732A96 Advanced Machine Learning

LAB 3: Reinforcement learning

Johannes Hedström

STIMA
Department of Computer and Information Science
Linköpings universitet

2024-10-15

Contents

```r
library(knitr)
library(ggplot2)
```

1 1

Q-Learning. The file RL Lab1.R in the course website contains a template of the Q- learning algorithm. You are asked to complete the implementation. We will work with a grid- world environment consisting of H x 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: $S = (x, y) \mid x \in 1, ..., H, y \in 1, ..., W$.
- Action space: $A = up, down, left, right$.

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.

```r
# given code
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 = 2.5,nudge_y = .35,na.rm = TRUE) +
          geom_text(aes(label = val2),size = 2.5,nudge_x = .35,na.rm = TRUE) +
          geom_text(aes(label = val3),size = 2.5,nudge_y = -.35,na.rm = TRUE) +
          geom_text(aes(label = val4),size = 2.5,nudge_x = -.35,na.rm = TRUE) +
          geom_text(aes(label = val5),size = 7.5) +
          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)))

}


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)
```

```
}
```

```
# greedy function
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.
  max_actions <- which(q_table[x,y,]==max(q_table[x,y,])) # picking out all max values

  max_actions[sample(seq_along(max_actions),1)]# Random if two actions are equally good in the table
  #sample(max_actions,1) Why does this choose 1 more often?!
  }

# Epsilon greedy function
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.
  if (runif(1)< 1-epsilon){
    GreedyPolicy(x,y) # best known action
  }
  else{
    sample(c(1,2,3,4),1) # random action
  }

}
```

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

  # Your code here.

  x <- start_state[1]
  y <- start_state[2]
  episode_correction <- c()
  repeat{
    # Follow policy, execute action, get reward.

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



    # correction
    corr <-(reward + gamma *max(q_table[new_pos[1],new_pos[2],])-q_table[x,y,action] )
    # Q-table update.
    q_table[x,y,action] <<- q_table[x,y,action] + alpha*corr
    # sum of corrections
    episode_correction <- c(episode_correction , corr)
    x <- new_pos[1]
    y <- new_pos[2]

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

}
```

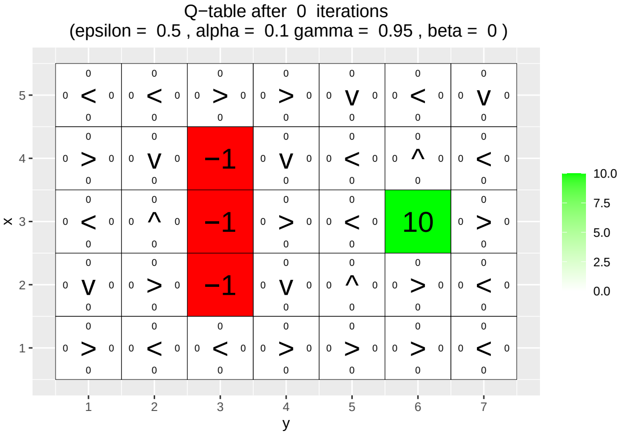
2 2

For our first environment, we will use H = 5 and W = 7. This environ- ment 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 in the file RL Lab1.R is used to visualize the environment and learned action values and policy. You will not have to modify this function, but read the comments in it to familiarize with how it can be used. When implementing Q-learning, the estimated values of Q(S, A) are commonly stored in a data-structured called Q-table. This is nothing but a tensor with one entry for each state- action pair. Since we have a H × W environment with four actions, we can use a 3D-tensor of dimensions H x W x 4 to represent our Q-table. Initialize all Q-values to 0. Run the function vis environment before proceeding further. Note that each non-terminal tile has four values. These represent the action values associated to the tile (state). Note also that each non-terminal tile has an arrow. This indicates the greedy policy for the tile (ties are broken at random).

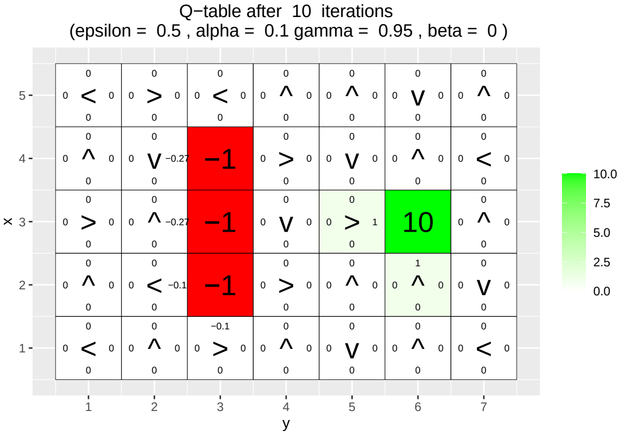
our are requested to carry out the following tasks.

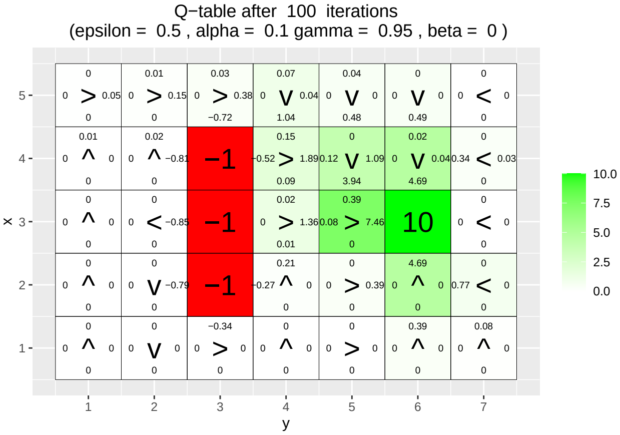
- Implement the greedy and ϵ -greedy policies in the functions GreedyPolicy and EpsilonGreedyPolicy of the file RL Lab1.R. The functions should break ties at random, i.e. they should sample uniformly from the set of actions with maximal Q-value.
- Implement the Q-learning algorithm in the function q learning of the file RL Lab1.R. The function should run one episode of the agent acting in the environment and update the Q-table accordingly. The function should return the episode reward and the sum of the temporal-difference correction terms R + maxa Q(S , a) − Q(S, A) for all steps in the episode. Note that a transition model taking as input is already implemented for you in the function transition model.
- Run 10000 episodes of Q-learning with $\epsilon = 0.5$, $\beta = 0$, $\alpha = 0.1$ and $\gamma = 0.95$. To do so, simply run the code provided in the file RL Lab1.R. The code visualizes the Q-table and a greedy policy derived from it after episodes 10, 100, 1000 and 10000. Answer the following questions:

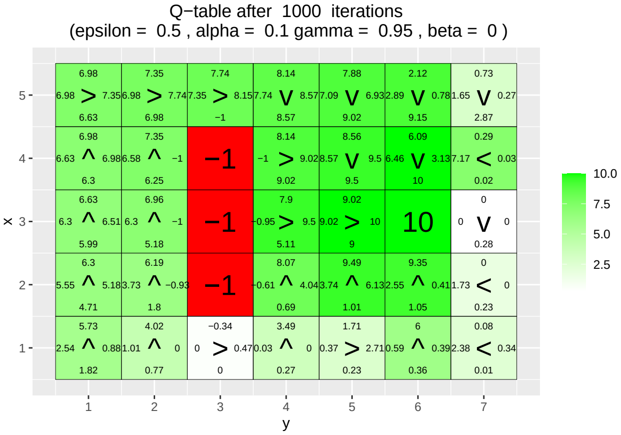
```r
# Environment A (learning)
set.seed(12345)
H <- 5
W <- 7

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()
```

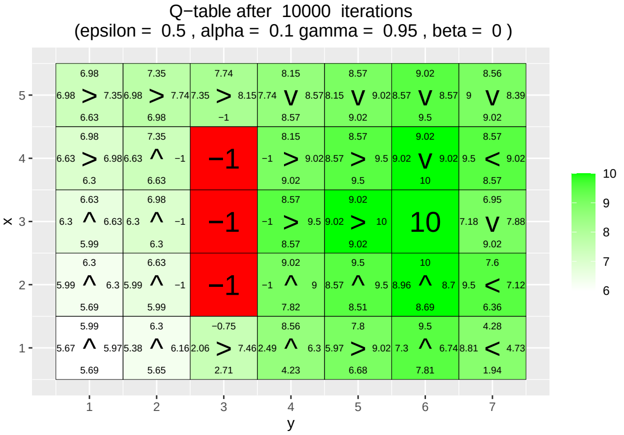

Q−table after 0 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)

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

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


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


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


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


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)

```
- What has the agent learned after the first 10 episodes ?
```

After the first 10 episodes the agent haven't learned that much, but it has updated the three boxes left of (-1) to not move to the right.

```
- Is the final greedy policy (after 10000 episodes) optimal for all states, i.e. not only for the initia
```

The final greedy policy isn't optimal for all states, its choosing a longer path to go over the obstacle then below, even though that the path below is shorter.

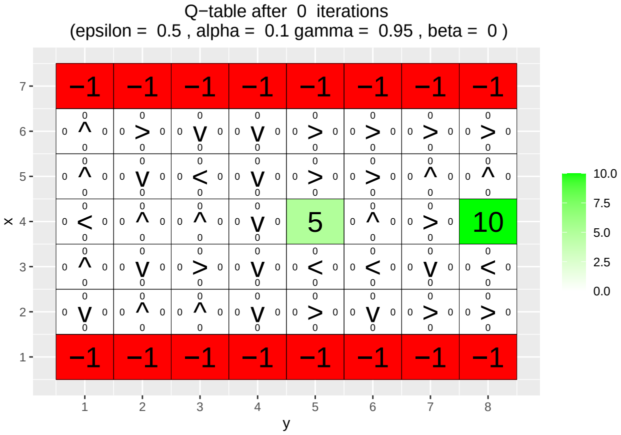
```
- Do the learned values in the Q-table reflect the fact that there are multiple paths (above and below t
```

The values are higher to go above the negative rewards, what could be done is to explore more when choosing the next action, seems like the left bottom corner isn't explored enough. What could be done is to increase epsilon and to increase the number of episodes.

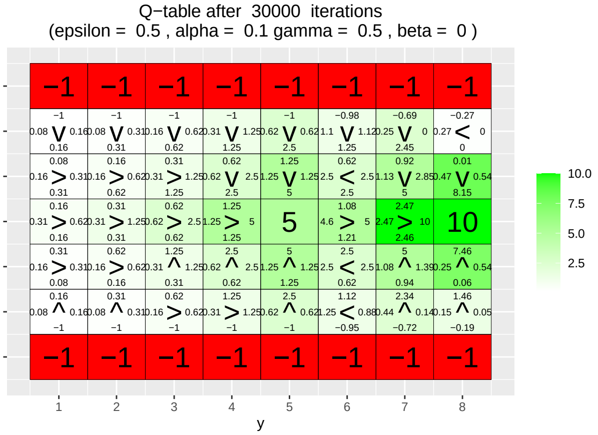
3 3

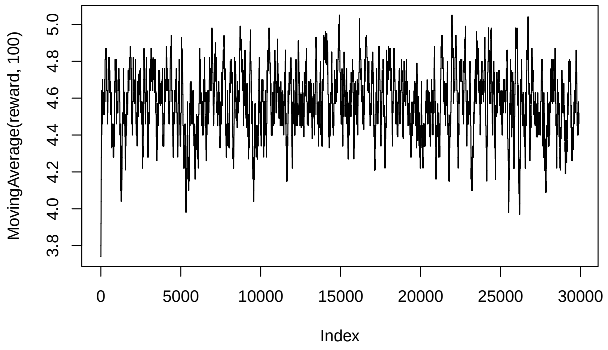
Environment B. This is a 7x8 environment where the top and bottom rows have negative rewards. In this environment, the agent starts each episode in the state (4, 1). There are two positive rewards, of 5 and 10. The reward of 5 is easily reachable, but the agent has to navigate around the first reward in order to find the reward worth 10. Your task is to investigate how the ϵ and 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 $\alpha = 0.1$. To do so, simply run the code provided in the file RL Lab1.R and explain your observations.

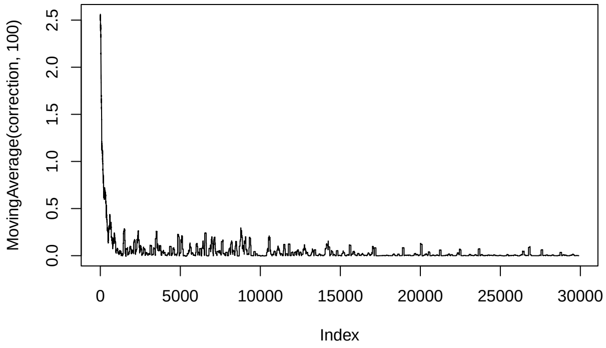
```r
# Environment B (the effect of epsilon and gamma)

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()
```

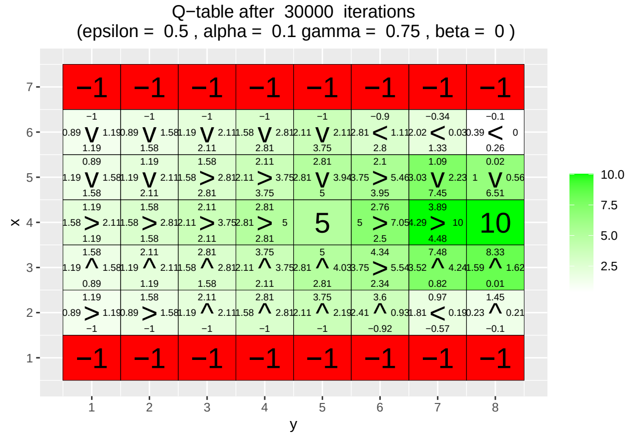


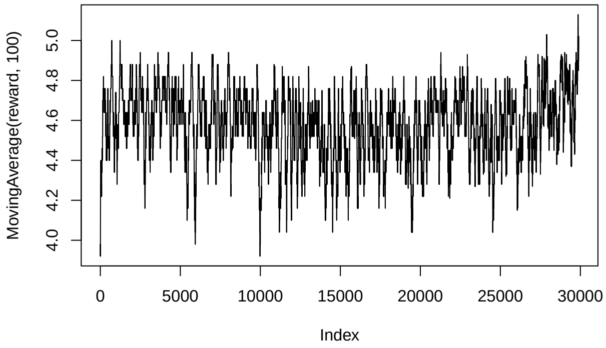
```
MovingAverage <- function(x, n){

  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")
}
```

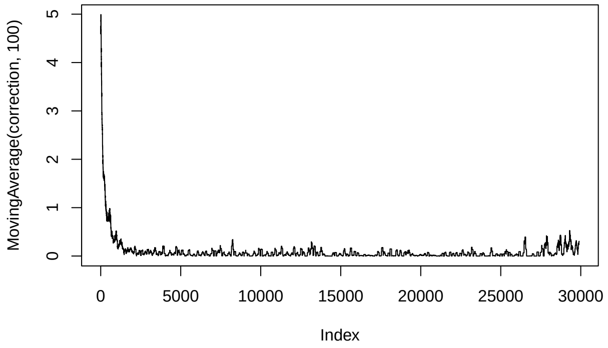

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
1
2
3
4
5
6
7
8
5
10
−1
10.0
7.5
5.0
2.5

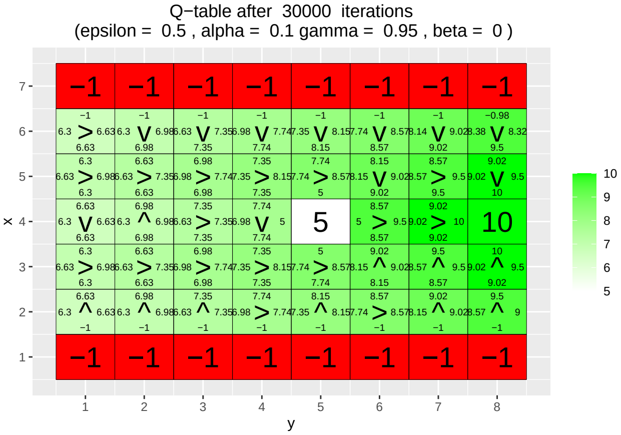

MovingAverage(reward, 100)
Index
3.8
4.0
4.2
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000

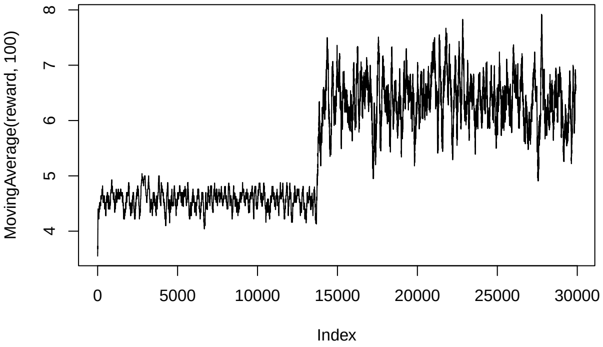

MovingAverage(correction, 100)
Index
0.0
0.5
1.0
1.5
2.0
2.5
0
5000
10000
15000
20000
25000
30000

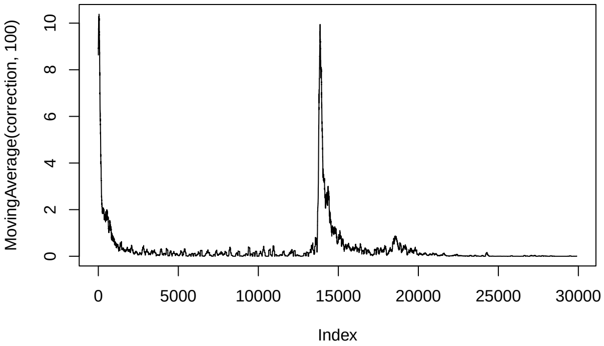

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
1
2
3
4
5
6
7
8
5
10
−1
10.0
7.5
5.0
2.5


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

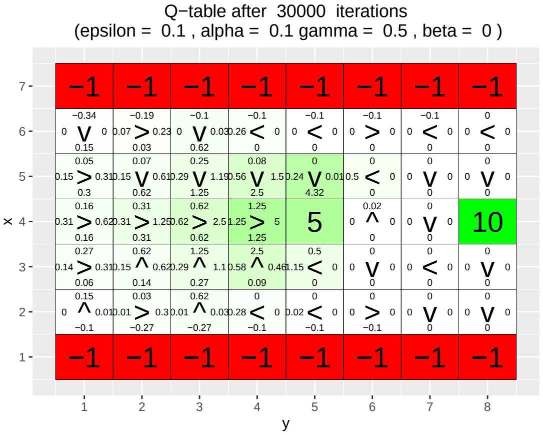

MovingAverage(correction, 100)
Index
0
1
2
3
4
5
0
5000
10000
15000
20000
25000
30000

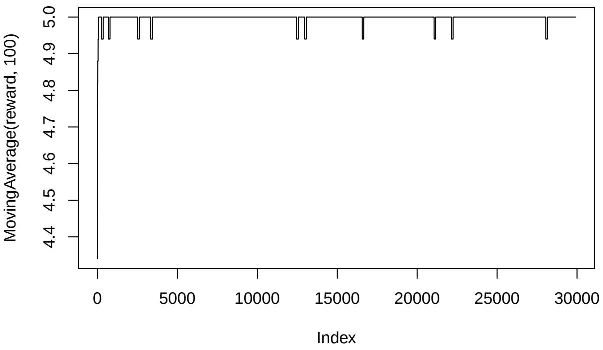


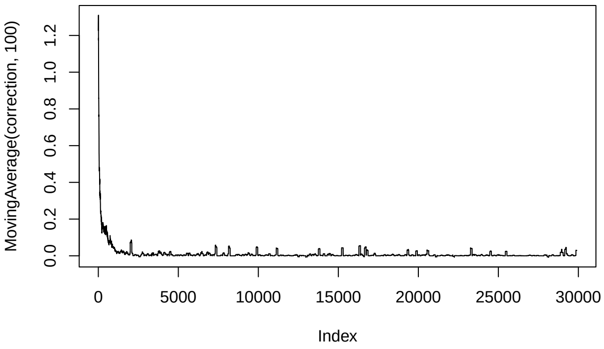


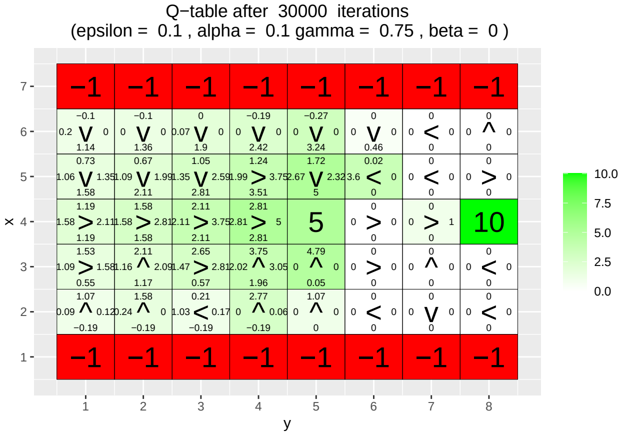


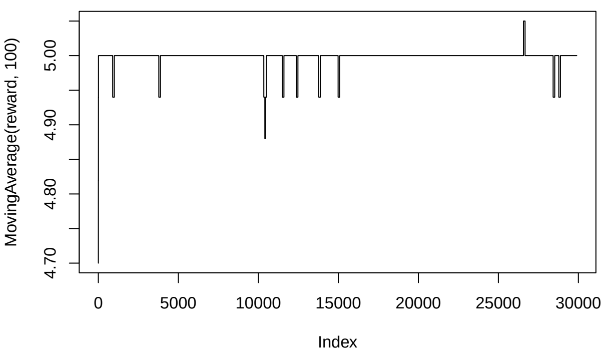
```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

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

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

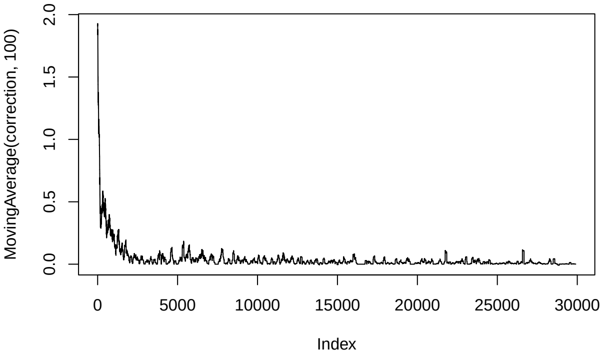

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
−1
5
10
10.0
7.5
5.0
2.5
0.0

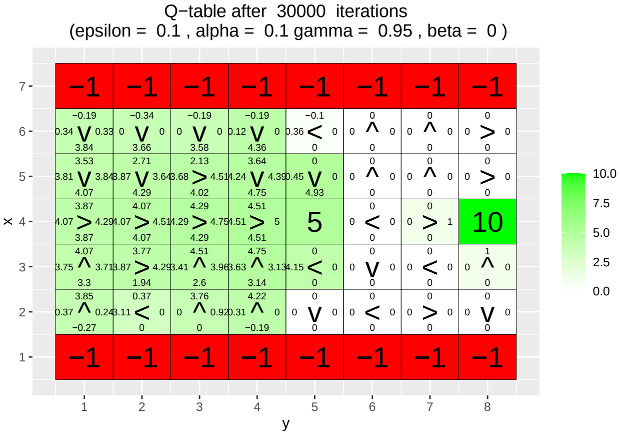

MovingAverage(reward, 100)
Index
4.4
4.5
4.6
4.7
4.8
4.9
5.0
0
5000
10000
15000
20000
25000
30000

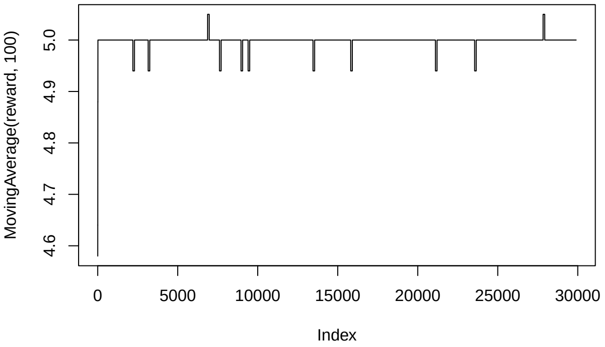

MovingAverage(correction, 100)
Index
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
5000
10000
15000
20000
25000
30000

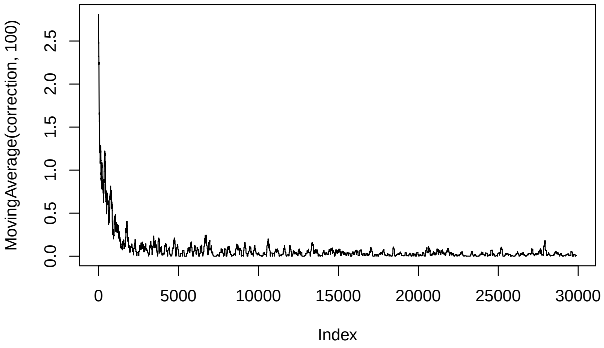

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
−1
5
10
10.0
7.5
5.0
2.5
0.0


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


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







3.1 Epsilon= 0.5, alpha=0.1, gamma=0.5, beta=0

High values for the moving average reward, correction gets close to zero around 5k episodes. Q table only gives reward to states close to 5 or 10. Most of the paths lead to 5.

3.2 Epsilon= 0.5, alpha=0.1, gamma=0.75, beta=0

The results looks to be close as for gamma = 0.5, two more states lead to 10 than previous.

3.3 Epsilon= 0.5, alpha=0.1, gamma=0.95, beta=0

We are now avoiding the 5 to get the higher reward in 10, gamma of 0.95 is making the agent to look for long term rewards(10) rather than short term(5) when starting at (4,1). Maybe this is learnt around 15k episodes as there is a big correction there and the average reward goes up at the same time. The agent have learned the optimal path to the reward of 10.

3.4 Epsilon= 0.1, alpha=0.1, gamma=0.5, beta=0

If we compare this to the one with epsilon = 0.5 and gamma = 0.5 we see that we don't learn that much and that we are not improving when looking at the moving average reward, the agent found out how to get to 5 then stuck by that. That is what can happen when epsilon is low. ## Epsilon= 0.1, alpha=0.1, gamma=0.75, beta=0

We haven't explored that much but we move to 10 easier than with gamma =0.5 and epsilon 0.1.

3.5 Epsilon= 0.1, alpha=0.1, gamma=0.95, beta=0

The same as before with epsilon =0.1, we don't explore that much, we can see that all the values on the right side of 5 is 0 in the Q-table. So even if the agent looks for long term reward, the low exploration makes it to go to 5 instead of 10.

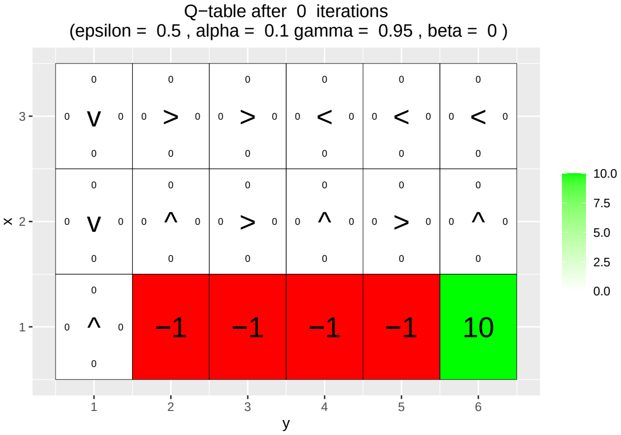
To summarize this, a low epsilon makes the agent too greedy and it's not exploring the environment enough, when starting in (4,1) it finds paths to 5 but not 10.

A higher gamma makes the agent more likely to find the 10, and a gamma of 0.5 makes the agent discount future reward more heavily making it prefer short term rewards more instead.

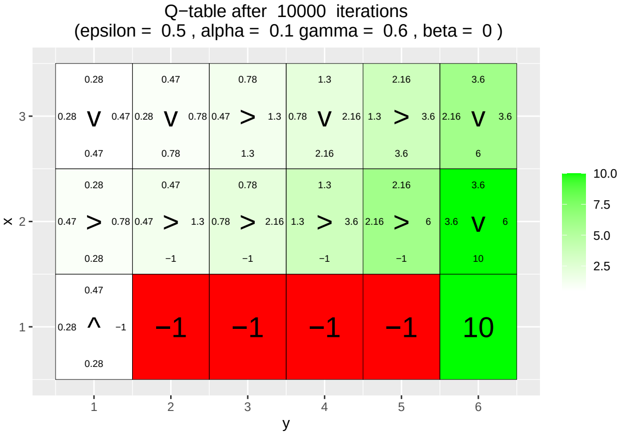
4 4

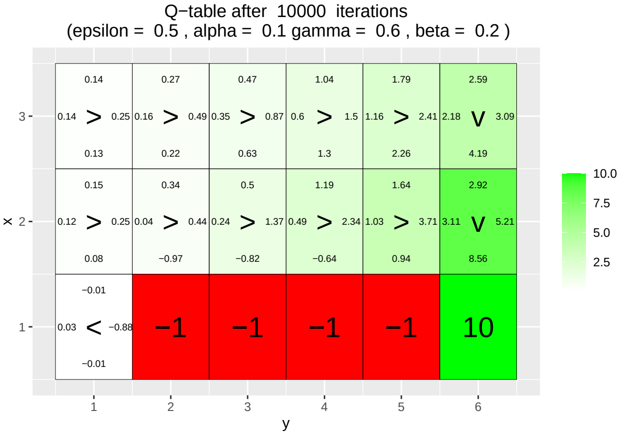
Environment C. This is a smaller 3 x 6 environment. Here the agent starts each episode in the state (1,1). Your task is to investigate how the β parameter affects the learned policy by running 10000 episodes of Q-learning with $\beta = 0, 0.2, 0.4, 0.66$, $\epsilon = 0.5$, $\gamma = 0.6$ and $\alpha = 0.1$. To do so, simply run the code provided in the file RL Lab1.R and explain your observations.

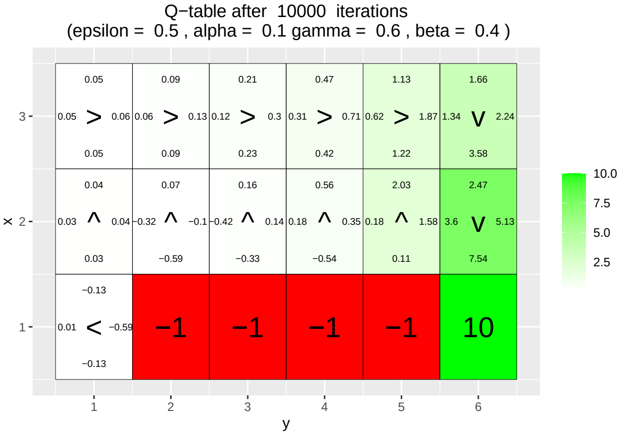
```r
# Environment C (the effect of beta).

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()
```

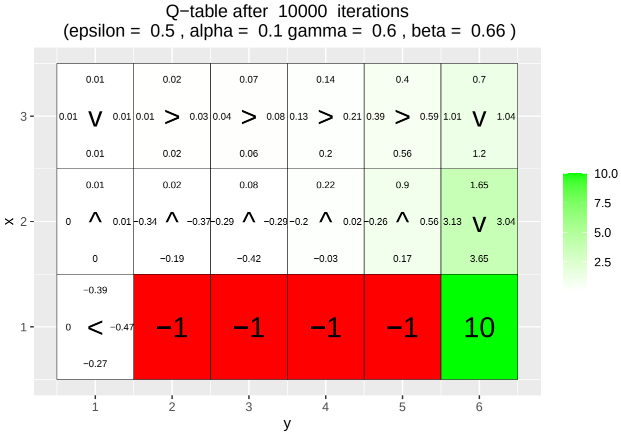


```
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

  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)
}
```









When beta is 0 the policy is deterministic and optimal for all states to find the goal. When beta then increases the environment becomes more random as beta is the probability to slip either right or left. So the optimal policy to find the goal for beta = 0 might not be the same for higher betas as the agent has to account for the probability of slipping. A shorter path to the goal but close to negative rewards could then lead to lower expected discounted reward because of this risk. It starts to get obvious for beta = 0.4 as the policy is more cautious, avoiding edges close to the negative rewards. For 0.6 it has almost started to prioritize getting away from the negative rewards over than getting to the goal.

5 5

Link to José's jupyter NB

The file RL Lab2 Colab.ipynb in the course website contains an implementation of the REINFORCE algorithm. The file also contains the result of running the code, so that you do not have to run it. So, you do not need to run it if you do not want.1 Your task is to study the code and the results obtained, and answer some questions. We will work with a 4 x 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.

6 6

Environment D. 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 in the code in the file RL Lab2 Colab.ipynb. The results provided in the file correspond to running the REINFORCE algorithm for 5000 episodes with $\beta = 0$ and $\gamma = 0.95$. Each training episode uses a random goal position from train goals. The initial position for the episode is also chosen at random. When training is completed, the code validates the learned policy for the goal positions in val goals. This is done by with the help of the function vis prob, which shows the grid, goal position and learned policy. Note that each non-terminal tile has four values. These represent the action probabilities associated to the tile (state). Note also that each non- terminal tile has an arrow. This indicates the action with the largest probability for the tile (ties are broken at random). Finally, answer the following questions:

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

```r
knitr::include_graphics('pic1.png')
```

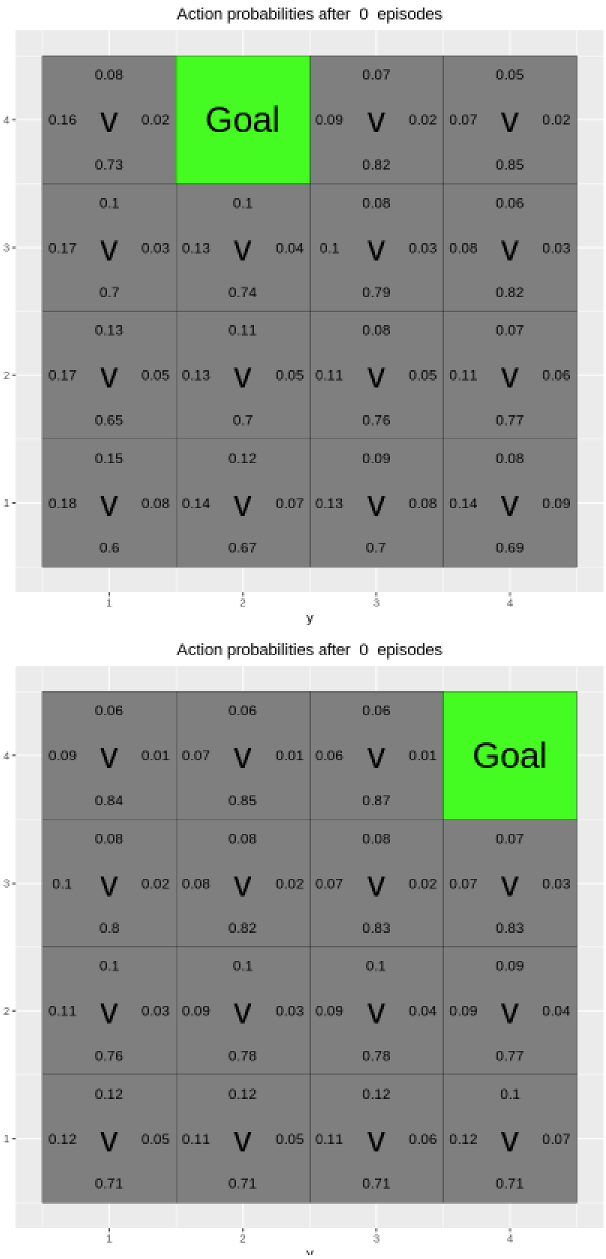
```r
knitr::include_graphics('pic2.png')
```

The policy looks okay/decent, most of the time it finds the goal, but not every state is optimal.

But in general its a good policy as we have learnt to generalize the model, as its able to find goals its not trained on.

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

Q-learning would have a problem to generalize to new data. But if we were to go through with the the algorithm we would have problems as we need to add all of the goal positions in the Q-table, making it significantly increase in size, and this would make the algorithm take time to converge and create storage space problems. This would make the algorithm "work", but its not generalizing as it would need to train on every possible goal.



Action probabilities after 0 episodes

Action probabilities after 0 episodes

Figure 1: Example before training

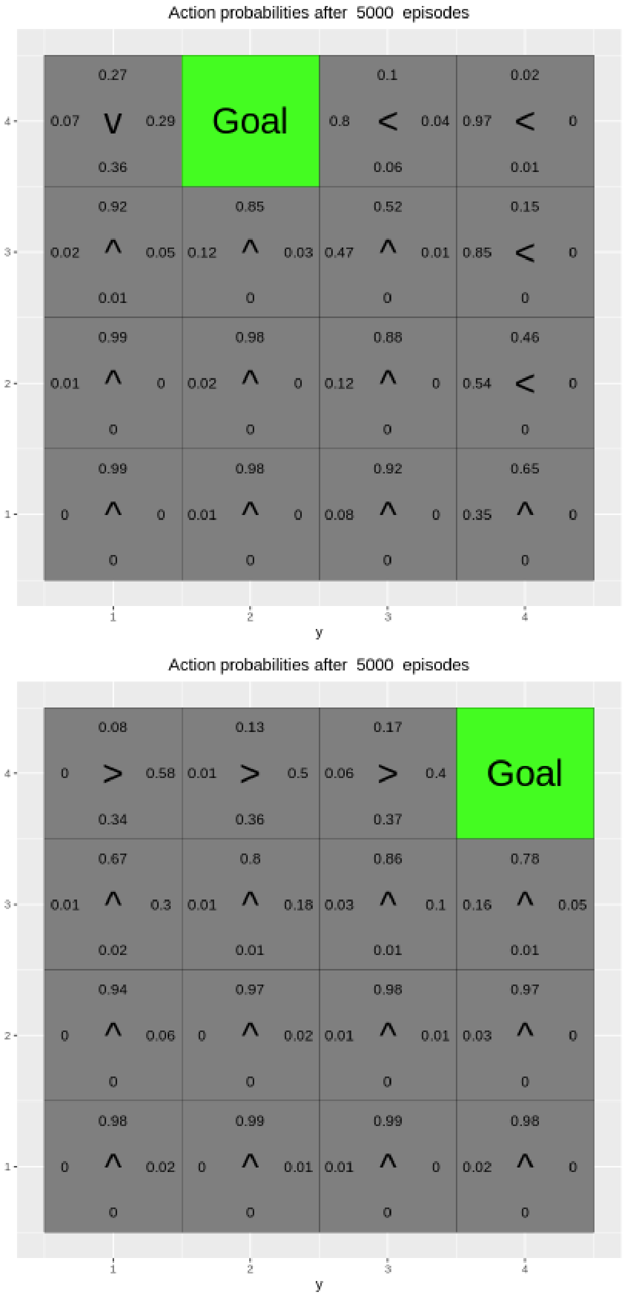


Figure 2: Example after training

7 7

Environment E. In this task, the goals for training are all from the top row of the grid. The validation goals are three positions from the rows below. To solve this task, simply study the code and results provided in the file RL Lab2 Colab.ipynb and answer the following questions:

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

The agent hasn't learned a good policy, we are trying to generalize the model but have only trained on data from the same row(4). We need to have a good sample of training data that can represent the environment.

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

The agents differ as the first one has trained on each row and each column 2 times each so it has learnt to explore much of the world. But the second one has only learnt from goals in one row making it having problems to generalize to the whole environment.