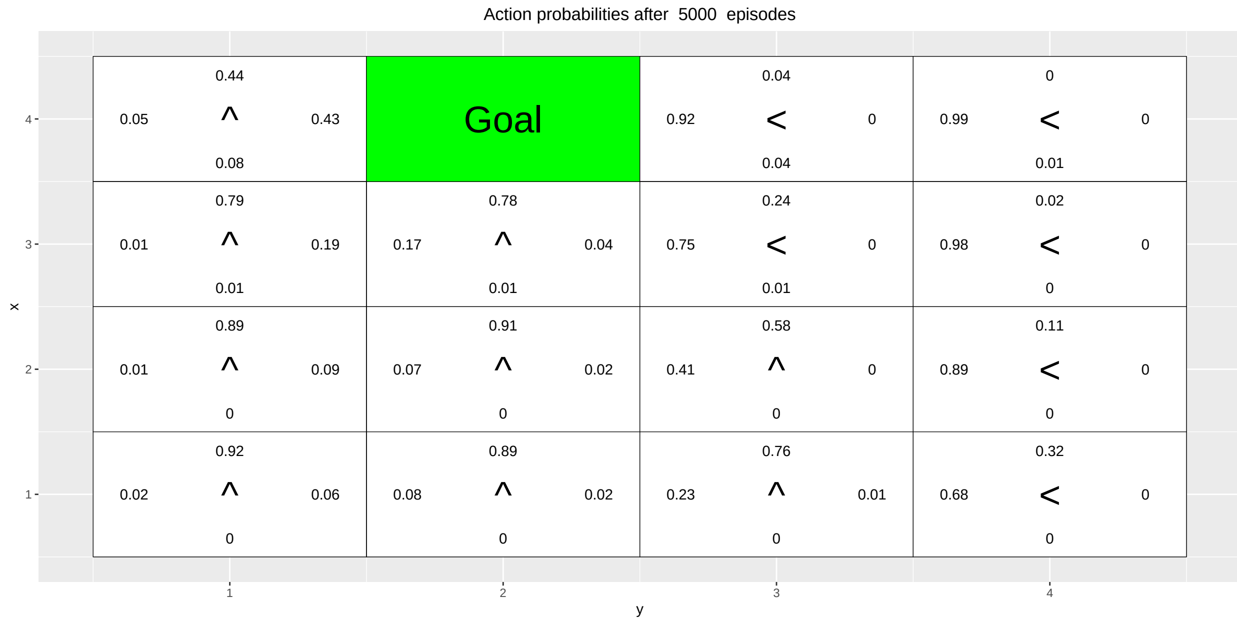
```
## episode 2310
## episode 2320
## episode 2330
## episode 2340
## episode 2350
## episode 2360
## episode 2370
## episode 2380
## episode 2390
## episode 2400
## episode 2410
## episode 2420
## episode 2430
## episode 2440
## episode 2450
## episode 2460
## episode 2470
## episode 2480
## episode 2490
## episode 2500
## episode 2510
## episode 2520
## episode 2530
## episode 2540
## episode 2550
## episode 2560
## episode 2570
## episode 2580
## episode 2590
## episode 2600
## episode 2610
## episode 2620
## episode 2630
## episode 2640
## episode 2650
## episode 2660
## episode 2670
## episode 2680
## episode 2690
## episode 2700
## episode 2710
## episode 2720
## episode 2730
## episode 2740
## episode 2750
## episode 2760
## episode 2770
## episode 2780
## episode 2790
## episode 2800
## episode 2810
## episode 2820
## episode 2830
## episode 2840
```

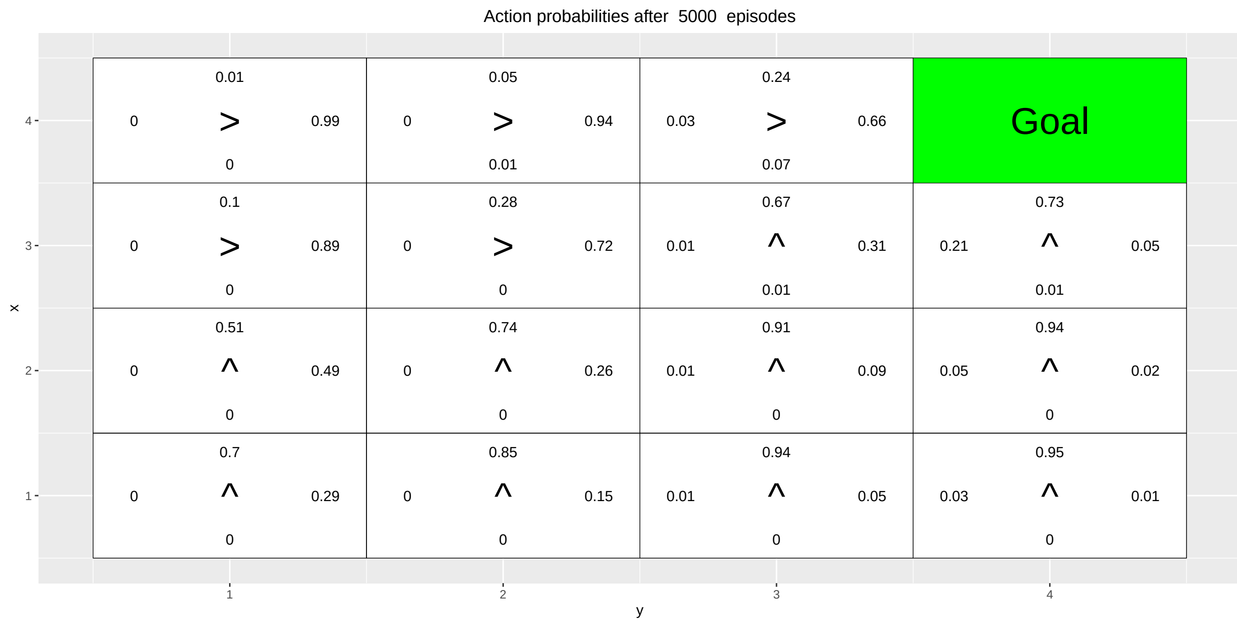
```
## episode 2850
## episode 2860
## episode 2870
## episode 2880
## episode 2890
## episode 2900
## episode 2910
## episode 2920
## episode 2930
## episode 2940
## episode 2950
## episode 2960
## episode 2970
## episode 2980
## episode 2990
## episode 3000
## episode 3010
## episode 3020
## episode 3030
## episode 3040
## episode 3050
## episode 3060
## episode 3070
## episode 3080
## episode 3090
## episode 3100
## episode 3110
## episode 3120
## episode 3130
## episode 3140
## episode 3150
## episode 3160
## episode 3170
## episode 3180
## episode 3190
## episode 3200
## episode 3210
## episode 3220
## episode 3230
## episode 3240
## episode 3250
## episode 3260
## episode 3270
## episode 3280
## episode 3290
## episode 3300
## episode 3310
## episode 3320
## episode 3330
## episode 3340
## episode 3350
## episode 3360
## episode 3370
## episode 3380
```

```
## episode 3390
## episode 3400
## episode 3410
## episode 3420
## episode 3430
## episode 3440
## episode 3450
## episode 3460
## episode 3470
## episode 3480
## episode 3490
## episode 3500
## episode 3510
## episode 3520
## episode 3530
## episode 3540
## episode 3550
## episode 3560
## episode 3570
## episode 3580
## episode 3590
## episode 3600
## episode 3610
## episode 3620
## episode 3630
## episode 3640
## episode 3650
## episode 3660
## episode 3670
## episode 3680
## episode 3690
## episode 3700
## episode 3710
## episode 3720
## episode 3730
## episode 3740
## episode 3750
## episode 3760
## episode 3770
## episode 3780
## episode 3790
## episode 3800
## episode 3810
## episode 3820
## episode 3830
## episode 3840
## episode 3850
## episode 3860
## episode 3870
## episode 3880
## episode 3890
## episode 3900
## episode 3910
## episode 3920
```

```
## episode 3930
## episode 3940
## episode 3950
## episode 3960
## episode 3970
## episode 3980
## episode 3990
## episode 4000
## episode 4010
## episode 4020
## episode 4030
## episode 4040
## episode 4050
## episode 4060
## episode 4070
## episode 4080
## episode 4090
## episode 4100
## episode 4110
## episode 4120
## episode 4130
## episode 4140
## episode 4150
## episode 4160
## episode 4170
## episode 4180
## episode 4190
## episode 4200
## episode 4210
## episode 4220
## episode 4230
## episode 4240
## episode 4250
## episode 4260
## episode 4270
## episode 4280
## episode 4290
## episode 4300
## episode 4310
## episode 4320
## episode 4330
## episode 4340
## episode 4350
## episode 4360
## episode 4370
## episode 4380
## episode 4390
## episode 4400
## episode 4410
## episode 4420
## episode 4430
## episode 4440
## episode 4450
## episode 4460
```

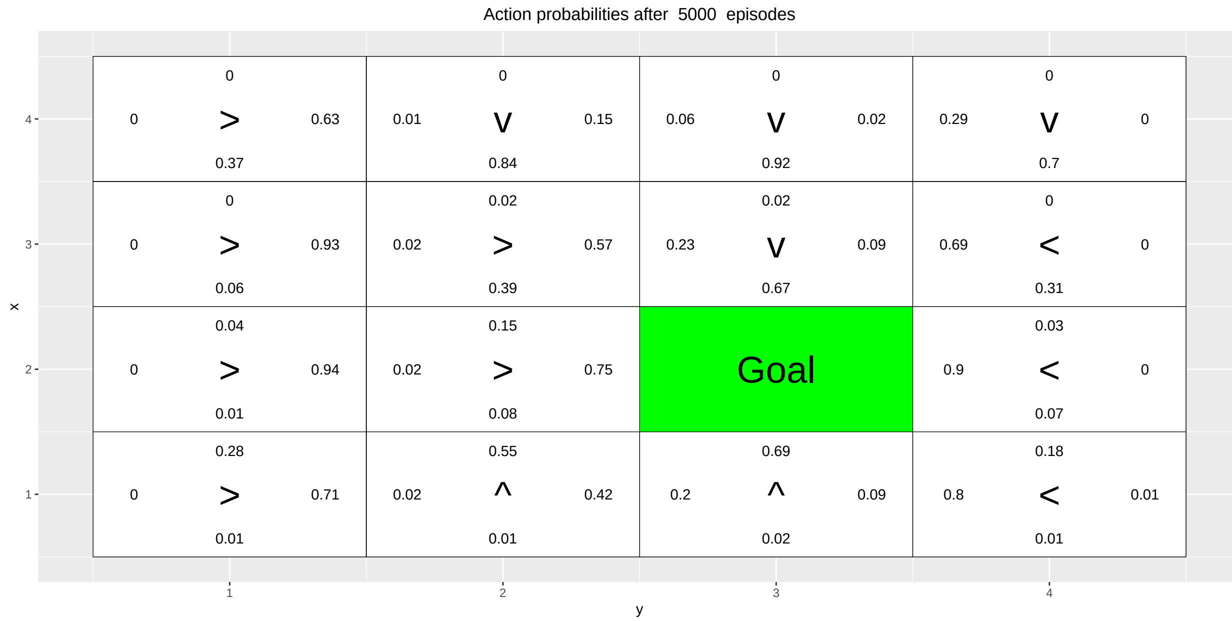
```
## episode 4470
## episode 4480
## episode 4490
## episode 4500
## episode 4510
## episode 4520
## episode 4530
## episode 4540
## episode 4550
## episode 4560
## episode 4570
## episode 4580
## episode 4590
## episode 4600
## episode 4610
## episode 4620
## episode 4630
## episode 4640
## episode 4650
## episode 4660
## episode 4670
## episode 4680
## episode 4690
## episode 4700
## episode 4710
## episode 4720
## episode 4730
## episode 4740
## episode 4750
## episode 4760
## episode 4770
## episode 4780
## episode 4790
## episode 4800
## episode 4810
## episode 4820
## episode 4830
## episode 4840
## episode 4850
## episode 4860
## episode 4870
## episode 4880
## episode 4890
## episode 4900
## episode 4910
## episode 4920
## episode 4930
## episode 4940
## episode 4950
## episode 4960
## episode 4970
## episode 4980
## episode 4990
## episode 5000
```

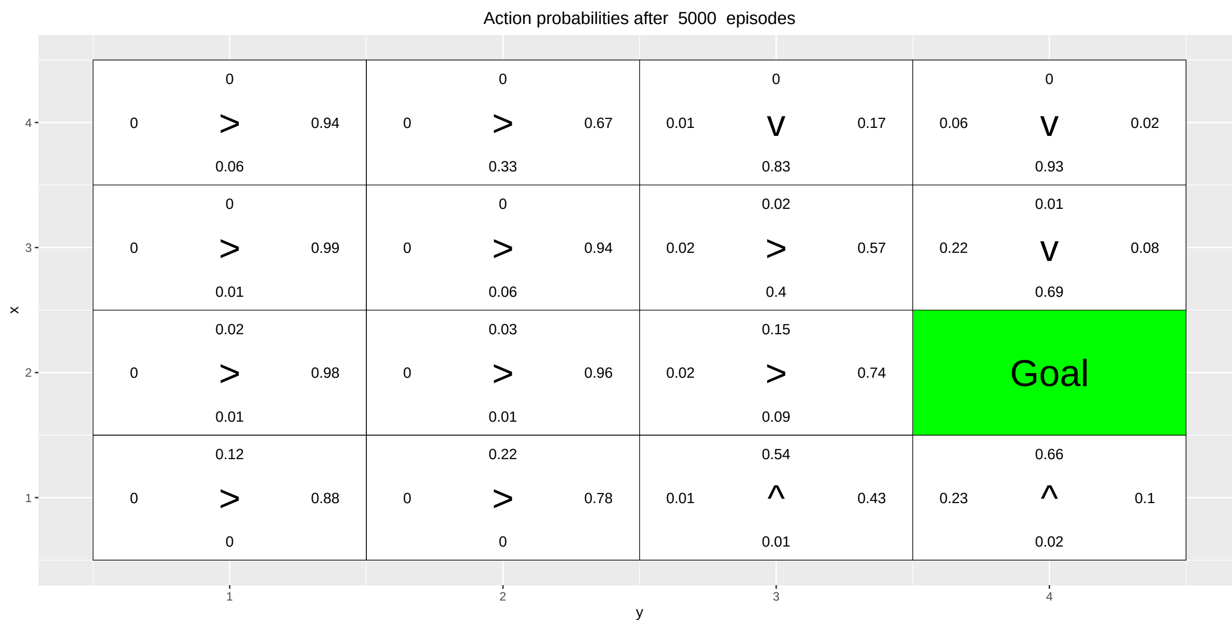

Action probabilities after 5000 episodes
0.44
0.05 ^ 0.43
0.08
Goal
0.04
0.92 < 0
0.04
0
0.99 < 0
0.01
0.79
0.01 ^ 0.19
0.01
0.78
0.17 ^ 0.04
0.01
0.24
0.75 < 0
0.01
0.02
0.98 < 0
0
0.89
0.01 ^ 0.09
0
0.91
0.07 ^ 0.02
0
0.58
0.41 ^ 0
0
0.11
0.89 < 0
0
0.92
0.02 ^ 0.06
0
0.89
0.08 ^ 0.02
0
0.76
0.23 ^ 0.01
0
0.32
0.68 < 0
0
x
4
3
2
1
y
1
2
3
4

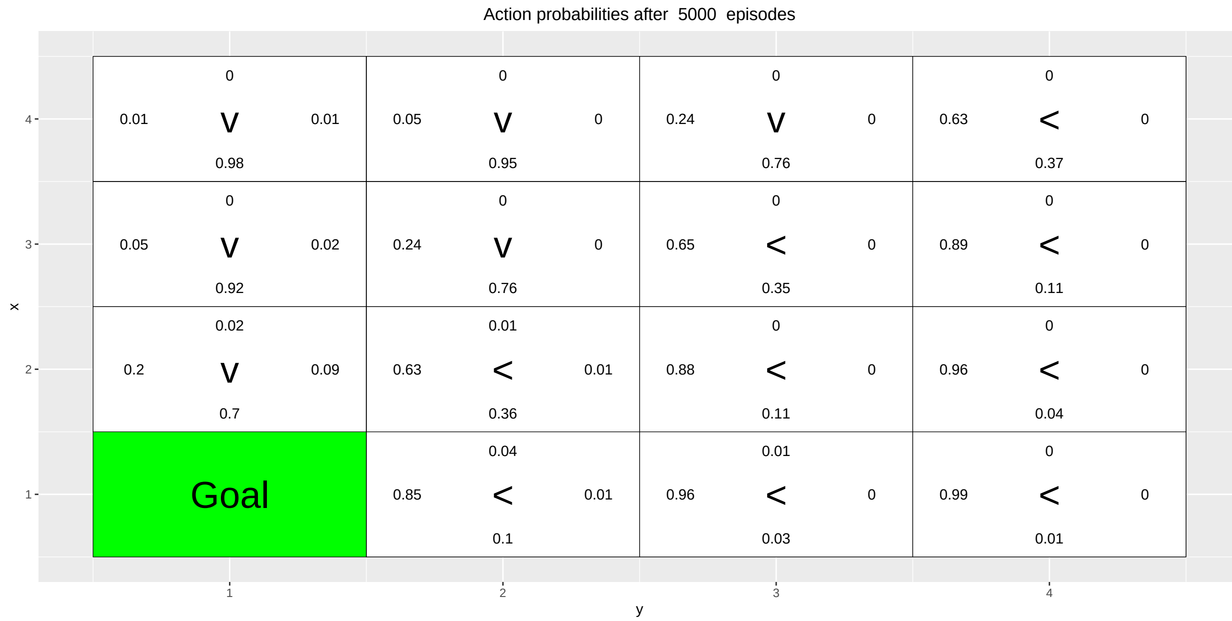

Action probabilities after 5000 episodes
0.01
0 > 0.99
0
0.05
0 > 0.94
0.01
0.24
0.03 > 0.66
0.07
Goal
0.1
0 > 0.89
0
0.28
0 > 0.72
0
0.67
0.01 ^ 0.31
0.01
0.73
0.21 ^ 0.05
0.01
0.51
0 ^ 0.49
0
0.74
0 ^ 0.26
0
0.91
0.01 ^ 0.09
0
0.94
0.05 ^ 0.02
0
0.7
0 ^ 0.29
0
0.85
0 ^ 0.15
0
0.94
0.01 ^ 0.05
0
0.95
0.03 ^ 0.01
0
x
4
3
2
1
y
1
2
3
4

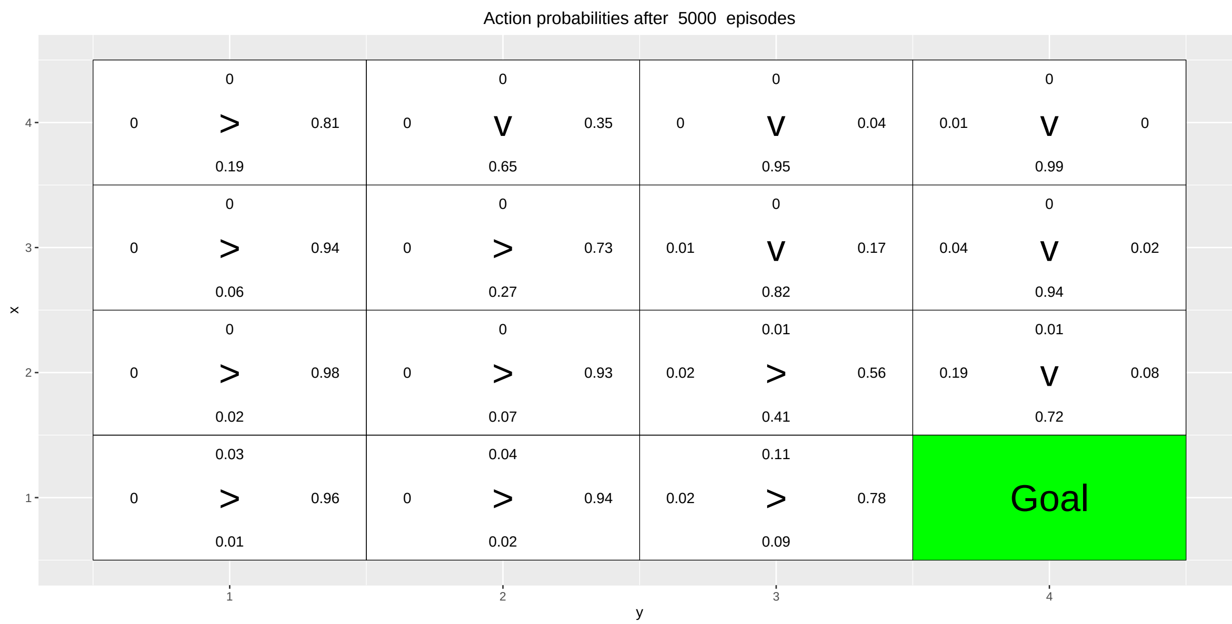

Action probabilities after 5000 episodes
0.04 0.02 0 0
0.05 V 0.4 0.32 V 0.06 0.77 < 0 0.96 < 0
0.51 0.6 0.22 0.04
0.25 0.04 0
0.03 > 0.64 Goal 0.91 < 0 0.99 < 0
0.08 0.05 0.01
0.67 0.75 0.23 0.02
0.02 ^ 0.31 0.17 ^ 0.06 0.76 < 0 0.97 < 0
0.01 0.01 0.01 0
0.82 0.85 0.55 0.09
0.02 ^ 0.16 0.1 ^ 0.04 0.43 ^ 0.01 0.9 < 0
0.01 0.01 0.01 0
x
4
3
2
1
1 2 3 4
y


Action probabilities after 5000 episodes
0.01 0.03 0.02 0
0 > 0.91 0.03 > 0.55 0.3 V 0.08 0.76 < 0
0.08 0.4 0.6 0.24
0.05 0.2 0.03
0 > 0.94 0.03 > 0.7 Goal 0.91 < 0
0.01 0.07 0.05
0.29 0.62 0.73 0.21
0 > 0.71 0.01 ^ 0.36 0.19 ^ 0.07 0.78 < 0
0 0.01 0.01 0.01
0.66 0.84 0.89 0.53
0 ^ 0.34 0.01 ^ 0.14 0.07 ^ 0.04 0.46 ^ 0.01
0 0 0 0
x
4
3
2
1
1 2 3 4
y


Action probabilities after 5000 episodes
Goal
x
y


Action probabilities after 5000 episodes
Goal
x
y

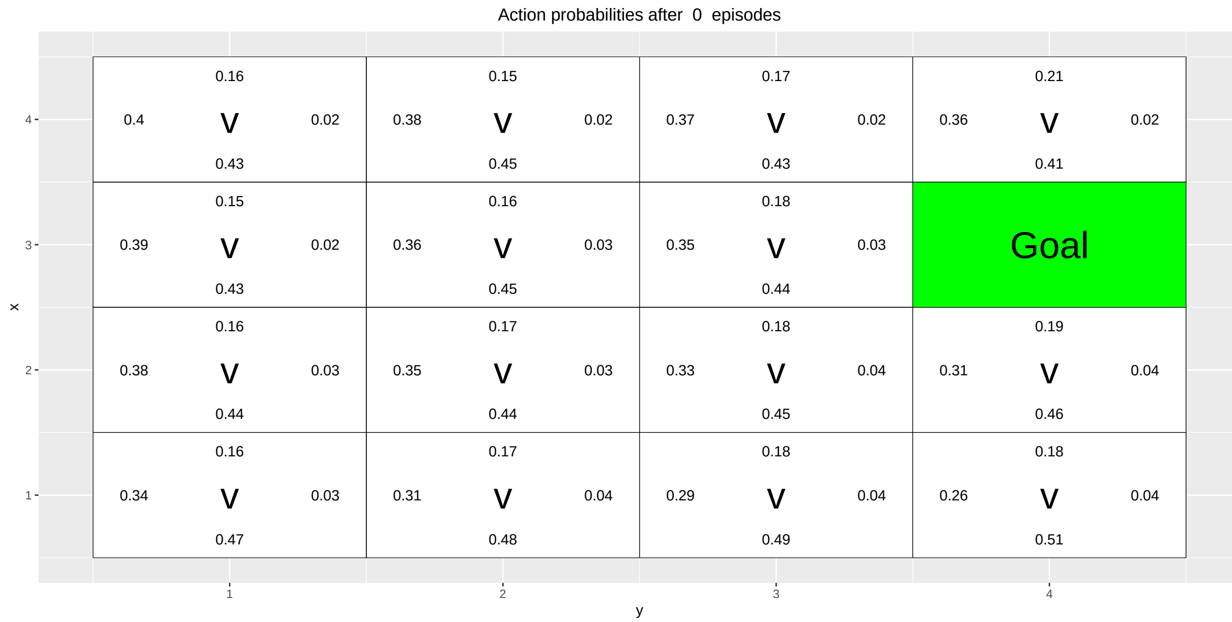


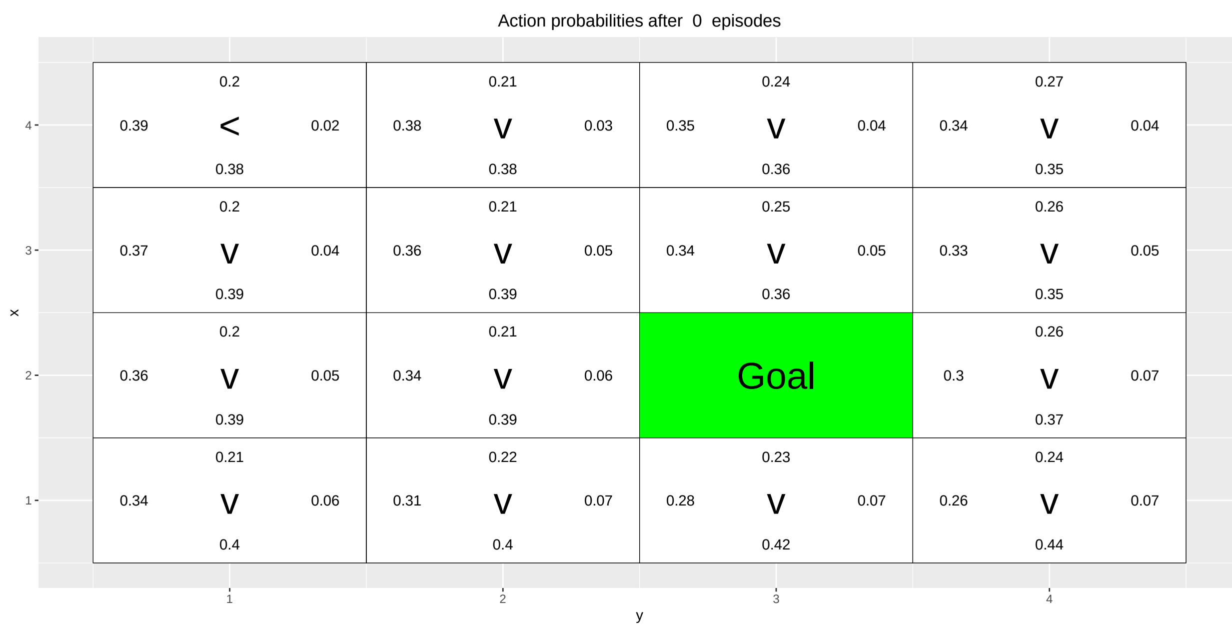


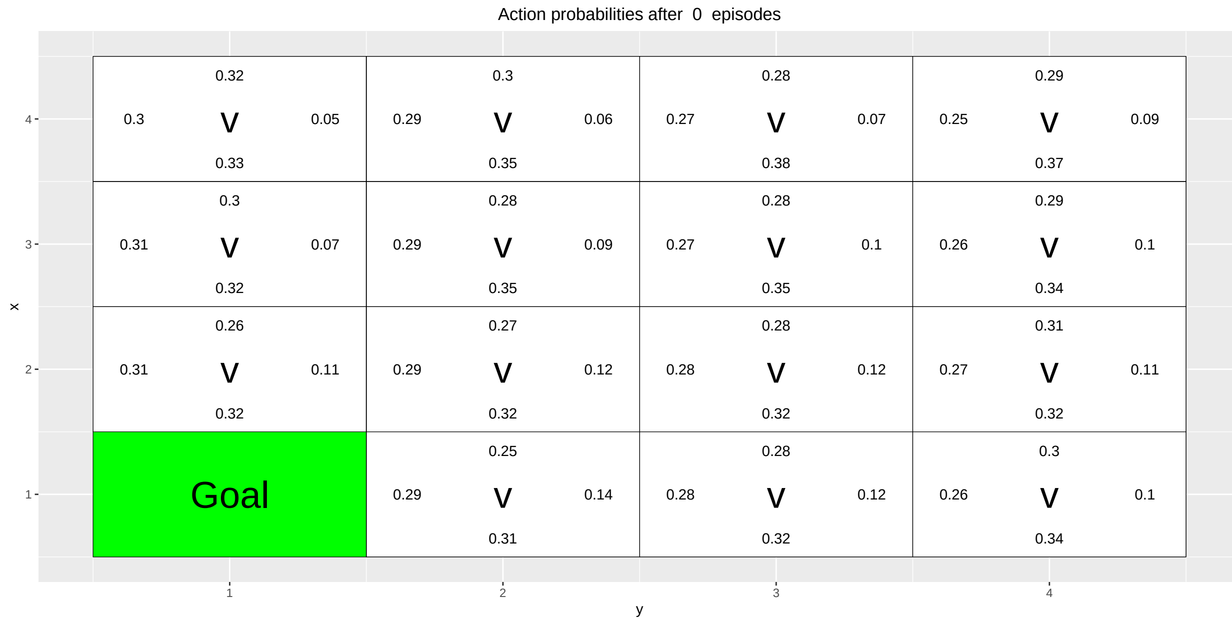
The results indicate that the learned model has a good performance since the model has been trained using all states as the *goal*. In this manner, the generalized model has accurate parameters for this environment with all states having a non-zero reward once. In every validation attempt, the arrows are pointed to the right direction which is an indication of the model accuracy.

Q-learning is not be a good approach for this task since the location of the goal keeps changing. In Q-learning we do not estimate parameters and therefore we do not have a generalized model to apply on a validation set. Q-learning works with q_tables which are deterministic to a specific environment with specific goals and so are not suitable for situations where the target state changes.

Environment E








```
## episode 10
## episode 20
## episode 30
## episode 40
## episode 50
## episode 60
## episode 70
## episode 80
## episode 90
## episode 100
## episode 110
## episode 120
## episode 130
## episode 140
## episode 150
## episode 160
## episode 170
## episode 180
## episode 190
## episode 200
## episode 210
## episode 220
## episode 230
## episode 240
## episode 250
## episode 260
## episode 270
## episode 280
## episode 290
## episode 300
## episode 310
## episode 320
## episode 330
## episode 340
```

```
## episode 350
## episode 360
## episode 370
## episode 380
## episode 390
## episode 400
## episode 410
## episode 420
## episode 430
## episode 440
## episode 450
## episode 460
## episode 470
## episode 480
## episode 490
## episode 500
## episode 510
## episode 520
## episode 530
## episode 540
## episode 550
## episode 560
## episode 570
## episode 580
## episode 590
## episode 600
## episode 610
## episode 620
## episode 630
## episode 640
## episode 650
## episode 660
## episode 670
## episode 680
## episode 690
## episode 700
## episode 710
## episode 720
## episode 730
## episode 740
## episode 750
## episode 760
## episode 770
## episode 780
## episode 790
## episode 800
## episode 810
## episode 820
## episode 830
## episode 840
## episode 850
## episode 860
## episode 870
## episode 880
```

```
## episode 890
## episode 900
## episode 910
## episode 920
## episode 930
## episode 940
## episode 950
## episode 960
## episode 970
## episode 980
## episode 990
## episode 1000
## episode 1010
## episode 1020
## episode 1030
## episode 1040
## episode 1050
## episode 1060
## episode 1070
## episode 1080
## episode 1090
## episode 1100
## episode 1110
## episode 1120
## episode 1130
## episode 1140
## episode 1150
## episode 1160
## episode 1170
## episode 1180
## episode 1190
## episode 1200
## episode 1210
## episode 1220
## episode 1230
## episode 1240
## episode 1250
## episode 1260
## episode 1270
## episode 1280
## episode 1290
## episode 1300
## episode 1310
## episode 1320
## episode 1330
## episode 1340
## episode 1350
## episode 1360
## episode 1370
## episode 1380
## episode 1390
## episode 1400
## episode 1410
## episode 1420
```

```
## episode 1430
## episode 1440
## episode 1450
## episode 1460
## episode 1470
## episode 1480
## episode 1490
## episode 1500
## episode 1510
## episode 1520
## episode 1530
## episode 1540
## episode 1550
## episode 1560
## episode 1570
## episode 1580
## episode 1590
## episode 1600
## episode 1610
## episode 1620
## episode 1630
## episode 1640
## episode 1650
## episode 1660
## episode 1670
## episode 1680
## episode 1690
## episode 1700
## episode 1710
## episode 1720
## episode 1730
## episode 1740
## episode 1750
## episode 1760
## episode 1770
## episode 1780
## episode 1790
## episode 1800
## episode 1810
## episode 1820
## episode 1830
## episode 1840
## episode 1850
## episode 1860
## episode 1870
## episode 1880
## episode 1890
## episode 1900
## episode 1910
## episode 1920
## episode 1930
## episode 1940
## episode 1950
## episode 1960
```

```
## episode 1970
## episode 1980
## episode 1990
## episode 2000
## episode 2010
## episode 2020
## episode 2030
## episode 2040
## episode 2050
## episode 2060
## episode 2070
## episode 2080
## episode 2090
## episode 2100
## episode 2110
## episode 2120
## episode 2130
## episode 2140
## episode 2150
## episode 2160
## episode 2170
## episode 2180
## episode 2190
## episode 2200
## episode 2210
## episode 2220
## episode 2230
## episode 2240
## episode 2250
## episode 2260
## episode 2270
## episode 2280
## episode 2290
## episode 2300
## episode 2310
## episode 2320
## episode 2330
## episode 2340
## episode 2350
## episode 2360
## episode 2370
## episode 2380
## episode 2390
## episode 2400
## episode 2410
## episode 2420
## episode 2430
## episode 2440
## episode 2450
## episode 2460
## episode 2470
## episode 2480
## episode 2490
## episode 2500
```

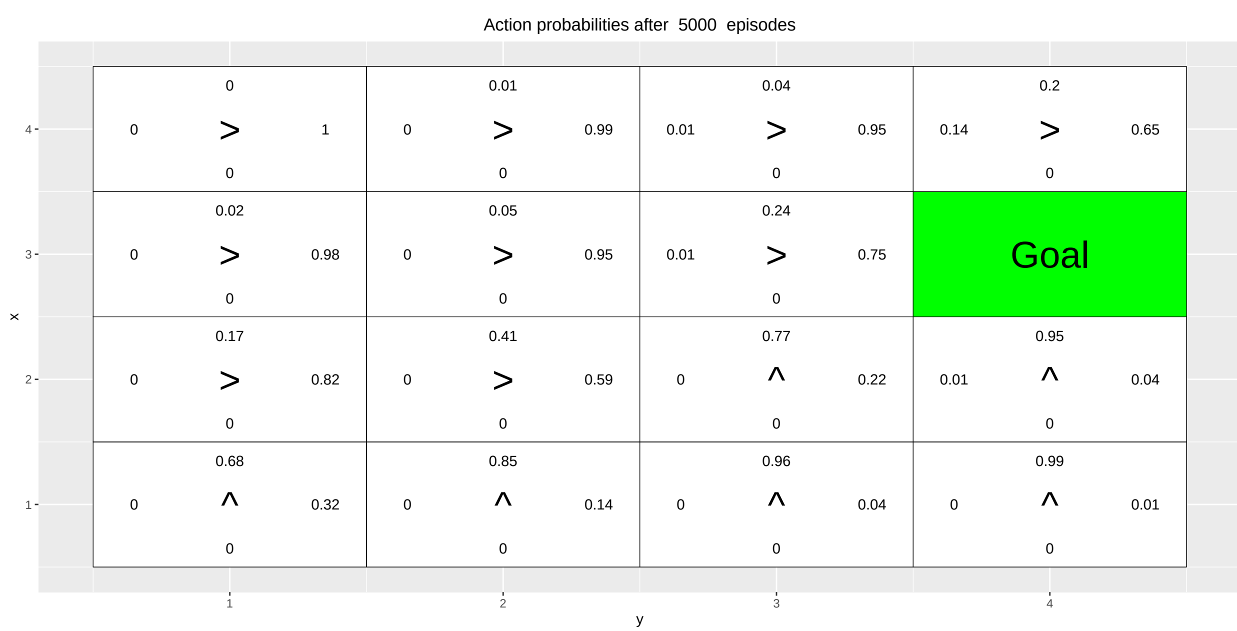
```
## episode 2510
## episode 2520
## episode 2530
## episode 2540
## episode 2550
## episode 2560
## episode 2570
## episode 2580
## episode 2590
## episode 2600
## episode 2610
## episode 2620
## episode 2630
## episode 2640
## episode 2650
## episode 2660
## episode 2670
## episode 2680
## episode 2690
## episode 2700
## episode 2710
## episode 2720
## episode 2730
## episode 2740
## episode 2750
## episode 2760
## episode 2770
## episode 2780
## episode 2790
## episode 2800
## episode 2810
## episode 2820
## episode 2830
## episode 2840
## episode 2850
## episode 2860
## episode 2870
## episode 2880
## episode 2890
## episode 2900
## episode 2910
## episode 2920
## episode 2930
## episode 2940
## episode 2950
## episode 2960
## episode 2970
## episode 2980
## episode 2990
## episode 3000
## episode 3010
## episode 3020
## episode 3030
## episode 3040
```

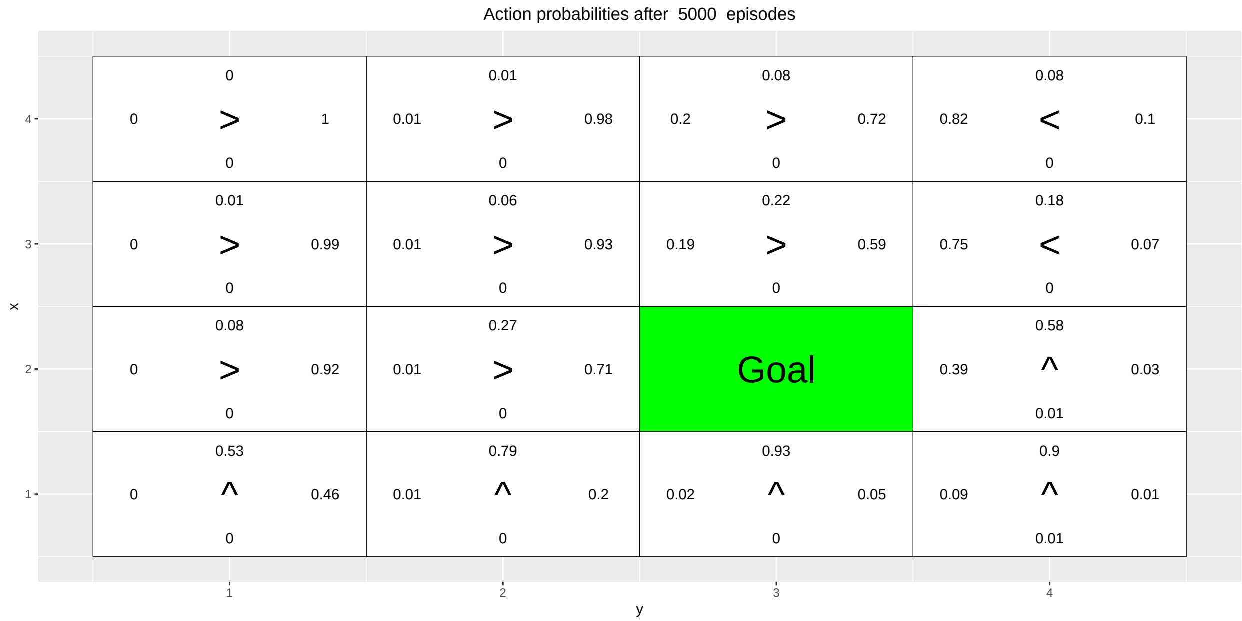
```
## episode 3050
## episode 3060
## episode 3070
## episode 3080
## episode 3090
## episode 3100
## episode 3110
## episode 3120
## episode 3130
## episode 3140
## episode 3150
## episode 3160
## episode 3170
## episode 3180
## episode 3190
## episode 3200
## episode 3210
## episode 3220
## episode 3230
## episode 3240
## episode 3250
## episode 3260
## episode 3270
## episode 3280
## episode 3290
## episode 3300
## episode 3310
## episode 3320
## episode 3330
## episode 3340
## episode 3350
## episode 3360
## episode 3370
## episode 3380
## episode 3390
## episode 3400
## episode 3410
## episode 3420
## episode 3430
## episode 3440
## episode 3450
## episode 3460
## episode 3470
## episode 3480
## episode 3490
## episode 3500
## episode 3510
## episode 3520
## episode 3530
## episode 3540
## episode 3550
## episode 3560
## episode 3570
## episode 3580
```

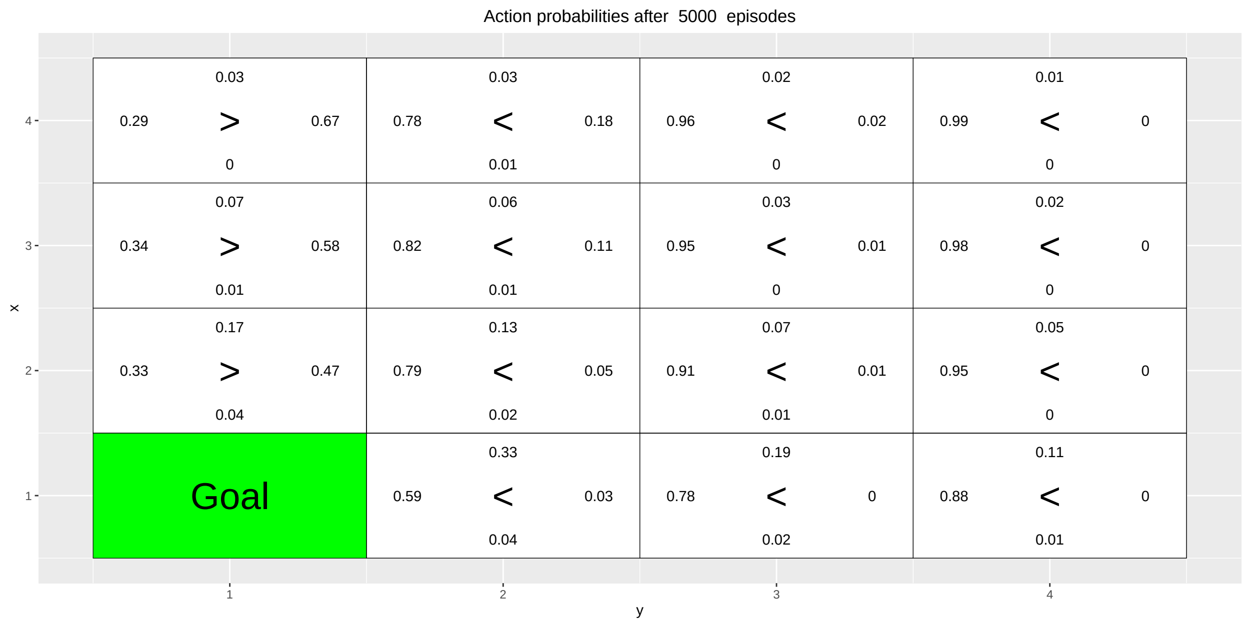
```
## episode 3590
## episode 3600
## episode 3610
## episode 3620
## episode 3630
## episode 3640
## episode 3650
## episode 3660
## episode 3670
## episode 3680
## episode 3690
## episode 3700
## episode 3710
## episode 3720
## episode 3730
## episode 3740
## episode 3750
## episode 3760
## episode 3770
## episode 3780
## episode 3790
## episode 3800
## episode 3810
## episode 3820
## episode 3830
## episode 3840
## episode 3850
## episode 3860
## episode 3870
## episode 3880
## episode 3890
## episode 3900
## episode 3910
## episode 3920
## episode 3930
## episode 3940
## episode 3950
## episode 3960
## episode 3970
## episode 3980
## episode 3990
## episode 4000
## episode 4010
## episode 4020
## episode 4030
## episode 4040
## episode 4050
## episode 4060
## episode 4070
## episode 4080
## episode 4090
## episode 4100
## episode 4110
## episode 4120
```

```
## episode 4130
## episode 4140
## episode 4150
## episode 4160
## episode 4170
## episode 4180
## episode 4190
## episode 4200
## episode 4210
## episode 4220
## episode 4230
## episode 4240
## episode 4250
## episode 4260
## episode 4270
## episode 4280
## episode 4290
## episode 4300
## episode 4310
## episode 4320
## episode 4330
## episode 4340
## episode 4350
## episode 4360
## episode 4370
## episode 4380
## episode 4390
## episode 4400
## episode 4410
## episode 4420
## episode 4430
## episode 4440
## episode 4450
## episode 4460
## episode 4470
## episode 4480
## episode 4490
## episode 4500
## episode 4510
## episode 4520
## episode 4530
## episode 4540
## episode 4550
## episode 4560
## episode 4570
## episode 4580
## episode 4590
## episode 4600
## episode 4610
## episode 4620
## episode 4630
## episode 4640
## episode 4650
## episode 4660
```

```
## episode 4670
## episode 4680
## episode 4690
## episode 4700
## episode 4710
## episode 4720
## episode 4730
## episode 4740
## episode 4750
## episode 4760
## episode 4770
## episode 4780
## episode 4790
## episode 4800
## episode 4810
## episode 4820
## episode 4830
## episode 4840
## episode 4850
## episode 4860
## episode 4870
## episode 4880
## episode 4890
## episode 4900
## episode 4910
## episode 4920
## episode 4930
## episode 4940
## episode 4950
## episode 4960
## episode 4970
## episode 4980
## episode 4990
## episode 5000
```







Given that the model is trained only for the first row, it means that the probabilities or the learned parameters would not yield accurate results when applied to non-visited states. The results approve this as we can see that the arrows in rows 2 or 3 are not pointed to the optimal direction and it might take the robot a long time to find the goal state.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)

# By Jose M. Peña and Joel Oskarsson.
# For teaching purposes.
# jose.m.pena@liu.se.
```

```
##########################################################################################
# Q-learning
##########################################################################################

# install.packages("ggplot2")
# install.packages("vctrs")
library(ggplot2)

# If you do not see four arrows in line 16, then do the following:
# File/Reopen with Encoding/UTF-8

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

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

  # Visualize an environment with rewards.
  # Q-values for all actions are displayed on the edges of each tile.
  # The (greedy) policy for each state is also displayed.
  #
  # Args:
  #   iterations, epsilon, alpha, gamma, beta (optional): for the figure title.
  #   reward_map (global variable): a HxW array containing the reward given at each state.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #   H, W (global variables): environment dimensions.

  df <- expand.grid(x=1:H,y=1:W)
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,1],NA),df$x,df$y)
  df$val1 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,2],NA),df$x,df$y)
  df$val2 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,3],NA),df$x,df$y)
  df$val3 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,q_table[x,y,4],NA),df$x,df$y)
  df$val4 <- as.vector(round(foo, 2))
  foo <- mapply(function(x,y)
    ifelse(reward_map[x,y] == 0,arrows[GreedyPolicy(x,y)],reward_map[x,y]),df$x,df$y)
  df$val5 <- as.vector(foo)
  foo <- mapply(function(x,y) ifelse(reward_map[x,y] == 0,max(q_table[x,y,]),
                                     ifelse(reward_map[x,y]<0,NA,reward_map[x,y])),df$x,df$y)
  df$val6 <- as.vector(foo)

  print(ggplot(df,aes(x = y,y = x)) +
          scale_fill_gradient(low = "white", high = "green", na.value = "red", name = "") +
          geom_tile(aes(fill=val6)) +
          geom_text(aes(label = val1),size = 4,nudge_y = .35,na.rm = TRUE) +
          geom_text(aes(label = val2),size = 4,nudge_x = .35,na.rm = TRUE) +
          geom_text(aes(label = val3),size = 4,nudge_y = -.35,na.rm = TRUE) +
          geom_text(aes(label = val4),size = 4,nudge_x = -.35,na.rm = TRUE) +
          geom_text(aes(label = val5),size = 10) +
```

```r
          geom_tile(fill = 'transparent', colour = 'black') +
          ggtitle(paste("Q-table after ",iterations," iterations\n",
                        "(epsilon = ",epsilon,", alpha = ",alpha,"gamma = ",gamma,", beta = ",beta,")")) +
          theme(plot.title = element_text(hjust = 0.5)) +
          scale_x_continuous(breaks = c(1:W),labels = c(1:W)) +
          scale_y_continuous(breaks = c(1:H),labels = c(1:H)))

}

GreedyPolicy <- function(x, y){

  # Get a greedy action for state (x,y) from q_table.
  #
  # Args:
  #   x, y: state coordinates.
  #   q_table (global variable): a HxWx4 array containing Q-values for each state-action pair.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.
  # Your code here.
  r <- q_table[x,y,]
  best <- which(r == max(r))
  return(sample(best, 1))
}

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

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

  # Your code here.
  r <- q_table[x,y,]
  best <- which(r == max(r))
  if(runif(1) < epsilon) {
    actions <- c(1,2,3,4)
    return(sample(actions, 1))
  }
  else {
    return(sample(best, 1))
  }
}

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

  # Computes the new state after given action is taken. The agent will follow the action
  # with probability (1-beta) and slip to the right or left with probability beta/2 each.
  #
```

```r
  # Args:
  #   x, y: state coordinates.
  #   action: which action the agent takes (in {1,2,3,4}).
  #   beta: probability of the agent slipping to the side when trying to move.
  #   H, W (global variables): environment dimensions.
  #
  # Returns:
  #   The new state after the action has been taken.

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

  return (foo)
}

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

  # Perform one episode of Q-learning. The agent should move around in the
  # environment using the given transition model and update the Q-table.
  # The episode ends when the agent reaches a terminal state.
  #
  # Args:
  #   start_state: array with two entries, describing the starting position of the agent.
  #   epsilon (optional): probability of acting greedily.
  #   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.
  x <- start_state[1]
  y <- start_state[2]
  episode_correction <- 0

  repeat{
    # Follow policy, execute action, get reward.
    a <- GreedyPolicy(x,y)
    A <- EpsilonGreedyPolicy(x,y,epsilon)
    trstate <- transition_model(x,y,A,beta)
    reward <- reward_map[trstate[1],trstate[2]]

    # Q-table update.
    correction <- alpha*(reward+gamma*max(q_table[trstate[1],trstate[2],]) - q_table[x,y,A])
```

```
    episode_correction <- episode_correction + correction
    q_table[x,y,A] <<- q_table[x,y,A] + correction
    x <- trstate[1]
    y <- trstate[2]
    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))
  }
}


GreedyPolicy <- function(x, y){
  r <- q_table[x,y,]
  best <- which(r == max(r))
  return(sample(best, 1))
}


EpsilonGreedyPolicy <- function(x, y, epsilon){
  r <- q_table[x,y,]
  best <- which(r == max(r))
  if(runif(1) < epsilon) {
    actions <- c(1,2,3,4)
    return(sample(actions, 1))
  }
  else {
    return(sample(best, 1))
  }
}


q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma = 0.95, beta = 0){
  x <- start_state[1]
  y <- start_state[2]
  episode_correction <- 0

  repeat{
    # Follow policy, execute action, get reward.
    a <- GreedyPolicy(x,y)
    A <- EpsilonGreedyPolicy(x,y,epsilon)
    trstate <- transition_model(x,y,A,beta)
    reward <- reward_map[trstate[1],trstate[2]]

    # Q-table update.
    correction <- alpha*(reward+gamma*max(q_table[trstate[1],trstate[2],]) - q_table[x,y,A])
    episode_correction <- episode_correction + correction
    q_table[x,y,A] <<- q_table[x,y,A] + correction
    x <- trstate[1]
    y <- trstate[2]
    if(reward!=0)
      # End episode.
      return (c(reward,episode_correction))
  }
```

```
}


# 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)
}


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])
```

```r
    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")
}



# Environment C (the effect of beta).

H <- 3
W <- 6

reward_map <- matrix(0, nrow = H, ncol = W)
reward_map[1,2:5] <- -1
reward_map[1,6] <- 10

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

vis_environment()

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

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

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


# By Jose M. Peña and Joel Oskarsson.
# For teaching purposes.
# jose.m.pena@liu.se.
```

```
##############################################################################################
# REINFORCE
##############################################################################################
library(tensorflow)
#install_tensorflow()
library(keras)

# install.packages("keras")


# install.packages("ggplot2")
# install.packages("vctrs")
library(ggplot2)

# If you do not see four arrows in line 19, then do the following:
# File/Reopen with Encoding/UTF-8

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){

  # Visualize an environment with rewards.
  # Probabilities for all actions are displayed on the edges of each tile.
  # The (greedy) policy for each state is also displayed.
  #
  # Args:
  #   goal: goal coordinates, array with 2 entries.
  #   episodes, epsilon, alpha, gamma, beta (optional): for the figure title.
  #   H, W (global variables): environment dimensions.

  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)
```

```r
  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){

  # 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)
}

DeepPolicy_dist <- function(x, y, goal_x, goal_y){

  # Get distribution over actions for state (x,y) and goal (goal_x,goal_y) from the deep policy.
  #
  # Args:
  #   x, y: state coordinates.
  #   goal_x, goal_y: goal coordinates.
  #   model (global variable): NN encoding the policy.
  #
  # Returns:
```

```
  #   A distribution over actions.

  foo <- matrix(data = c(x,y,goal_x,goal_y), nrow = 1)

  # return (predict_proba(model, x = foo))
  return (predict_on_batch(model, x = foo)) # Faster.

}

DeepPolicy <- function(x, y, goal_x, goal_y){

  # Get an action for state (x,y) and goal (goal_x,goal_y) from the deep policy.
  #
  # Args:
  #   x, y: state coordinates.
  #   goal_x, goal_y: goal coordinates.
  #   model (global variable): NN encoding the policy.
  #
  # Returns:
  #   An action, i.e. integer in {1,2,3,4}.

  foo <- DeepPolicy_dist(x,y,goal_x,goal_y)

  return (sample(1:4, size = 1, prob = foo))

}

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

  # Train the policy network on a rolled out trajectory.
  #
  # Args:
  #   states: array of states visited throughout the trajectory.
  #   actions: array of actions taken throughout the trajectory.
  #   goal: goal coordinates, array with 2 entries.
  #   gamma: discount factor.

  # Construct batch for training.
  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)

}
```

```r
reinforce_episode <- function(goal, gamma = 0.95, beta = 0){

  # Rolls out a trajectory in the environment until the goal is reached.
  # Then trains the policy using the collected states, actions and rewards.
  #
  # Args:
  #   goal: goal coordinates, array with 2 entries.
  #   gamma (optional): discount factor.
  #   beta (optional): probability of slipping in the transition model.

  # Randomize starting position.
  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')
```

```r
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))
}

show_validation(5000)


# Environment E (training with top row goal positions)

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))
}

show_validation(5000)
```