

Report Reinforcement Learning

Aman Kumar Nayak

10/7/2020

Question 1

Q-Learning

Q-Learning is an off-policy Reinforcement Learning algorithm. It uses table to store Q-Values of all possible state-action possible pairs. Table is updated using Bellman equation while action selection can be done using ϵ - greedy policy or some other policy.

We will work with a grid world environment consisting of $H \times W$ tiles laid out in a 2-dimensional grid. An agent acts by moving up, down, left or right in the grid-world. This corresponds to the following Markov decision process:

State Space : $S = \{(x,y) | x \in \{1, \dots, H\}, y \in \{1, \dots, H\}\}$ Action Space : $A = \{\text{up, down, left, right}\}$

Additionally, we assume state space to be fully observable. The reward function is a deterministic function of the state and does not depend on the actions taken by the agent. We assume the agent gets the reward as soon as it moves to a state. The transition model is defined by the agent moving in the direction chosen with probability $(1-\beta)$. The agent might also slip and end up moving in the direction to the left or right of its chosen action, each with probability $\beta/2$. The transition model is unknown to the agent, forcing us to resort to model-free solutions. The environment is episodic and all states with a non-zero reward are terminal. Throughout this lab we use integer representations of the different actions: Up=1, right=2, down=3 and left=4

Environment A

For our first environment, we will use $H = 5$ and $W = 7$. This environment includes a reward of 10 in state (3,6) and a reward of -1 in states (2,3), (3,3) and (4,3). We specify the rewards using a reward map in the form of a matrix with one entry for each state. States with no reward will simply have a matrix entry of 0. The agent starts each episode in the state (3,1). The function `vis_environment` is used to visualize the environment and learned action values and policy.

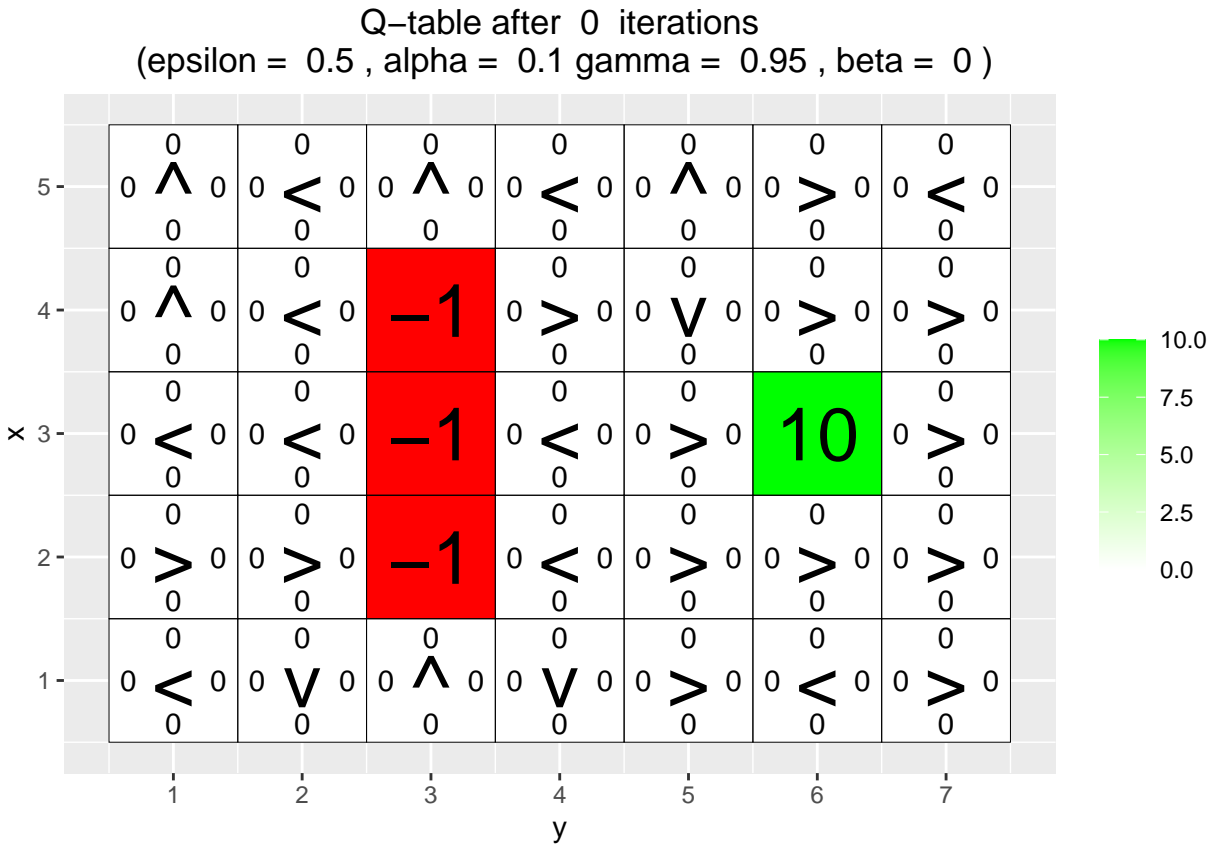
Q-Learning is off-policy, it evaluates one policy (target policy) while observing behavior policy. Finding optimal action value function Q_* under arbitrary behavior policy is achieved by policy iteration. Q converges to optimal action value function $Q_*(S, A)$ and its greedy converges to optimal policy under appropriate choice of learning rate (α) over time.

$$Q_*(S, A) = Q(S, A) + \alpha(R + \gamma(\max_a Q(S', a)) - Q(S, A))$$

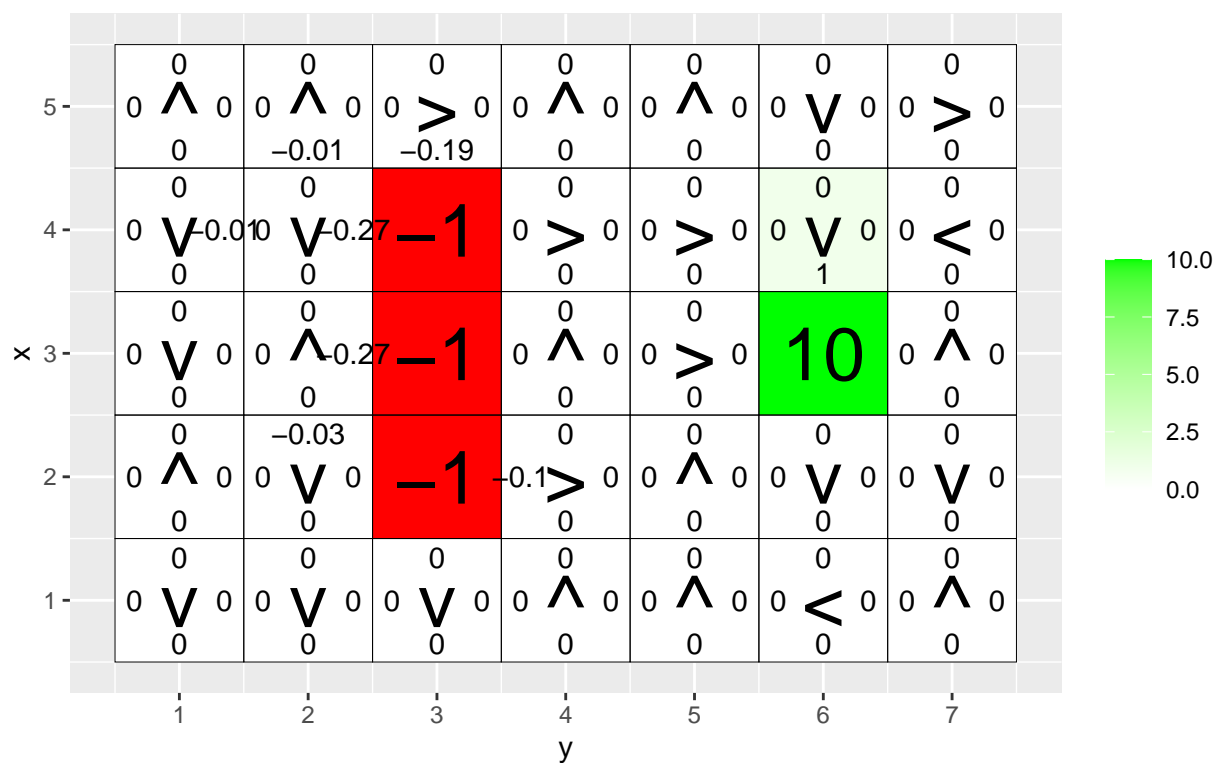
Here :

S : is current state A : Action chosen by ϵ -Greedy policy S' : Transition state based on current state and chosen action A R : is Reward for moving to transition state S' a : Action chosen by Greedy Policy α :

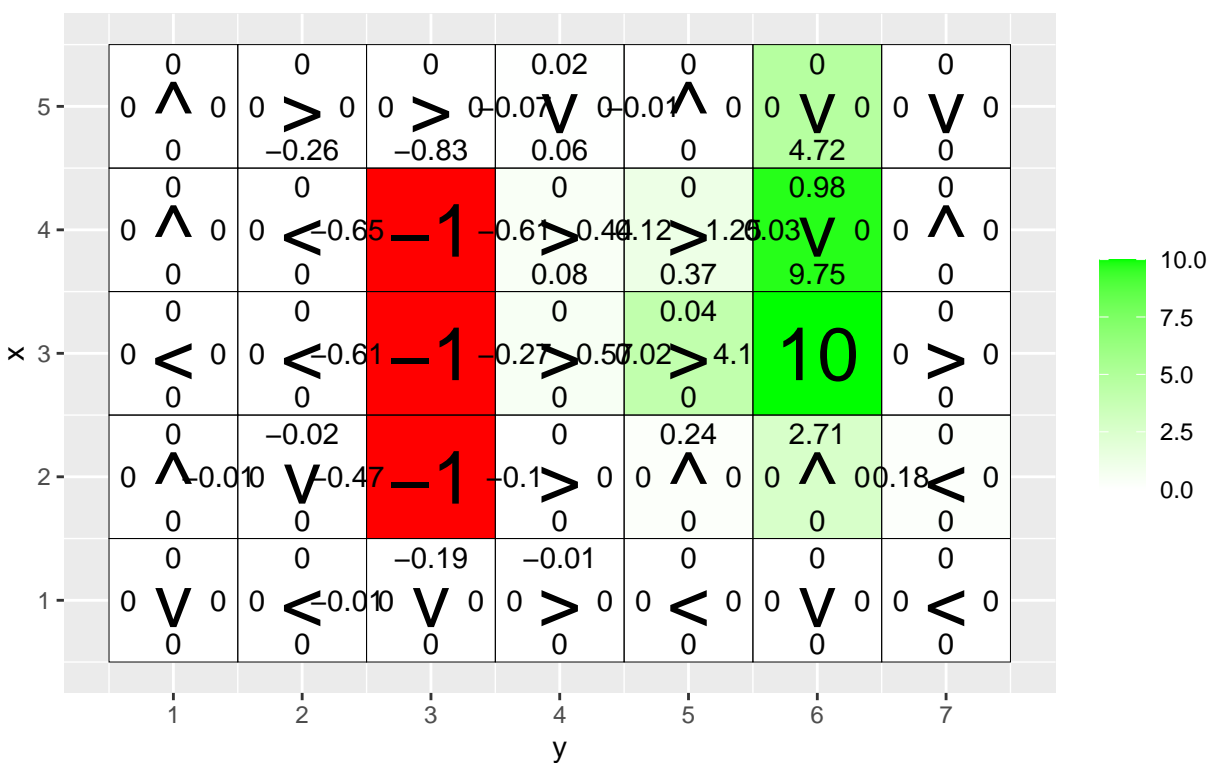
Learning Rate γ : Discount factor, ranges between (0 , 1) which control importance of future rewards in comparison to current one.

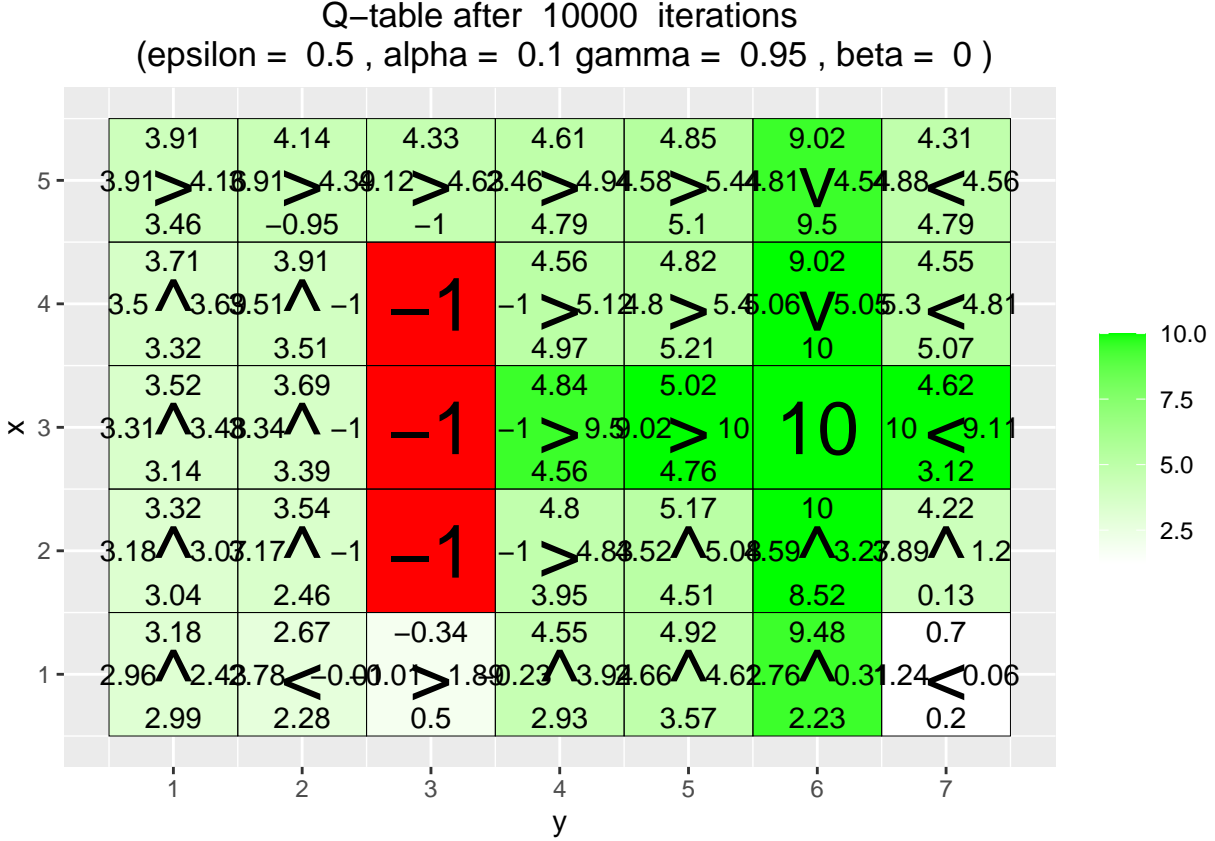


Q-table after 10 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)



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





What has the agent learned after the first 10 episodes ?

From the above Q-Table named “Q-Table after 10 Iterations” with ten iterations, we have a small number of episodes and, the agent is not able to learn optimal paths towards maximum reward location as it is in the exploring stage, so there is no significant learning.

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

From plot “Q-Table after 10000 Iterations”, the model has learned multiple paths towards the maximum reward position in 10000 episodes, and thus when left in any grid position, it will converge to the maximum reward position. The final greedy policy is optimal in terms of reaching to highest reward grid position when we have a high number of episodes in the learning phase.

Does the agent learn that there are multiple paths to get to the positive reward ? If not, what could be done to make the agent learn this ?

In “Q-Table after 10000 Iterations”, looking at maximum rewards for each grid, it is observed that the agent did learn multiple paths to reach to maximum reward position in the grid based on where it started.

Now if we want to explore more paths during during learning, we can set ϵ parameter (Exploration probability) with much more higher value. It is explained below in more details.

Environment B

Investigate how the ϵ and γ 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 $\alpha = 0.1$

Exploration vs Exploitation

Exploration allows an agent to improve its current knowledge about each action, hopefully leading to long-term benefit. Improving the accuracy of the estimated action-values, enables an agent to make more informed decisions in the future.

Exploitation on the other hand, chooses the greedy action to get the most reward by exploiting the agent's current action-value estimates. But by being greedy with respect to action-value estimates, may not actually get the most reward and lead to sub-optimal behavior.

When an agent explores, it gets more accurate estimates of action-values. And when it exploits, it might get more reward. It cannot, however, choose to do both simultaneously.

Use of ϵ parameter:

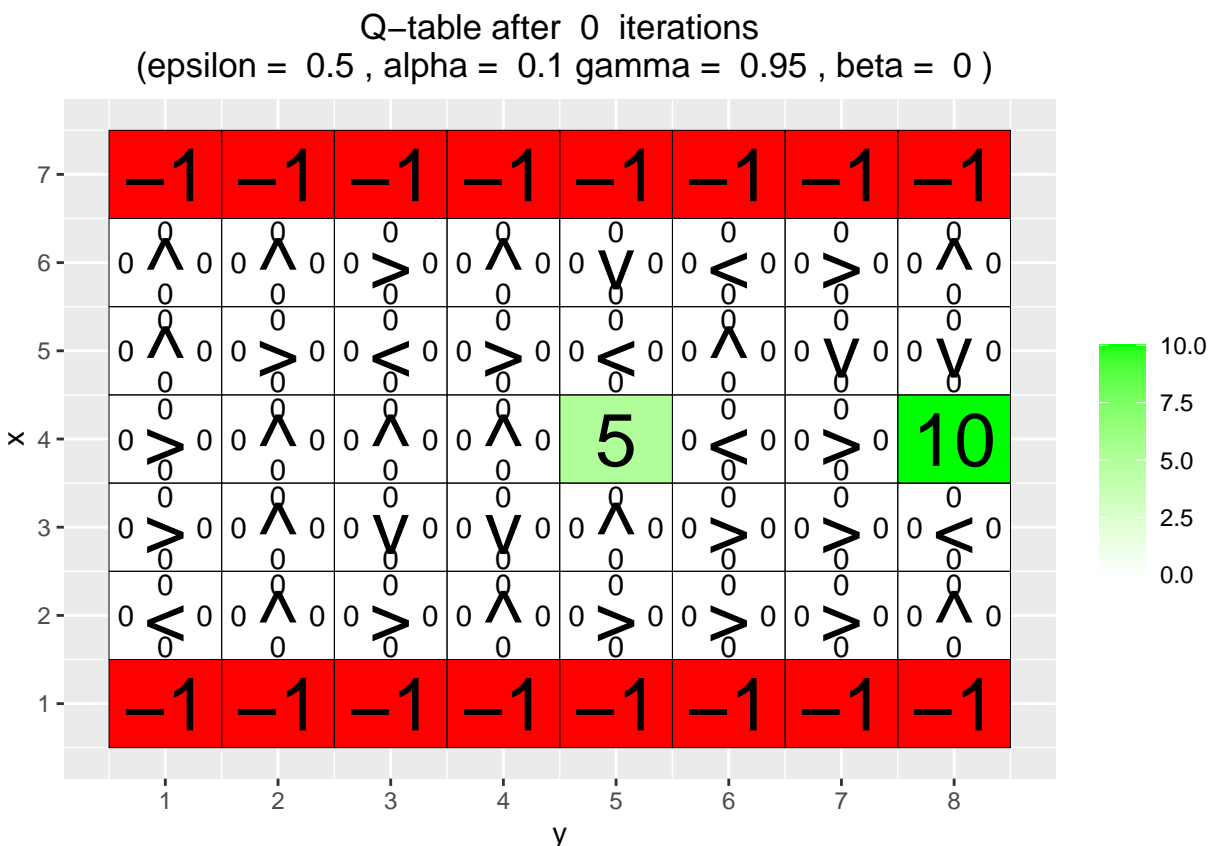
ϵ -Greedy is a simple method to balance exploration and exploitation by choosing between exploration and exploitation randomly.

Now with probability of $1-\epsilon$ we decide to exploit and with probability of ϵ we decide to explore.

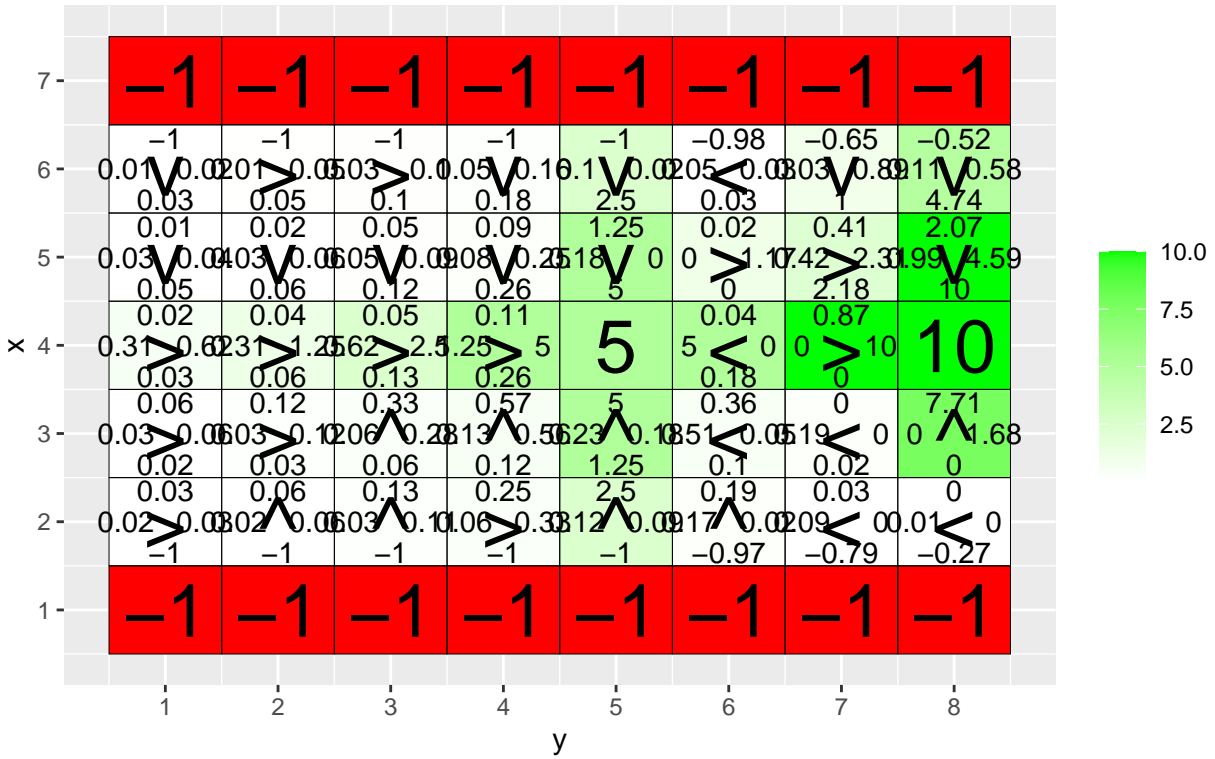
Use of γ (discount factor) parameter:

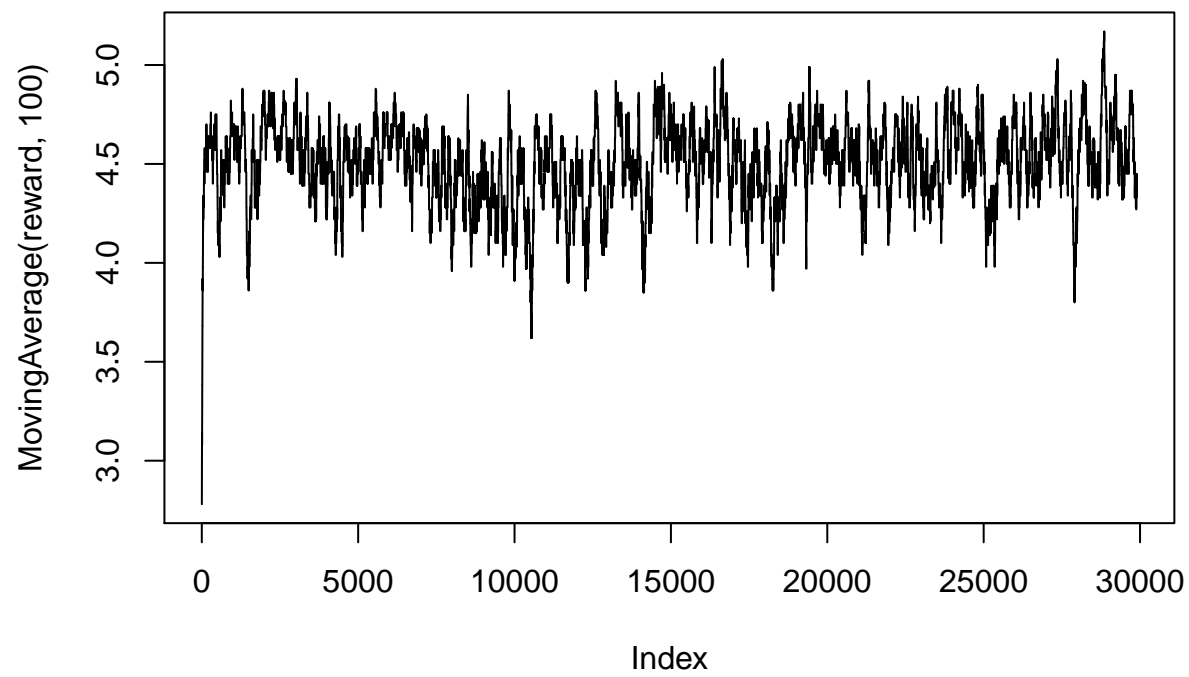
The discount factor essentially determines how much the reinforcement learning agents cares about rewards in the distant future relative to those in the immediate future. If $\gamma = 0$, the agent will be completely shortsighted and only learn about actions that produce an immediate reward. If $\gamma = 1$, the agent will evaluate each of its actions based on the sum total of all of its future rewards.

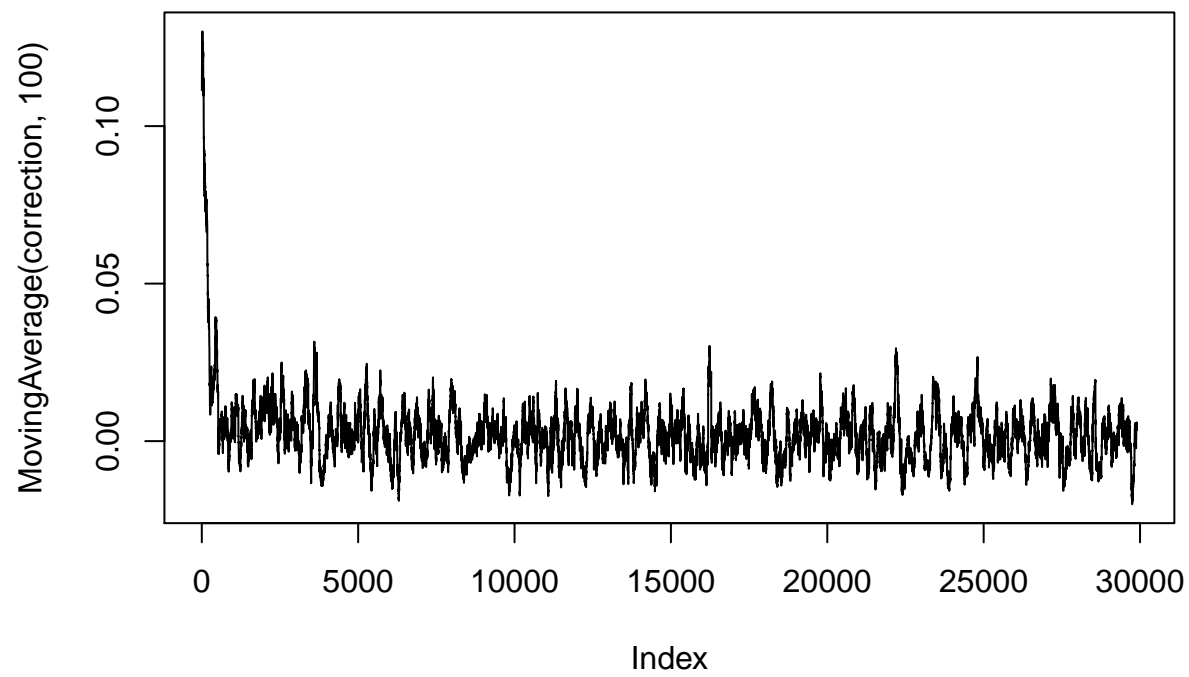
Epsilon set as 0.5



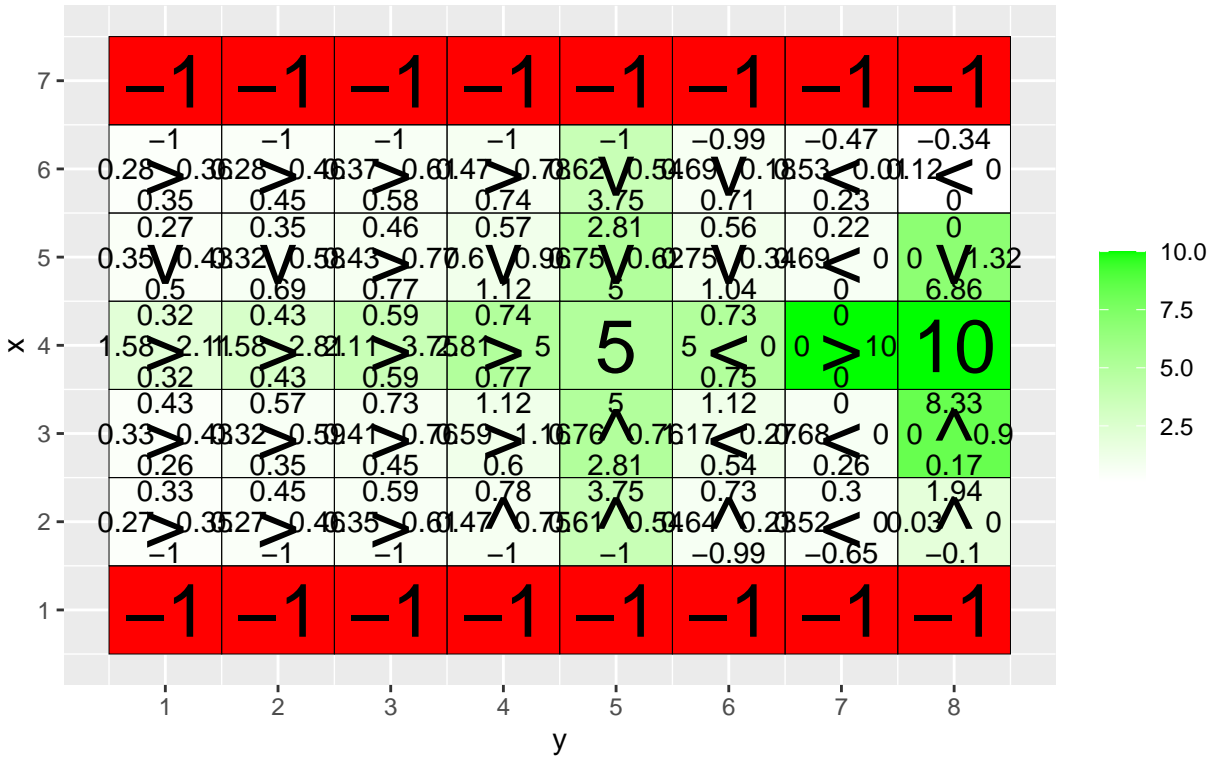
Q-table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.5 , beta = 0)

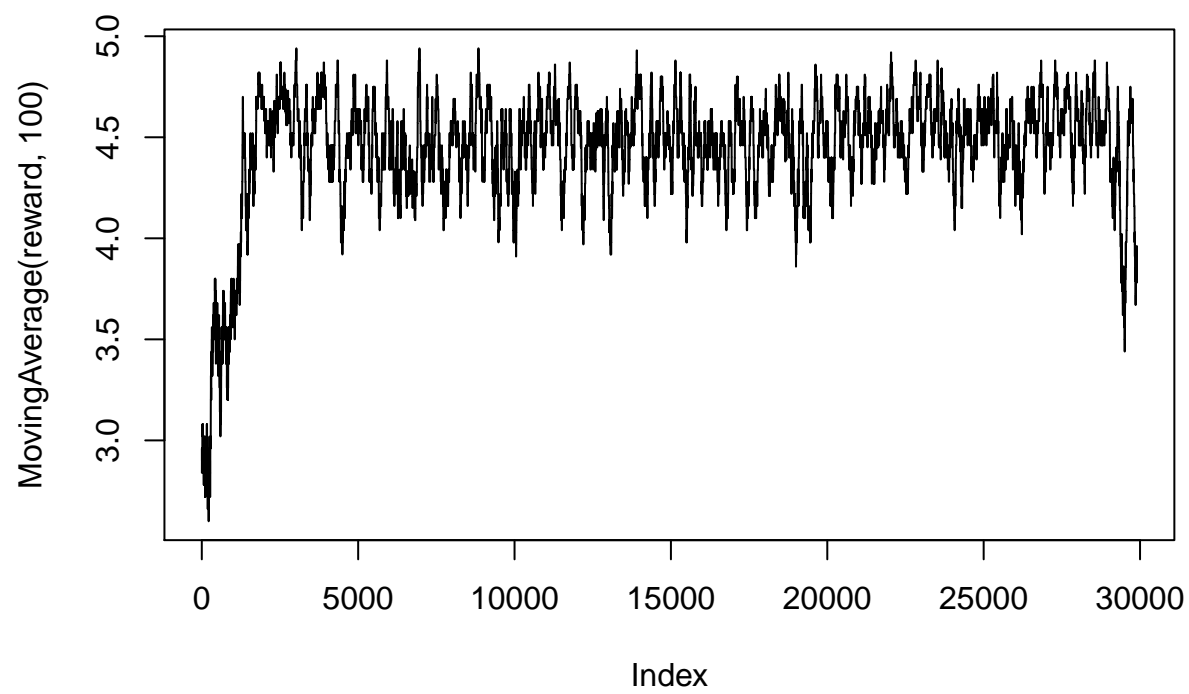


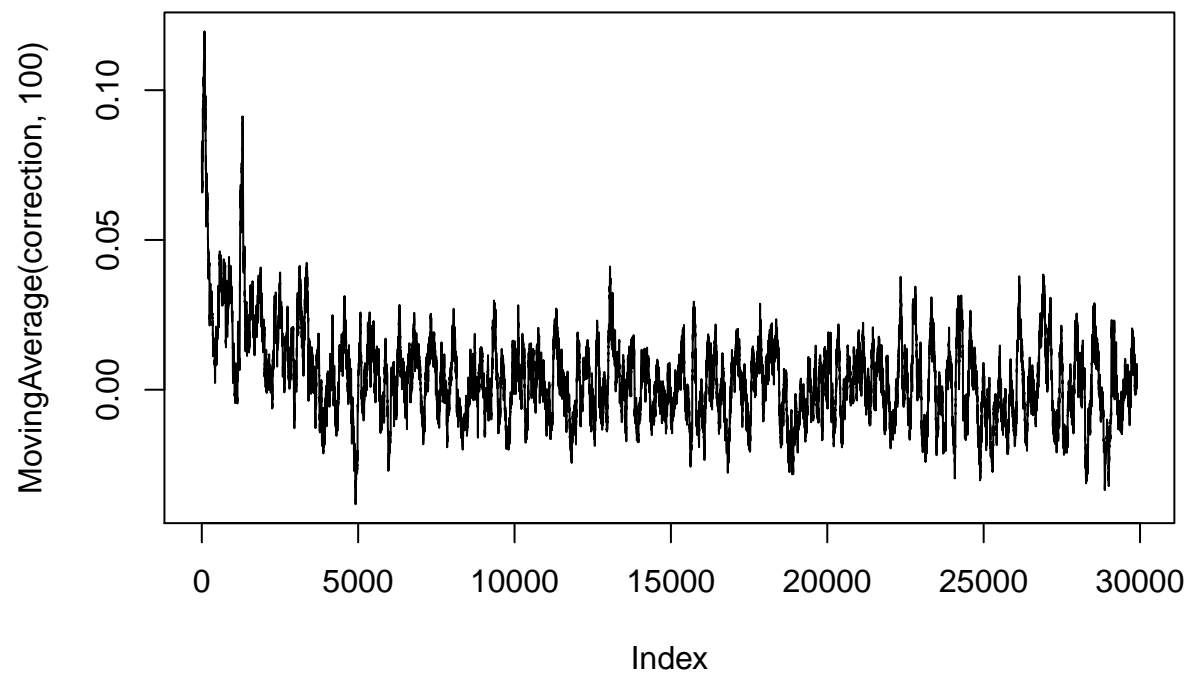




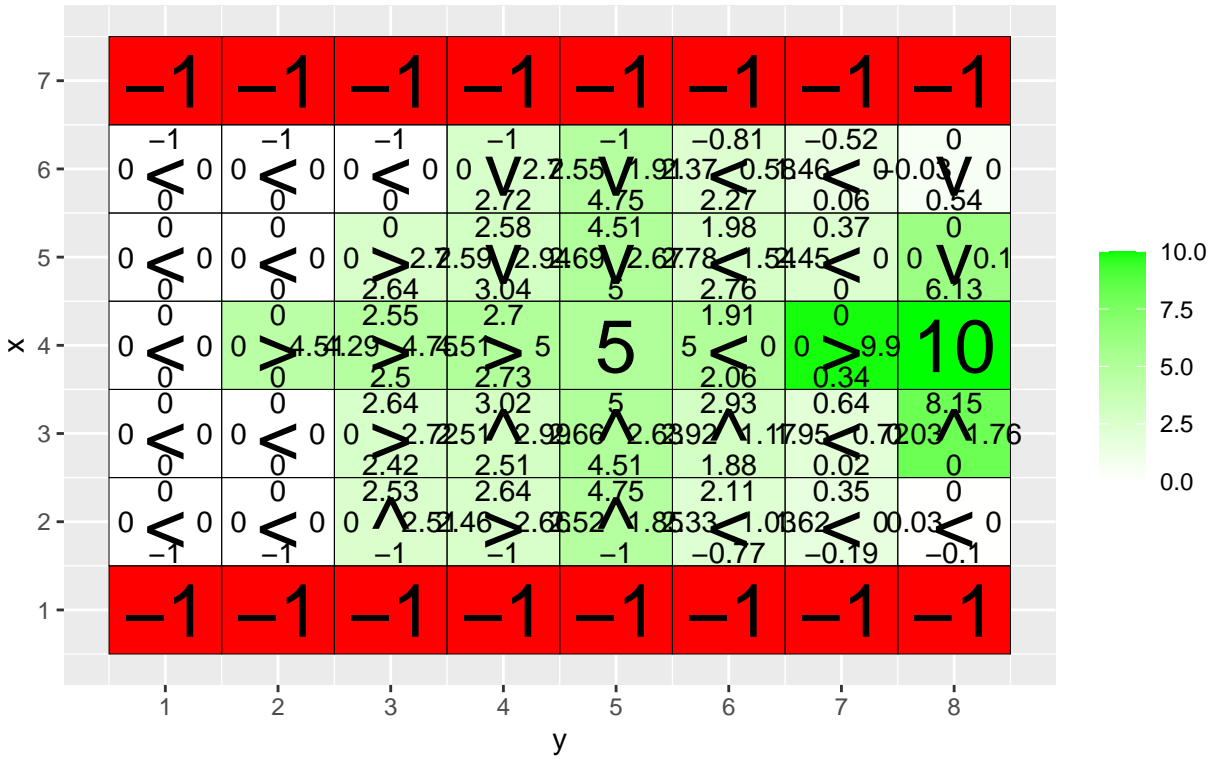
Q-table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)

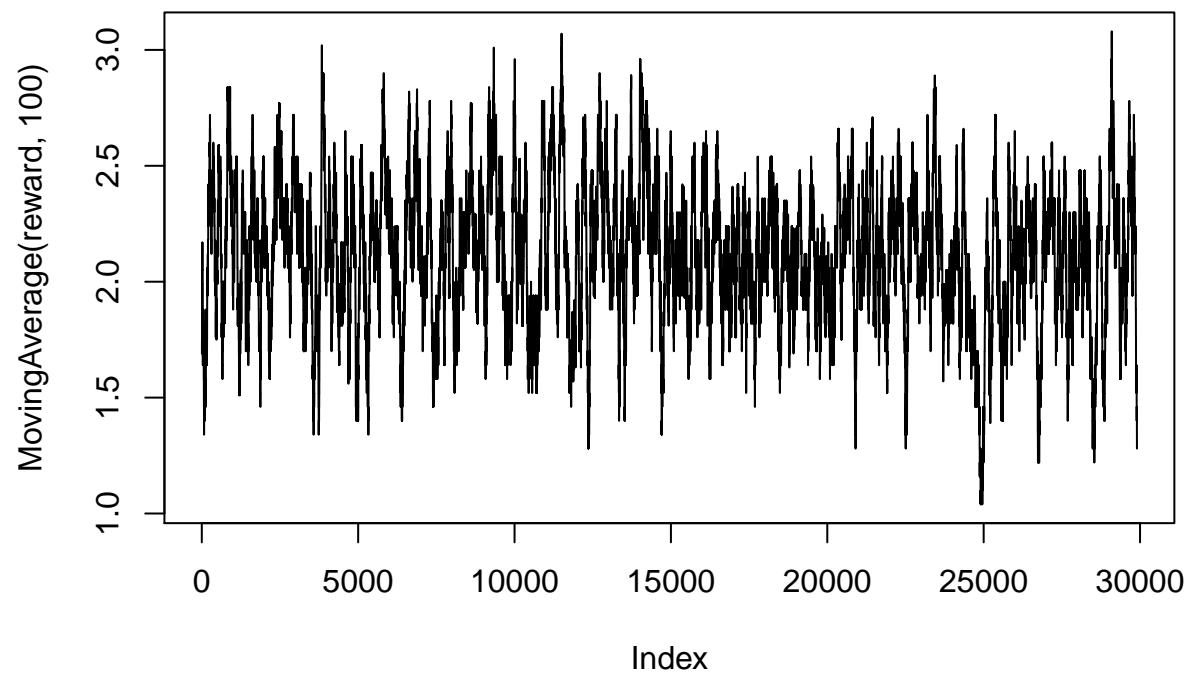


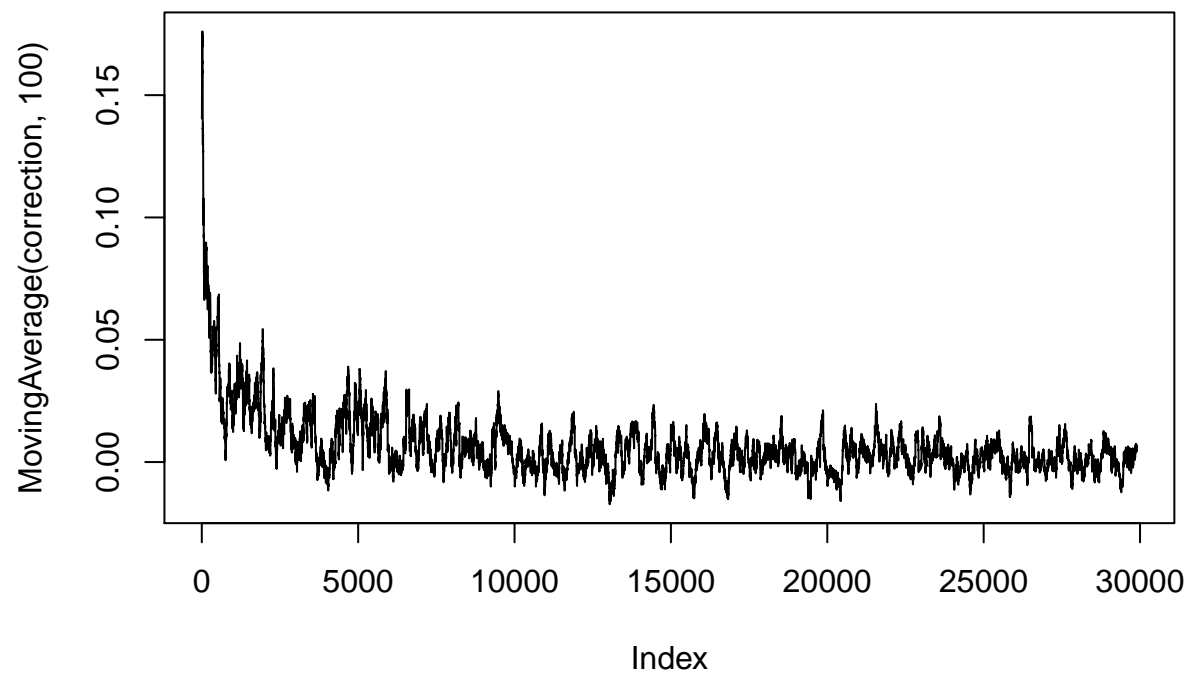




Q-table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)

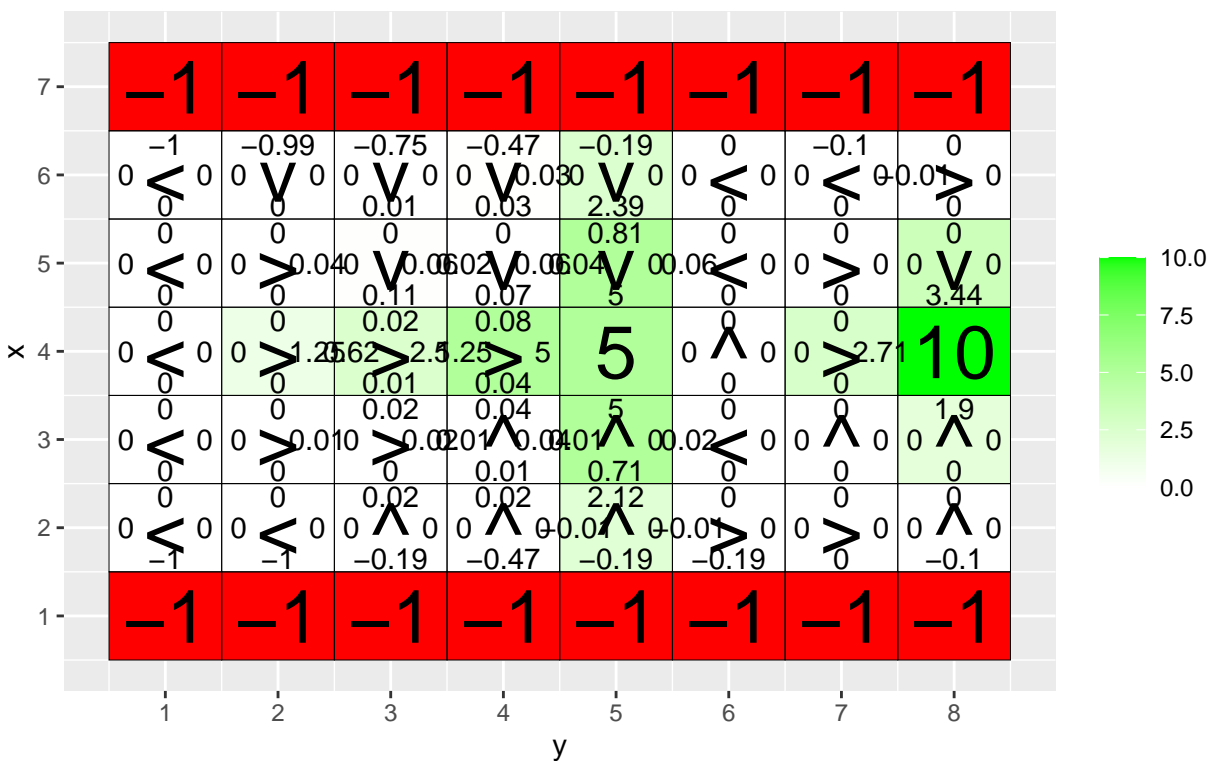


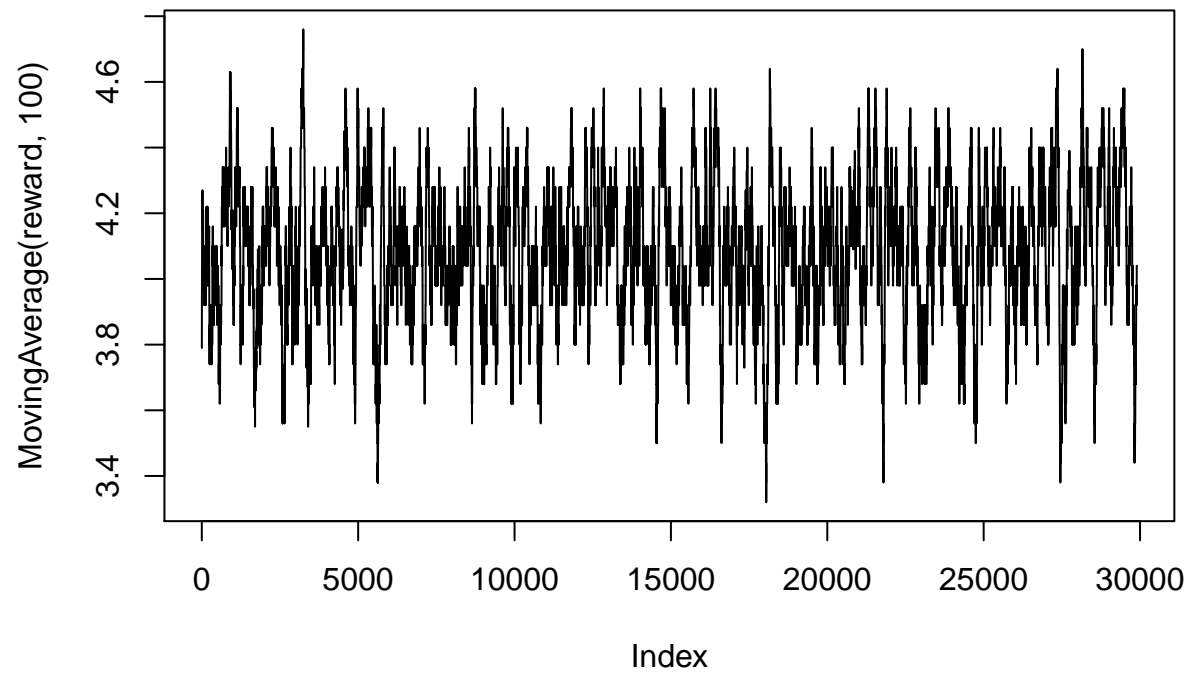


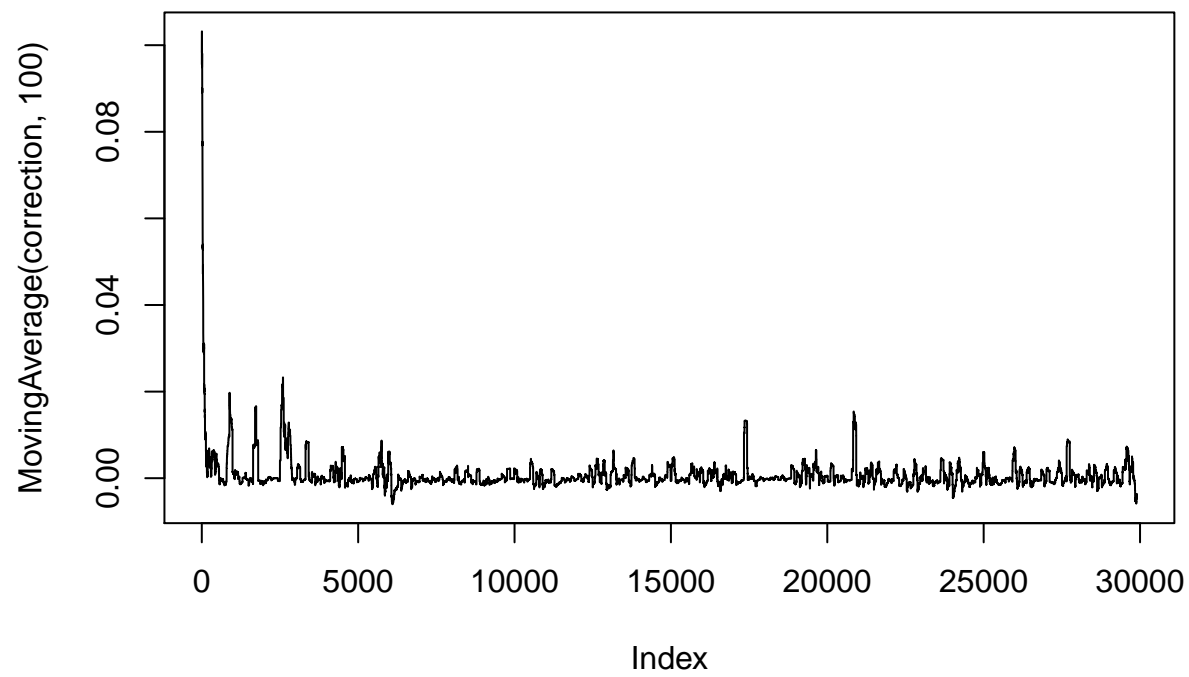


```
## Epsilon set as 0.1
```

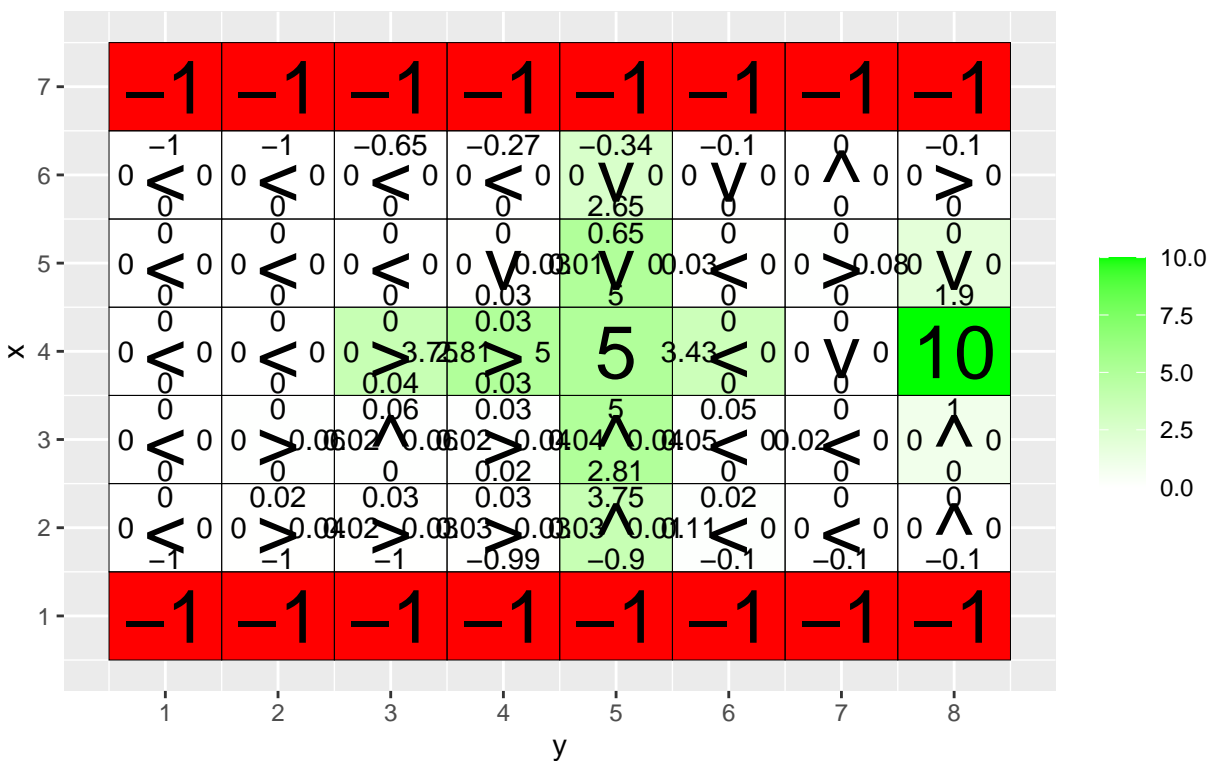

Q-table after 30000 iterations
 (epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)

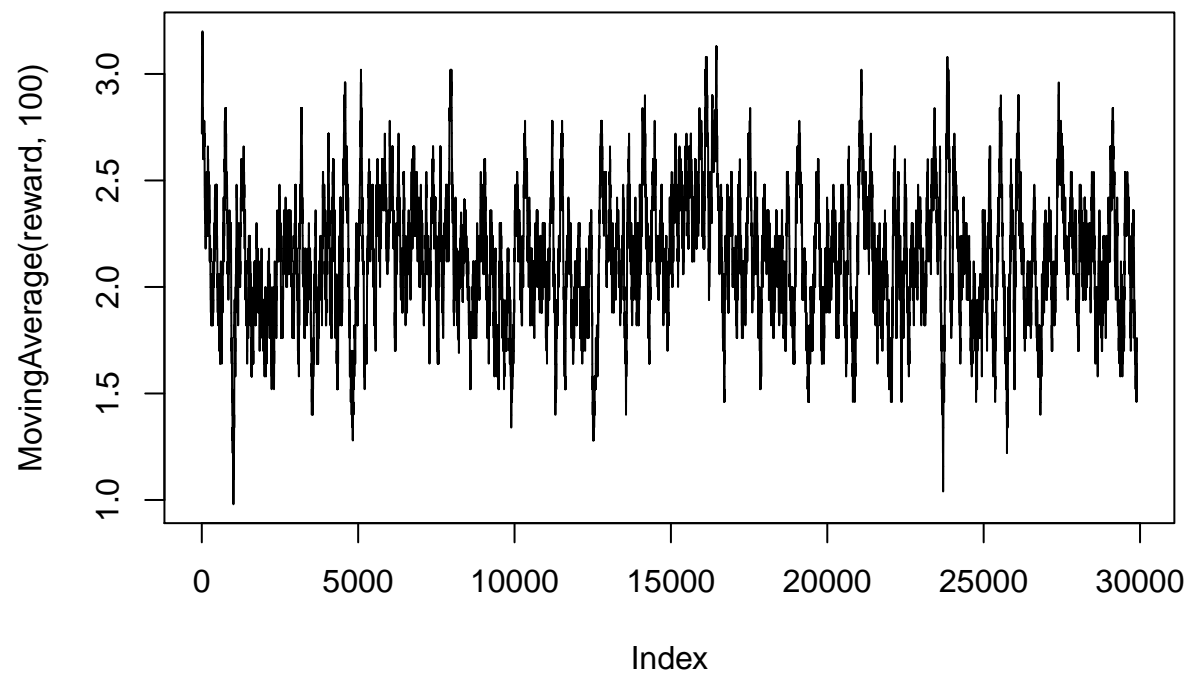


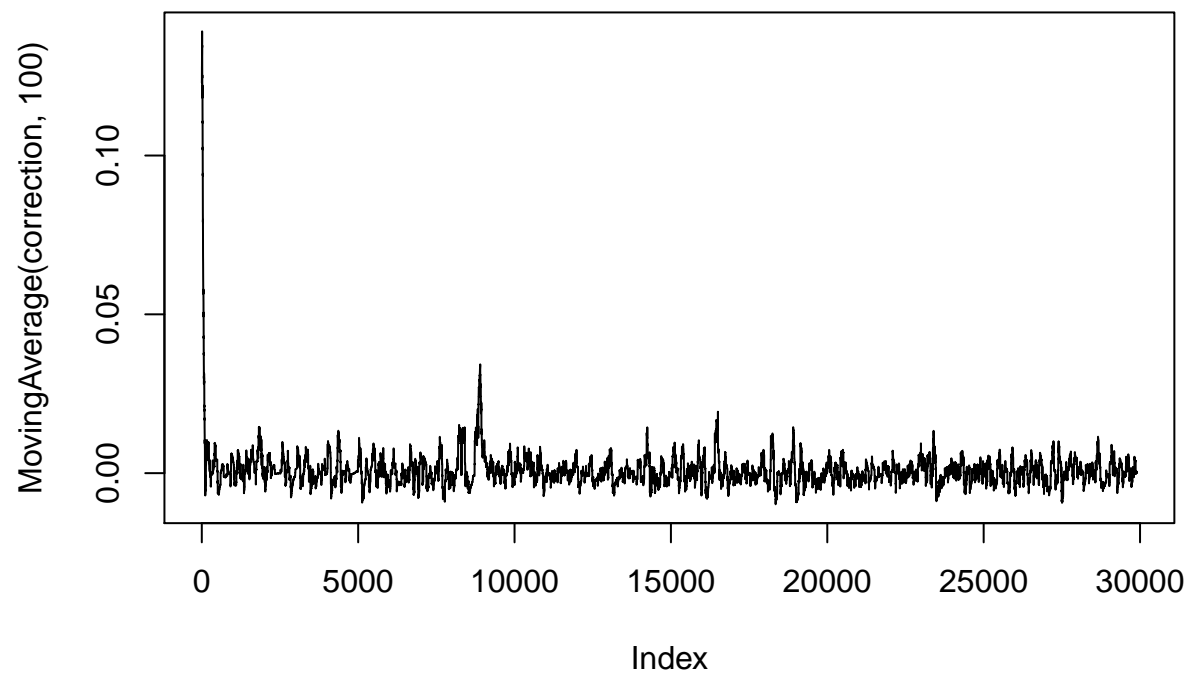




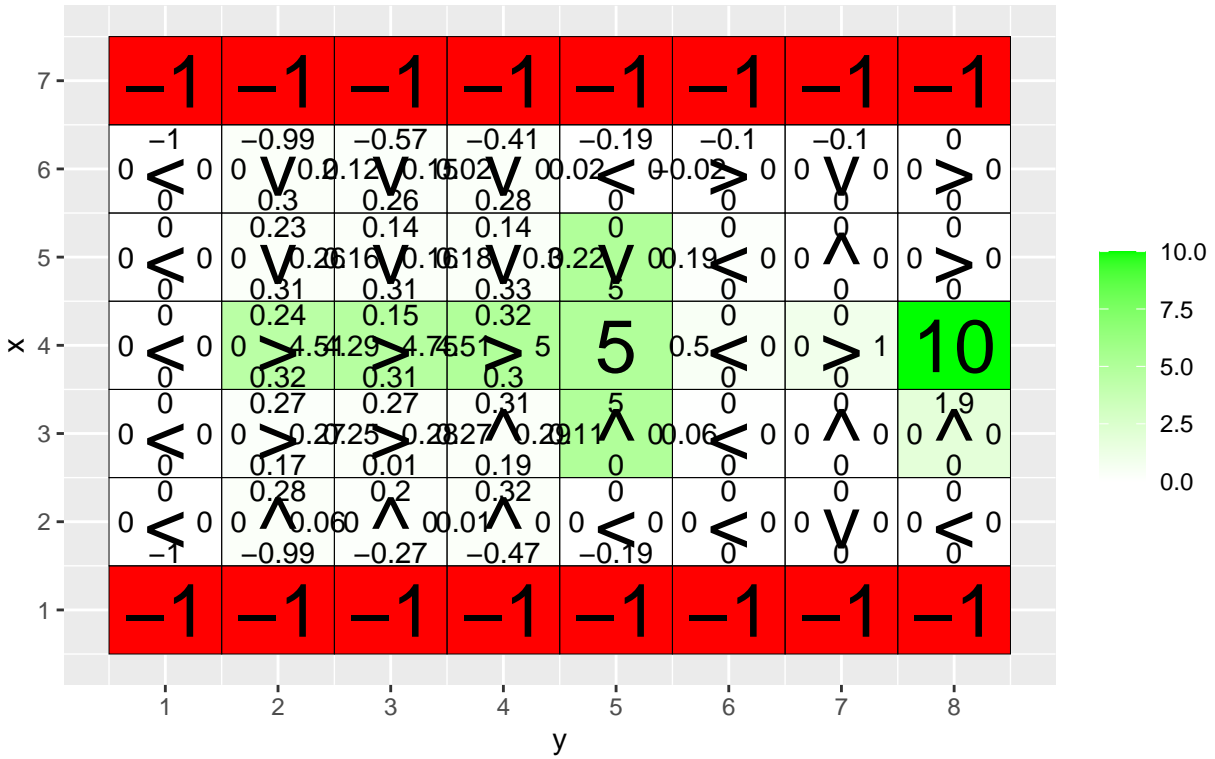
Q-table after 30000 iterations
 (epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)

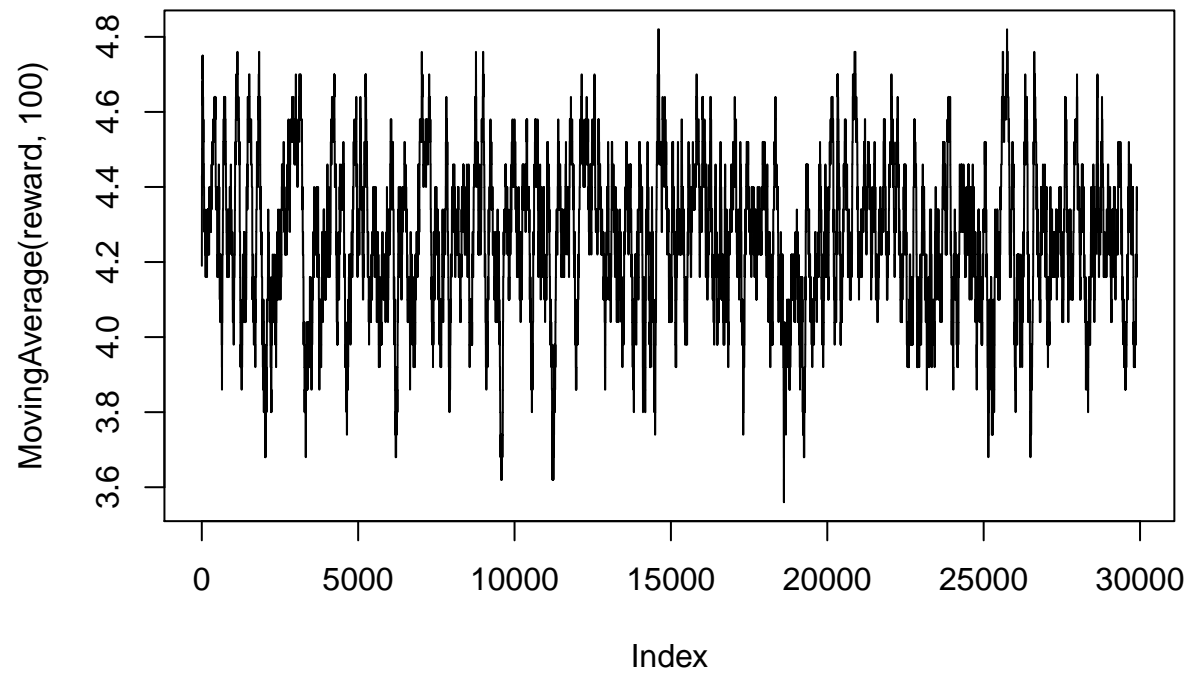


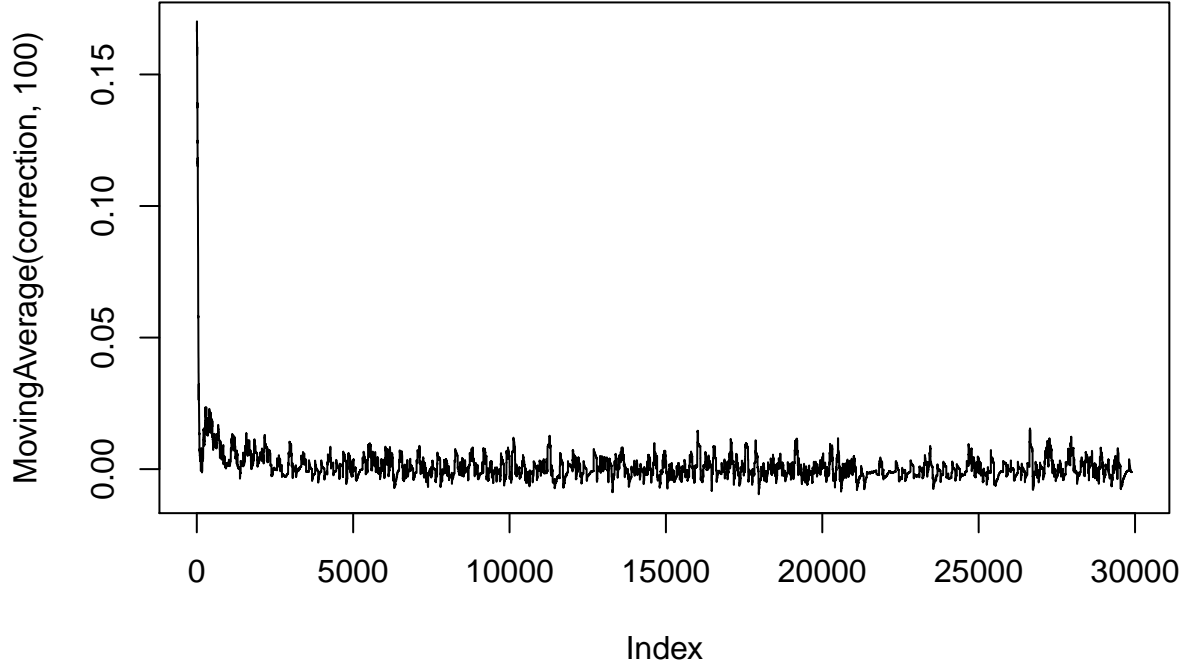




Q-table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.95 , beta = 0)







Epsilon Impact

When $\epsilon = 0.5$, we give equal probability to exploit and explore thus it can be seen that we can find multiple directions towards high reward stages while when we set $\epsilon = 0.1$, the system is more exploiting in nature and select path with only high rewards and thus we are not able to find multiple directions towards high reward stages which can be seen by less number of green grids when $\epsilon = 0.1$.

Gamma Impact

With different values of γ (discount factor) it can be seen that correction is low for $\gamma = 0.5$ while since it is looking for a high reward in near future only and when we increase this to $\gamma = 0.75$ we can see the higher variance in correction (when ϵ is 0.5) as now we are more interested in long term rewards than a short term one and when we increase it $\gamma = 0.95$, we are giving very very high value to future rewards and thus the system is more tuned for future rewards.

Combined Impact of Epsilon and Gamma

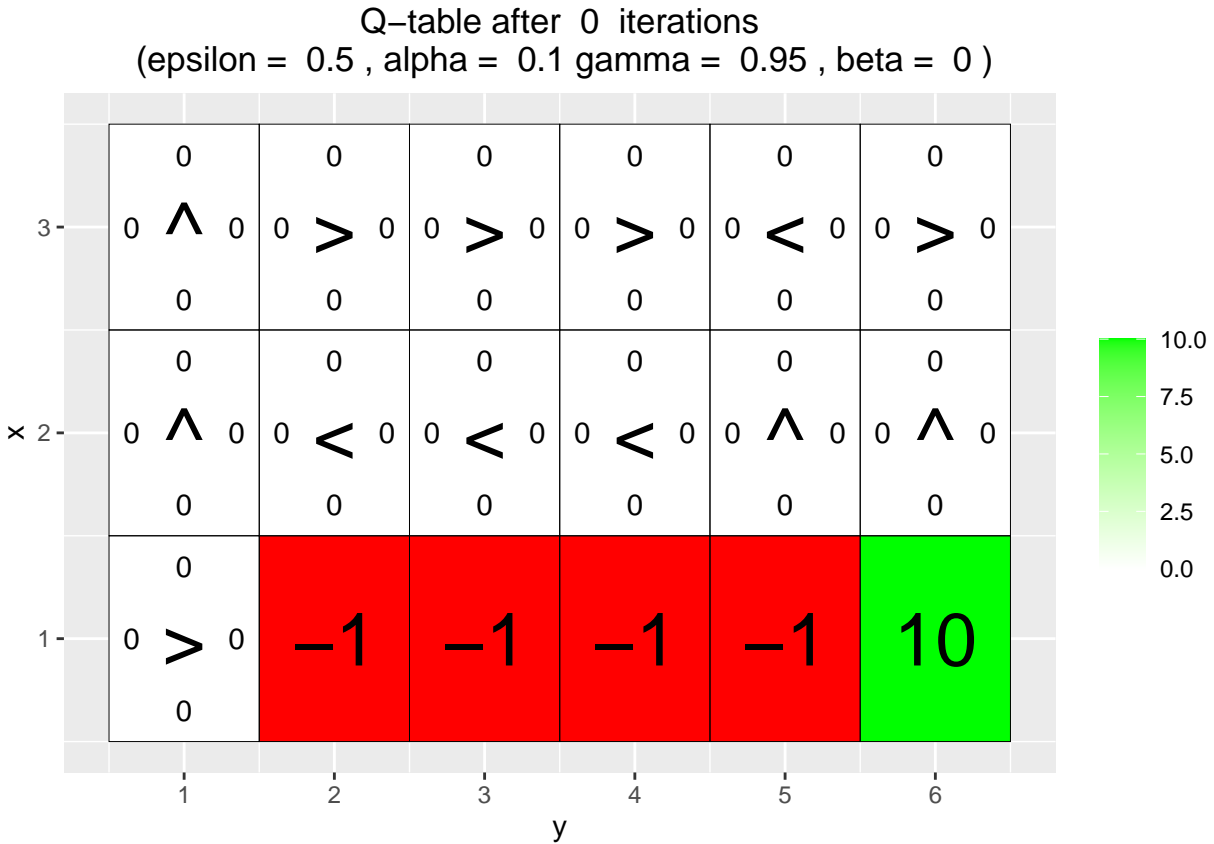
When we have ϵ as 0.5 and we keep on increasing γ , since model have balance exploration and exploitation and with every increase in γ we can see that system is giving more and more value to future rewards thus we get large number of episodes converging at max reward in grid which is 10.

But when we have ϵ as 0.1 and even when we increase γ , system being exploiting ($1-\epsilon$ is high) and not exploring more we can see higher number of episodes converging at reward grid 5 which is not gloabl optimal location.

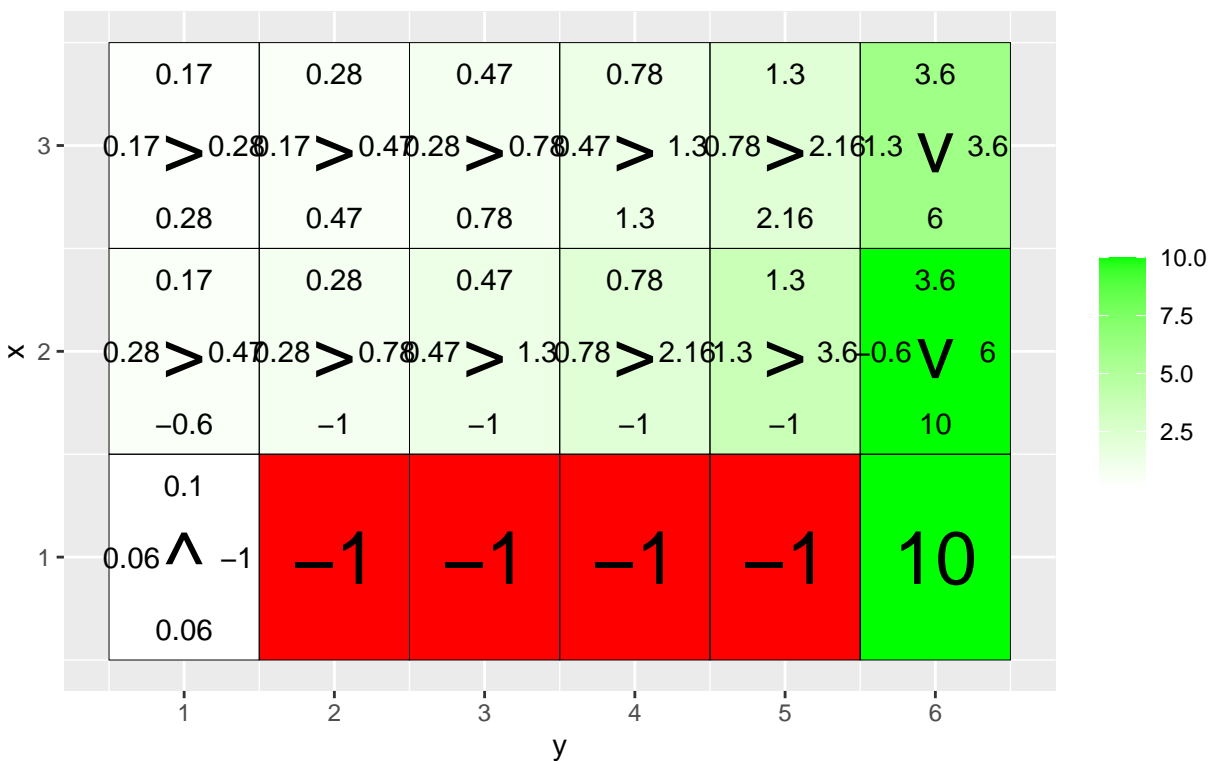
Environment C (the effect of beta)

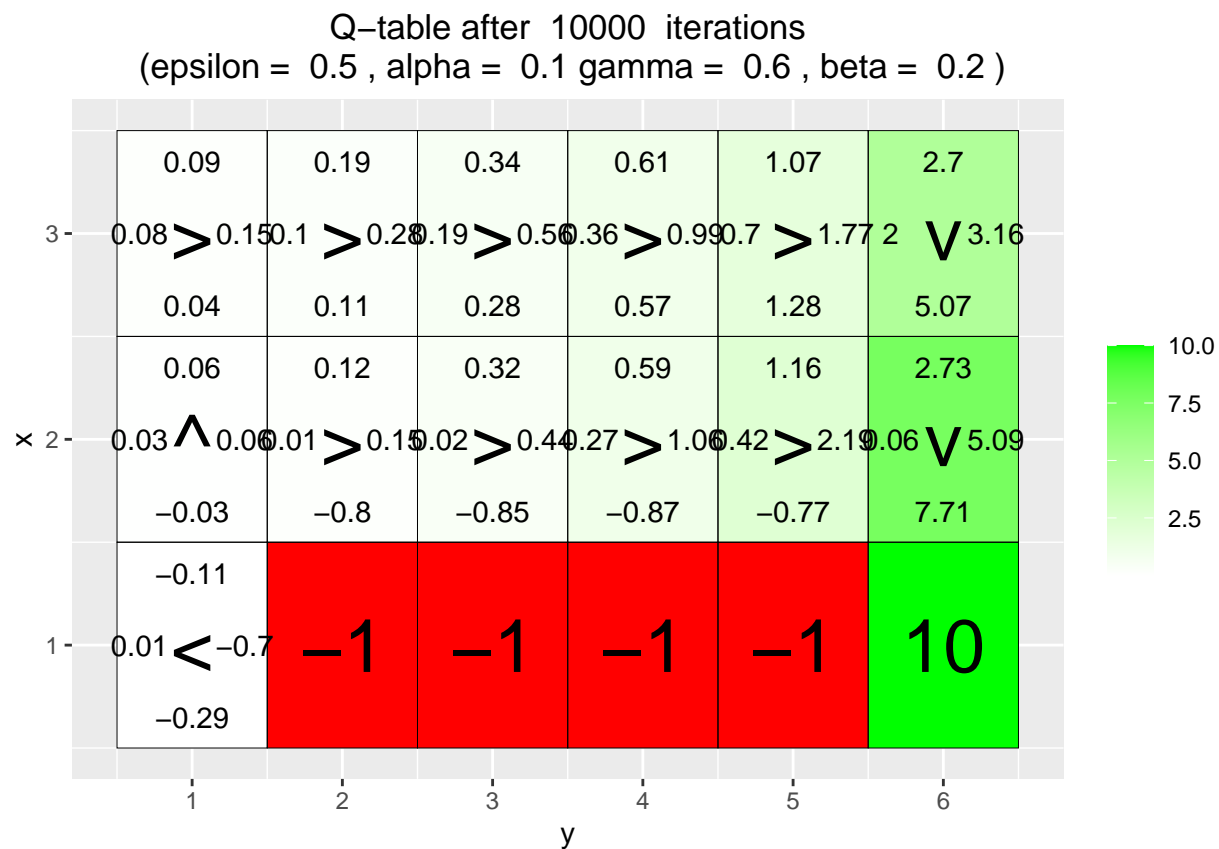
Your task is to investigate how the β parameter affects the learned policy by running 10000 episodes of Q-learning with $\beta = 0, 0.2, 0.4, 0.66$, $\epsilon = 0.5$, $\gamma = 0.6$ and $\alpha = 0.1$

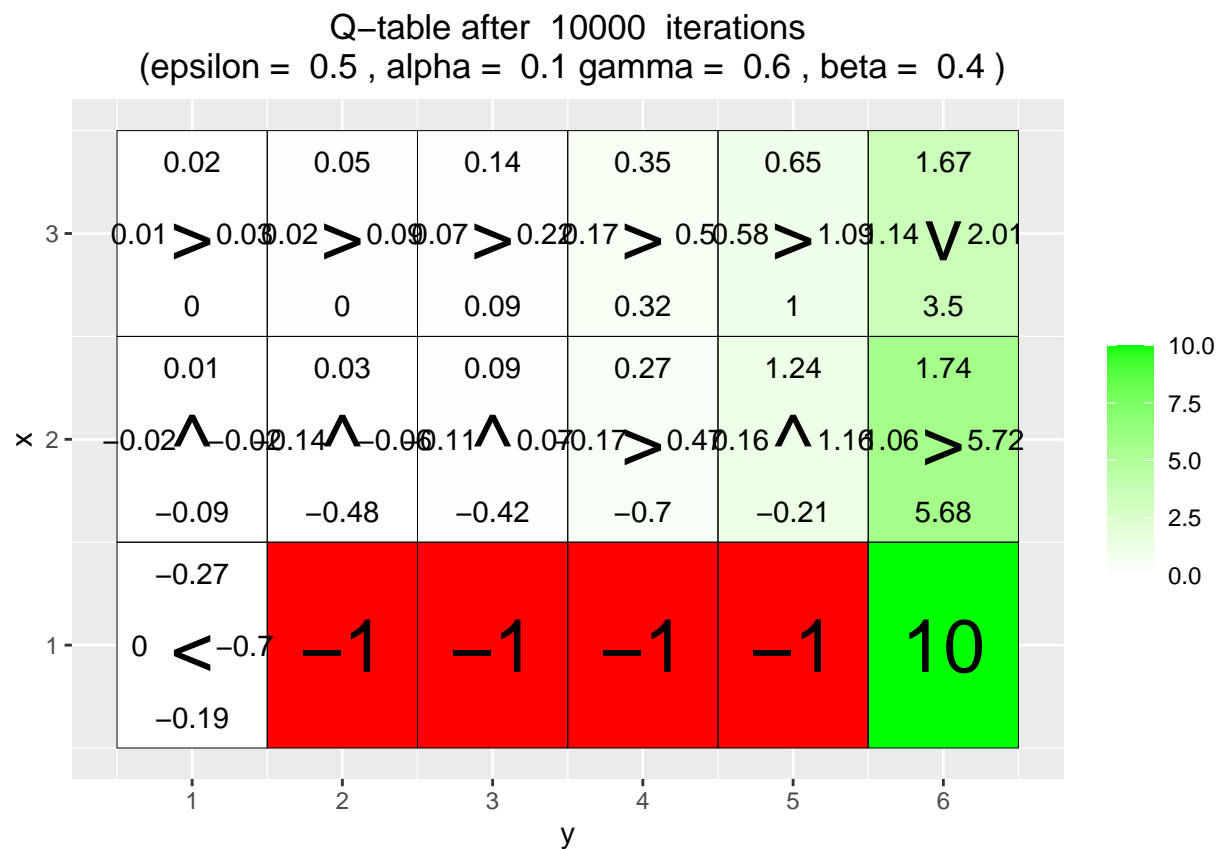
Since the transition model is defined by the agent moving in the direction chosen with probability $(1-\beta)$. The agent might also slip and end up moving in the direction to the left or right of its chosen action, each with probability $\beta/2$



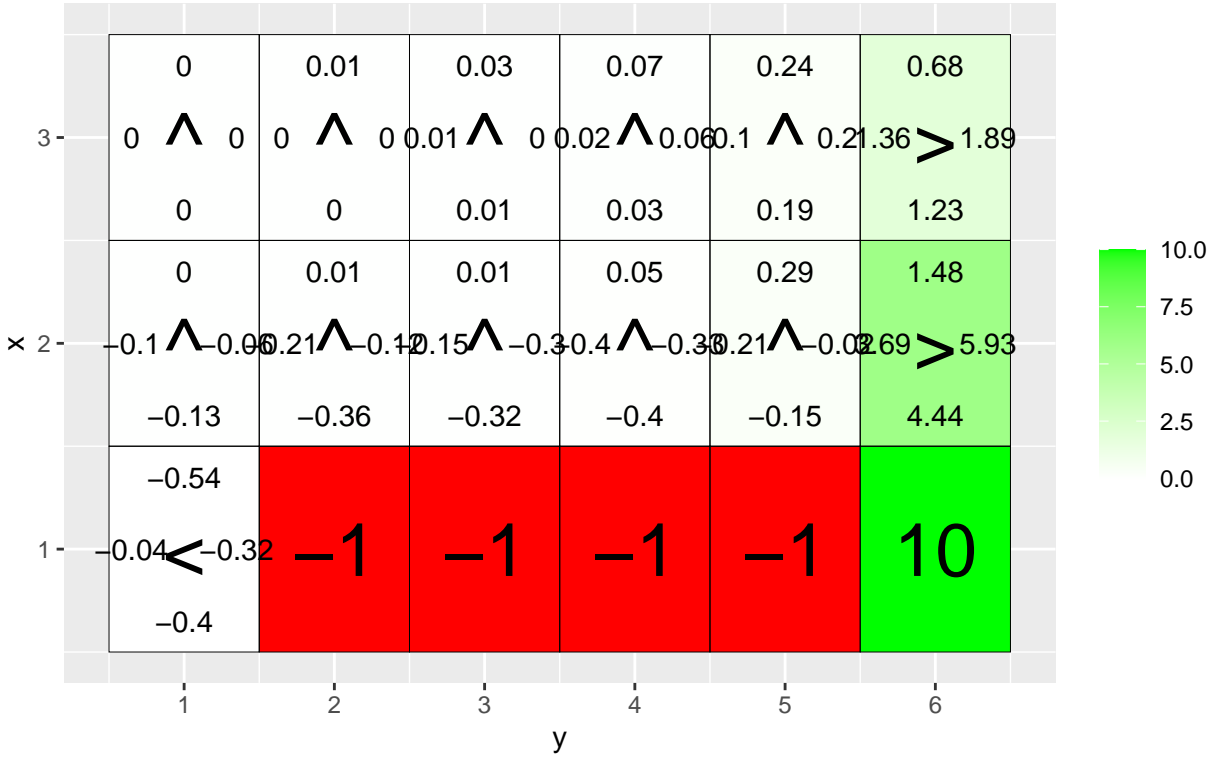
Q-table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0)







Q-table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.66)



As we are increasing β , we can see that model will not move in chosen direction and thus its movement is getting more and more random and with β at 0.6 (as movement in chosen direction have probability of $(1-\beta)$), it will mostly move towards right or left of chosen direction thus it might get harder for agent to reach optimal state in grid.

Thus β value can be used to cover uncertainty about action taken by agent in any given environment.

Question 2

Reinforcement Learning

Task Setup

We will work with a 4×4 grid. We want the agent to learn to navigate to a random goal position in the grid. The agent will start in a random position and it will be told the goal position. The agent receives a reward of 5 when it reaches the goal. Since the goal position can be any position, we need a way to tell the agent where the goal is. Since our agent does not have any memory mechanism, we provide the goal coordinates as part of the state at every time step, i.e. a state consists now of four coordinates: Two for the position of the agent, and two for the goal position. The actions of the agent can however only impact its own position, i.e. the actions do not modify the goal position. Note that the agent initially does not know that the last two coordinates of a state indicate the position with maximal reward, i.e. the goal position. It has to learn it. It also has to learn a policy to reach the goal position from the initial position. Moreover, the policy has to depend on the goal position, because it is chosen at random in each episode. Since we only have a single non-zero reward, we do not specify a reward map. Instead, the goal coordinates are passed to the functions that need to access the reward function.

Environment D (training with random goal positions)

Task: In this task, we will use eight goal positions for training and, then, validate the learned policy on the remaining eight possible goal positions. The training and validation goal positions are stored in the lists `train goals` and `val goals`.

Action probabilities after 0 episodes

x	4	0.12	0.51 ^	0.08	Goal		0.17	0.26 v	0.06	0.23	0.17 v	0.07
			0.29					0.5			0.53 v	
	3	0.15	0.46 ^	0.08	0.18	0.33 v	0.07	0.22	0.21 v	0.07	0.27	0.14 v
			0.31			0.42			0.49		0.5	0.09
2		0.18	0.4 ^	0.09	0.23	0.28 v	0.08	0.26	0.18 v	0.08	0.3	0.12 v
			0.33			0.4			0.47		0.49	0.1
1		0.22	0.36 ^	0.09	0.26	0.25 v	0.09	0.28	0.16 v	0.09	0.31	0.11 v
			0.32			0.4			0.47		0.48	0.1
					1	2		3			4	
		y										

4	0.11	0.53 ^ 0.31	0.06	0.14	0.36 v 0.43	0.07	0.19	0.21 v 0.53	0.07	Goal
3	0.14	0.46 ^ 0.33	0.07	0.19	0.3 v 0.43	0.08	0.23	0.17 v 0.52	0.08	0.27 v 0.55
2	0.18	0.39 ^ 0.33	0.09	0.24	0.24 v 0.43	0.09	0.27	0.13 v 0.51	0.09	0.31 v 0.52
1	0.23	0.32 v 0.34	0.11	0.28	0.19 v 0.43	0.1	0.31	0.1 v 0.48	0.1	0.34 v 0.48
		1			2			3		4

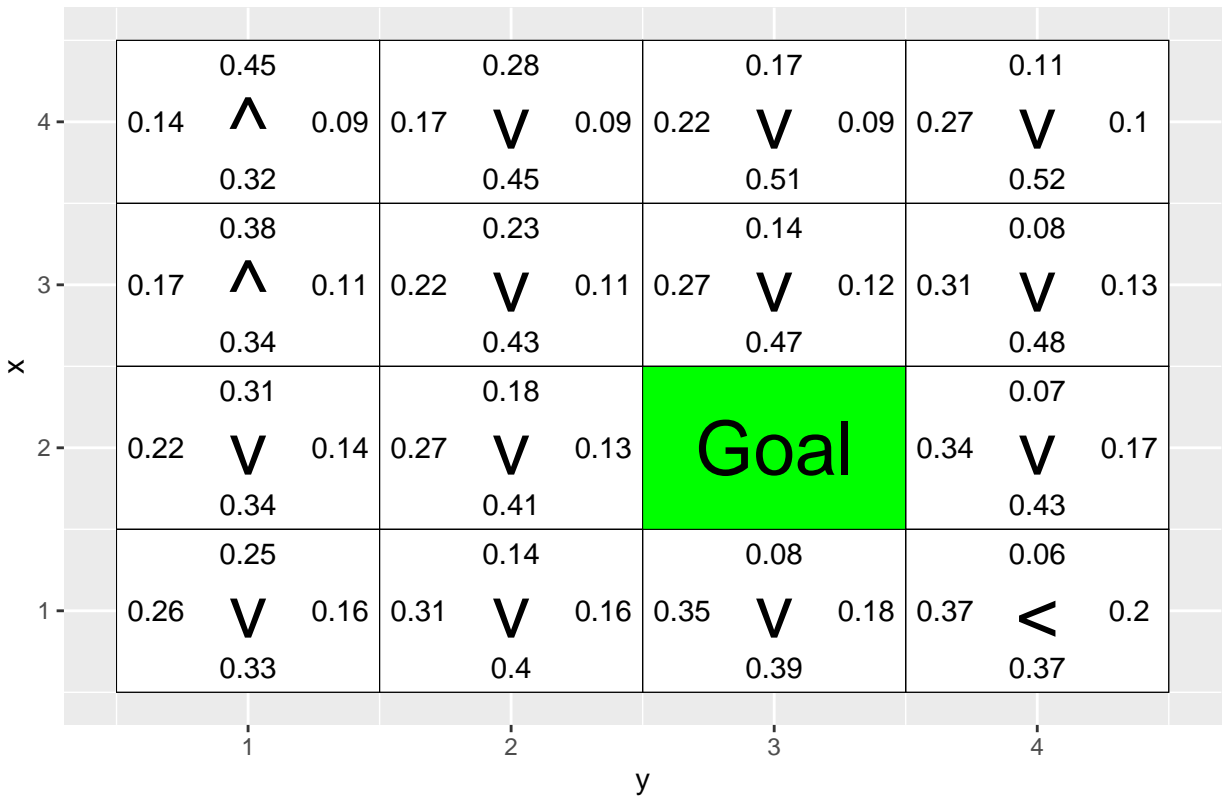
Action probabilities after 0 episodes

4	0.13	0.48 ^ 0.3	0.09	0.15	0.34 v 0.43	0.08	0.19	0.22 v 0.51	0.08	0.24	0.14 v 0.52	0.09	
3	0.15	0.42 ^ 0.33	0.1	Goal			0.25	0.18 v 0.48	0.1	0.29	0.12 v 0.47	0.12	
2	0.19	0.36 ^ 0.34	0.11	0.24	0.23 v 0.42	0.11	0.29	0.15 v 0.44	0.12	0.33	0.1 v 0.44	0.14	
1	0.24	0.3 v 0.34	0.12	0.28	0.19 v 0.41	0.12	0.31	0.12 v 0.43	0.14	0.35	0.09 v 0.42	0.14	
		1			2			3			4		
	x												y

Action probabilities after 0 episodes

4	0.12	0.49 ^	0.08	0.16	0.32 v	0.08	0.2	0.2 v	0.08	0.25	0.12 v	0.09	
		0.31			0.44			0.52			0.55		
3	0.15	0.42 ^	0.09	0.2	0.27 v	0.1	Goal				0.1	0.29	0.11 v
		0.34			0.44						0.5		
2	0.19	0.36 ^	0.11	0.25	0.21 v	0.11	0.29	0.12 v	0.12	0.33	0.08 v	0.14	
		0.34			0.43			0.47			0.46		
1	0.24	0.29 v	0.13	0.29	0.16 v	0.13	0.33	0.1 v	0.14	0.37	0.07 v	0.15	
		0.34			0.42			0.44			0.42		
		1			2			3			4		
		y											

Action probabilities after 0 episodes

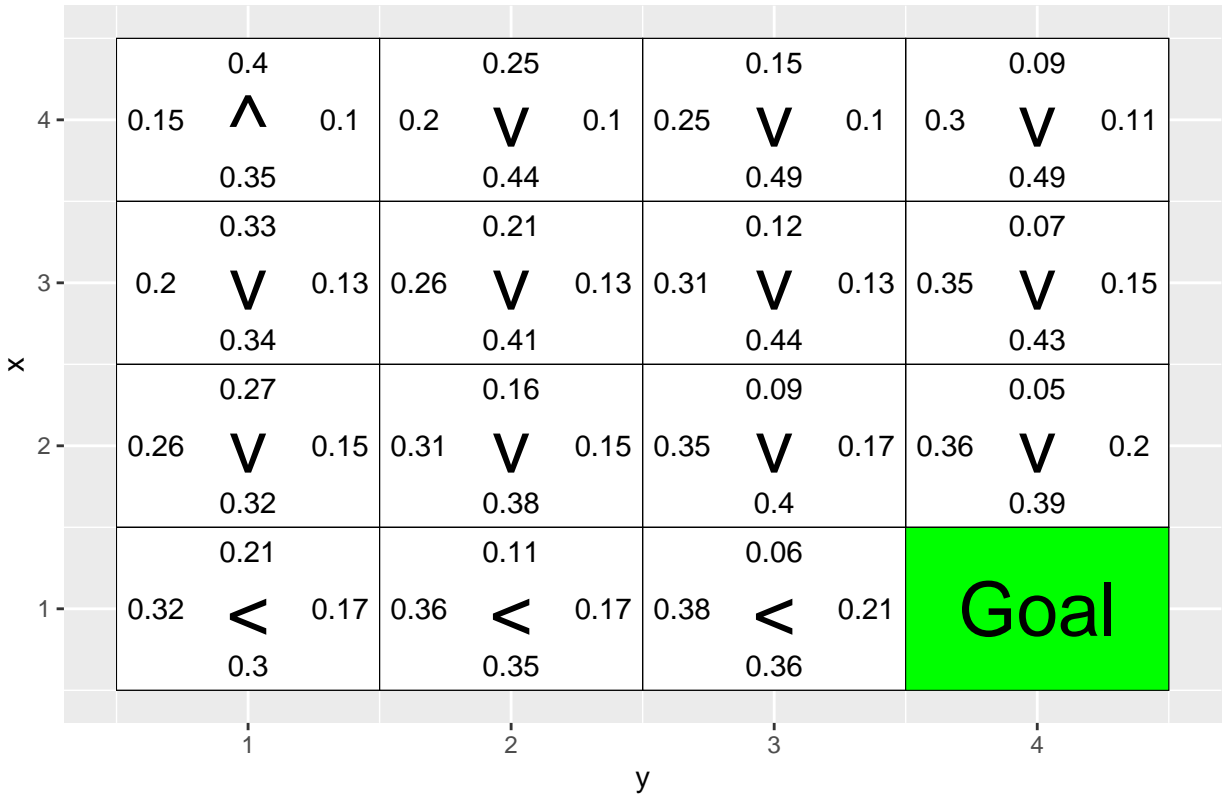


Action probabilities after 0 episodes

4	0.14	0.45 ^	0.09	0.19	0.28 v	0.09	0.23	0.17 v	0.09	0.28	0.1 v	0.1
		0.32			0.44			0.5		0.52		
3	0.18	0.37 ^	0.11	0.24	0.23 v	0.11	0.29	0.13 v	0.11	0.32	0.08 v	0.12
		0.33			0.42			0.47		0.48		
2	0.23	0.3 v	0.13	0.29	0.18 v	0.13	0.33	0.1 v	0.14	Goal		
		0.33			0.4			0.43				
1	0.28	0.24 v	0.16	0.33	0.13 v	0.15	0.36	0.07 v	0.17	0.38	0.04 v	0.19
		0.32			0.39			0.39		0.38		
		1			2			3		4		
		y										

4	0.15	0.4 ^ 0.32	0.13	0.18	0.29 v 0.41	0.12	0.2	0.19 v 0.51	0.1	0.22	0.14 v 0.53	0.11
3	0.18	0.35 ^ 0.33	0.14	0.21	0.23 v 0.43	0.13	0.24	0.16 v 0.46	0.14	0.27	0.12 v 0.45	0.15
2	0.21	0.28 v 0.35	0.16	0.26	0.18 v 0.38	0.18	0.3	0.13 v 0.38	0.19	0.35	0.09 v 0.38	0.17
1	Goal			0.3	0.16 v 0.32	0.22	0.34	0.11 < 0.33	0.22	0.4	0.07 < 0.34	0.19
	1			2			3			4		

Action probabilities after 0 episodes



```
## 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

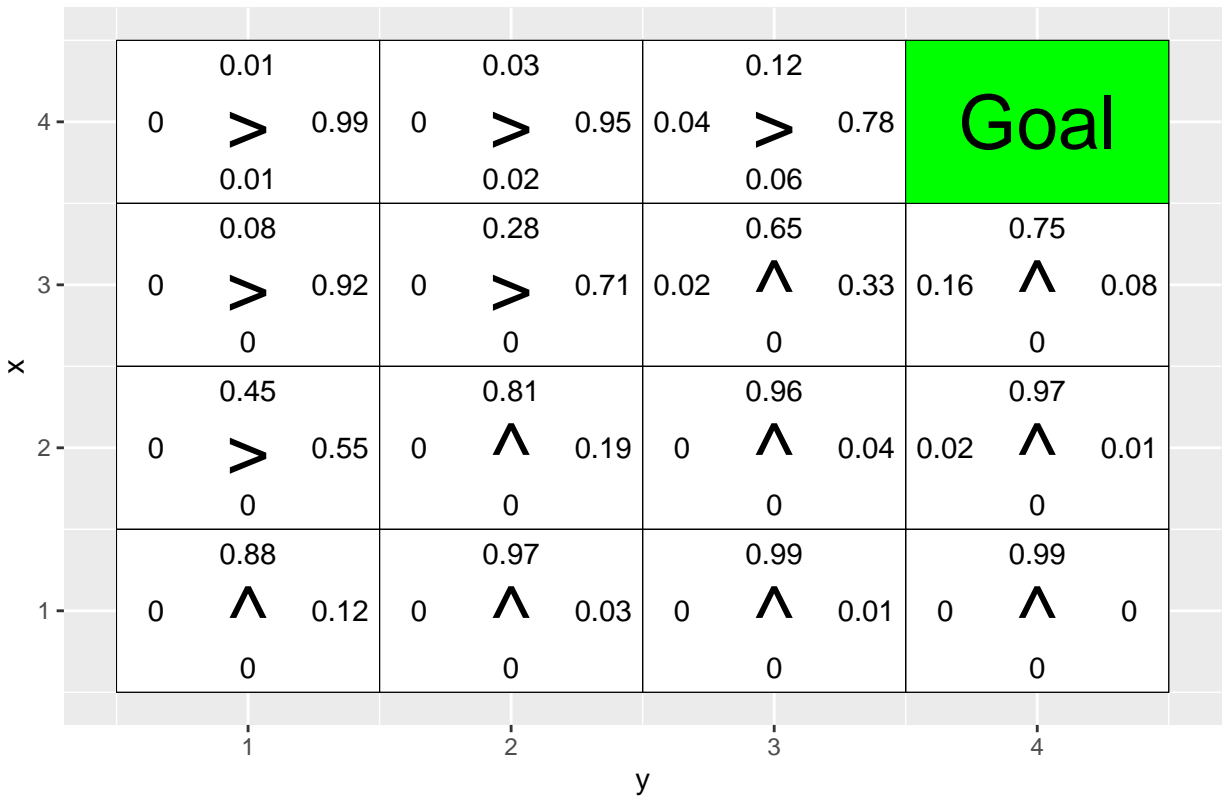
episode 4050
episode 4060
episode 4070
episode 4080
episode 4090
episode 4100
episode 4110
episode 4120
episode 4130
episode 4140
episode 4150
episode 4160
episode 4170
episode 4180
episode 4190
episode 4200
episode 4210
episode 4220
episode 4230
episode 4240
episode 4250
episode 4260
episode 4270
episode 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

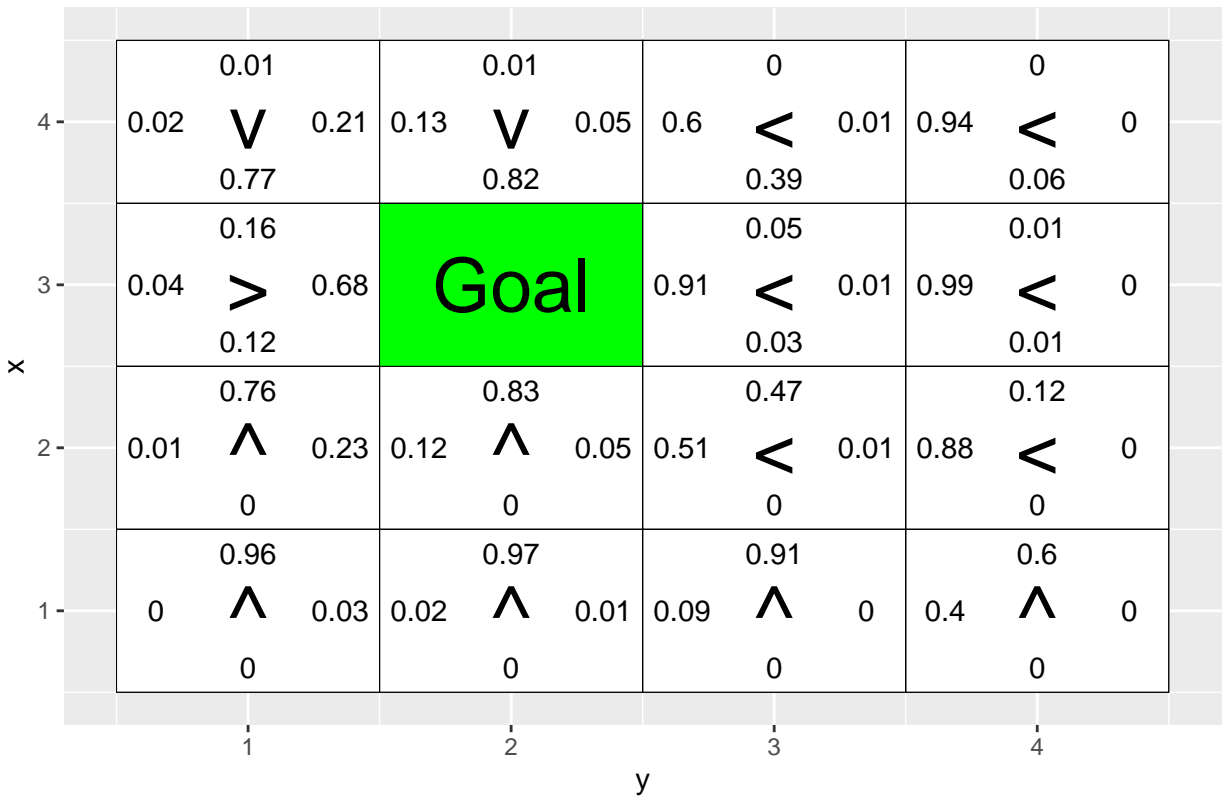
Action probabilities after 5000 episodes



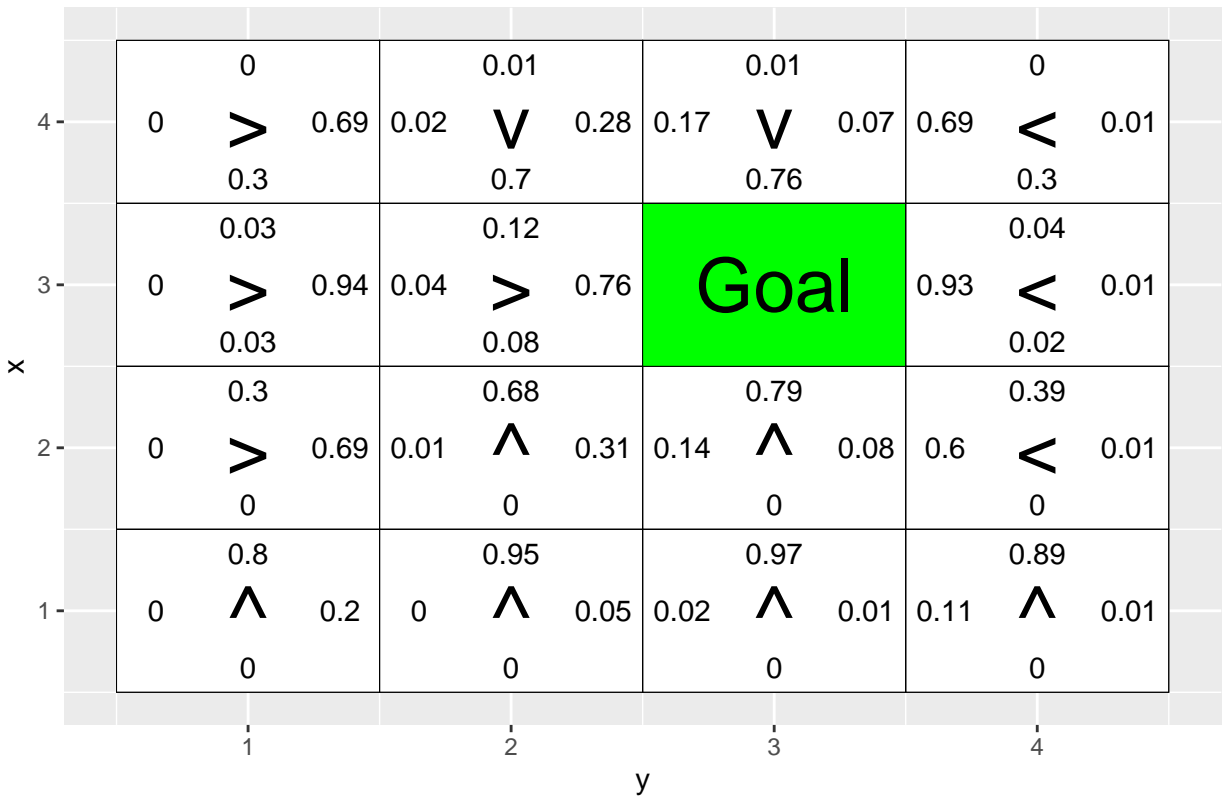
Action probabilities after 5000 episodes



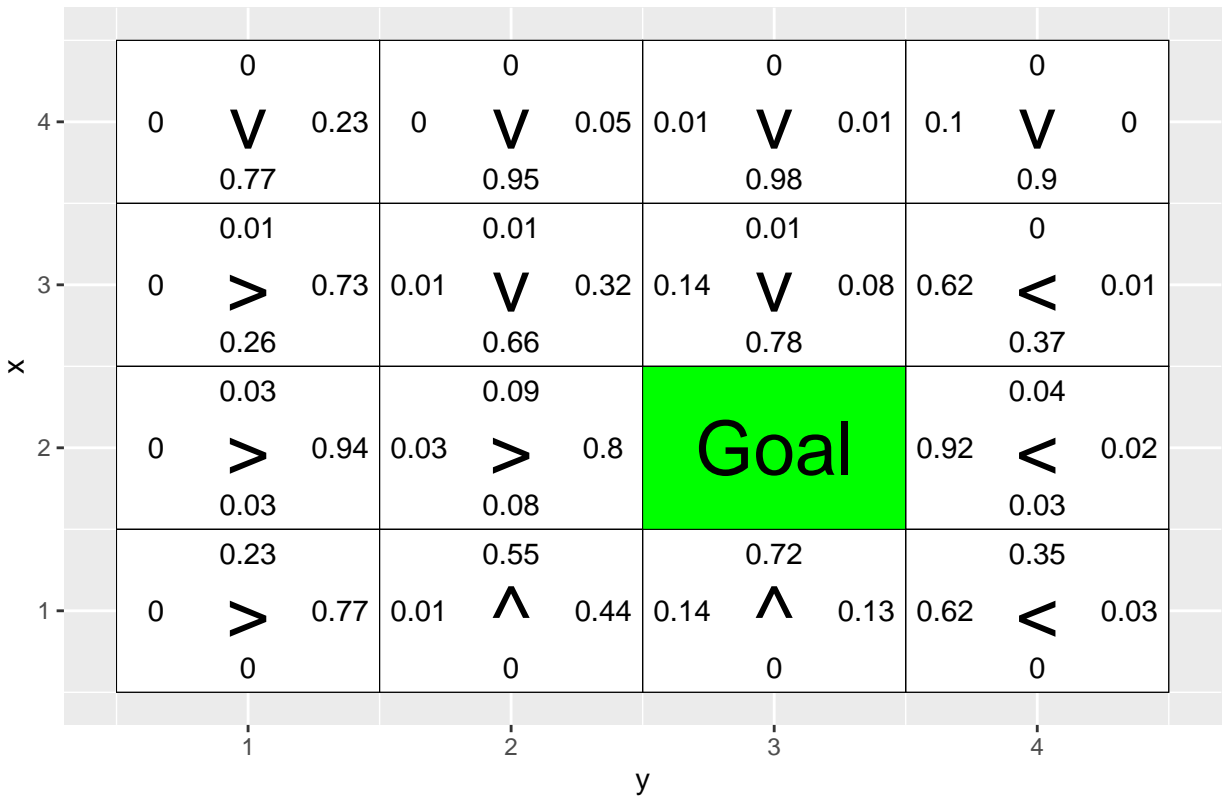
Action probabilities after 5000 episodes



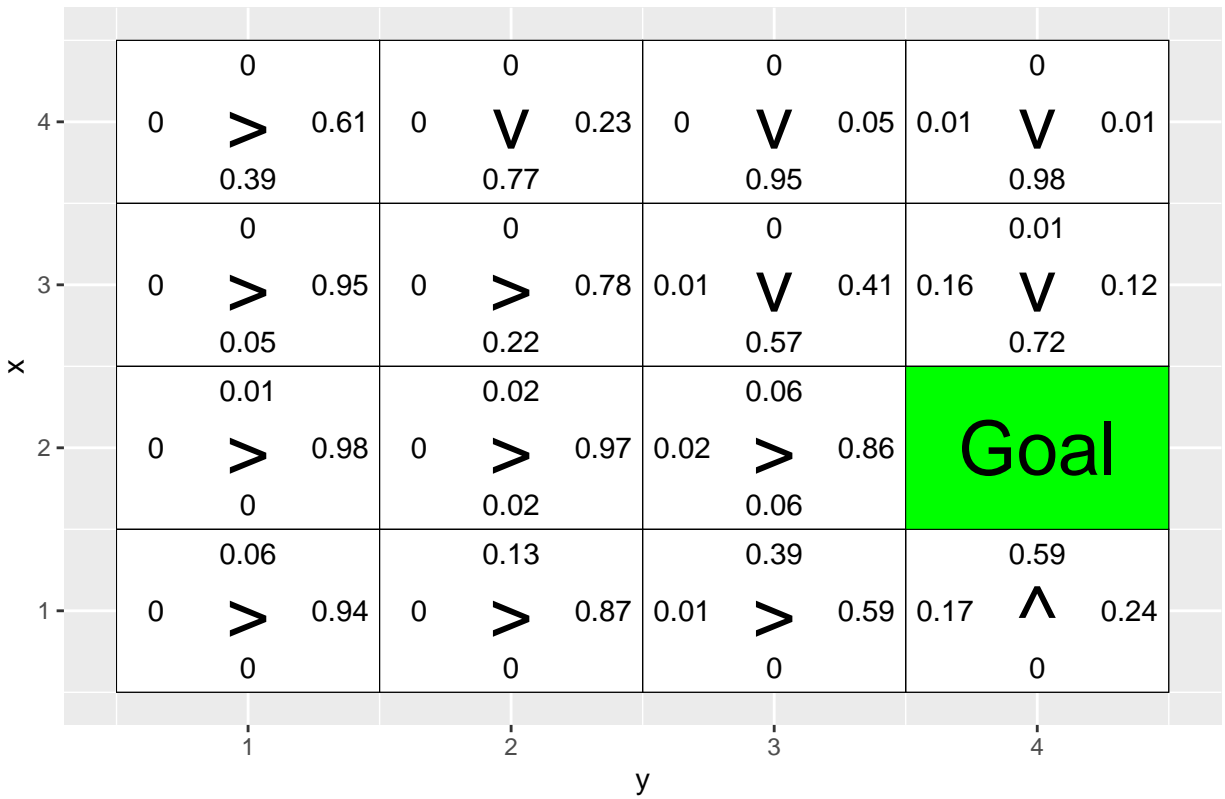
Action probabilities after 5000 episodes



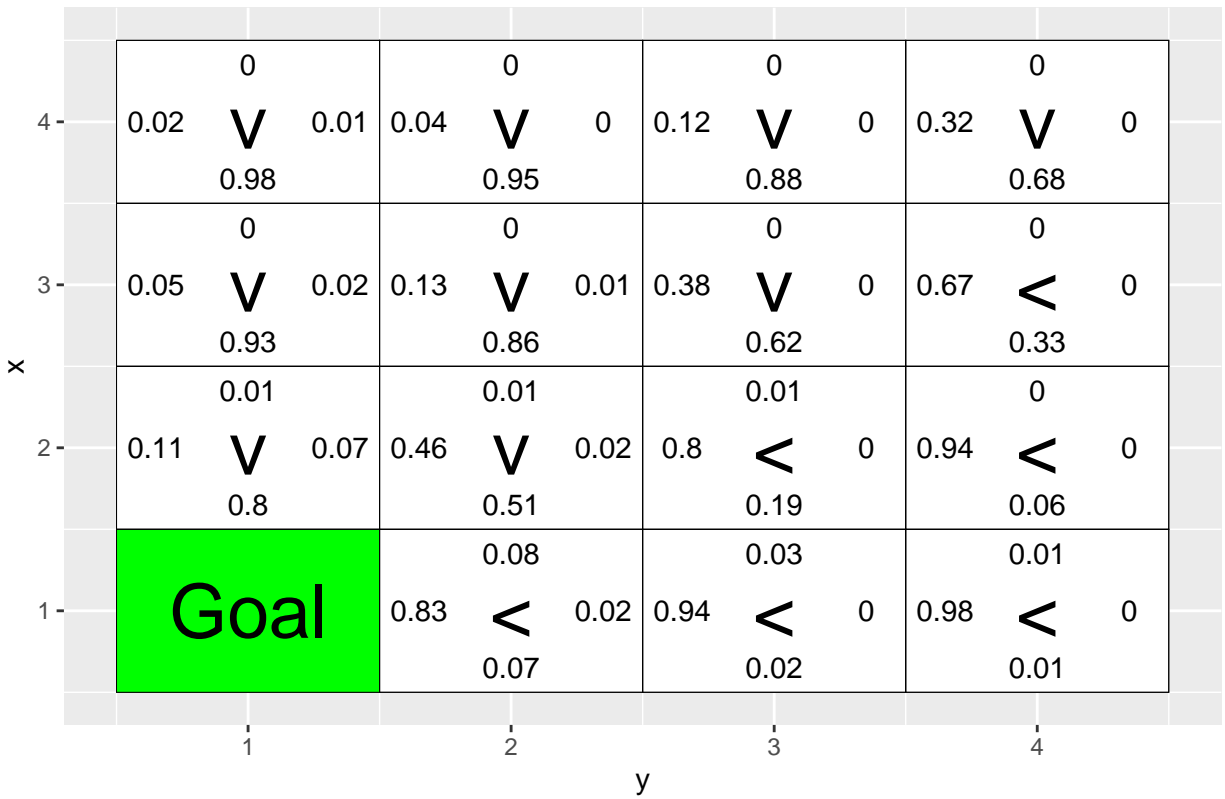
Action probabilities after 5000 episodes



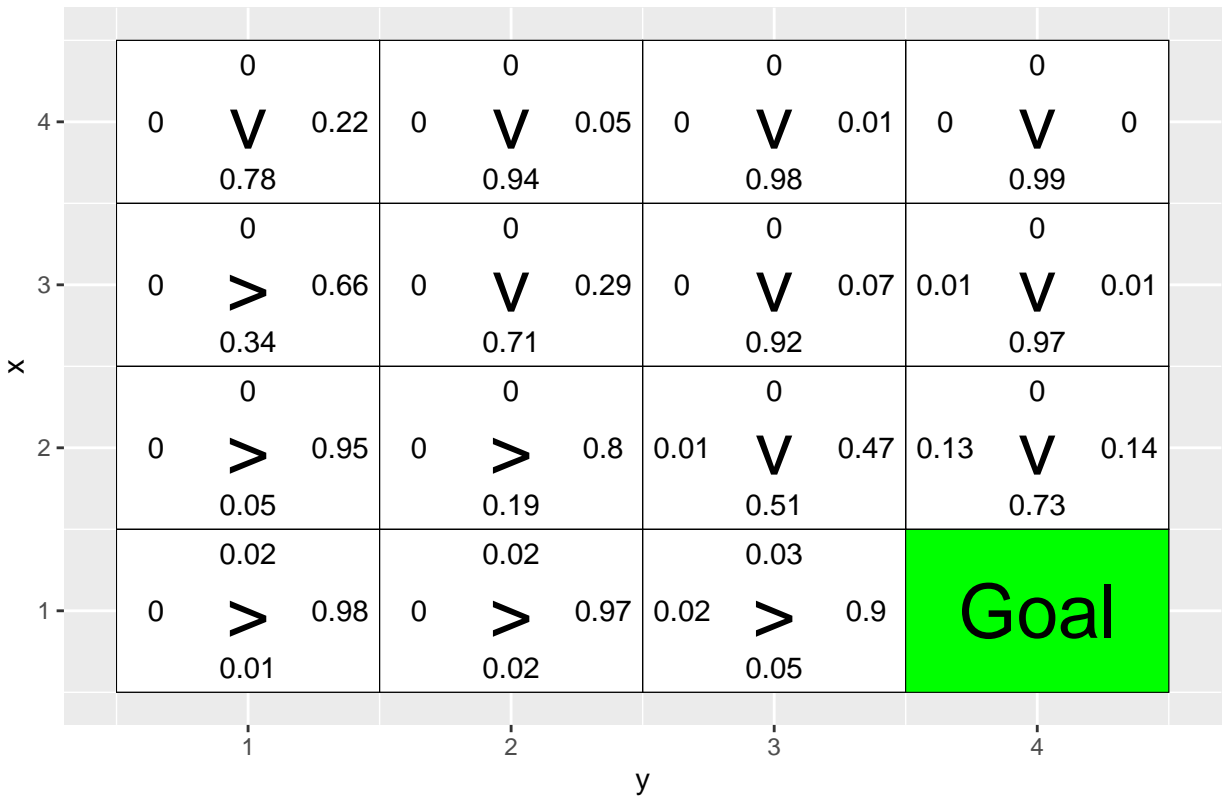
Action probabilities after 5000 episodes



Action probabilities after 5000 episodes



Action probabilities after 5000 episodes



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

It can be seen that the agent is able to learn the good policy as it had access to all grid positions (as training data) and thus it is able to generalize and learn good policy.

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

In Q-Learning we create a q-matrix which contain states and rewards for the state, now when we grid where the optimal location is kept on changing like for the above scenario, we cannot implement Q-Learning since the path to the optimal location will keep on changing and in order to do so, we have to train the model with different high reward positions which won't give us single generalized model. Thus applying Q-Learning for a similar problem is not the correct solution.

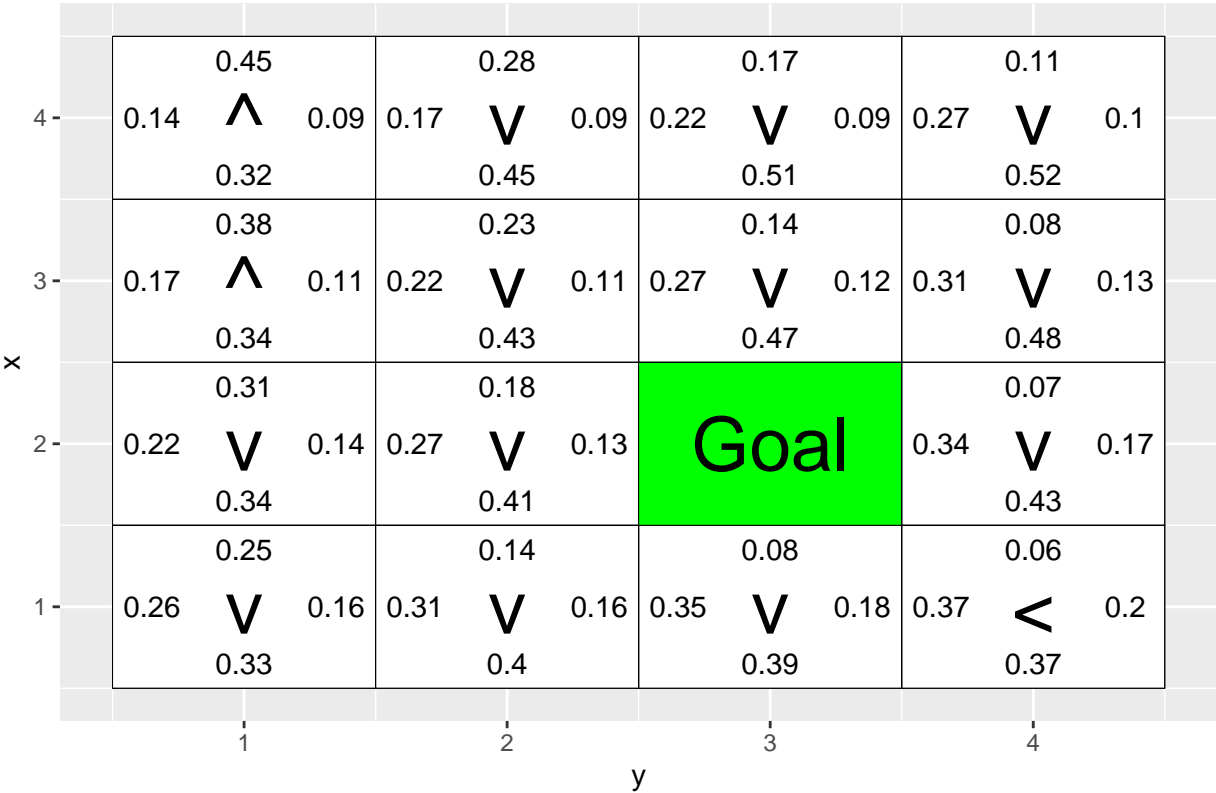
Environment E (training with top row goal positions)

Task : To repeat the previous experiments but this time the goals for training are all from the top row of the grid. The validation goals are three positions from the rows below.

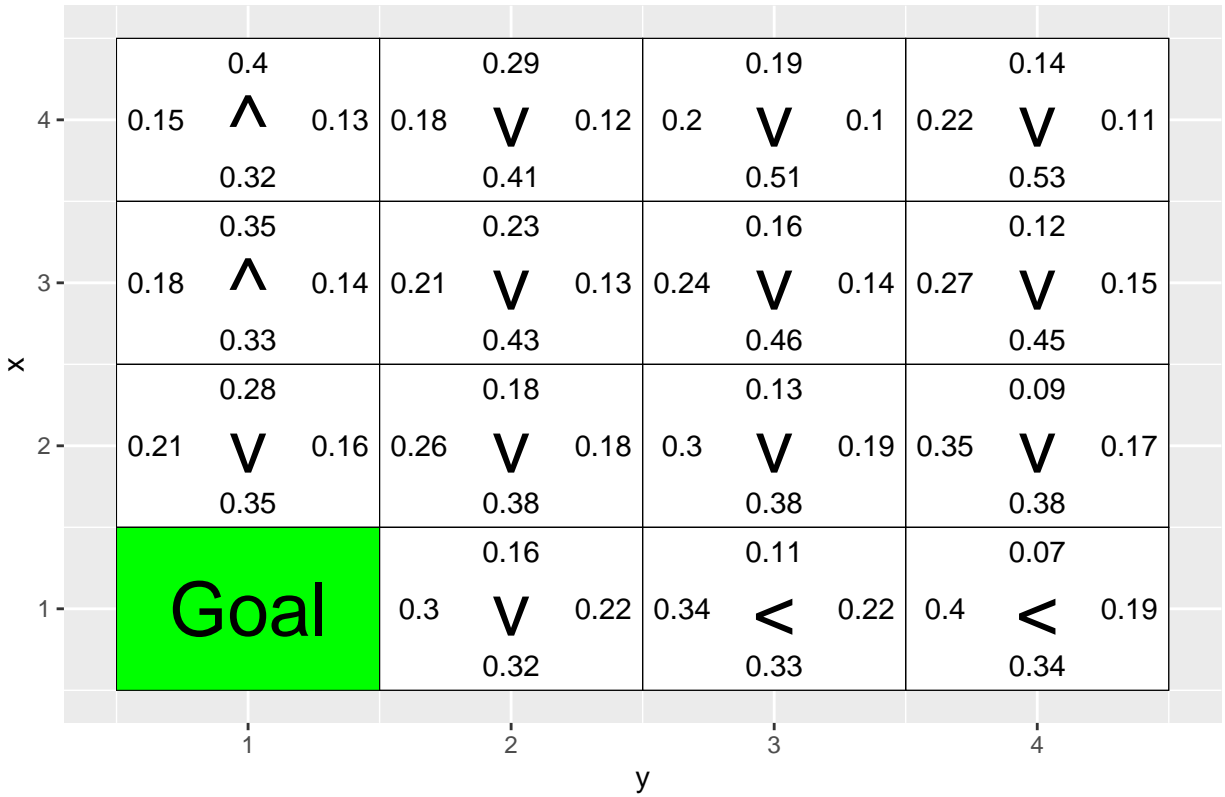
Action probabilities after 0 episodes

4	0.12	0.49 ^	0.07	0.17	0.31 v	0.08	0.21	0.19 v	0.08	0.25	0.11 v	0.08
		0.31			0.44			0.52			0.56	
3	0.16	0.42 ^	0.09	0.21	0.26 v	0.1	0.26	0.15 v	0.09	Goal		
		0.33			0.43			0.5				
2	0.2	0.35 ^	0.11	0.26	0.21 v	0.12	0.3	0.11 v	0.11	0.33	0.07 v	0.13
		0.33			0.42			0.48			0.47	
1	0.25	0.28 v	0.14	0.3	0.16 v	0.13	0.33	0.09 v	0.13	0.37	0.05 v	0.15
		0.33			0.41			0.45			0.43	
		1			2			3			4	
		y										

Action probabilities after 0 episodes



Action probabilities after 0 episodes



```

## 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

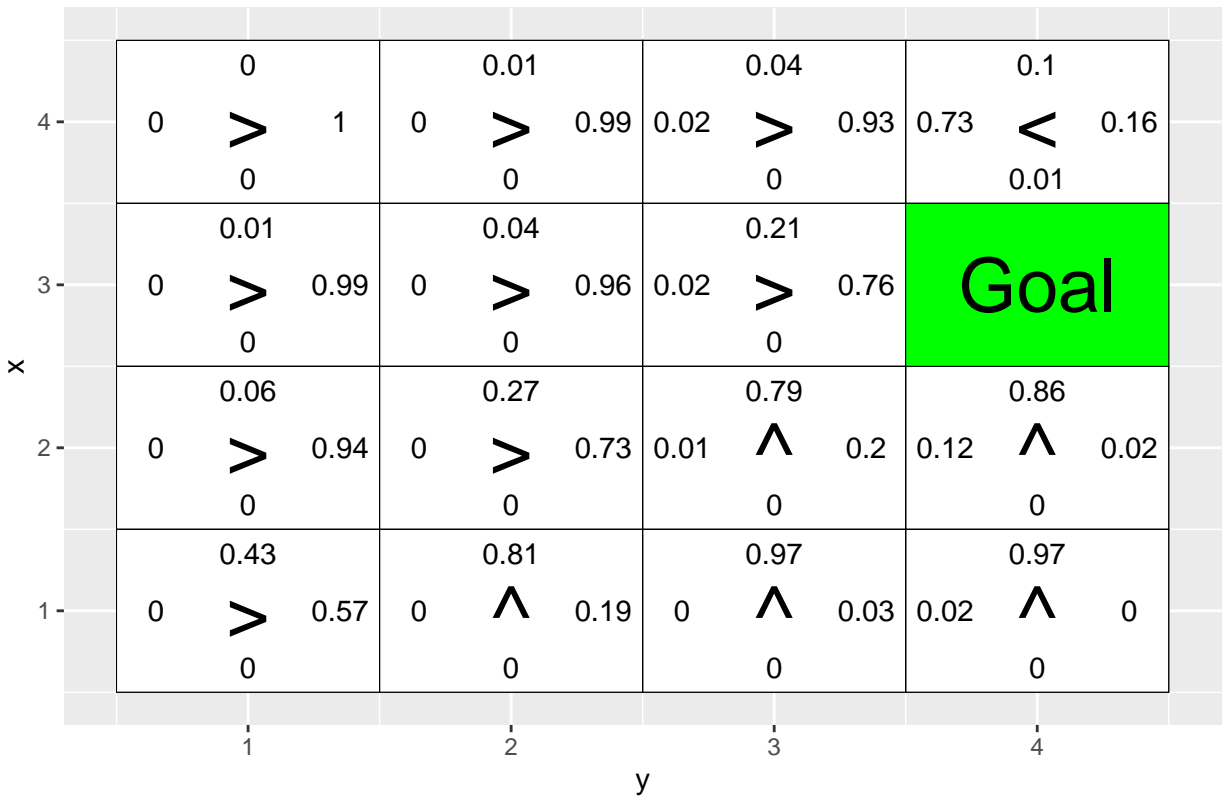
episode 2970
episode 2980
episode 2990
episode 3000
episode 3010
episode 3020
episode 3030
episode 3040
episode 3050
episode 3060
episode 3070
episode 3080
episode 3090
episode 3100
episode 3110
episode 3120
episode 3130
episode 3140
episode 3150
episode 3160
episode 3170
episode 3180
episode 3190
episode 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

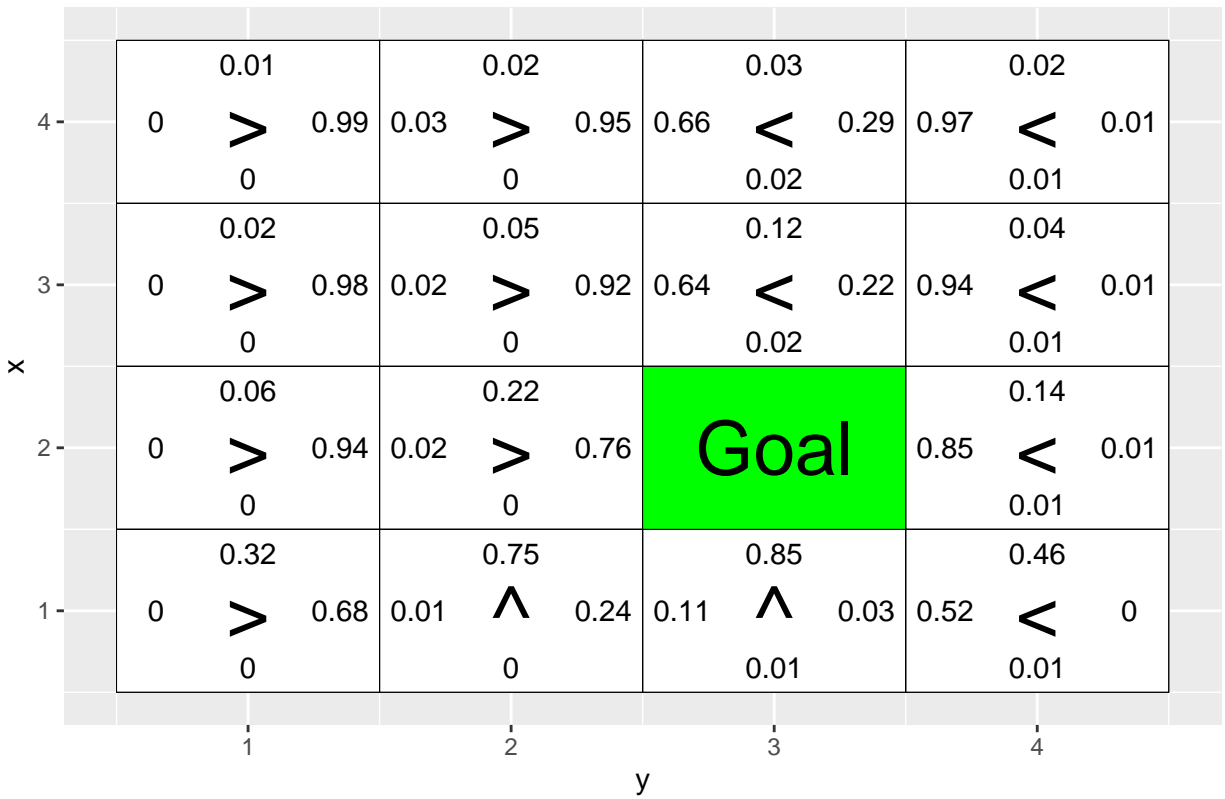
episode 4050
episode 4060
episode 4070
episode 4080
episode 4090
episode 4100
episode 4110
episode 4120
episode 4130
episode 4140
episode 4150
episode 4160
episode 4170
episode 4180
episode 4190
episode 4200
episode 4210
episode 4220
episode 4230
episode 4240
episode 4250
episode 4260
episode 4270
episode 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

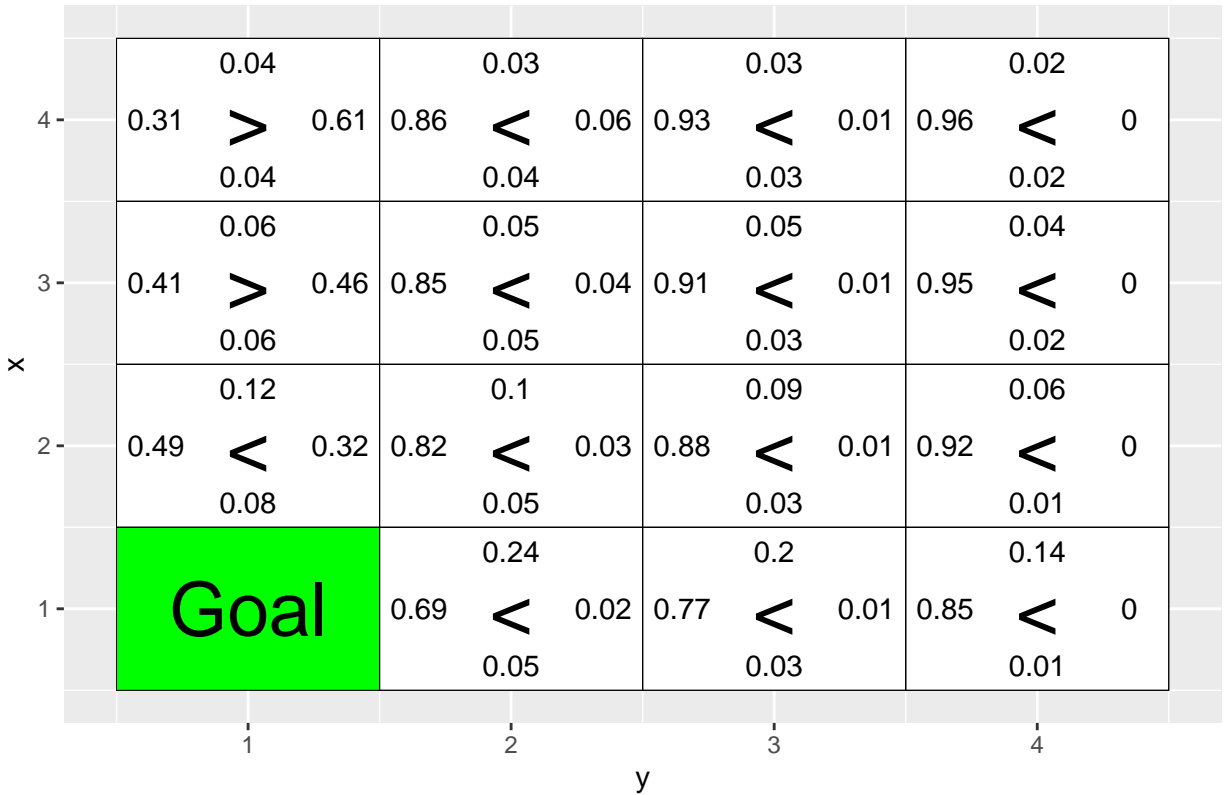
Action probabilities after 5000 episodes



Action probabilities after 5000 episodes



Action probabilities after 5000 episodes



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

Since the agent only had access to 1st row of the grid as data, the neural network was not able to learn all paths for the whole grid, so we are not able to create a generalized model because of which agent is not able to learn optimal policy.

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

In D agent was able to generalize based on given training data as it had access to the whole grid while it lacks to do it for E as it only had access to 1st row of the grid thus better generalization was obtained in D while E we were not able to do so because of which we have obtained different results. This can be seen by the path selected in case of validation for different goal positions.

References

- A Concise Introduction to Reinforcement Learning

*Epsilon-Greedy Algorithm in Reinforcement Learning

Appendix

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

library(ggplot2)

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

```

#arrows <- c("↑", "→", "↓", "←")
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) +
    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}.
Rewards = q_table[x,y,]
maxRewardIndx = which(Rewards == max(Rewards))
#randomly selecting max value and setting it as action

if (length(maxRewardIndx) == 1){
  InitSelected = maxRewardIndx
}else{
  InitSelected = sample(maxRewardIndx , size = 1)
}

return(InitSelected)
# Your code here.
}#GreedyPolicy

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.
  Rewards = q_table[x,y,]
  maxRewardIndx = which(Rewards == max(Rewards))

  #randomly selecting max value and setting it as action
  if (length(maxRewardIndx) == 1){
    InitSelected = maxRewardIndx
  }else{
    InitSelected = sample(maxRewardIndx , size = 1)
  }
  #unif
  p = runif(1)
  if(p < epsilon)
  {
    return(sample(c(1,2,3,4) , size = 1)) #e
  }else{
    return(InitSelected) #1-e
  }
}#EpsilonGreedyPolicy

```



```

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)
}

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 superassignment operator <<-.

  # Your code here.
  episode_correction = 0

  repeat{
    # Follow policy, execute action, get reward.
    x = start_state[1]
    y = start_state[2]
    A = EpsilonGreedyPolicy(x,y,epsilon)

```

```

a = GreedyPolicy(x,y)
#calculate new state
TransState = transition_model(x,y,A,beta)

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

NewCorrection = alpha * (reward + gamma*(q_table[TransState[1],TransState[2],a]) - q_table[x,y,A])
episode_correction = episode_correction + NewCorrection

# Q-table update.
q_table[x,y,A] <-< q_table[x,y,A] + NewCorrection

start_state = TransState

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

}

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)
}

#test run for additional 1000 run

# for(i in 1:100000){
#   foo <- q_learning(start_state = c(3,1))
#
#   if(any(i==c(10,100,1000,10000 , 100000)))
#     vis_environment(i)
# }

# Environment B (the effect of epsilon and gamma)

cat("Epsilon set as 0.5")

```

```

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")
}

cat("Epsilon set as 0.1")

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")
}

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)
}

# install.packages("keras")
#install.packages("tensorflow")
library(tensorflow)
library(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("↑", "→", "↓", "←")
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)
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)
}

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')

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)

```