

lab3

Andreas Hertin

2021-10-03

```
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.

  maxVal <- which(q_table[x, y, ] == max(q_table[x, y, ]))
  if(length(maxVal) > 1){
    return(sample(maxVal, 1))
  } else {
    return(maxVal)
  }
}

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.

  randomDraw <- runif(1, 0, 1)
  maxVal <- GreedyPolicy(x, y)
  if(randomDraw <= 1 - epsilon){
    return(maxVal)
  } else {
    return(sample(1:4,1))
  }
}

```

```

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

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

  curr_state <- start_state
  episode_correction <- 0

  repeat{
    # Follow policy, execute action, get reward.
    x <- curr_state[1]
    y <- curr_state[2]

```

```

    action <- EpsilonGreedyPolicy(x, y, epsilon)

    next_state <- transition_model(x, y, action, beta)

    next_x <- next_state[1]
    next_y <- next_state[2]

    reward <- reward_map[next_x, next_y]

    # Q-table update.

    tmp_ep_diff <- reward + gamma * max(q_table[next_x, next_y, ]) - q_table[x, y, action]

    episode_correction <- episode_correction + tmp_ep_diff

    q_table[x, y, action] <<- q_table[x, y, action] + alpha * tmp_ep_diff

    if(reward!=0){
      # End episode.
      return (c(reward,episode_correction))
    }
    curr_state <- next_state
  }
}

```

Q-Learning Environments

Environment A (learning)

```

H <- 5
W <- 7

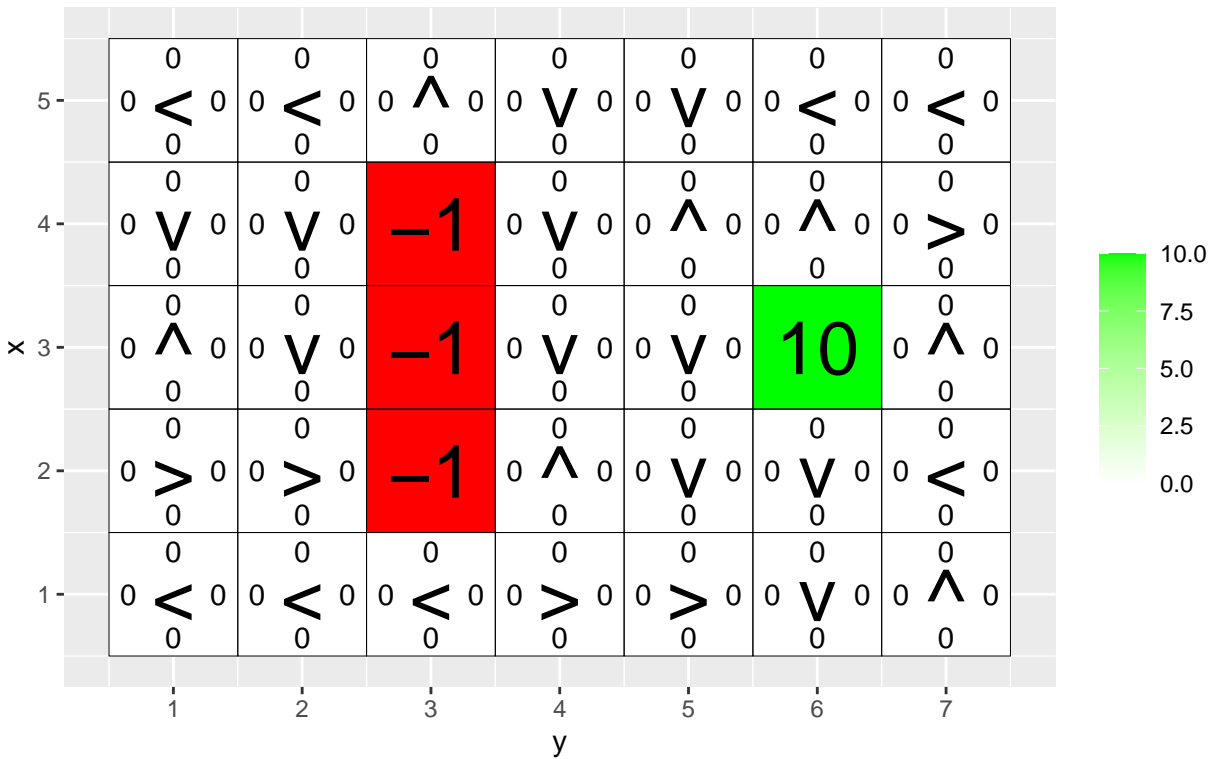
reward_map <- matrix(0, nrow = H, ncol = W)
reward_map[3,6] <- 10
reward_map[2:4,3] <- -1

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

vis_environment()

```

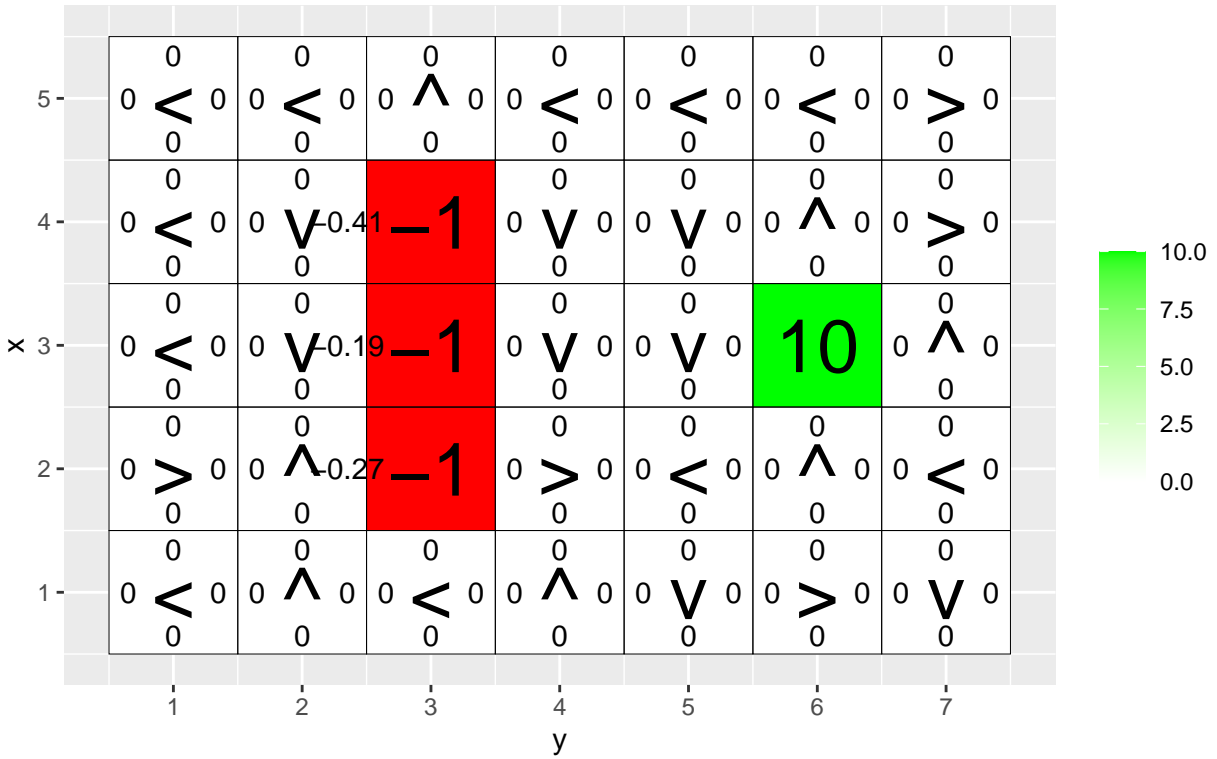
Q-table after 0 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)



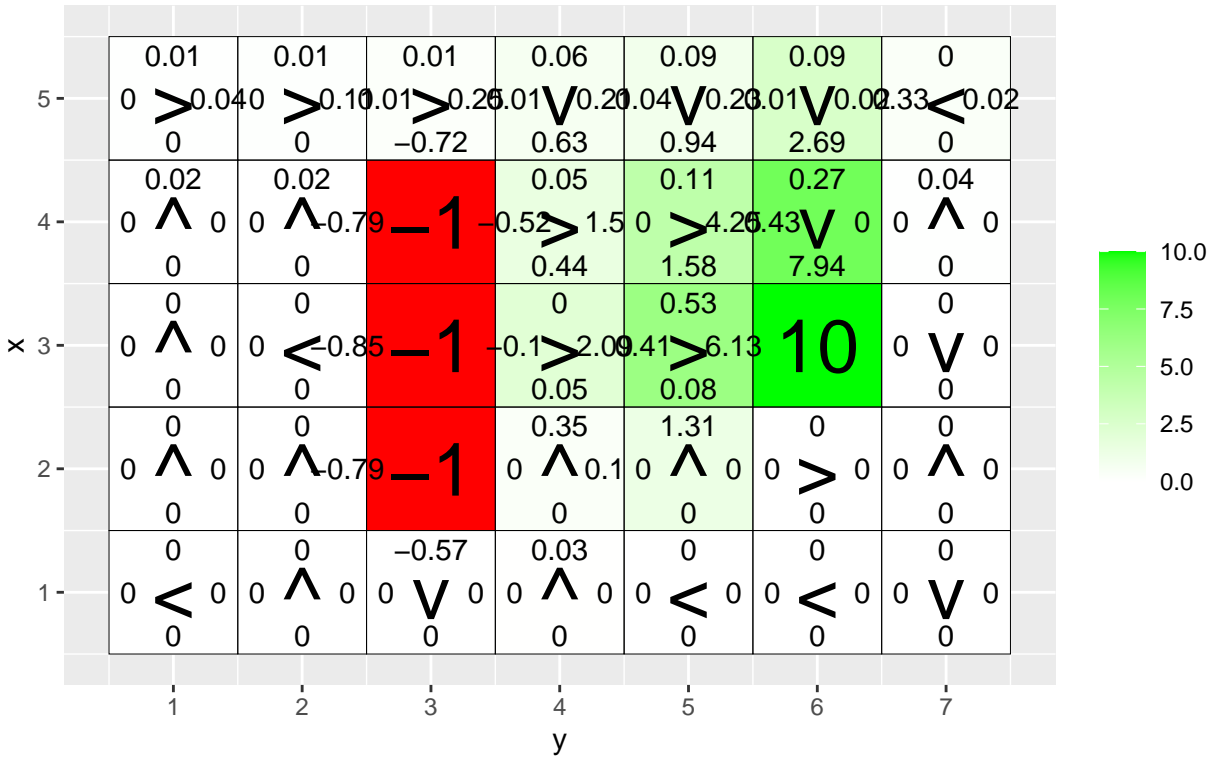
```
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))

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

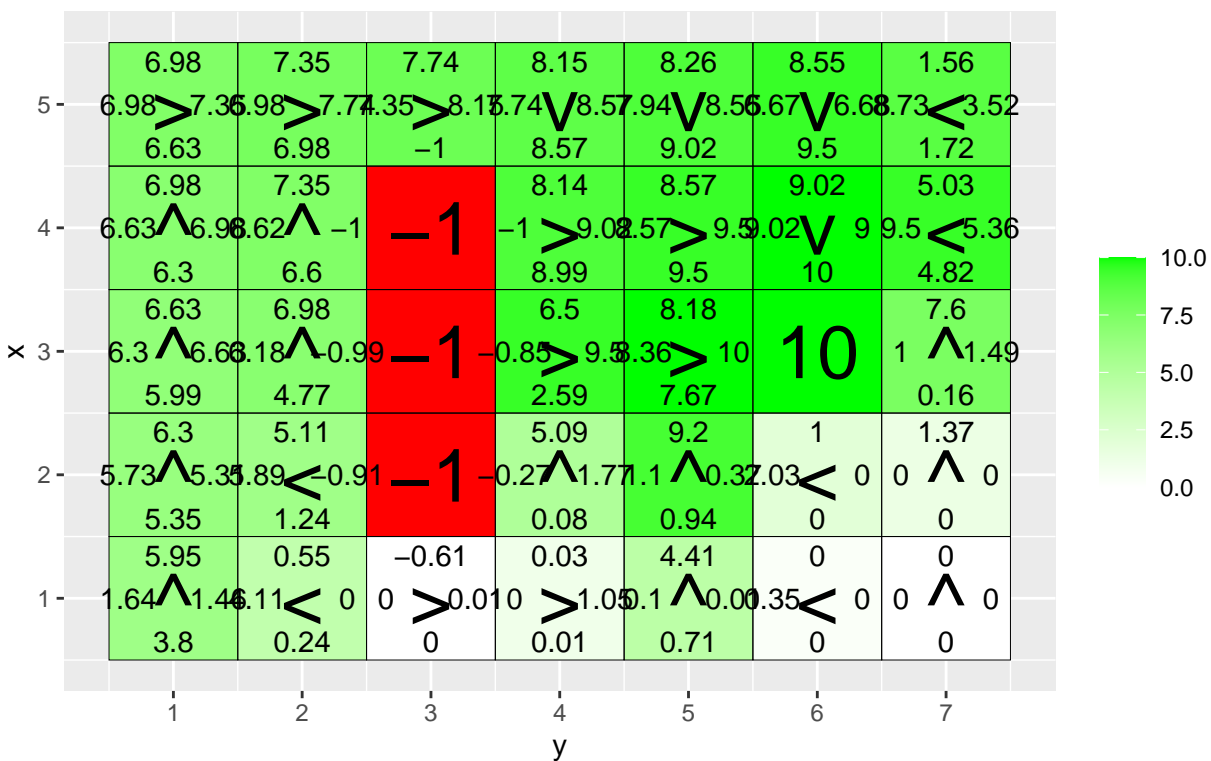
Q-table after 10 iterations
 (epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)

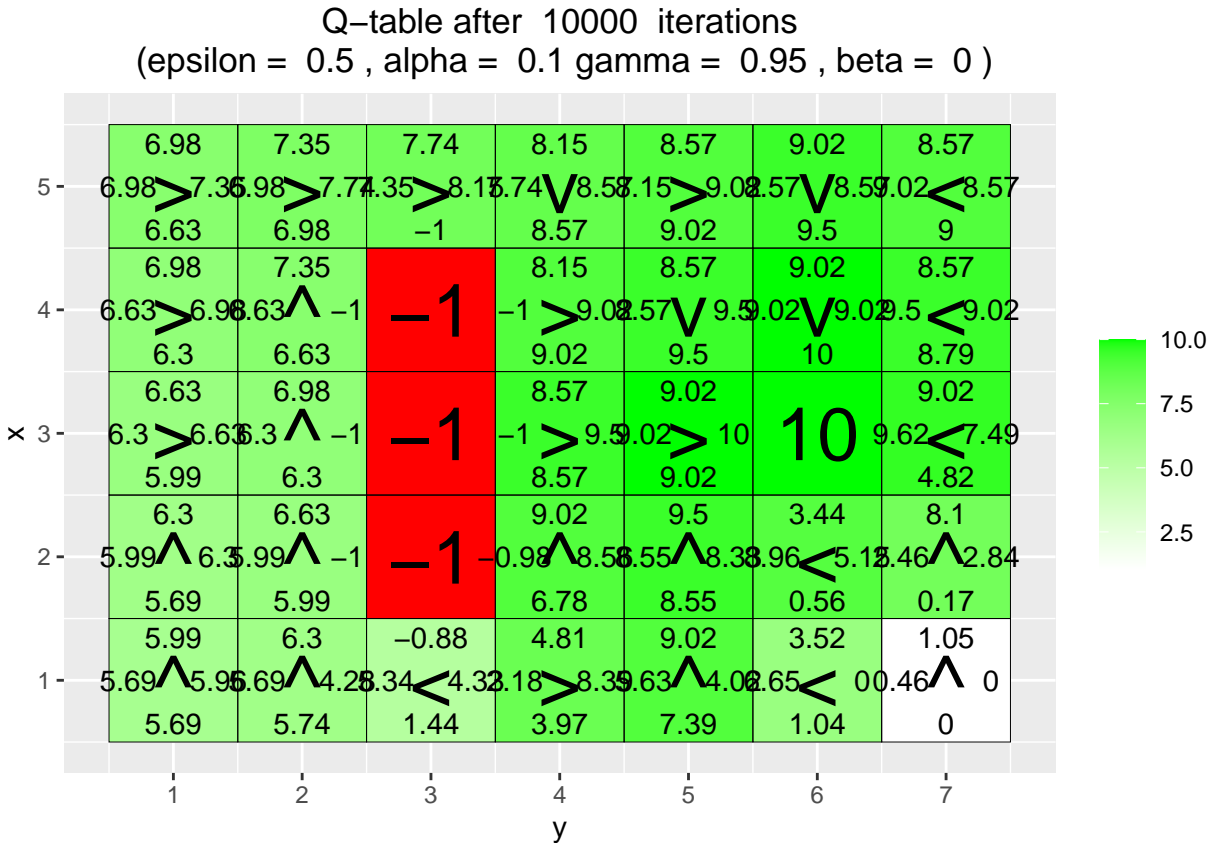


Q-table after 100 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)



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





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

Almost nothing, it has found that moving in to the -1 states are bad but it did not learn to move towards the +10 state.

- Is the final greedy policy (after 10000 episodes) optimal for all states, i.e. not only for the initial state ? Why / Why not ?

No it is not, an optimal policy would be to always move to the +10 state in the shortest possible way. We can see some states where this is not the case, for example in the corner where $x,y = 1,1$ and the neighboring states. The optimal route would be to go underneath the -1 block but it instead goes up and over.

- Do the learned values in the Q-table reflect the fact that there are multiple paths (above and below the negative rewards) to get to the positive reward ? If not, what could be done to make it happen ?

No, as stated in the previous question it does not go underneath the negative reward block. Doing more iterations could lead to it finding the path or increase the exploration a bit.

Environment B (the effect of epsilon and gamma)

```

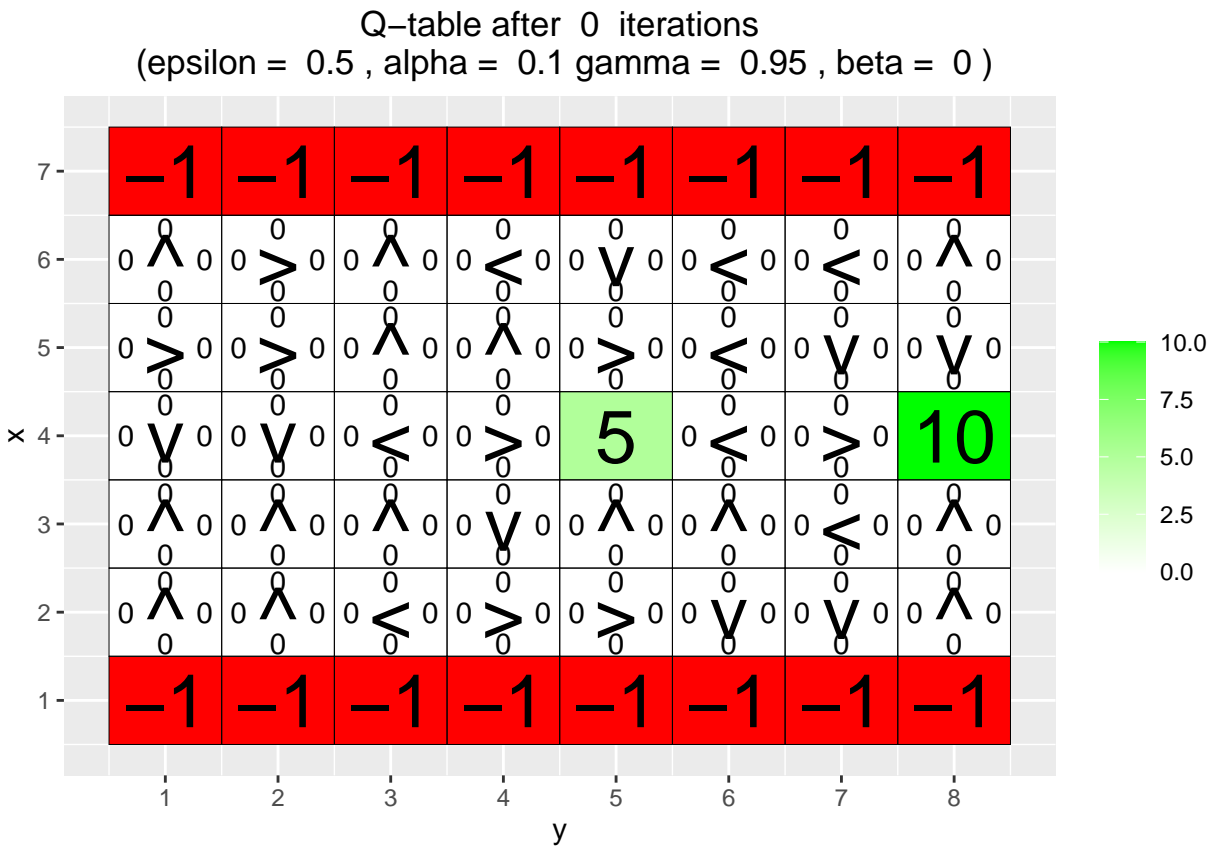
H <- 7
W <- 8

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

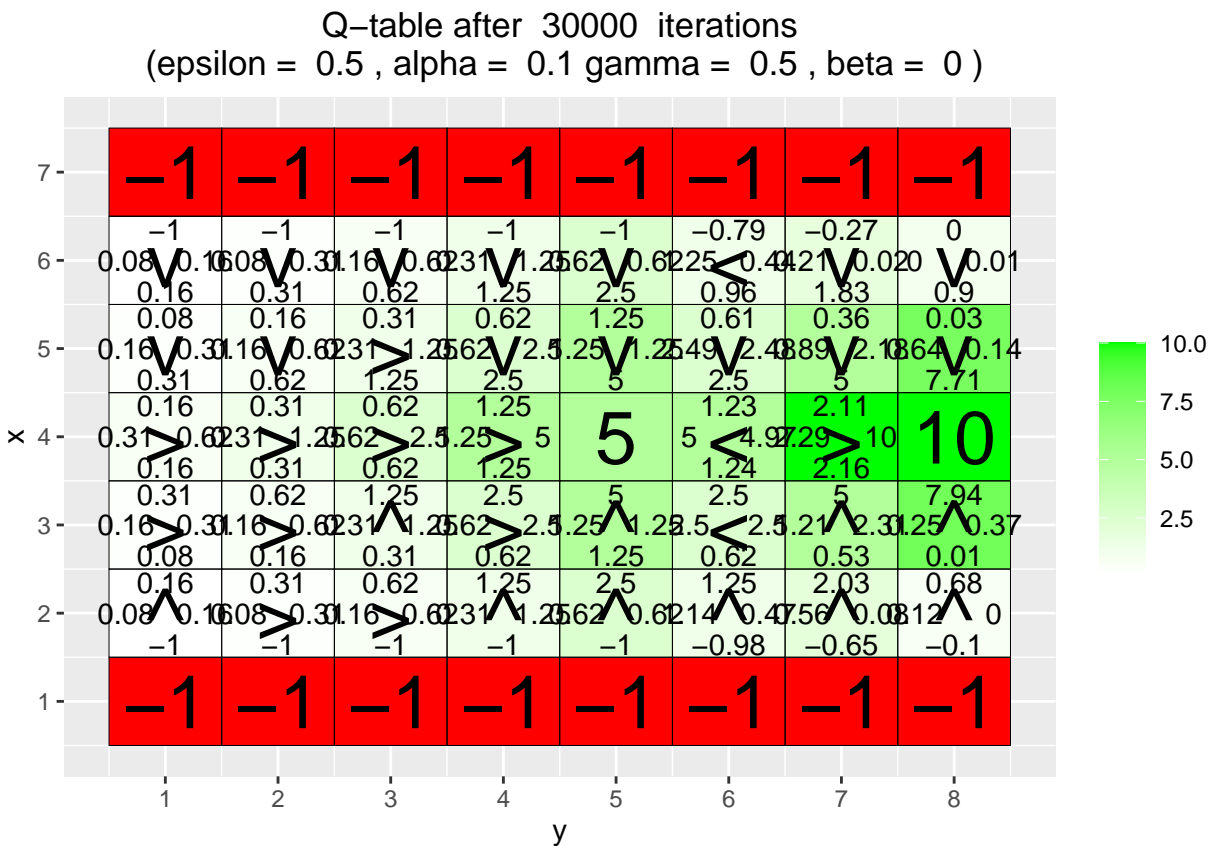
```

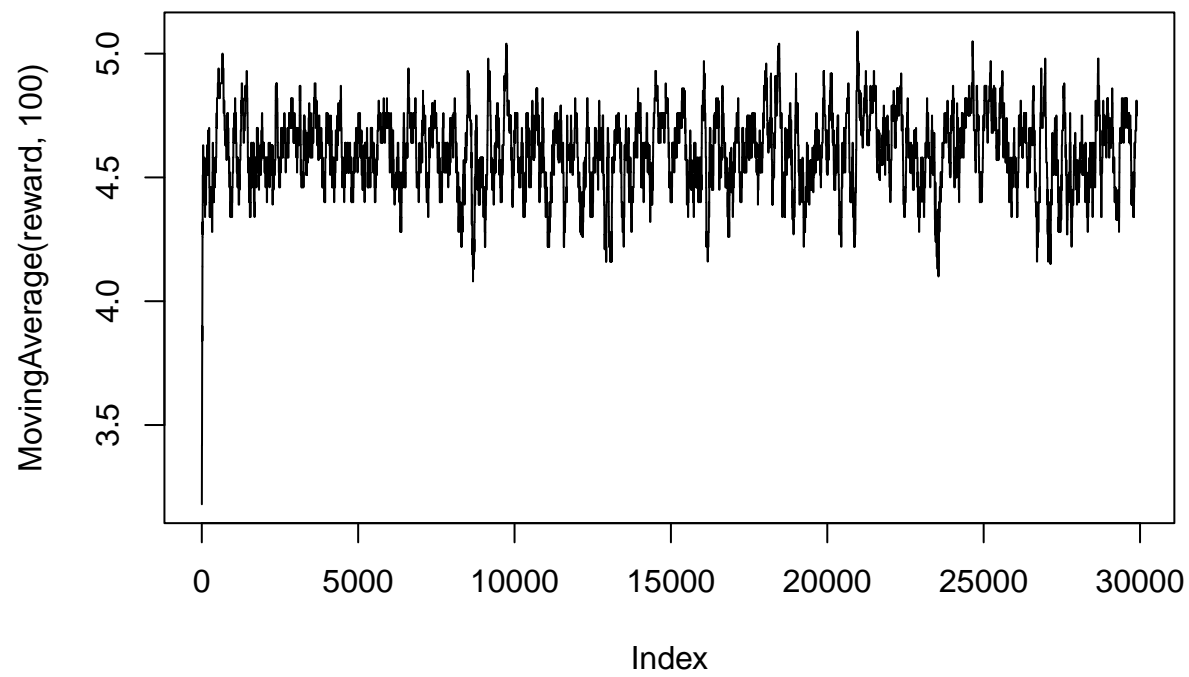
```

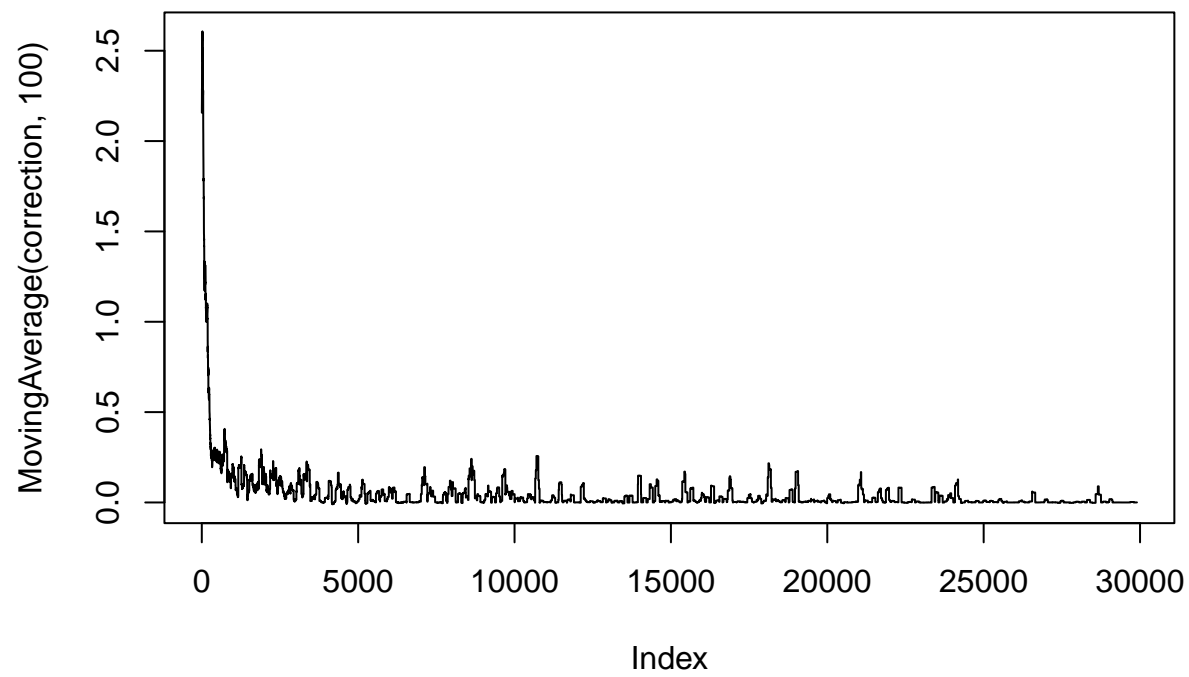
for(i in 1:30000){
  foo <- q_learning(gamma = j, start_state = c(4,1))
  reward <- c(reward,foo[1])
  correction <- c(correction,foo[2])
}

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

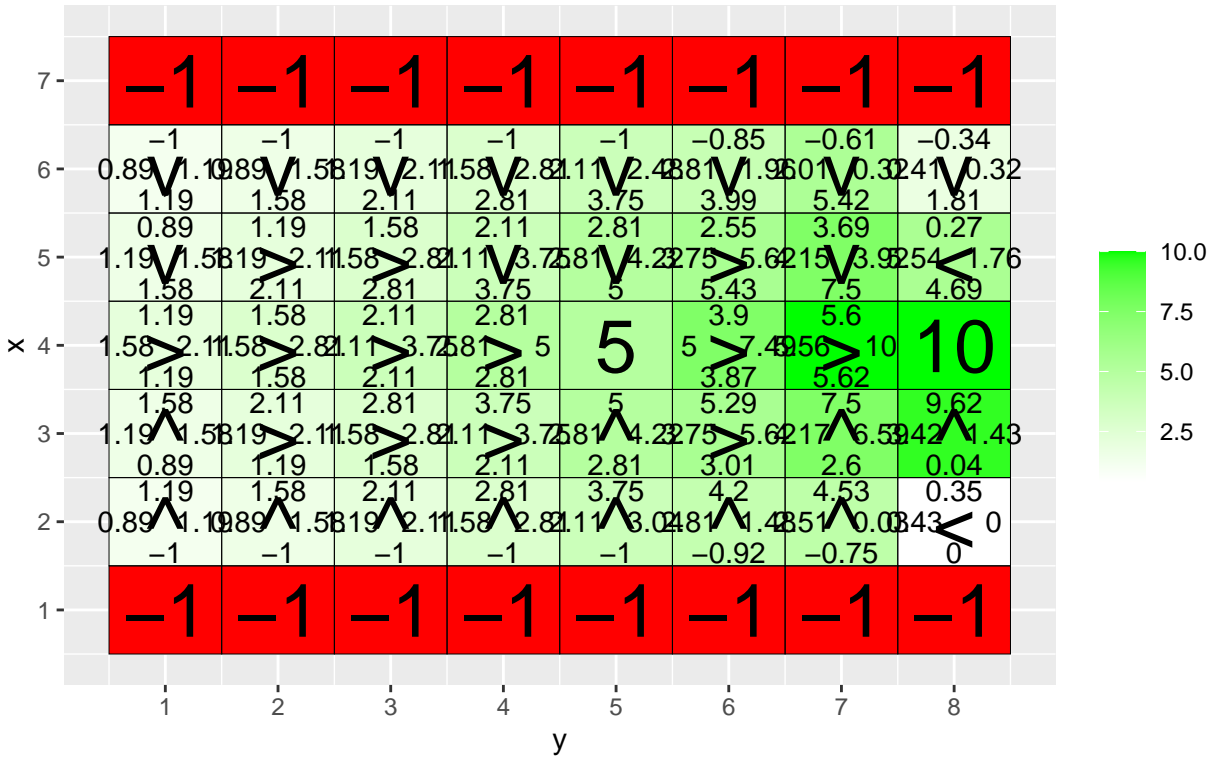
```

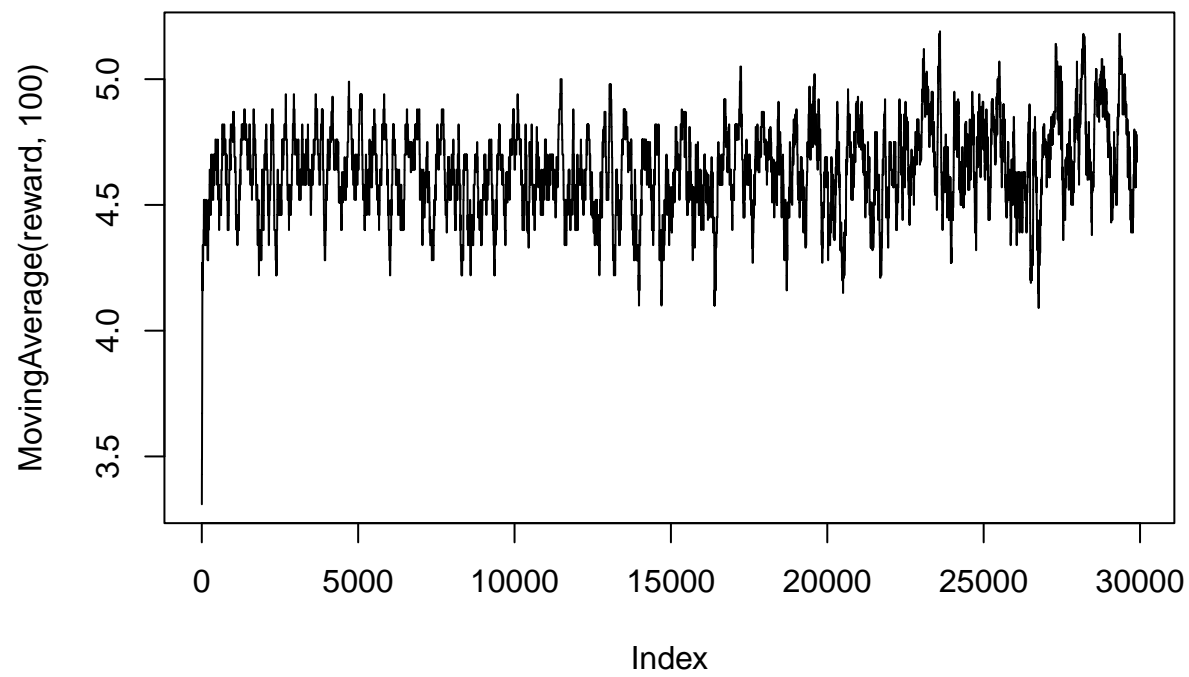


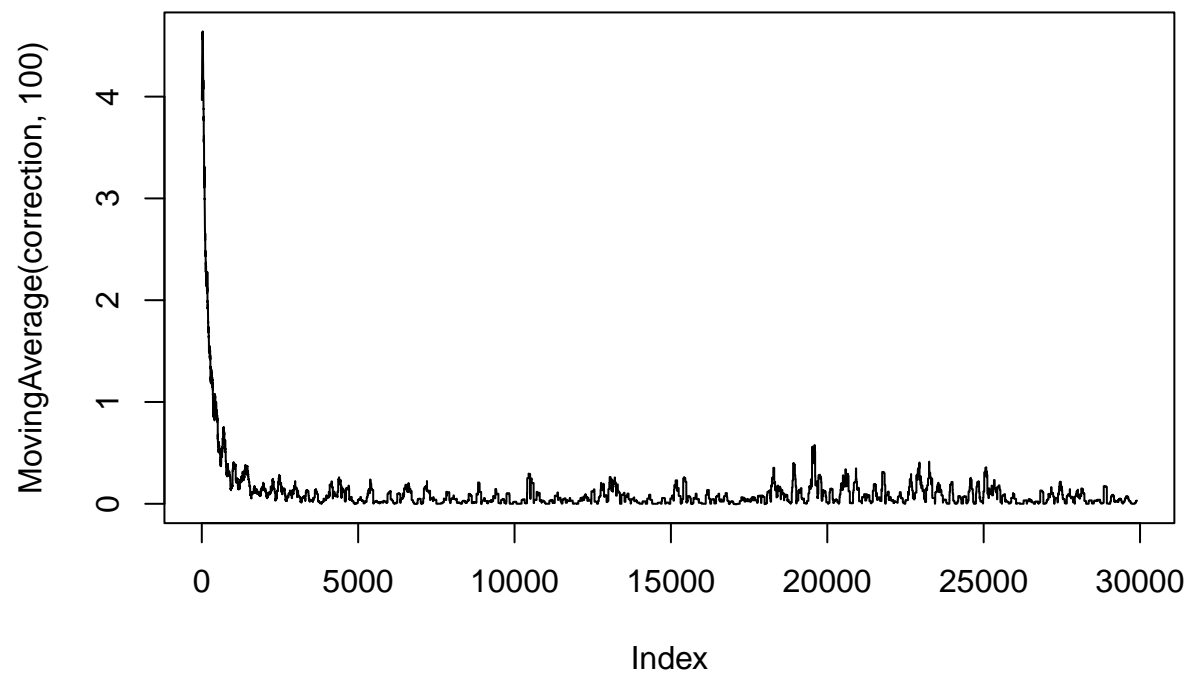




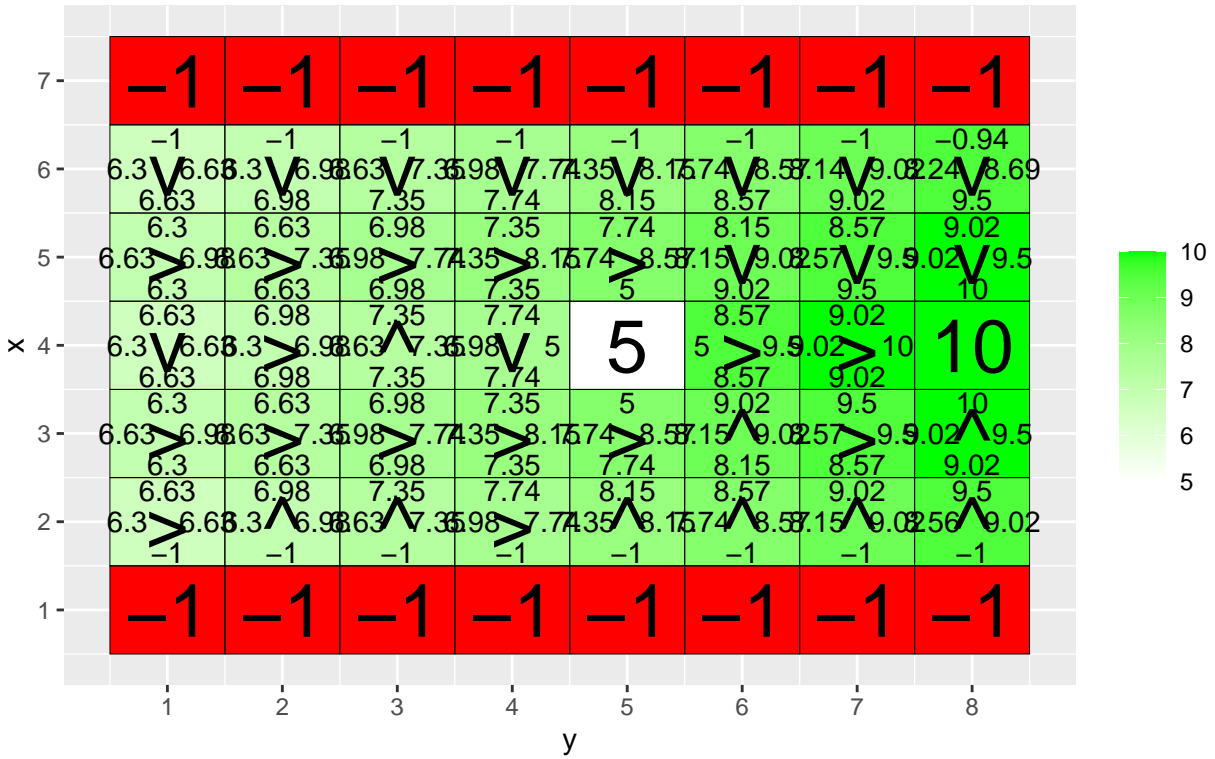
Q-table after 30000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.75 , beta = 0)

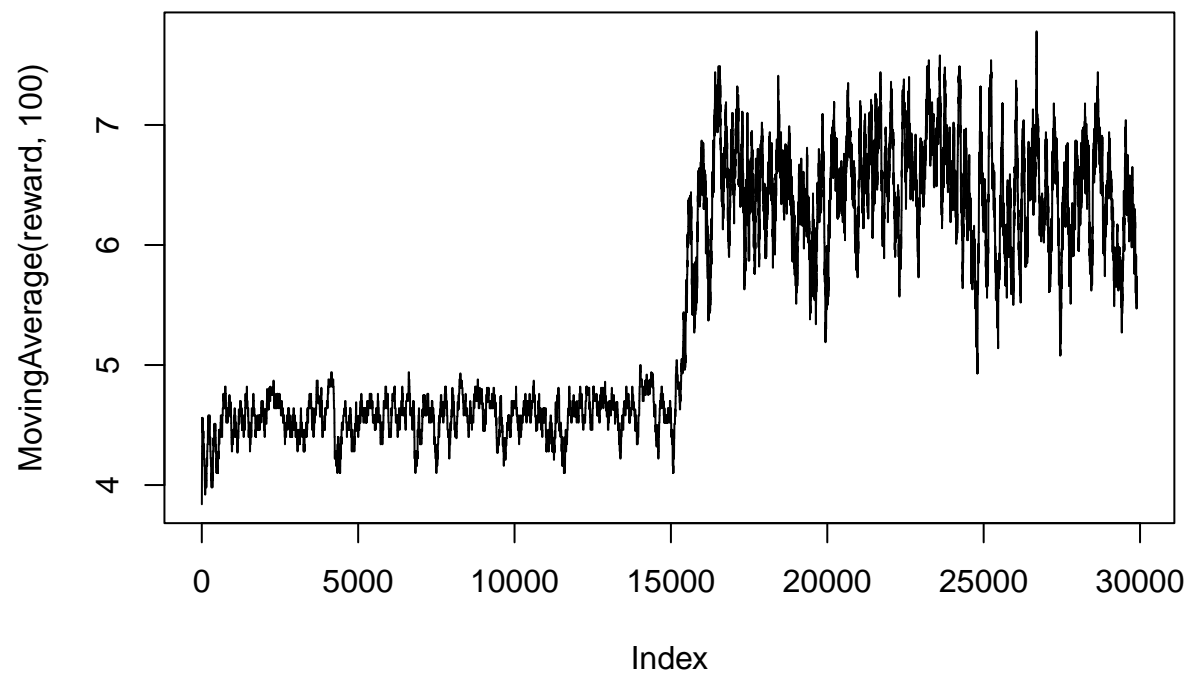


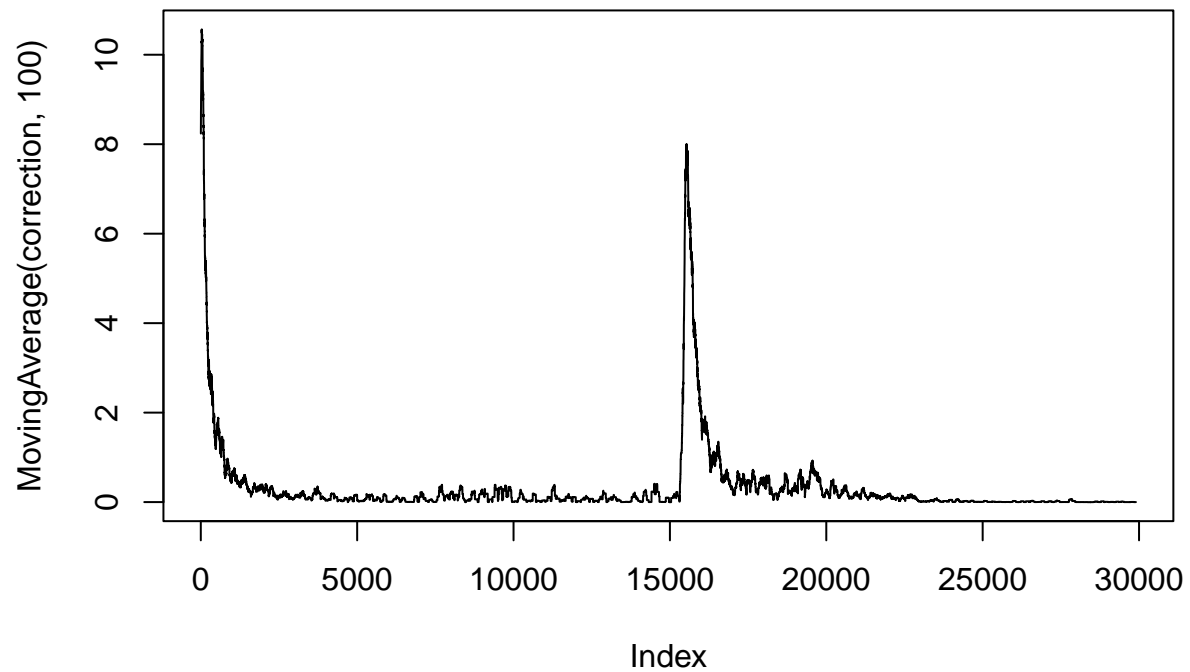




Q-table after 30000 iterations
 (epsilon = 0.5 , alpha = 0.1 gamma = 0.95 , beta = 0)





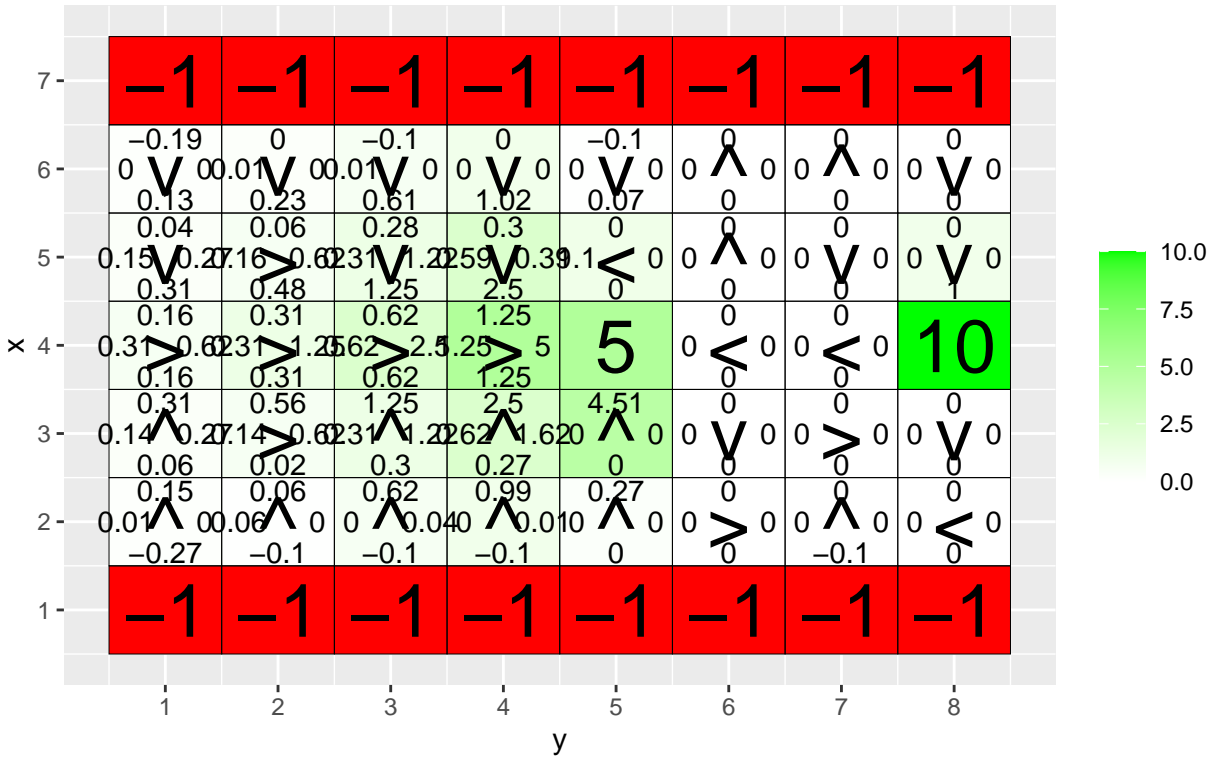


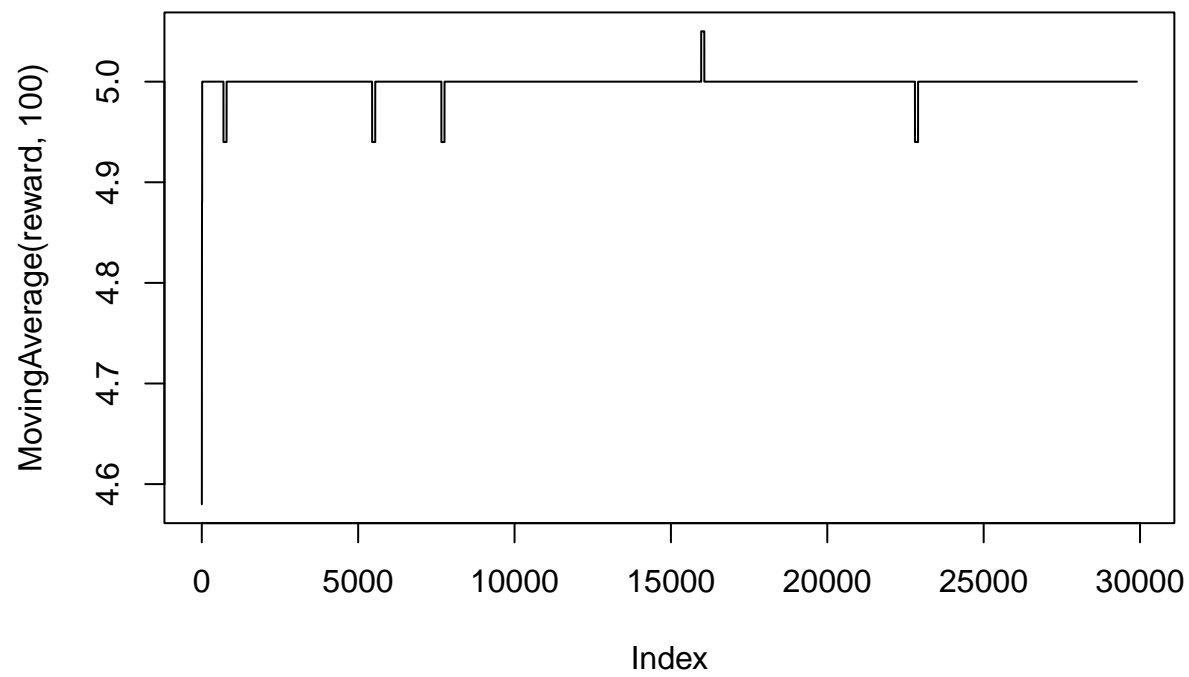
```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

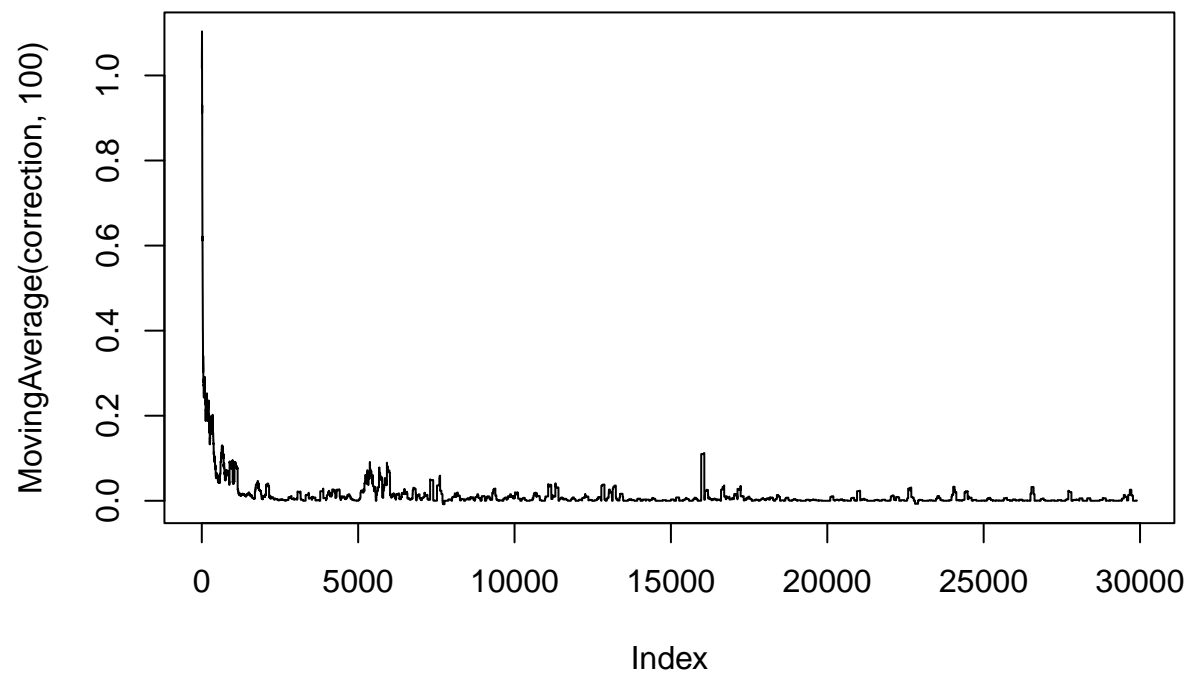
  for(i in 1:30000){
    foo <- q_learning(epsilon = 0.1, gamma = j, start_state = c(4,1))
    reward <- c(reward,foo[1])
    correction <- c(correction,foo[2])
  }

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

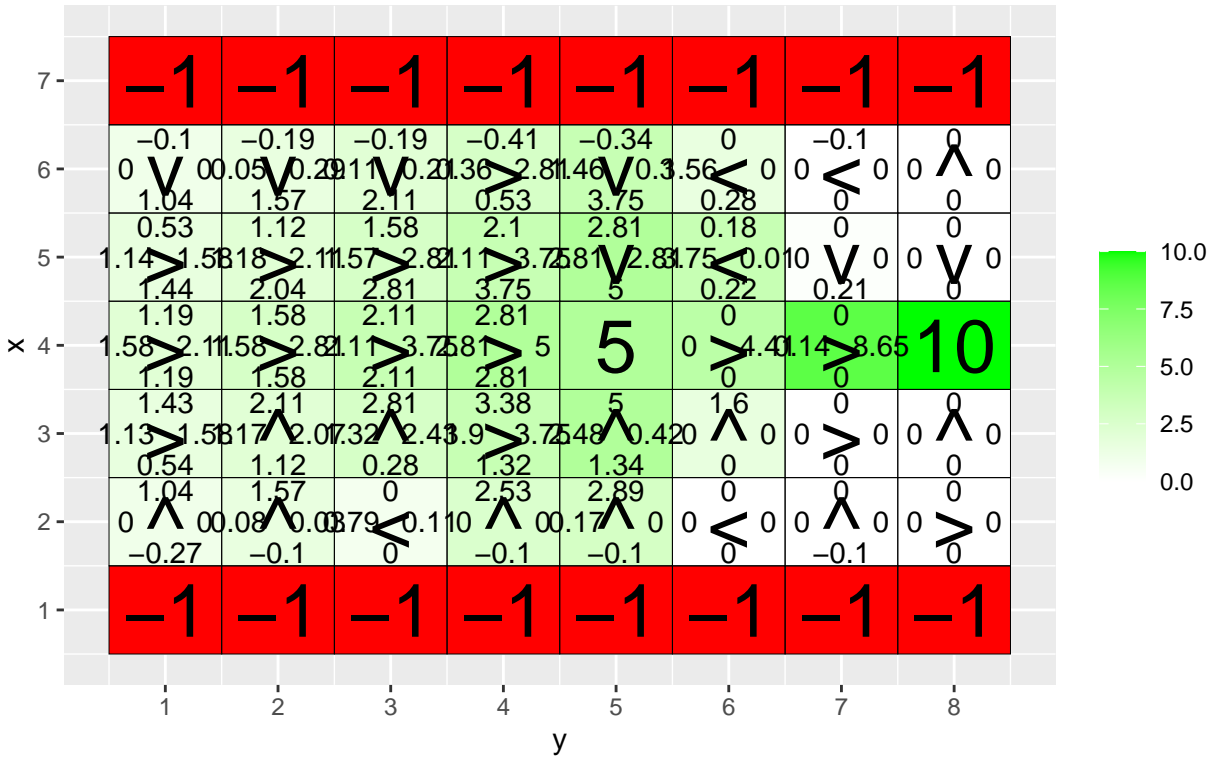
Q-table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.5 , beta = 0)

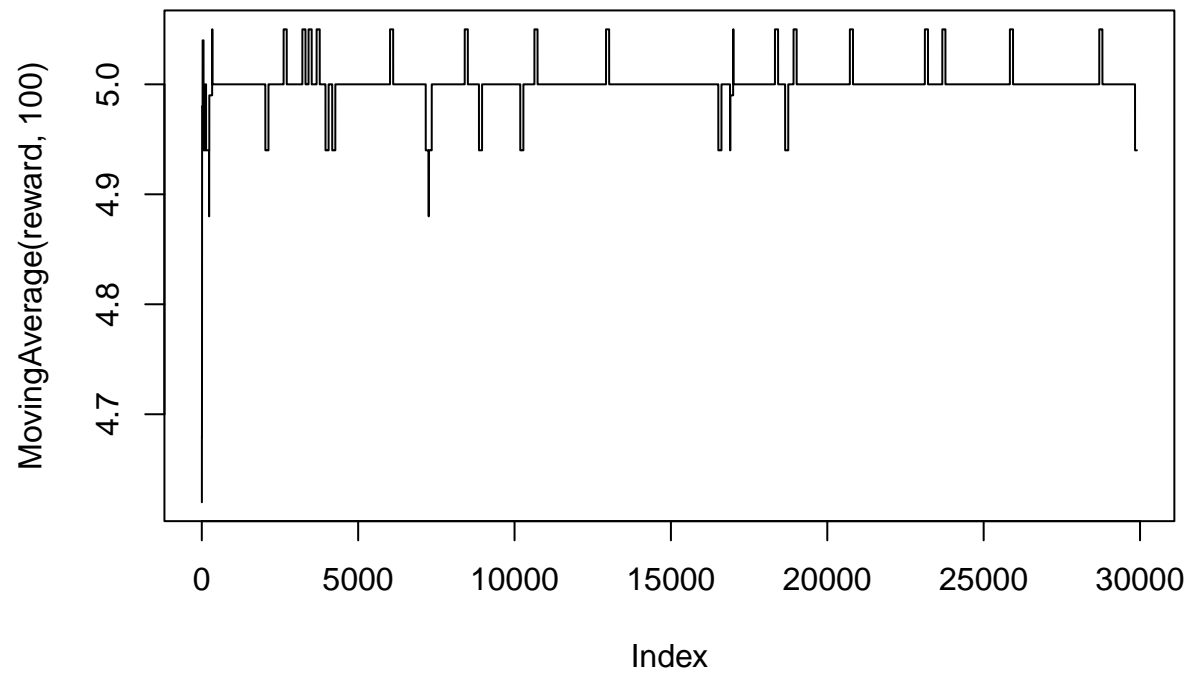


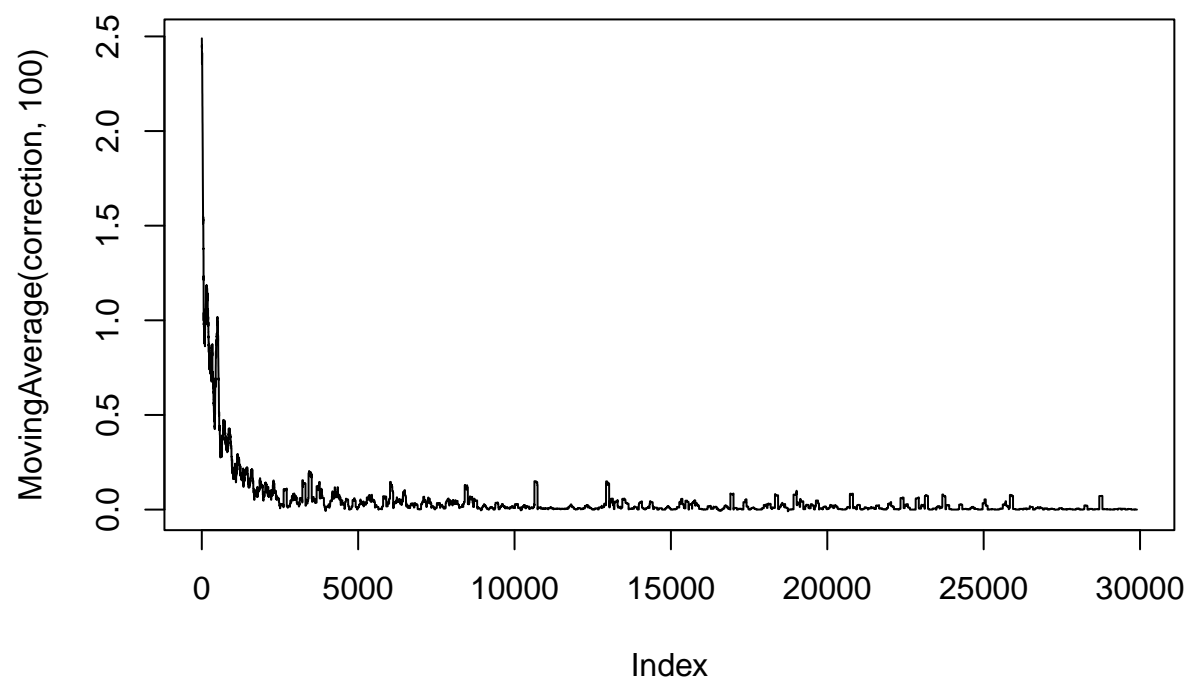




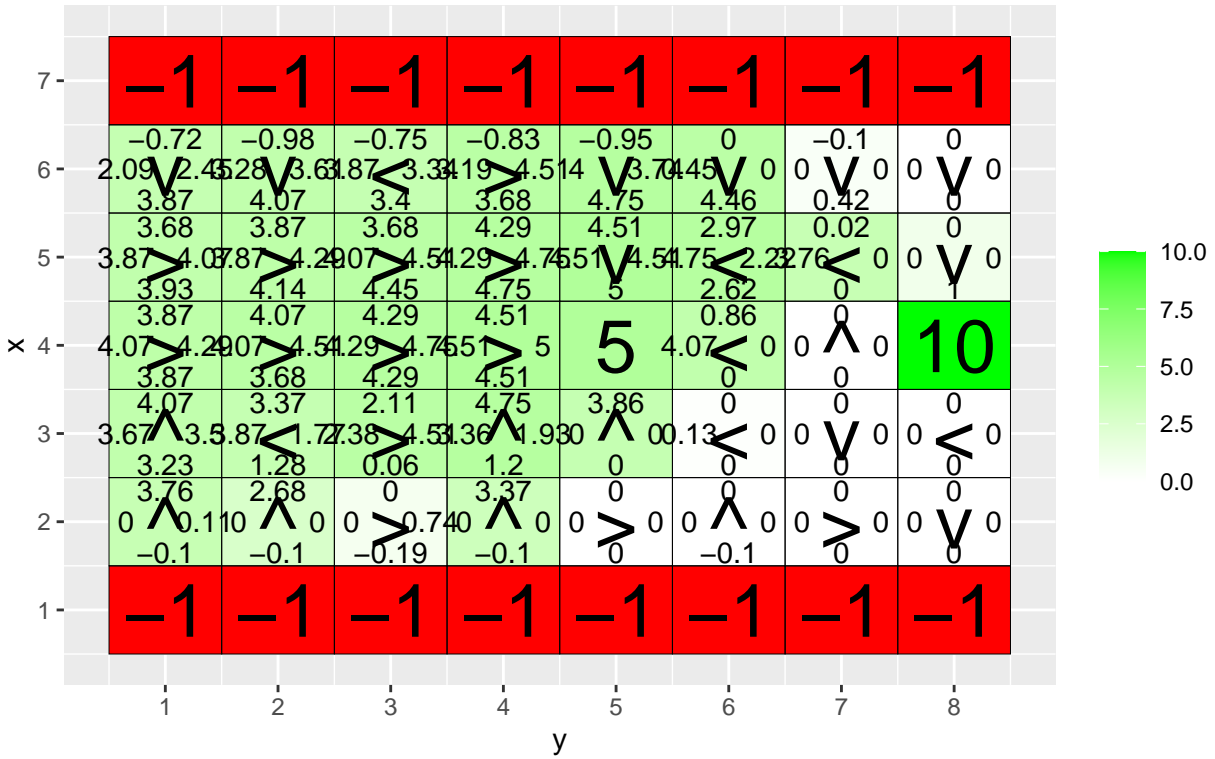
Q-table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.75 , beta = 0)

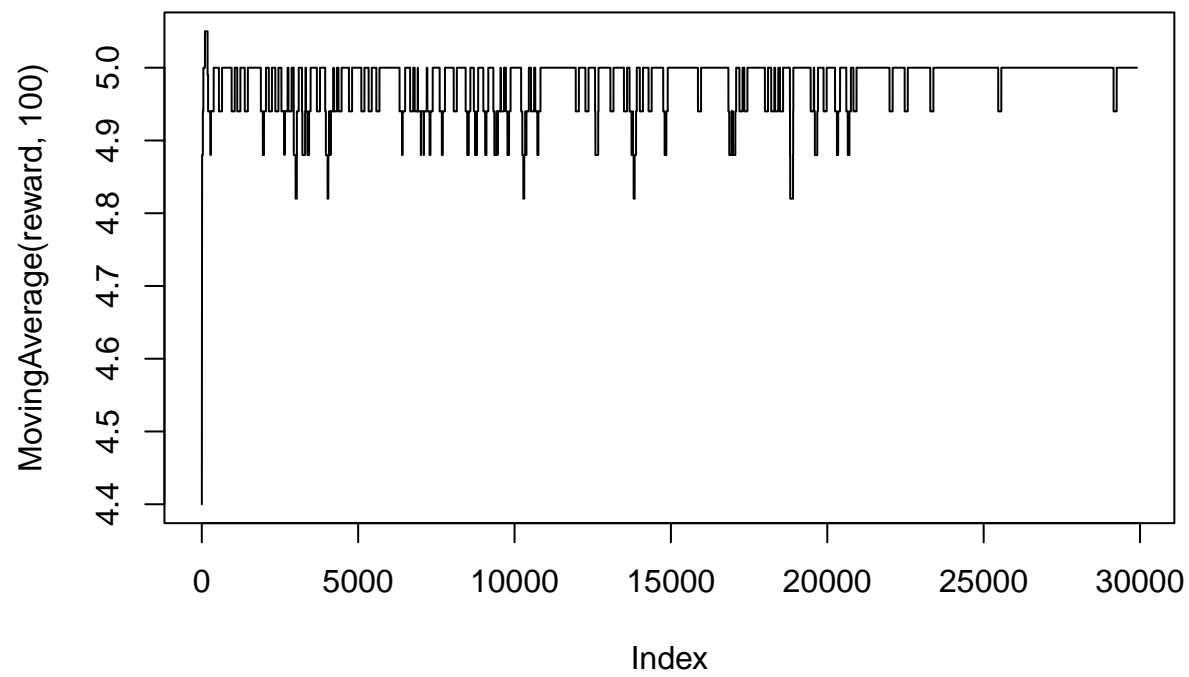


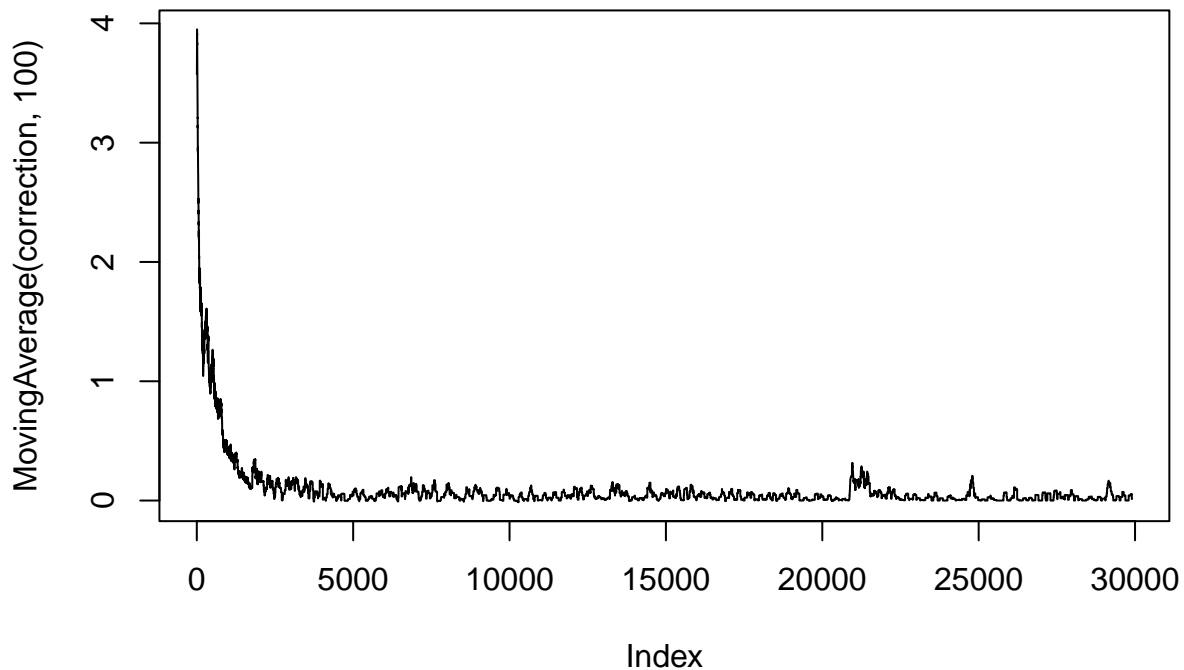




Q-table after 30000 iterations
(epsilon = 0.1 , alpha = 0.1 gamma = 0.95 , beta = 0)







When increasing the value for γ we increase the value of future rewards for the agent. With an increased ϵ we have increased chance of exploration, $\epsilon = 0.5$ would give a 50-50 chance of the agent exploring vs exploiting the current best reward.

We want a good mix of these two variables to not get stuck in states where rewards are “good enough” or take unnecessary paths.

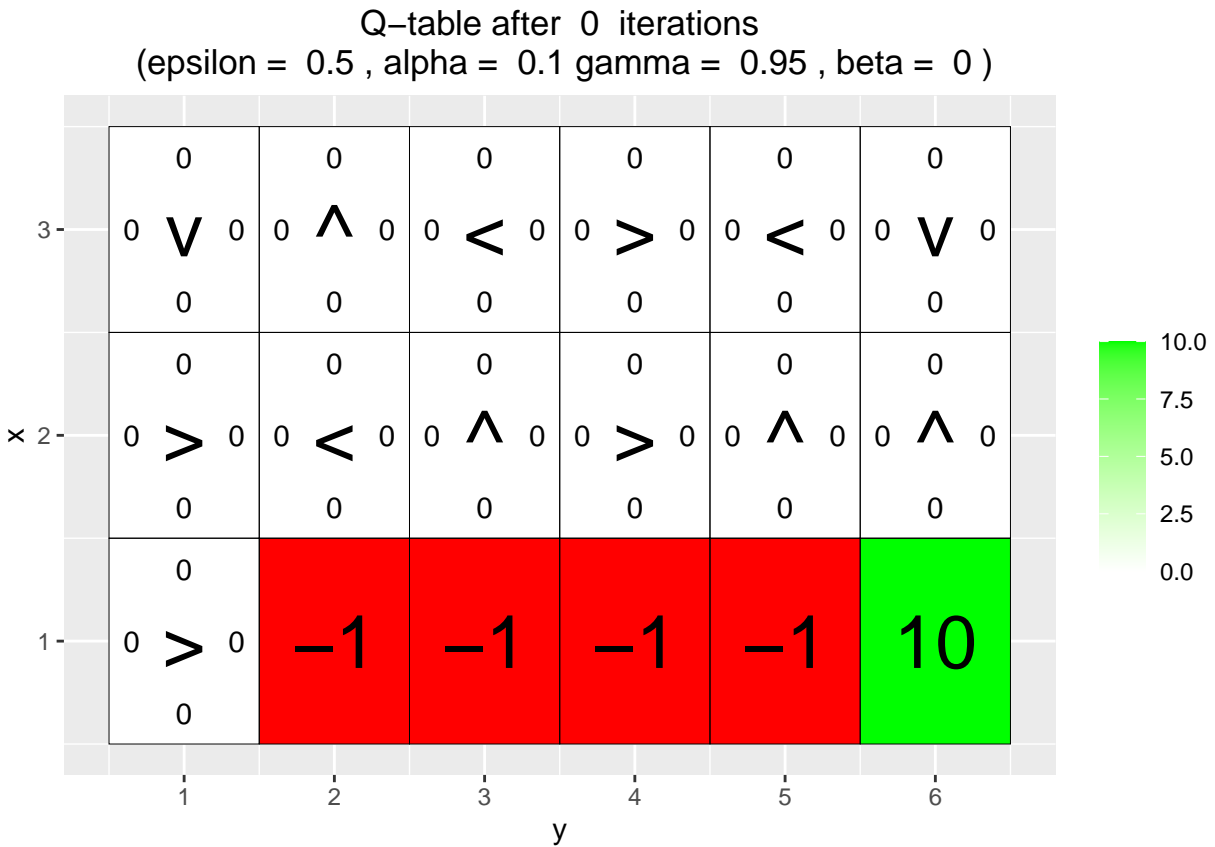
Environment C (the effect of beta).

```
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()
```

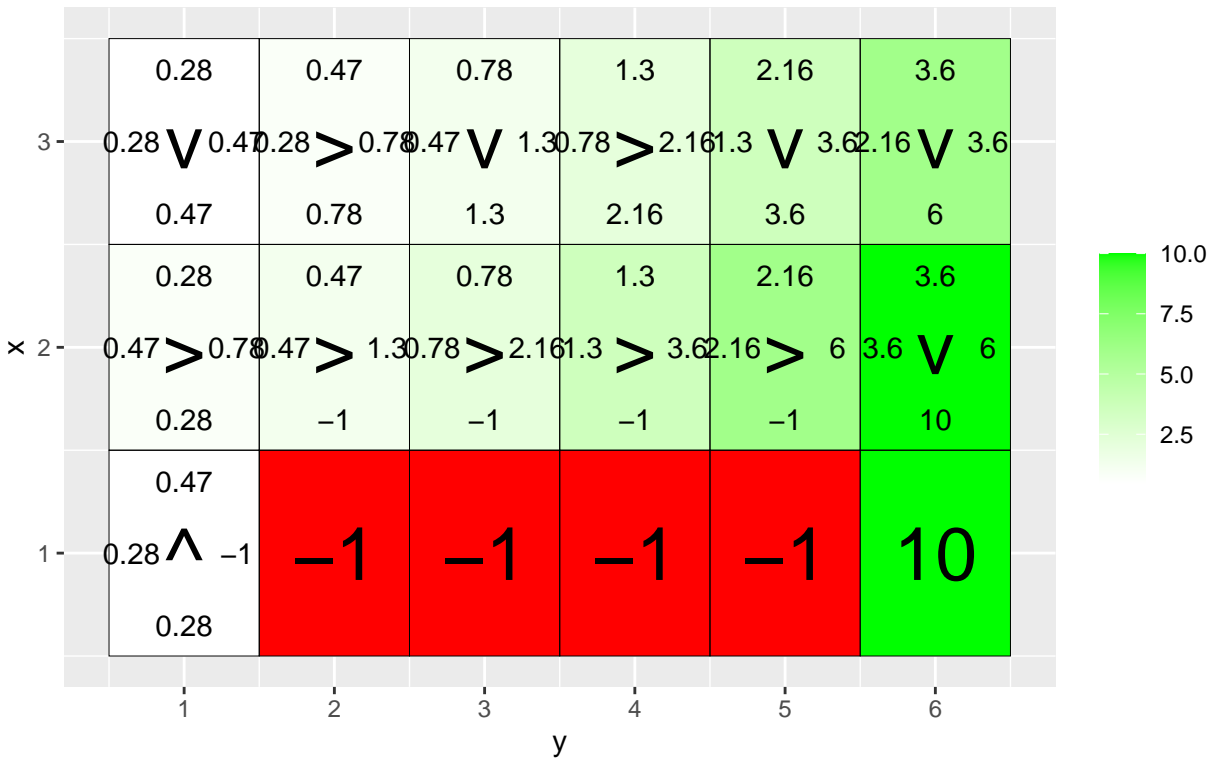


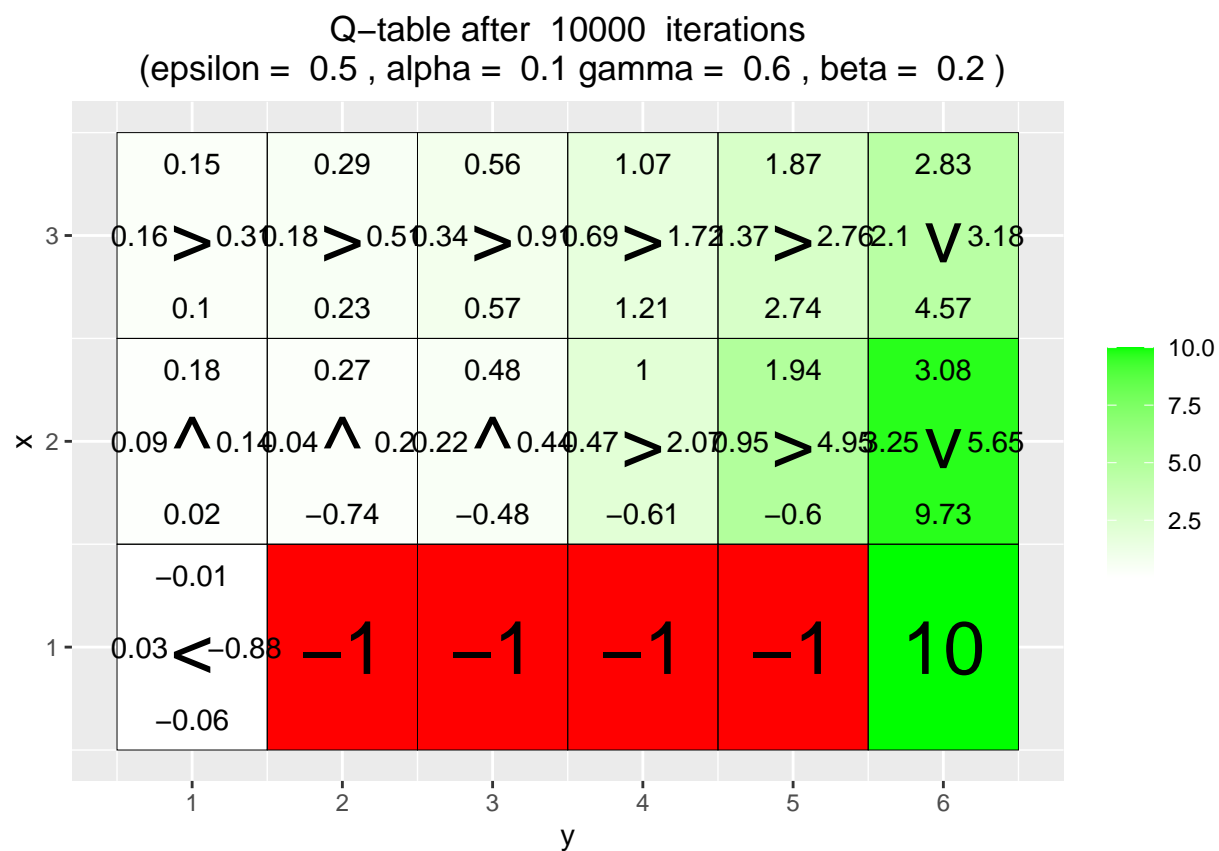
```
for(j in c(0,0.2,0.4,0.66)){
  q_table <- array(0,dim = c(H,W,4))

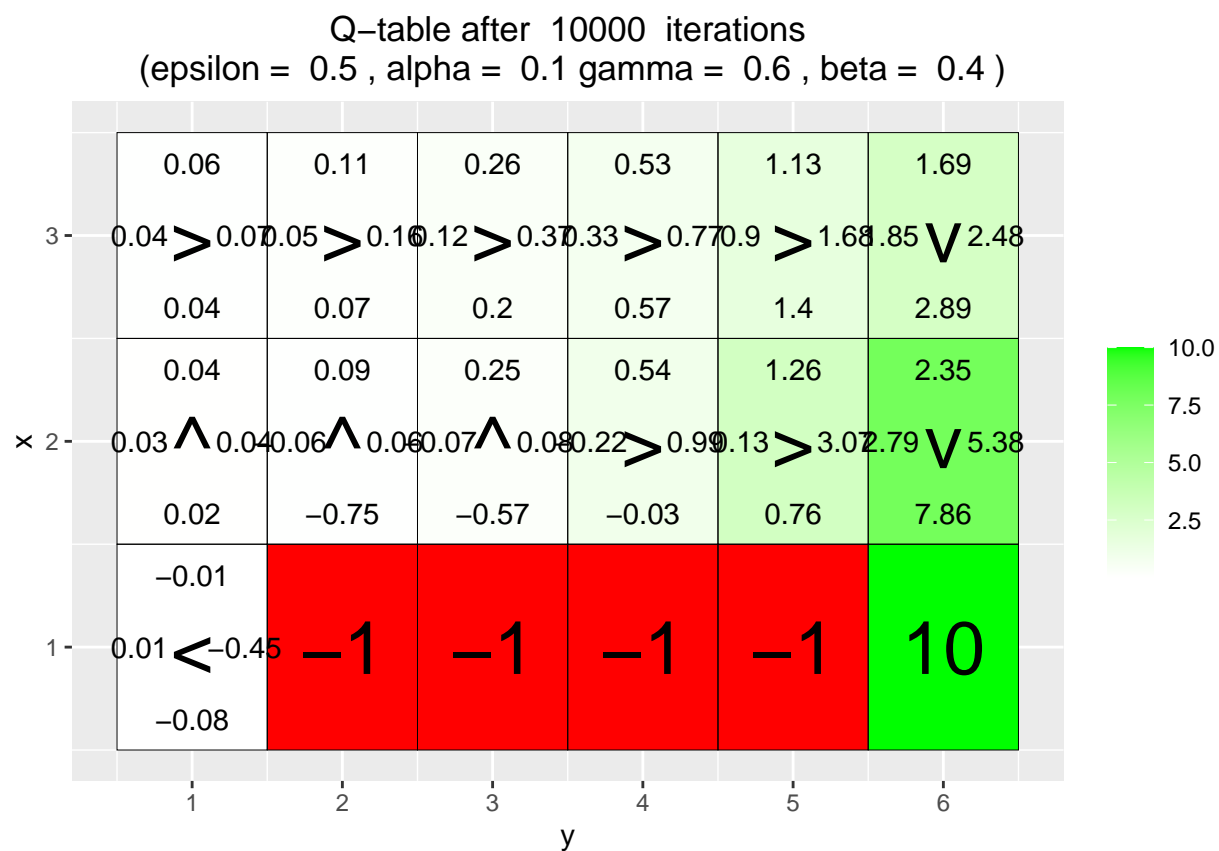
  for(i in 1:10000)
    foo <- q_learning(gamma = 0.6, beta = j, start_state = c(1,1))

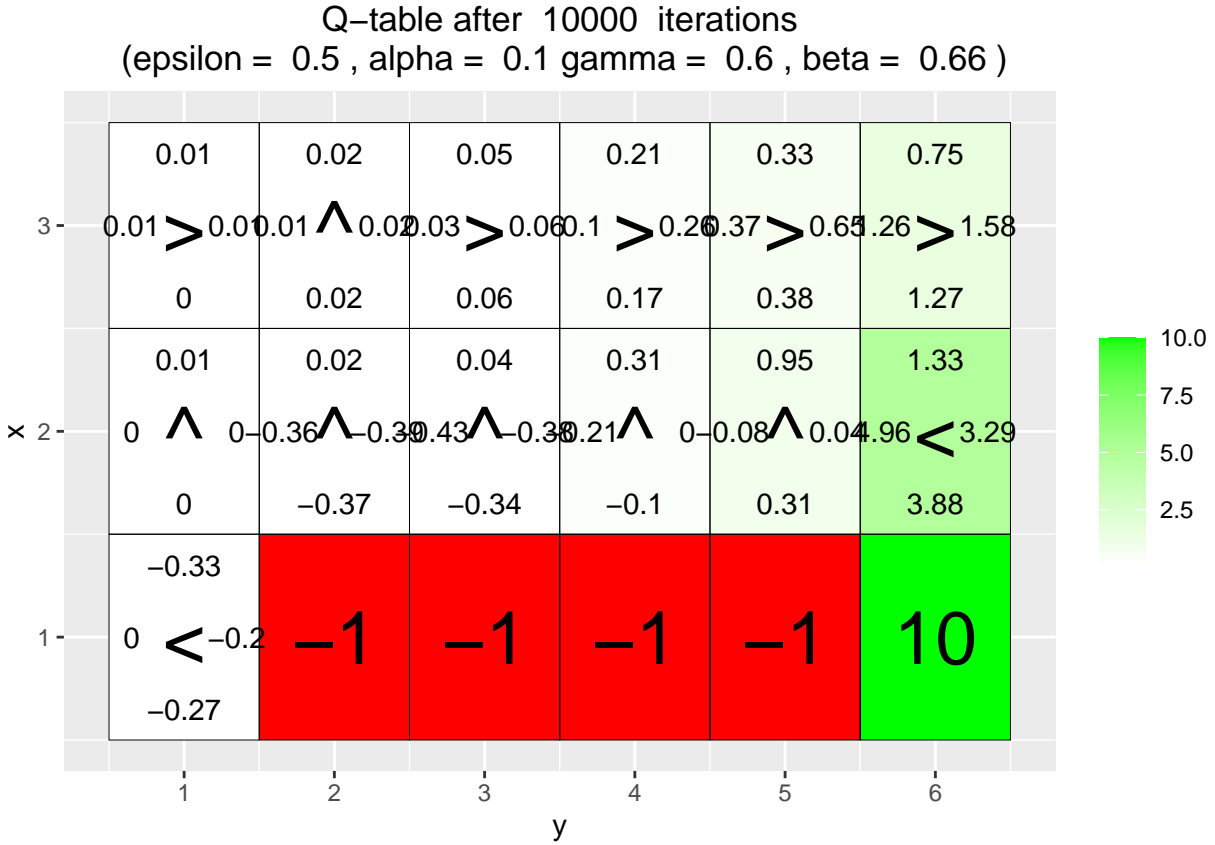
  vis_environment(i, gamma = 0.6, beta = j)
}
```

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









β is the slipping factor and it can make the agent slip left or right with equal probability from the state it wanted to go to. In the figures we can see that a higher β makes it less likely to slip into the negative rewards. It has done this by avoiding to go to the states close to the negative reward block