LAB 3

3.1

- You are provided with a gridworld environment where an agent can move up, down, left, or right. The agent starts each episode at a specific location and receives rewards based on the environment's layout.
- Implement the **Greedy** and **Epsilon-Greedy** policies in the functions `GreedyPolicy` and `EpsilonGreedyPolicy` respectively.
- Implement the **Q-learning** algorithm in the function `q_learning`, which will update the Q-table based on the agent's actions and rewards.

```
rm(list = ls())

# 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)
set.seed(1234)

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}.

  # Your code here.
  q_values = q_table[x, y, ]

  best_actions = which(q_values == max(q_values))
  if (length(best_actions) > 1) {
    action = sample(best_actions, 1)
  } else {
    action = best_actions
  }
  return (action)
}
```

```r
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 (epsilon >= runif(1)){
    action = sample(1:4, 1)
  } else {
    action = GreedyPolicy(x, y)
  }

  return (action)
}

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){

  #Initialize
  current_state = start_state
  episode_correction = 0
  reward = 0

  repeat{
    # Current state
```

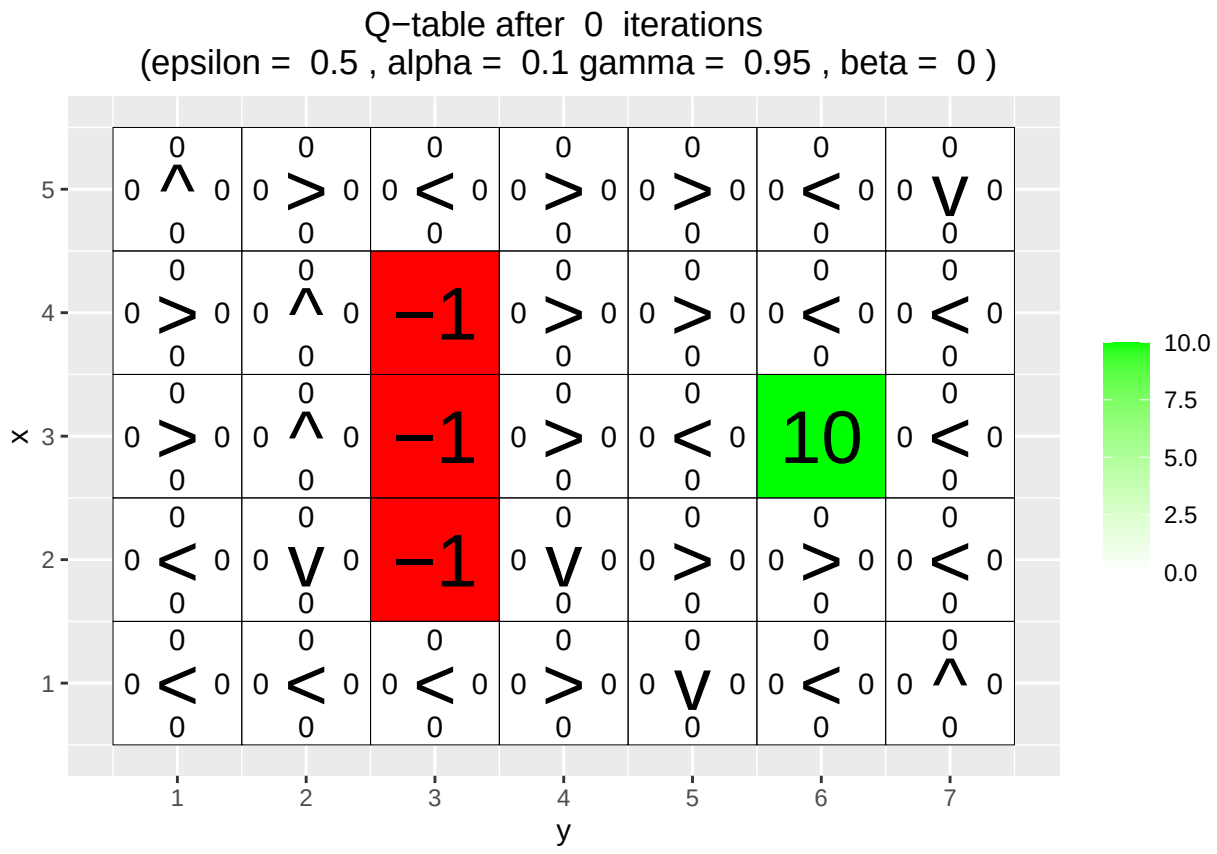
```
    x = current_state[1]
    y = current_state[2]

    # Action
    action = EpsilonGreedyPolicy(x, y, epsilon)

    # Update state
    next_state = transition_model(x, y, action, beta)
    next_x = next_state[1]
    next_y = next_state[2]

    # Reward
    reward = reward_map[next_x, next_y]

    # New max Q value
    max_q_next = max(q_table[next_x, next_y, ])

    # Correction
    correction = reward + gamma * max_q_next - q_table[x, y, action]

    # Update q_table based on correction
    q_table[x, y, action] <<- q_table[x, y, action] + alpha * correction

    # Accumulate corrections
    episode_correction = episode_correction + correction

    # Update state
    current_state = next_state

    # End the episode if a terminal state (non-zero reward) is reached
    if (reward != 0) {
      break
    }
  }

  return (c(reward, episode_correction))
}
```

3.2

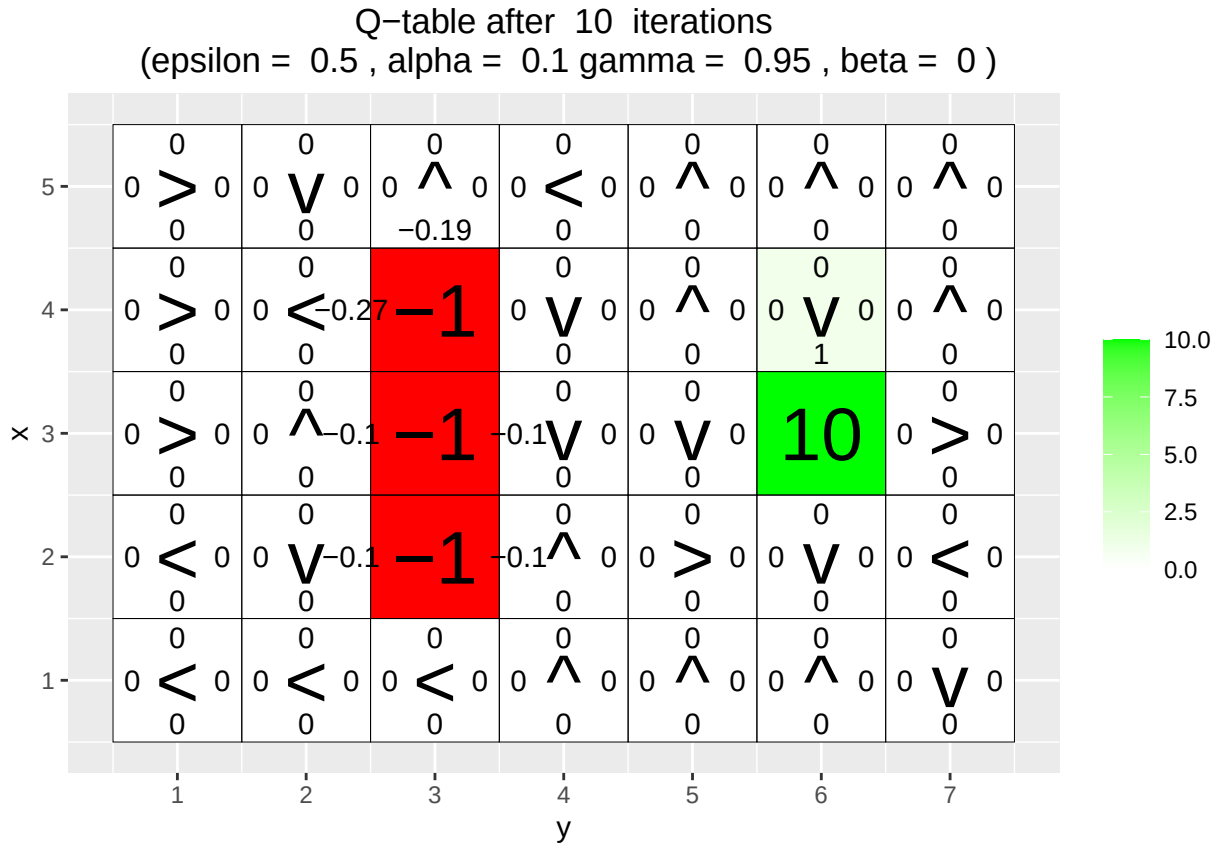
- Run 10000 episodes of Q-learning with $\epsilon = 0.5$, $\beta = 0$, $\alpha = 0.1$, and $\gamma = 0.95$. Visualize the results at various stages (after 10, 100, 1000, and 10000 episodes).

```
# Environment A (learning)
set.seed(1234)
H <- 5
W <- 7

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

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

vis_environment()
```

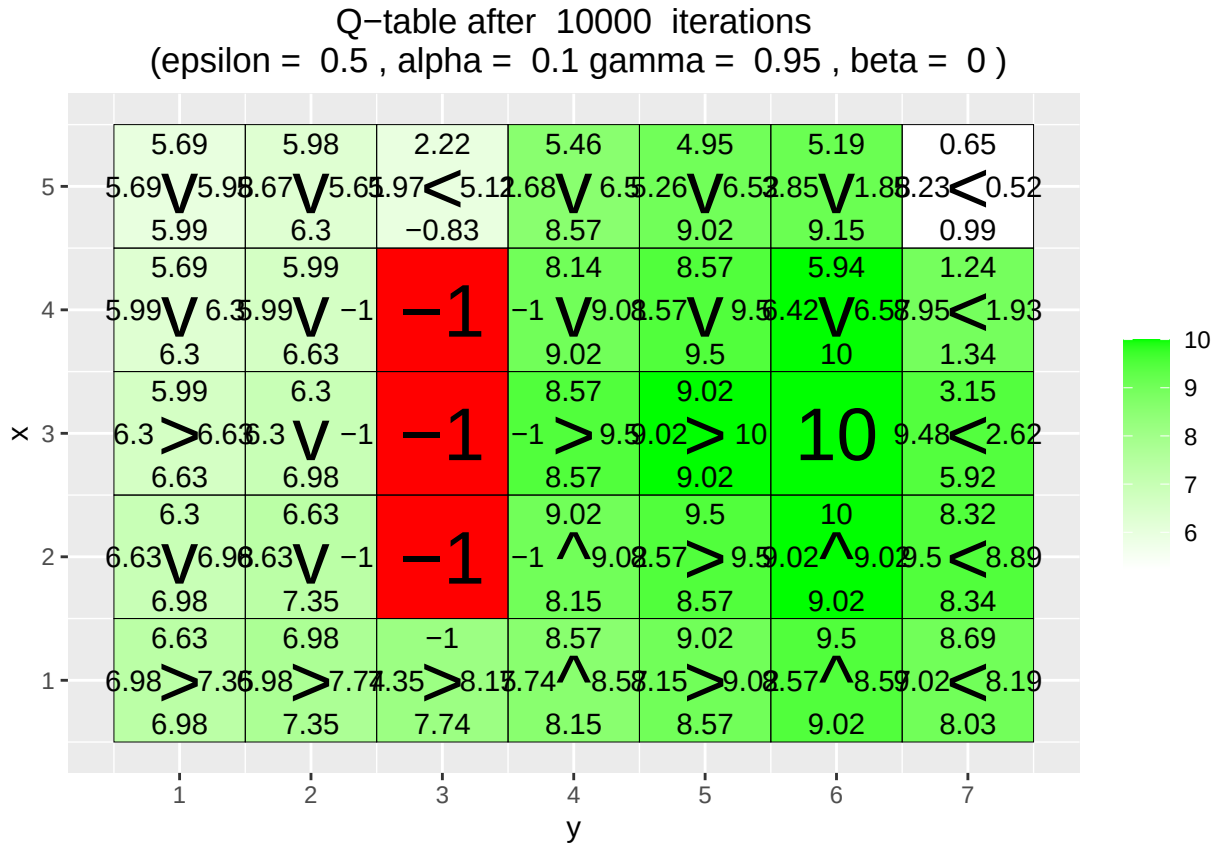



```
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))

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


Q−table after 10 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0
−0.19
−0.27
−0.1
−0.1
−0.1
−0.1
1
10


Q−table after 100 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0


Q−table after 1000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0



Q: What has the agent learned after the first 10 episodes ? A: To not enter(4, 3)

Q: Is the final greedy policy (after 10000 episodes) optimal for all states, i.e. not only for the initial state ? Why / Why not ? A: No. For example, the action policy for state (5, 3) is to go left, but the optimal action is right. This is due to the agent not having had the chance of exploring this state enough times.

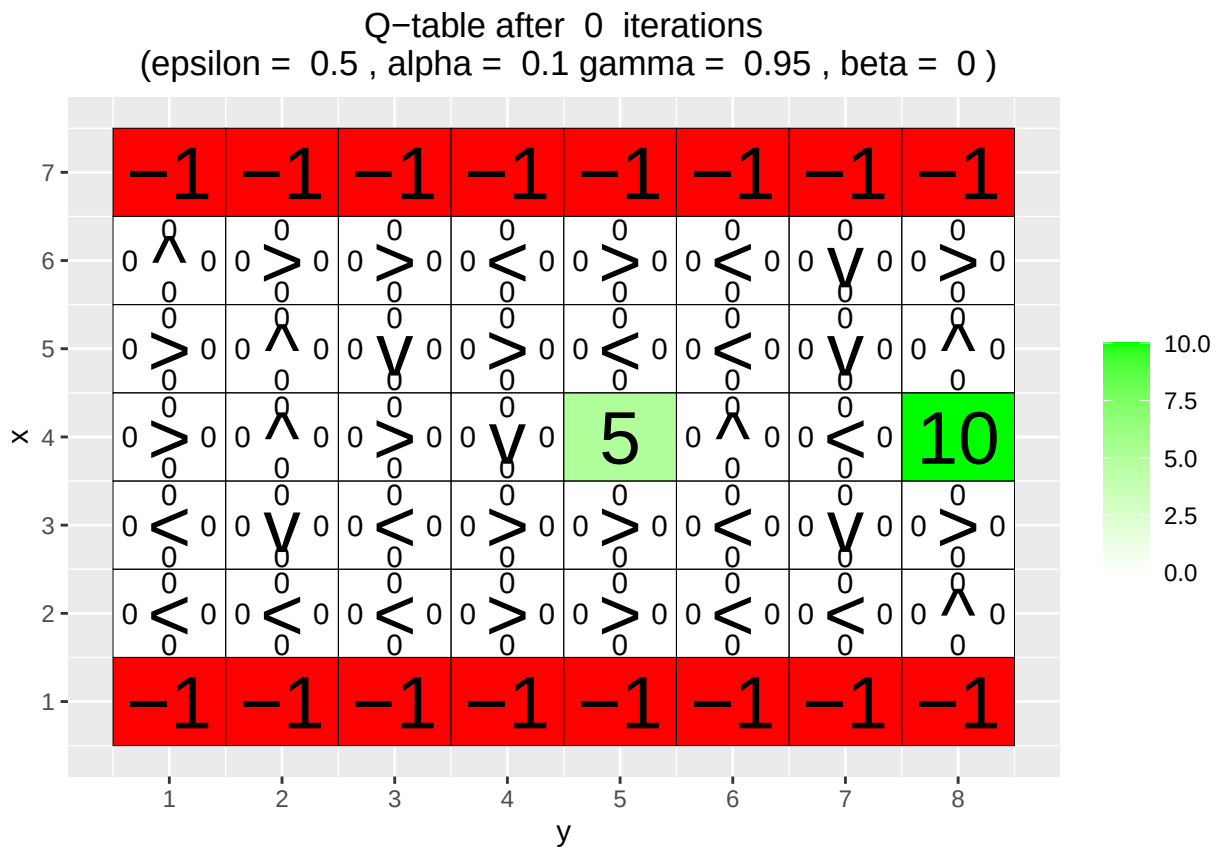
Q: Do the learned values in the Q-table reflect the fact that there are multiple paths (above and below the negative rewards) to get to the positive reward ? If not, what could be done to make it happen ? A: No, it always tries to go below. We could either enforce a greater epsilon, or change start_state to be further up (e.g (1, 5))

3.3

Environment B: - Investigate the effects of different values of ϵ and γ by running 30000 episodes of Q-learning with various configurations.

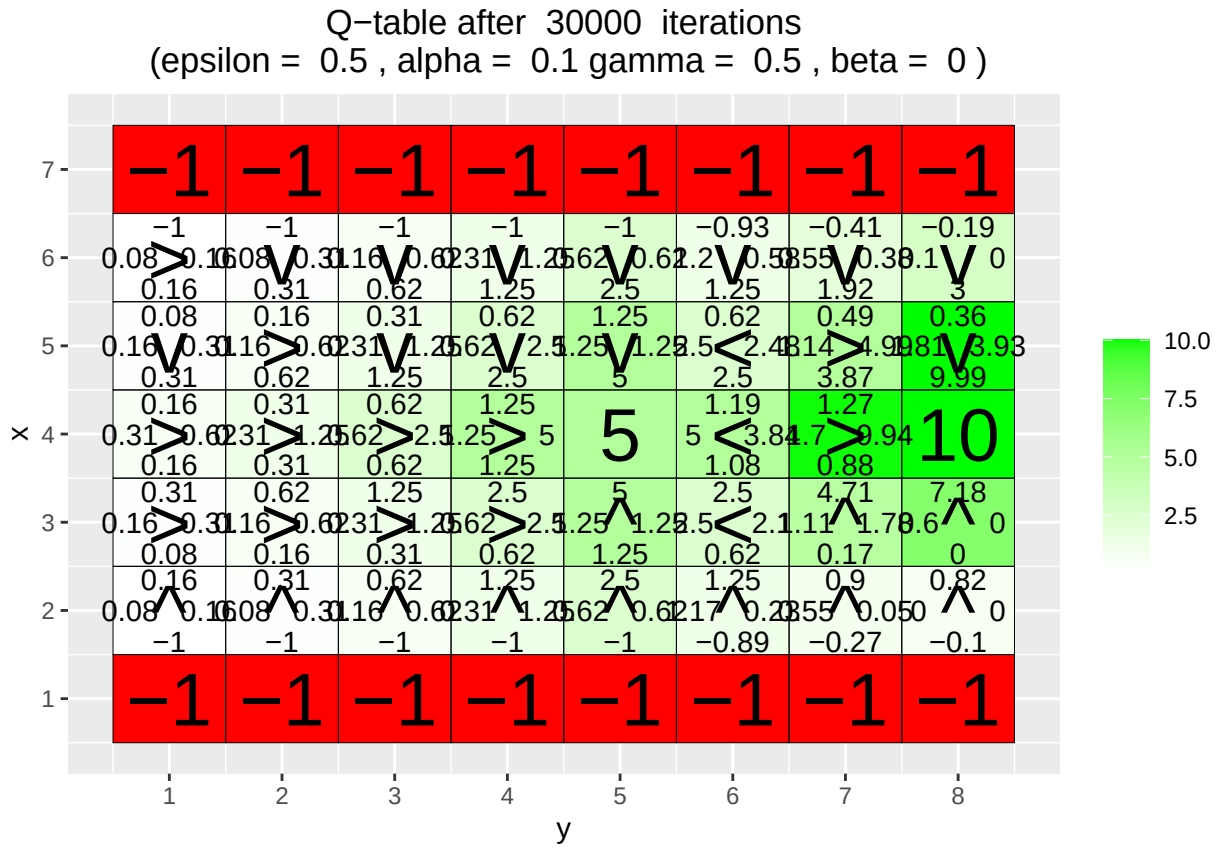
```
# Environment B (the effect of epsilon and gamma)
set.seed(1234)

H <- 7
W <- 8

reward_map <- matrix(0, nrow = H, ncol = W)
reward_map[1,] <- -1
reward_map[7,] <- -1
reward_map[4,5] <- 5
```

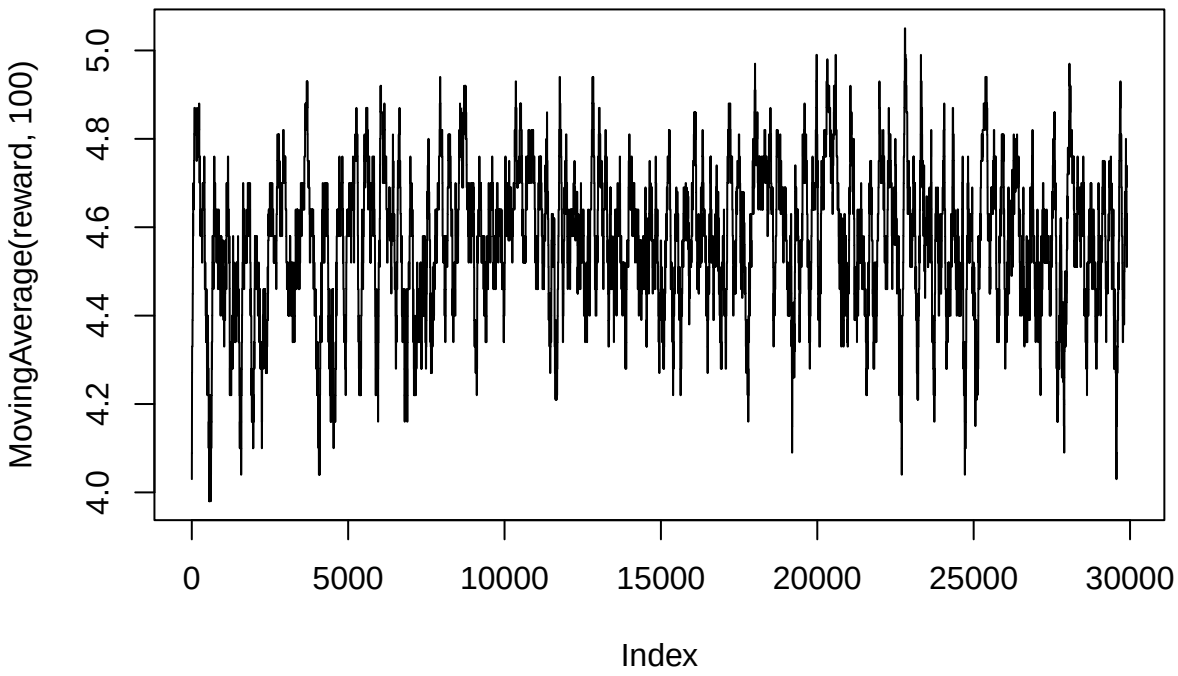
```
reward_map[4,8] <- 10

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

vis_environment()
```

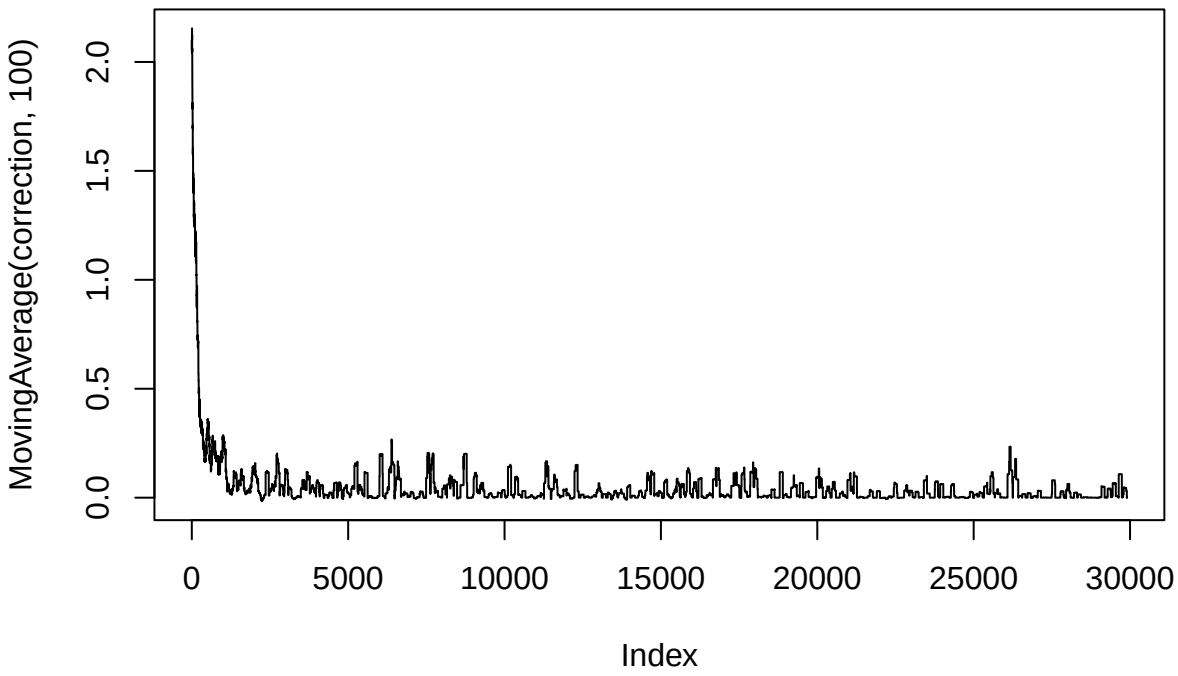


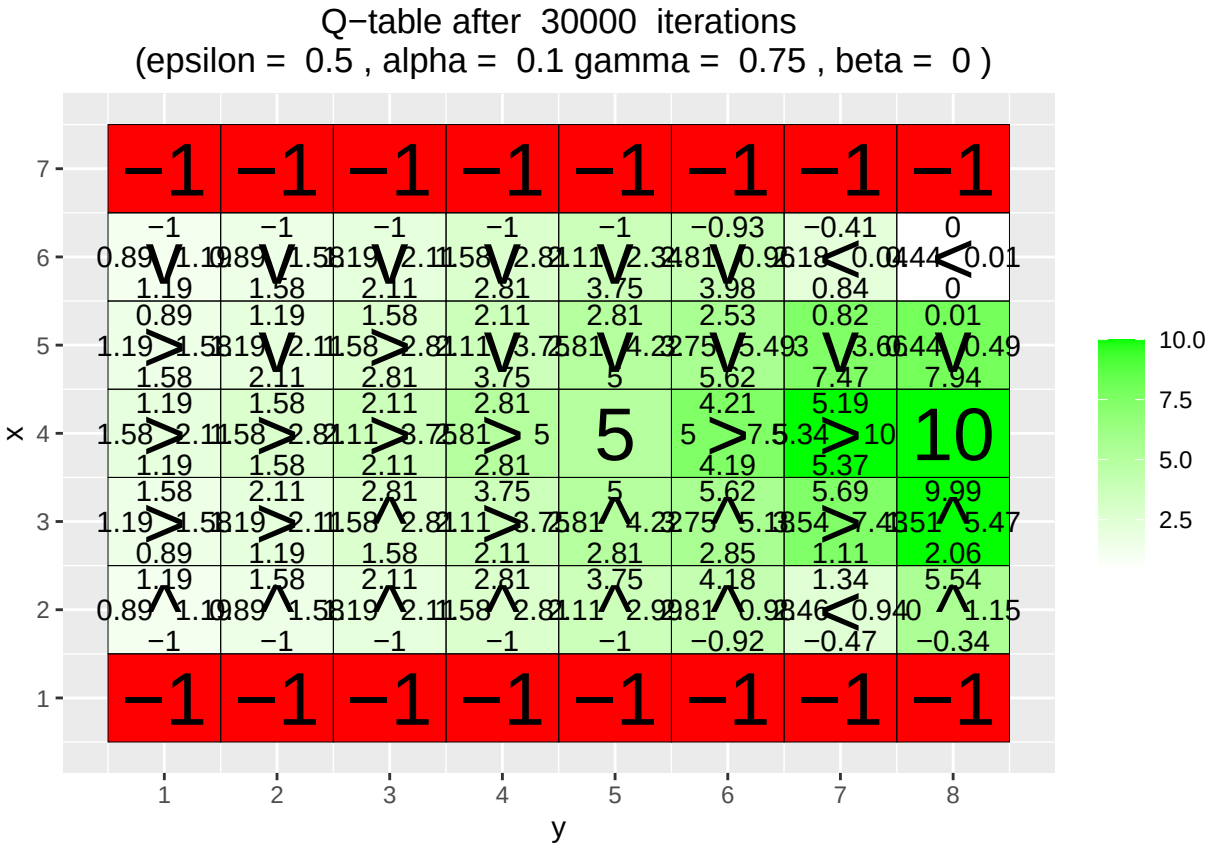

```
MovingAverage <- function(x, n){

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

  return (rsum)
}

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

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

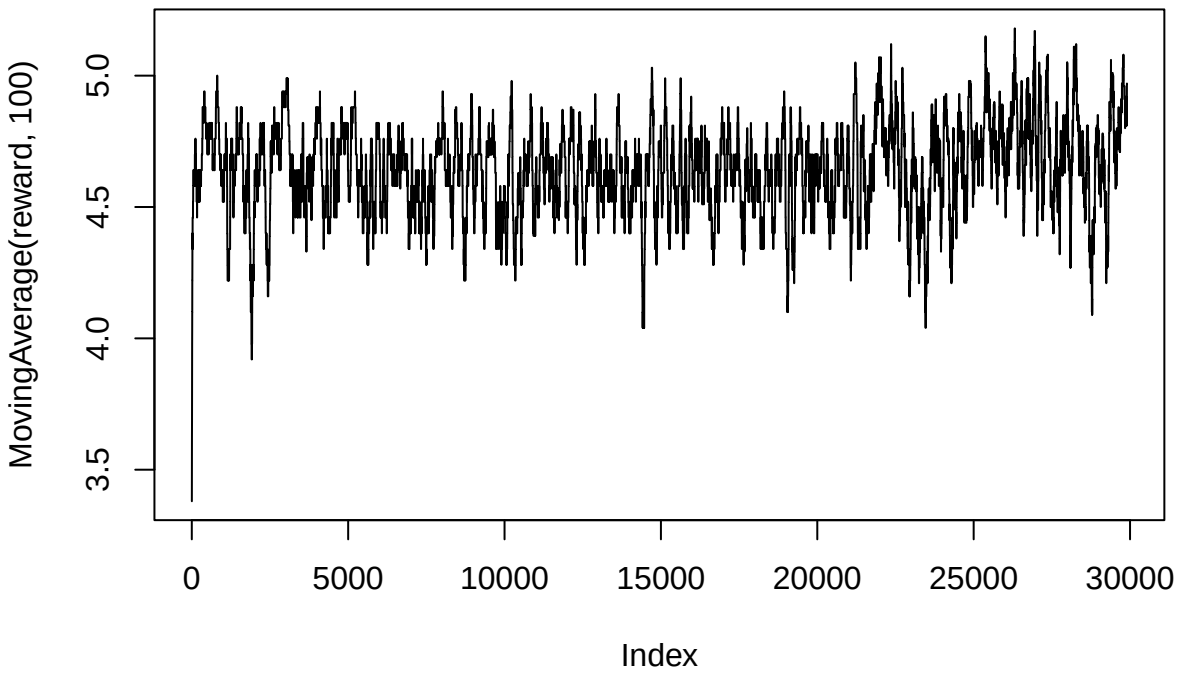
```
  vis_environment(i, gamma = j)
  plot(MovingAverage(reward,100),type = "l")
  plot(MovingAverage(correction,100),type = "l")
}
```

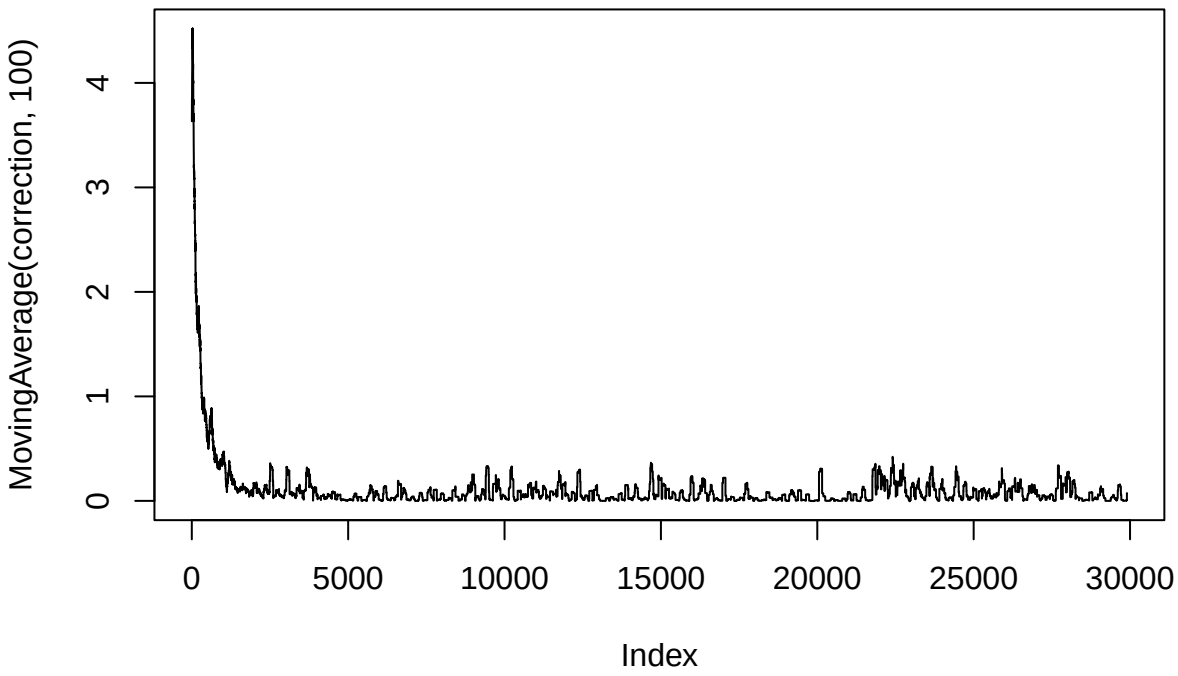


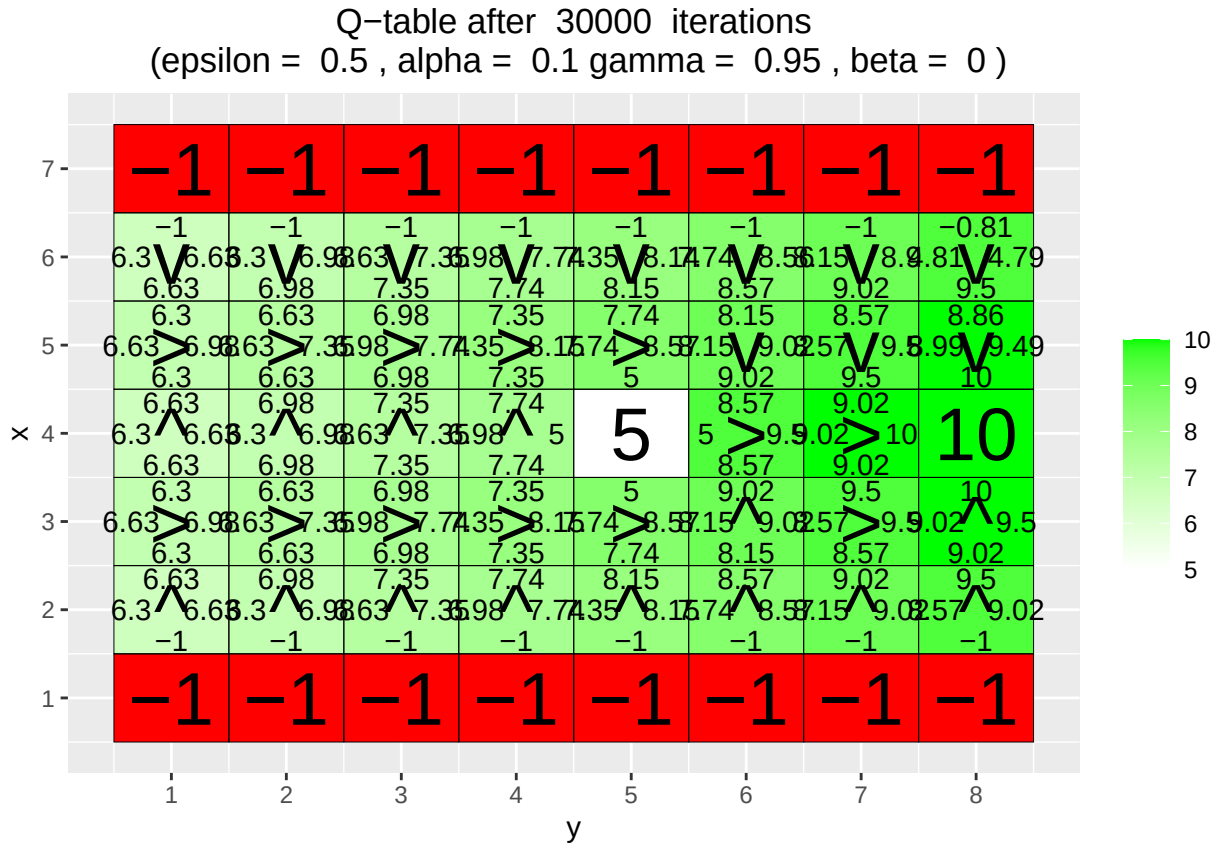

MovingAverage(reward, 100)
4.0
4.2
4.4
4.6
4.8
5.0
0
5000
10000
15000
20000
25000
30000
Index

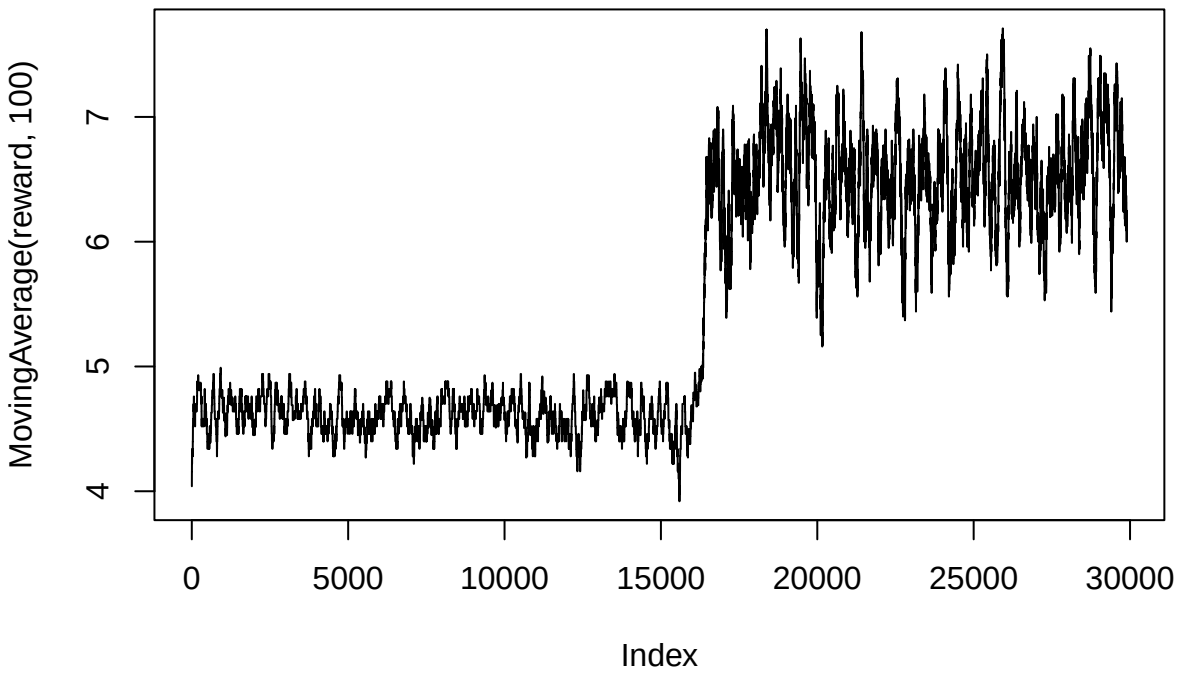

MovingAverage(correction, 100)
0.0
0.5
1.0
1.5
2.0
0
5000
10000
15000
20000
25000
30000
Index

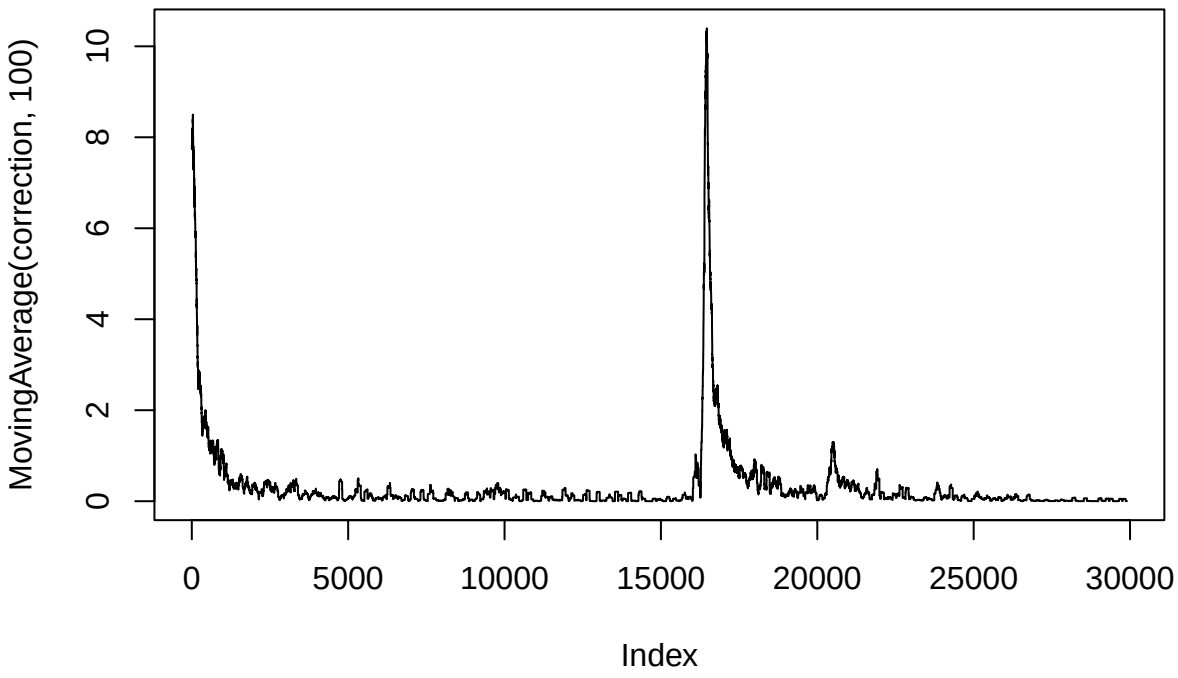

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5


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

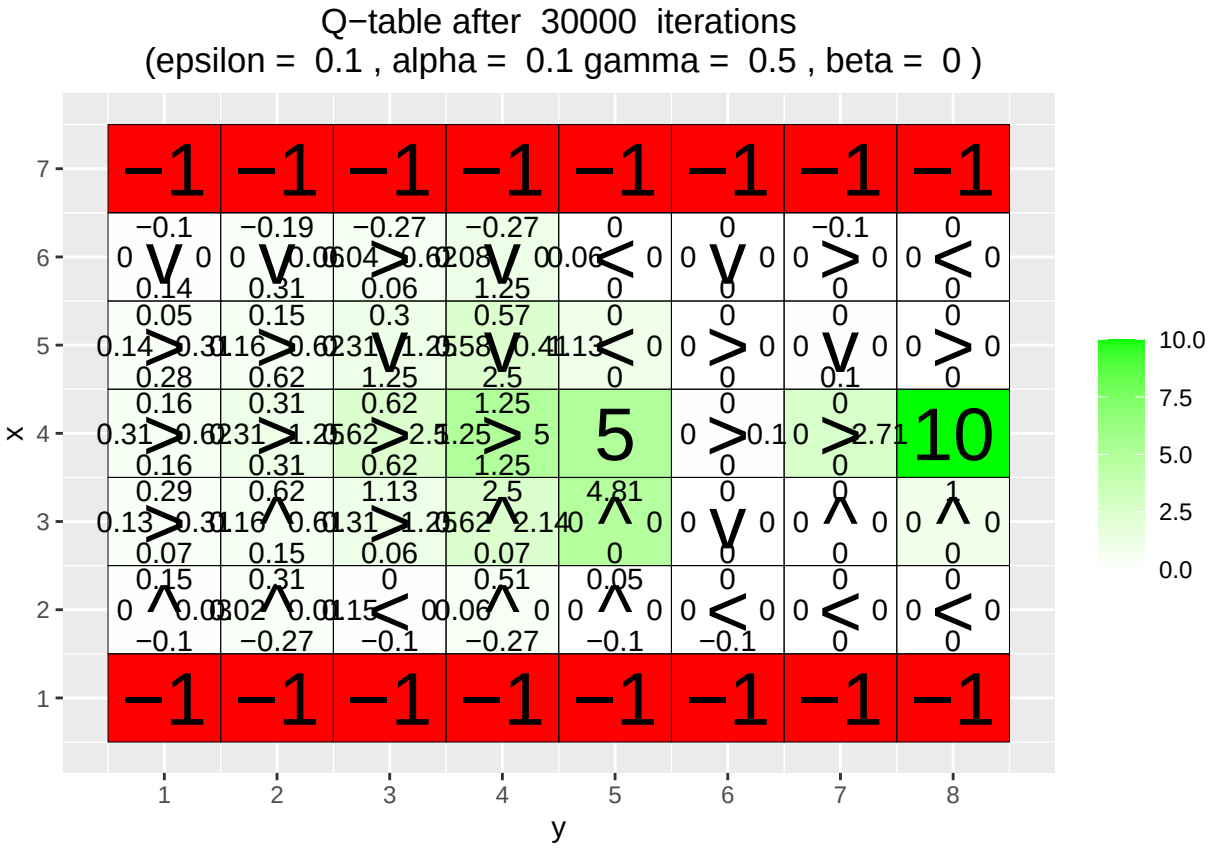

MovingAverage(correction, 100)
0
1
2
3
4
0
5000
10000
15000
20000
25000
30000
Index

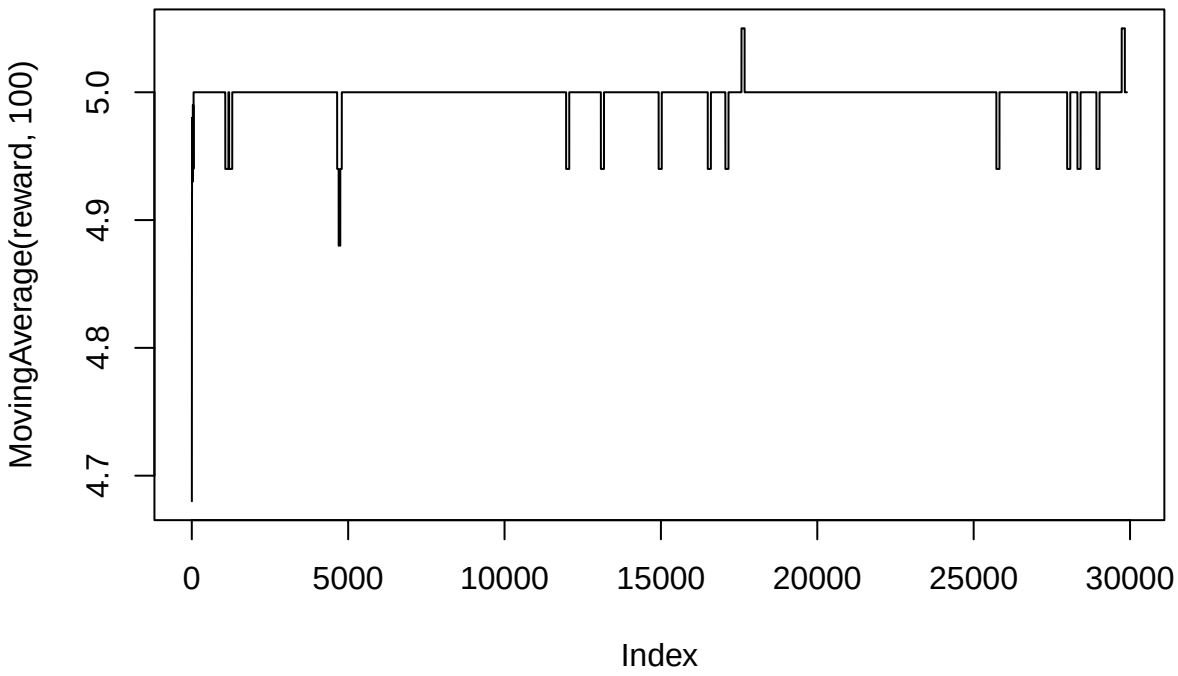

Q−table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)
x
y
10
9
8
7
6
5

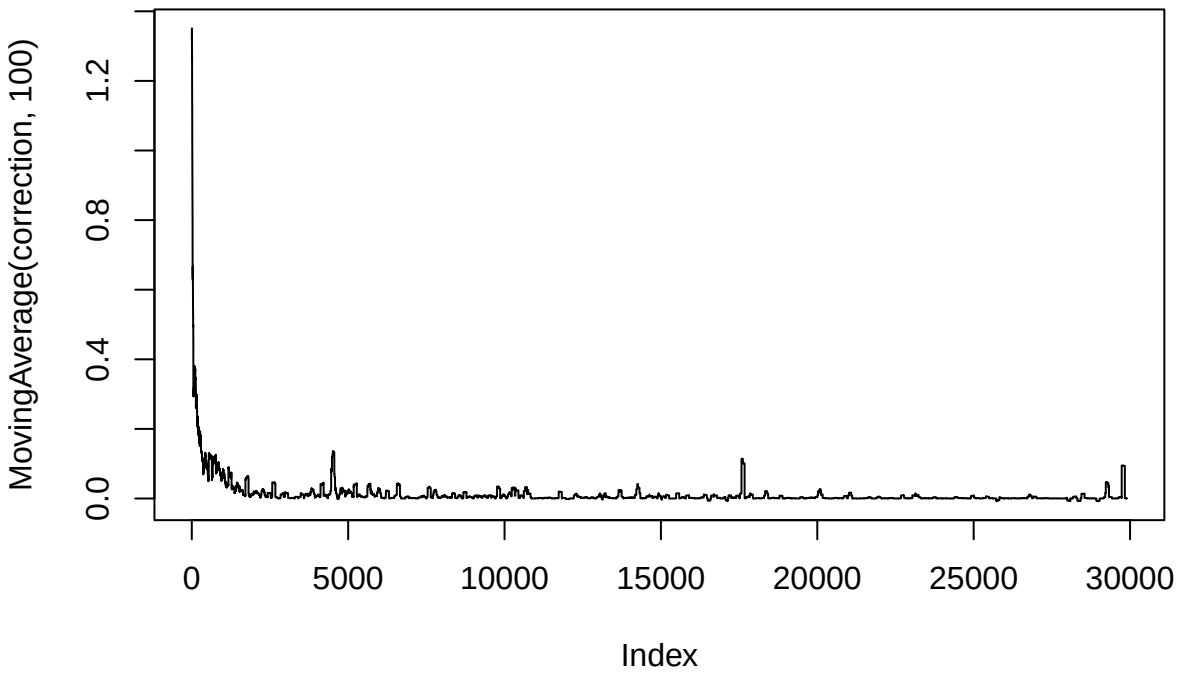

MovingAverage(reward, 100)
4
5
6
7
0
5000
10000
15000
20000
25000
30000
Index

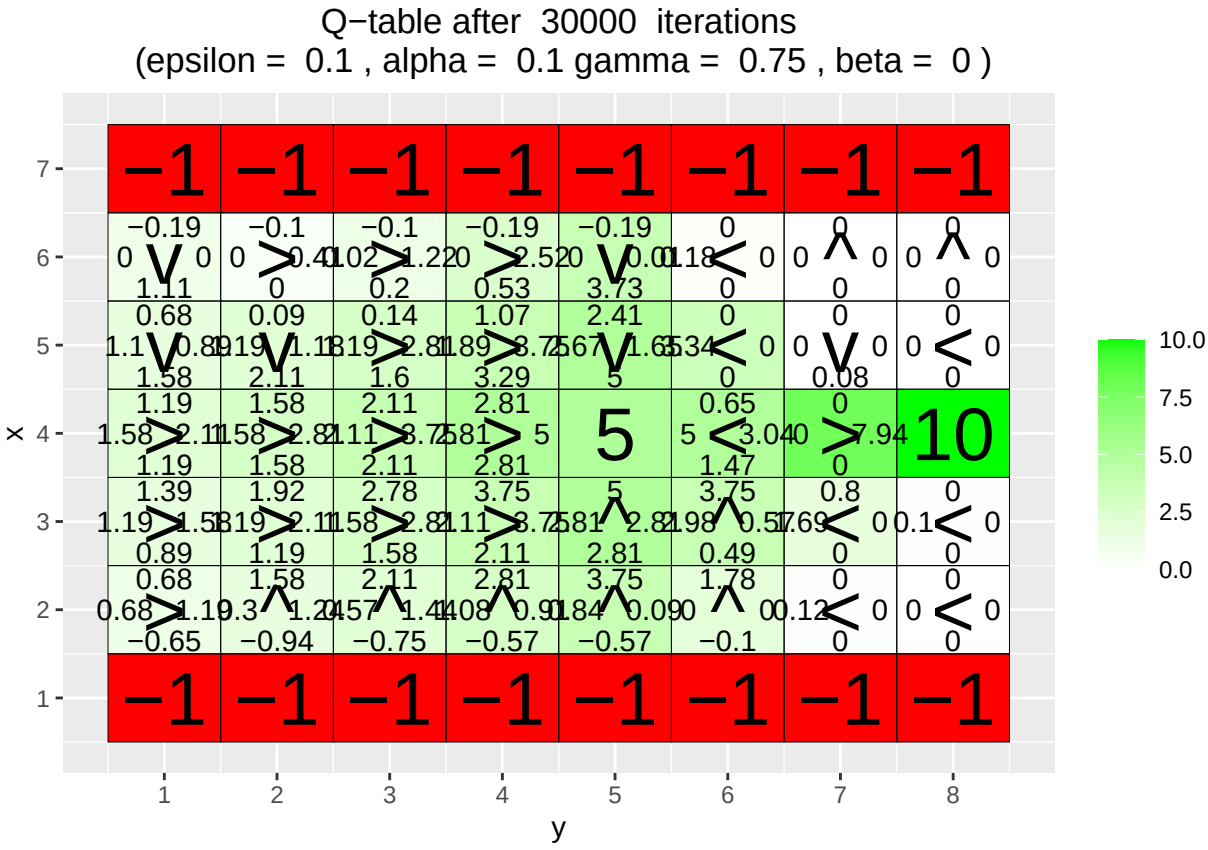


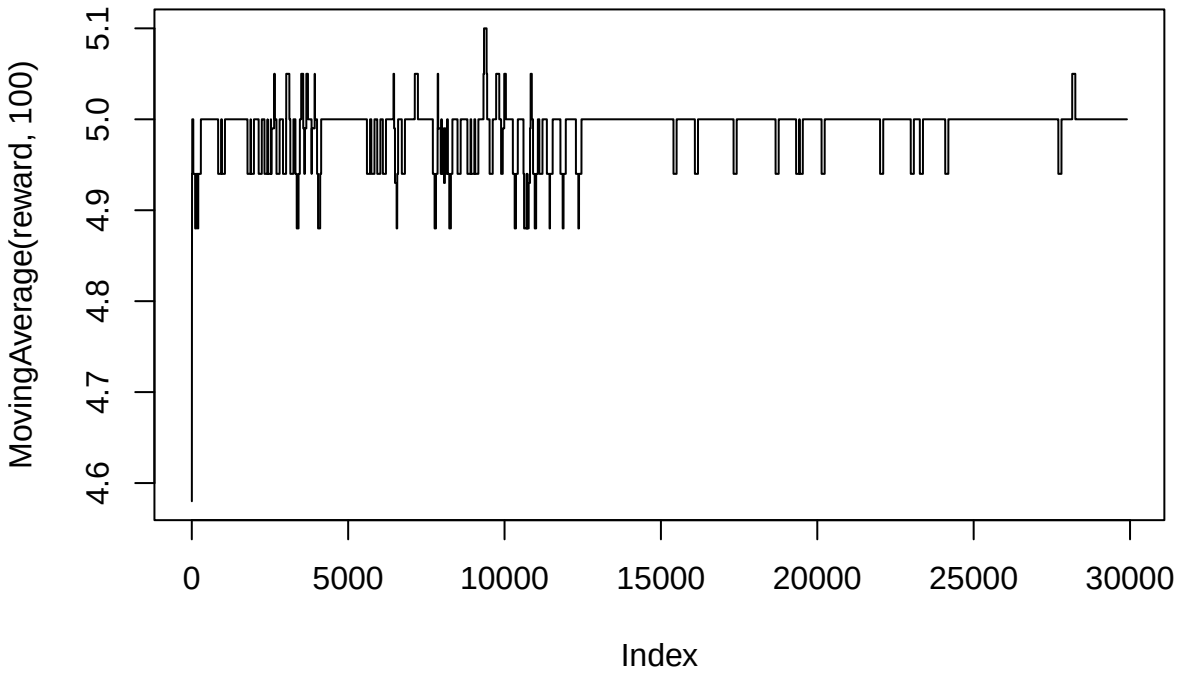

```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

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

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

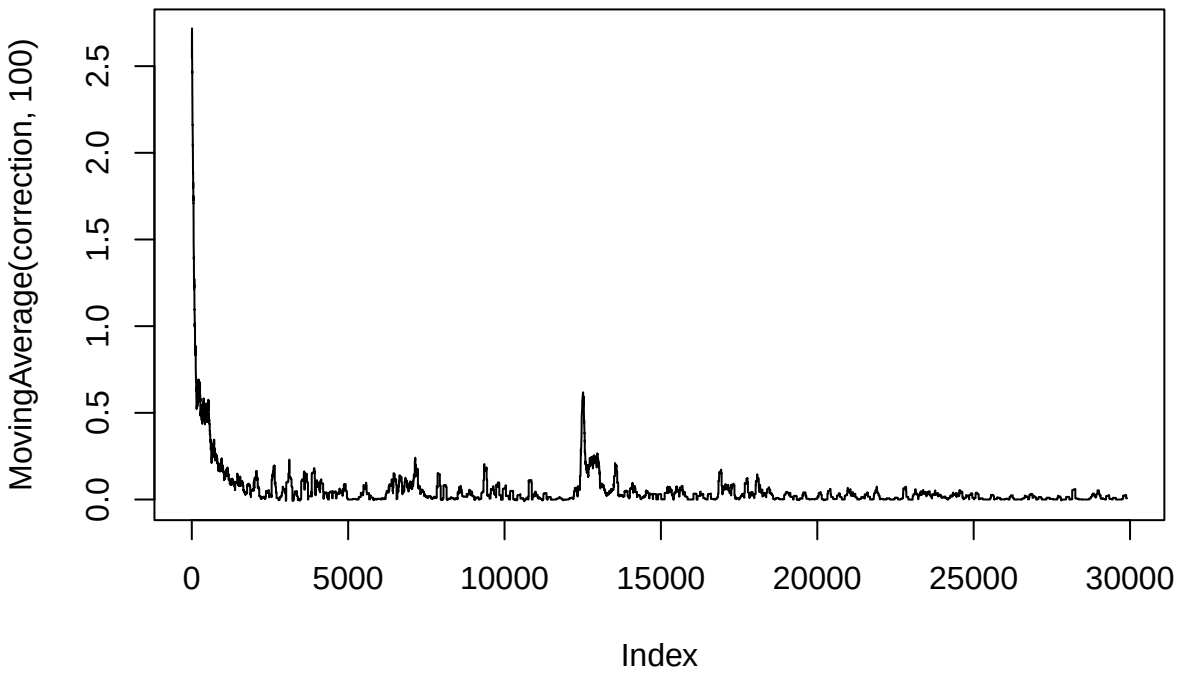

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0

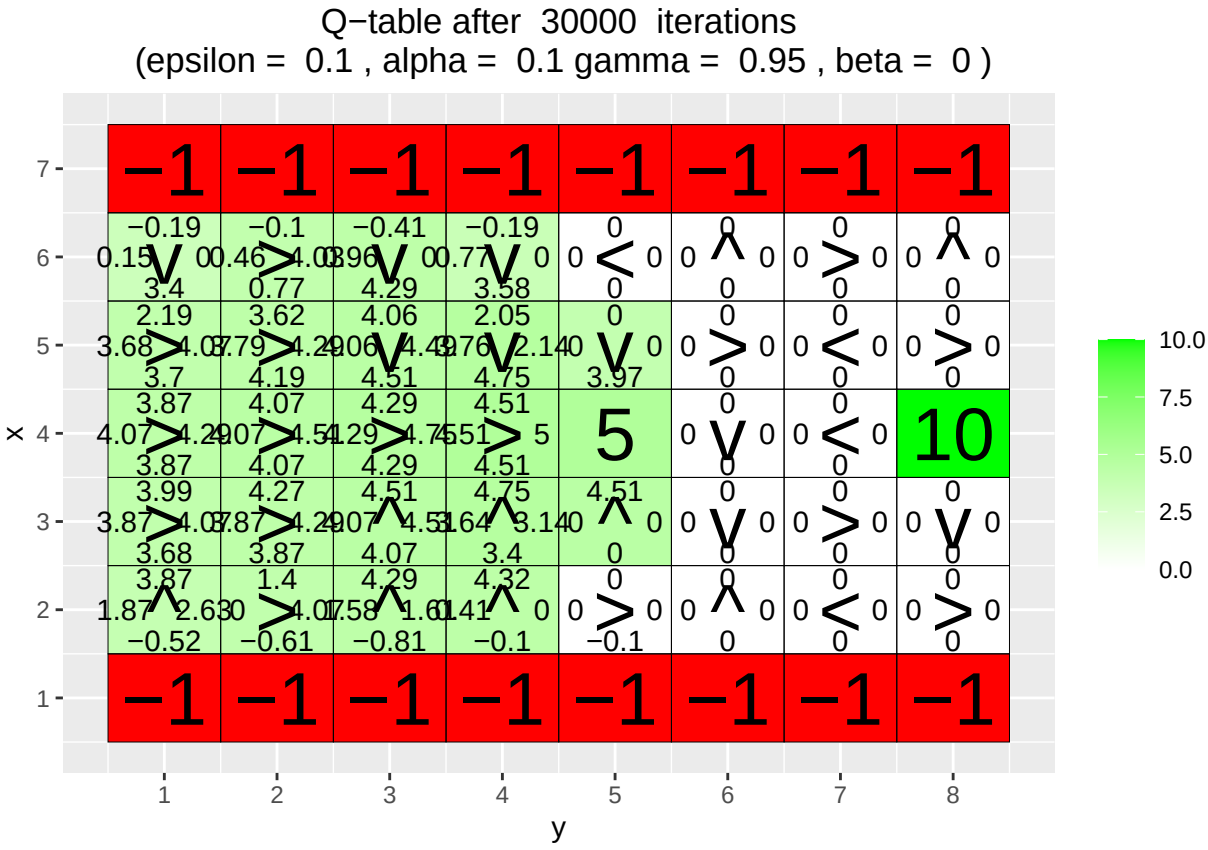

MovingAverage(reward, 100)
4.7
4.8
4.9
5.0
0
5000
10000
15000
20000
25000
30000
Index

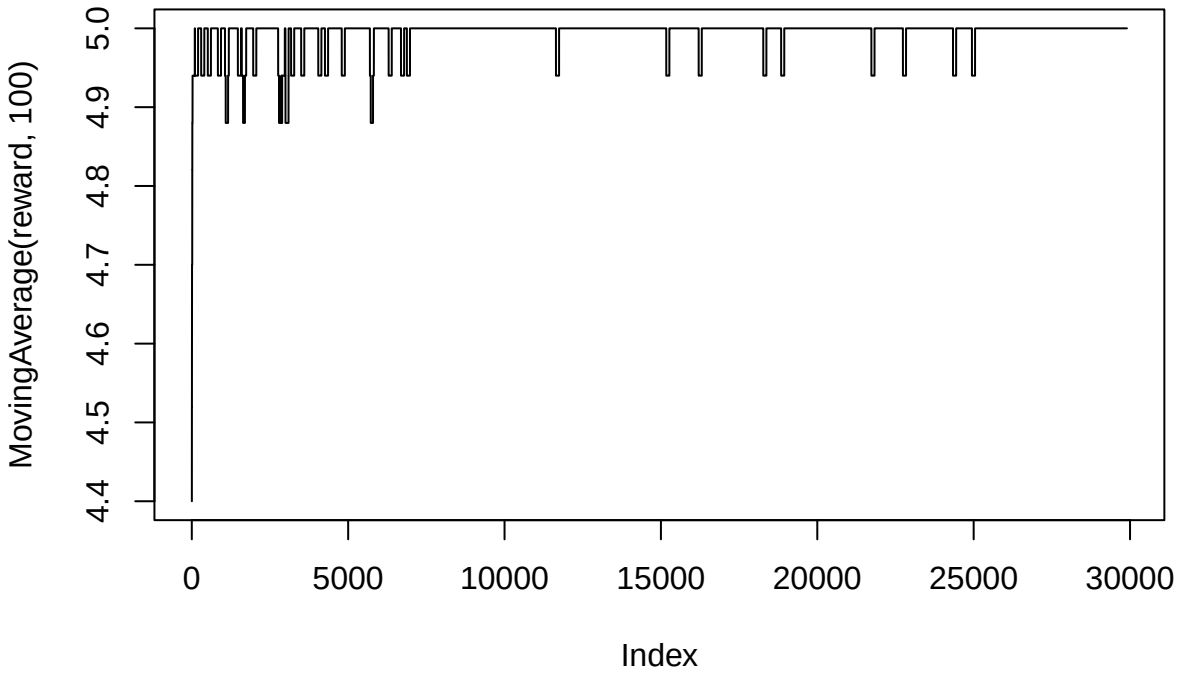

MovingAverage(correction, 100)
0.0
0.4
0.8
1.2
0
5000
10000
15000
20000
25000
30000
Index

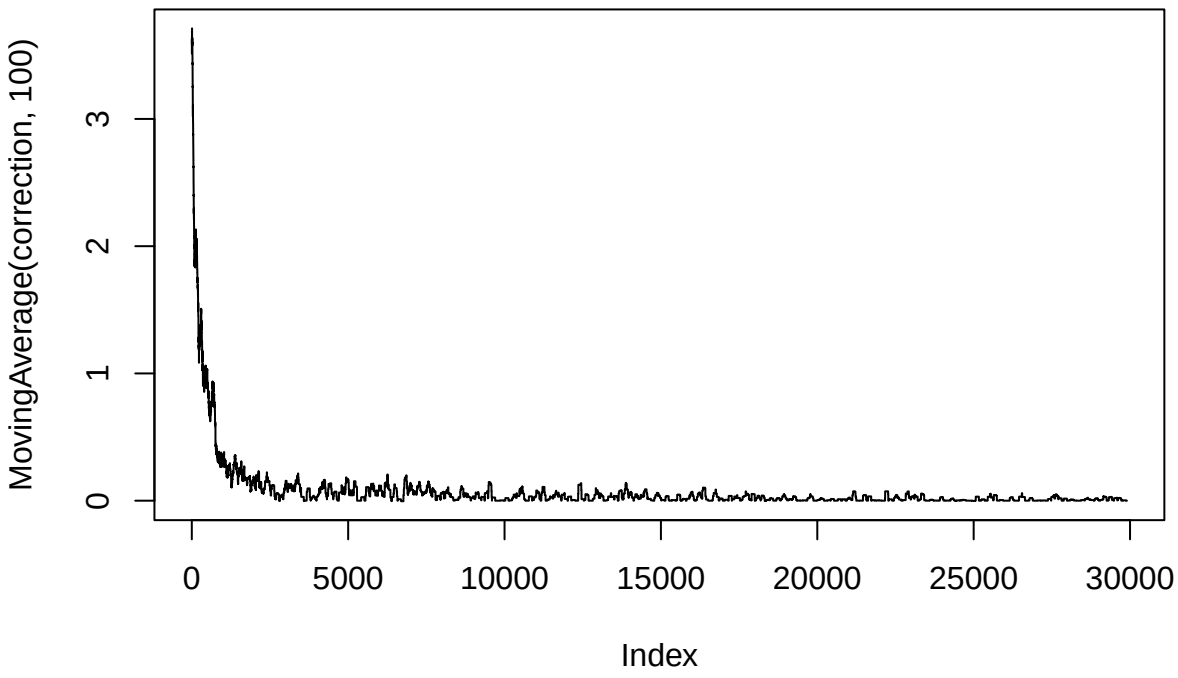

Q−table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)
x
y
10.0
7.5
5.0
2.5
0.0


MovingAverage(reward, 100)
4.6
4.7
4.8
4.9
5.0
5.1
0
5000
10000
15000
20000
25000
30000
Index


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


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


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

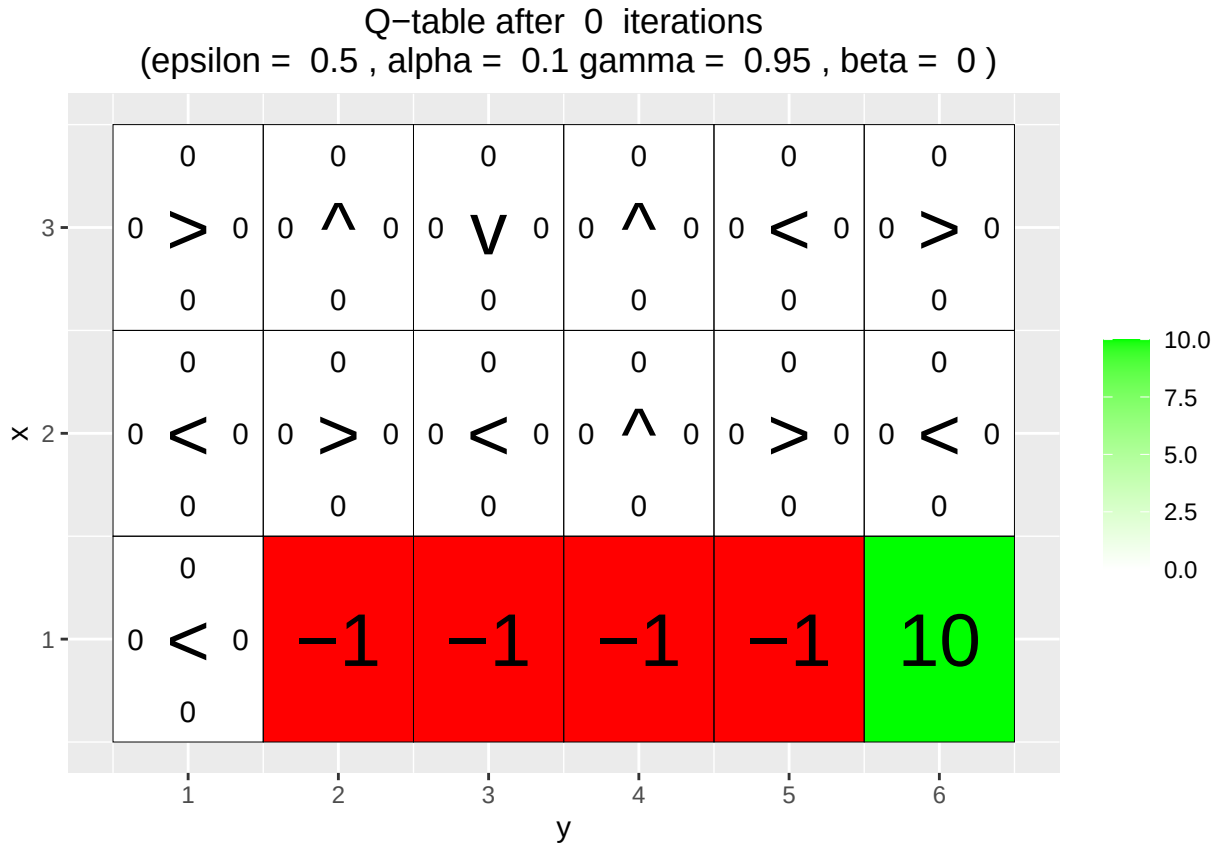


Q: Investigate how the ϵ and γ parameters affect the learned policy A: A high γ changes the action policy to be more prone going towards 10 over 5. A low ϵ leads to the agent not exploring certain states.

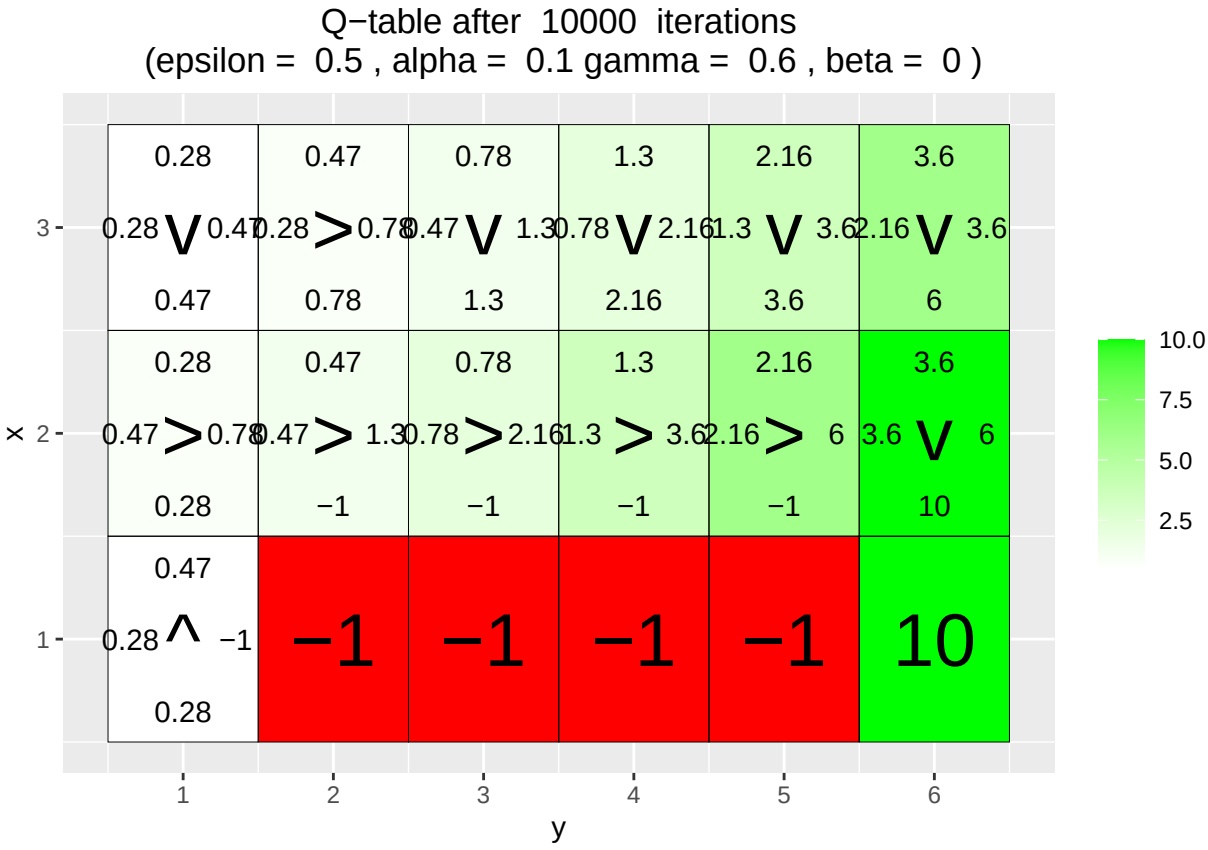
3.4

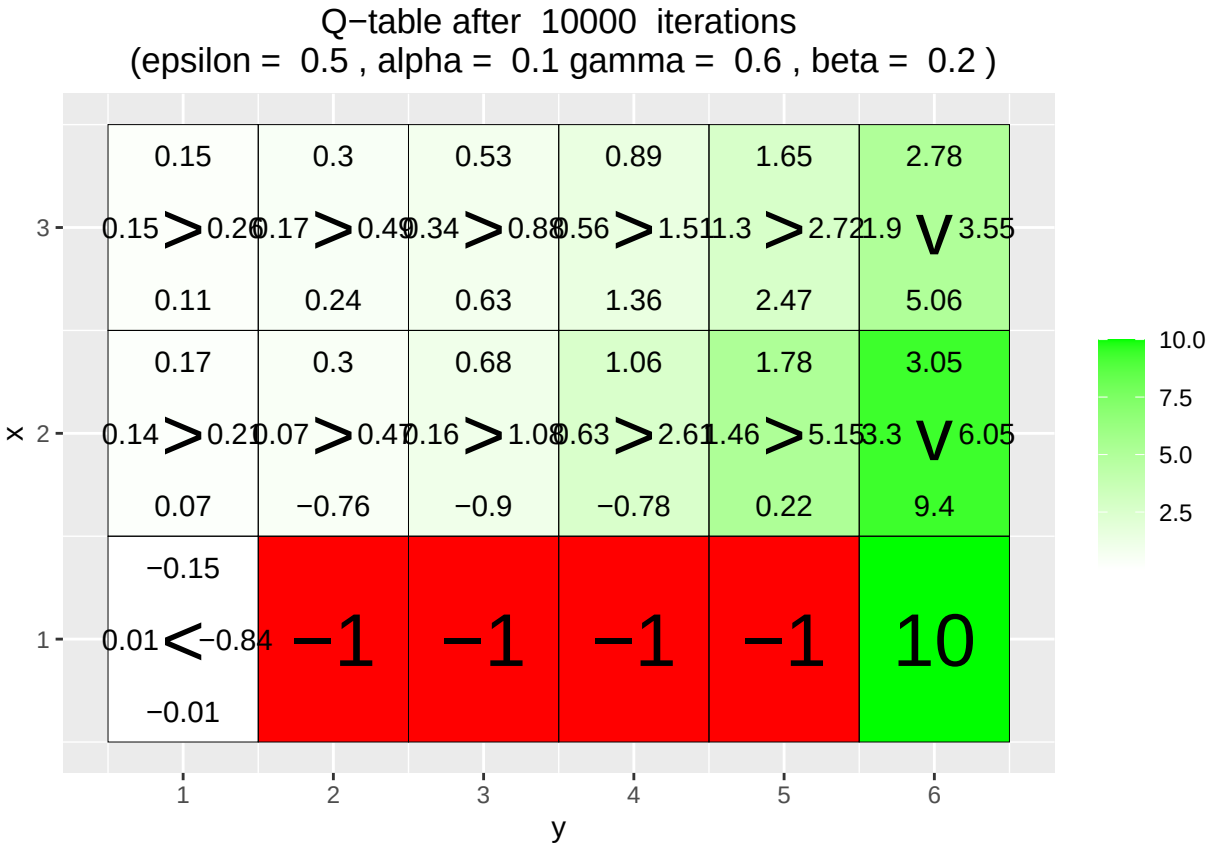
Environment C: - Investigate how the β parameter (which controls the probability of slipping) affects the learned policy by running 10000 episodes with different values of β.

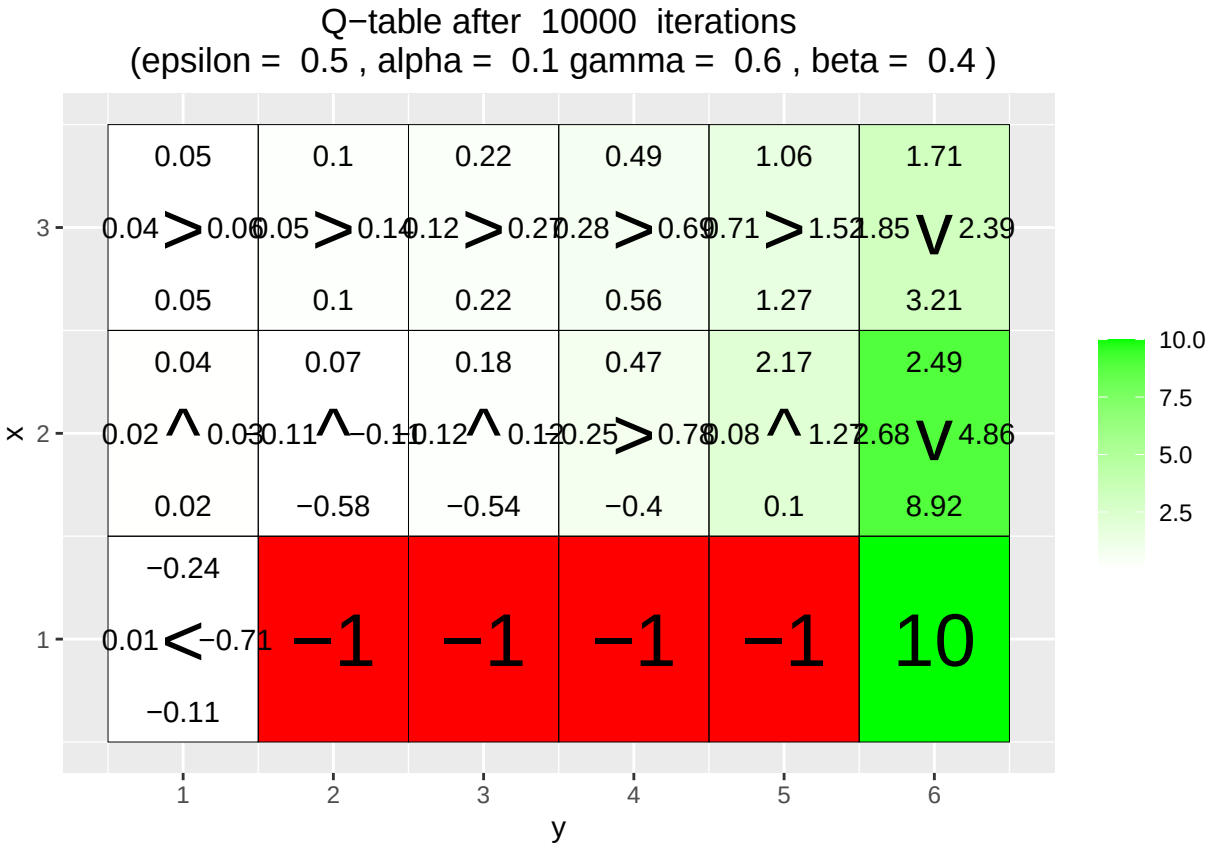
```
# Environment C (the effect of beta).
set.seed(1234)

H <- 3
W <- 6

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

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

vis_environment()
```

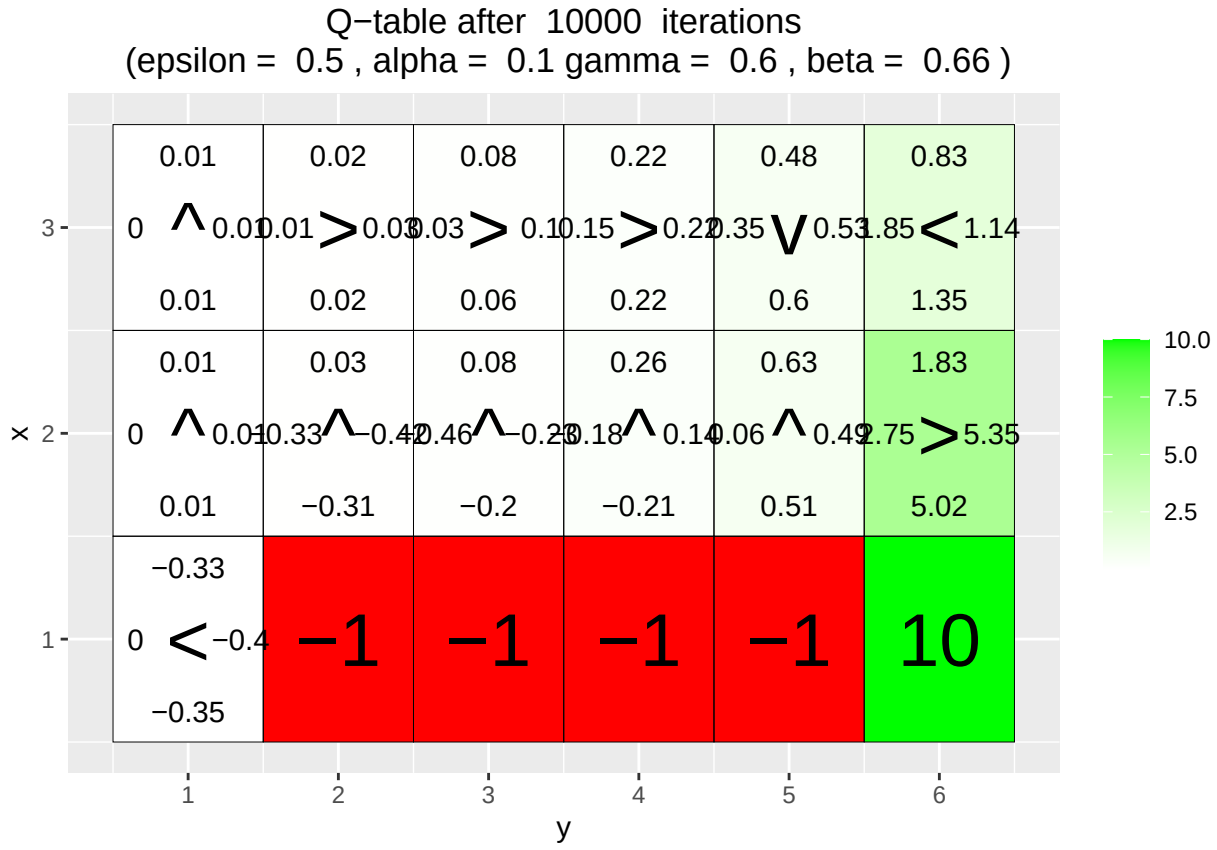



```
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

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

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


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0)
0.28
0.47
0.78
1.3
2.16
3.6
0.28 V 0.47
0.28 > 0.78
0.47 V 1.3
0.78 V 2.16
1.3 V 3.6
2.16 V 3.6
0.47
0.78
1.3
2.16
3.6
6
0.28
0.47
0.78
1.3
2.16
3.6
0.47 > 0.78
0.47 > 1.3
0.78 > 2.16
1.3 > 3.6
2.16 > 6
3.6 V 6
0.28
−1
−1
−1
−1
10
0.47
0.28 ^ −1
−1
−1
−1
−1
10
0.28
3
2
1
x
1
2
3
4
5
6
y
10.0
7.5
5.0
2.5


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)
0.15
0.3
0.53
0.89
1.65
2.78
0.15 > 0.26
0.17 > 0.49
0.34 > 0.88
0.56 > 1.51
1.3 > 2.72
1.9 V 3.55
0.11
0.24
0.63
1.36
2.47
5.06
0.17
0.3
0.68
1.06
1.78
3.05
0.14 > 0.21
0.07 > 0.47
0.16 > 1.08
0.63 > 2.61
1.46 > 5.15
3.3 V 6.05
0.07
−0.76
−0.9
−0.78
0.22
9.4
−0.15
0.01 < −0.84
−1
−1
−1
−1
10
−0.01
x
3
2
1
1
2
3
4
5
6
y
10.0
7.5
5.0
2.5


Q−table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.4)
0.05
0.1
0.22
0.49
1.06
1.71
0.04 > 0.06
0.05 > 0.14
0.12 > 0.27
0.28 > 0.69
0.71 > 1.52
1.85 V 2.39
0.05
0.1
0.22
0.56
1.27
3.21
0.04
0.07
0.18
0.47
2.17
2.49
0.02 ^ 0.03
0.11 ^ 0.11
0.12 ^ 0.12
0.25 > 0.78
0.08 ^ 1.27
2.68 V 4.86
0.02
−0.58
−0.54
−0.4
0.1
8.92
−0.24
0.01 < −0.71
−1
−1
−1
−1
10
−0.11
x
y
1
2
3
1
2
3
4
5
6
10.0
7.5
5.0
2.5



Q: Investigate how the β parameter affects the learned policy. A: Having a higher β worsen the model. This is due to it becoming more difficult for the agent to understand the task. Choosing an optimal action can lead to a negative reward.

3.6

Environment D: - Evaluate the agent's performance after training on eight goal positions and validating on the remaining eight. Answer questions regarding the learned policy's effectiveness.

https://github.com/STIMALiU/AdvMLCourse/blob/master/ReinforcementLearning/RL_Lab2_Colab.ipynb

Q: Has the agent learned a good policy? Why / Why not ? A: Yes, it is performing well finding the optimal policy. It seems to have learnt a policy taking to account the goal target.

Q: Could you have used the Q-learning algorithm to solve this task ? A: In theory yes, if it got to explore all combinations of states and goals, but Q-learning does not generalize.

2.7

Environment E: - Analyze the agent's policy when trained on goals from the top row and validated on goals from the lower rows. Compare the results with Environment D.

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.

Q: Has the agent learned a good policy? Why / Why not ? A: No, the agent tends to want to move upward, since that's what had the best return during training.

Q: If the results obtained for environments D and E differ, explain why. A: The agent in E is only trained on goals to the top, causing its policy to perform bad for goals at the bottom.