Advanced Machine Learning Lab 3

Simon Jorstedt, Marijn Jaarsma, Simge Cinar

2024-10-06

Statement of Contribution

We solved the assignment individually and compared our answers. We have similar answers for each question except for 2.6

Implementation

First let's define the necessary functions

```
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, ] # gives the values of 4 actions for a specific location
  #greedy_action <- which.max(q_values) # chooses the best action with maximum value
  #return(greedy_action)

  # Get the set of best actions, and sample one uniformly
  best_actions <- which(q_table[x,y,] == max(q_table[x,y,]))
  if (length(best_actions) == 1){return(best_actions)}
  else {return(sample(best_actions, 1))}
}
```

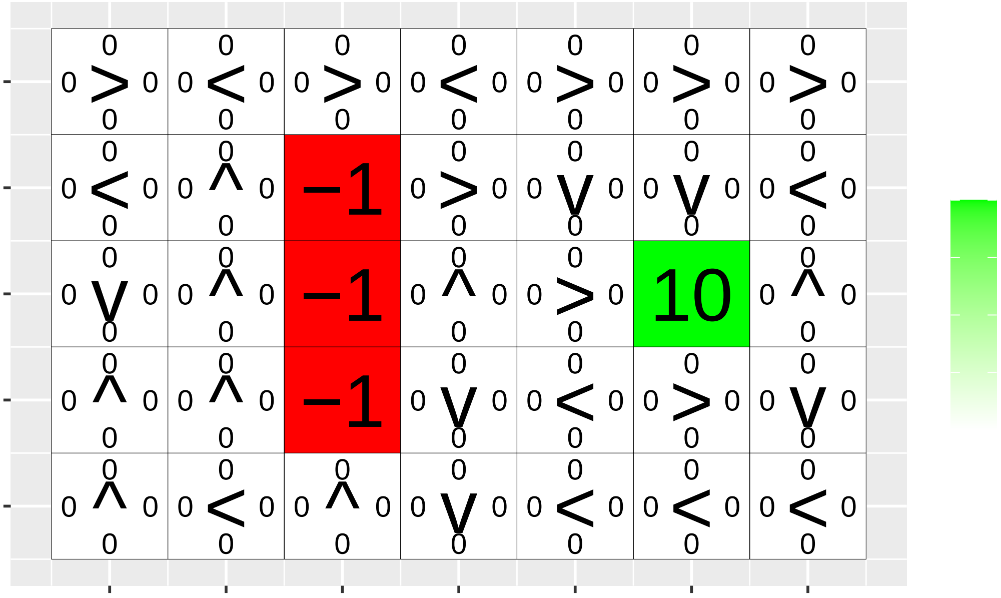
```
EpsilonGreedyPolicy <- function(x, y, epsilon){

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

  ## Your code here.
  ## explore with prob epsilon, greedy action wih prob 1-epsilon
  #rand_num <- runif(1)
  #action <- ifelse(rand_num < epsilon, sample(1:4, 1), GreedyPolicy(x,y))
  #return(action)

  # Take a random action
  if (runif(1) < epsilon){return(sample(1:4, 1))}

  # Take a greedy action
  return(GreedyPolicy(x,y))
}
```

```
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.

  # Initialize the variables
  state_x <- start_state[1]
  state_y <- start_state[2]
  episode_correction <- 0

  repeat{
    # Get the action
    action <- EpsilonGreedyPolicy(state_x, state_y, epsilon)

    # Get the new states after taking the action, goes to selection location with prob 1-beta
    new_state <- transition_model(state_x, state_y, action, beta)
    state_x_new <- new_state[1]
    state_y_new <- new_state[2]
```

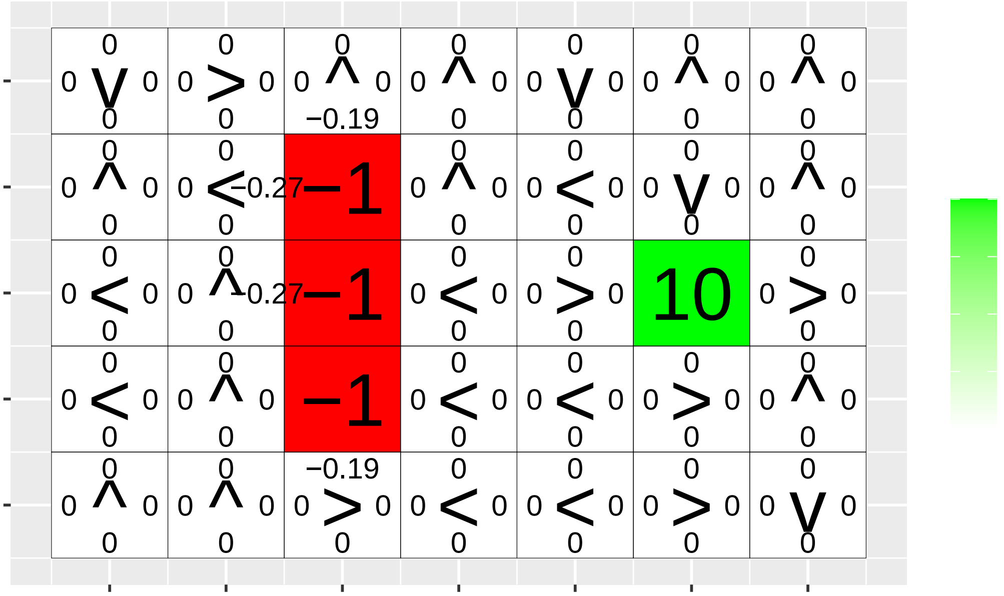
```r
    # Update the reward map
    reward <- reward_map[state_x_new, state_y_new]

    # Update Q-table
    step_correction <- reward + gamma * max(q_table[state_x_new, state_y_new, ]) - q_table[state_x, sta
    q_table[state_x, state_y, action] <<- q_table[state_x, state_y, action] + alpha * step_correction

    # Update total temporal difference correction
    episode_correction <- episode_correction + abs(step_correction)

    # Reset the state
    state_x <- state_x_new
    state_y <- state_y_new

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

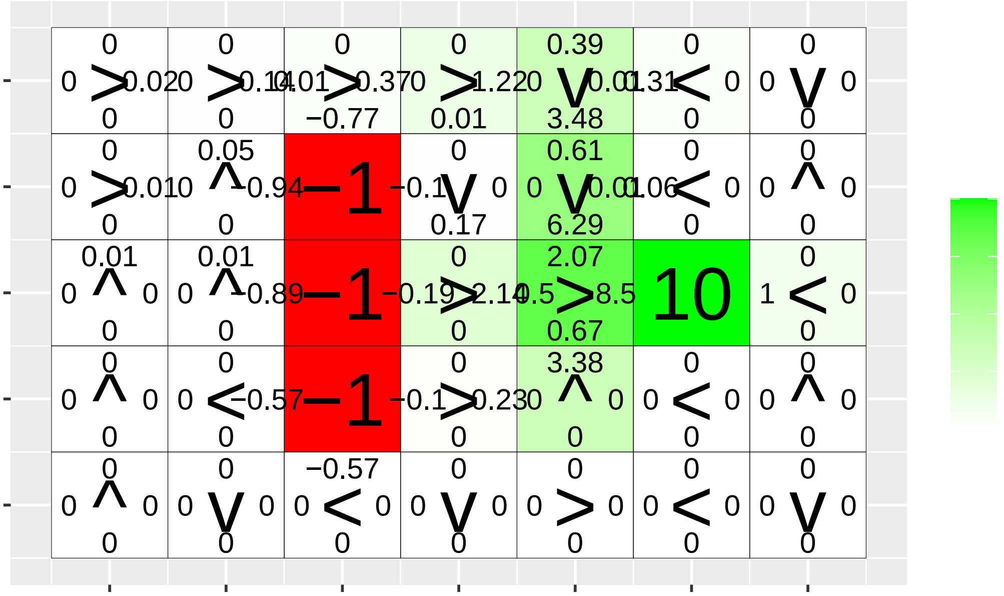
Question 2.2 - Environment A

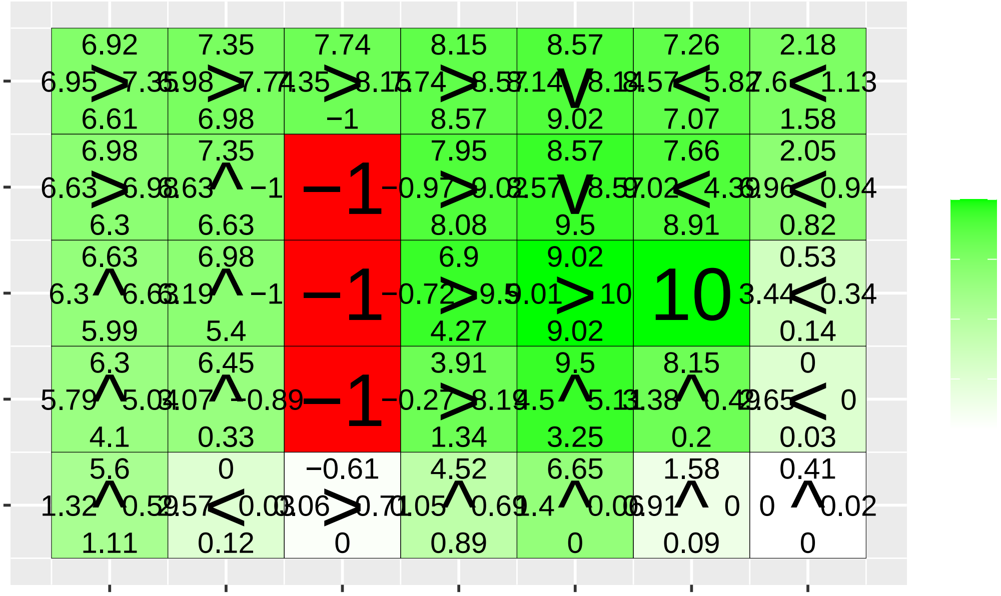
```r
H <- 5
W <- 7

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()
```

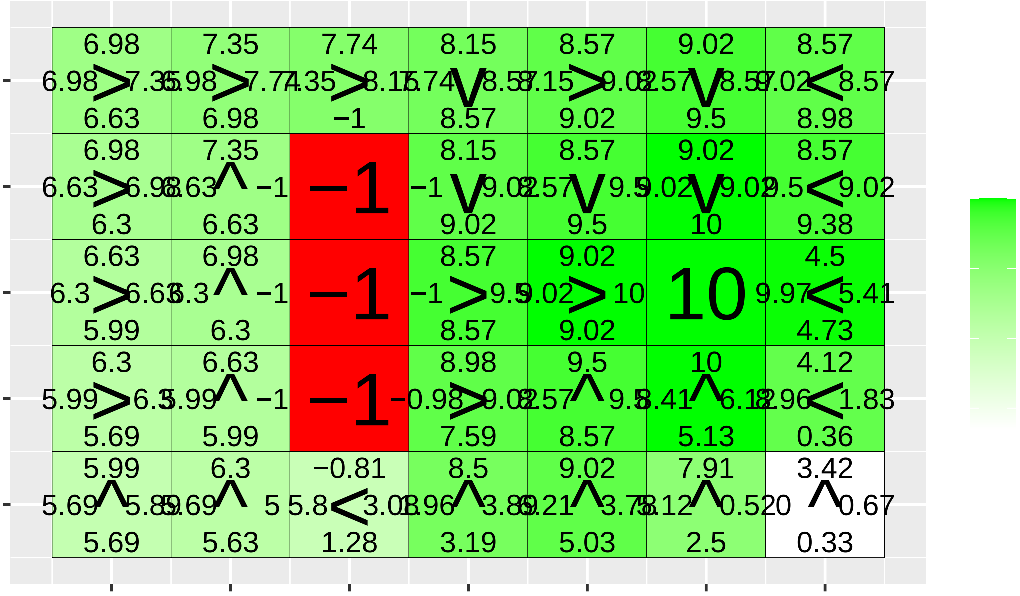


```r
for(i in 1:10000){
  foo <- q_learning(start_state = c(3,1))

  if(any(i==c(10,100,1000,10000)))
    vis_environment(i)
    cat("\n\n")
}
```







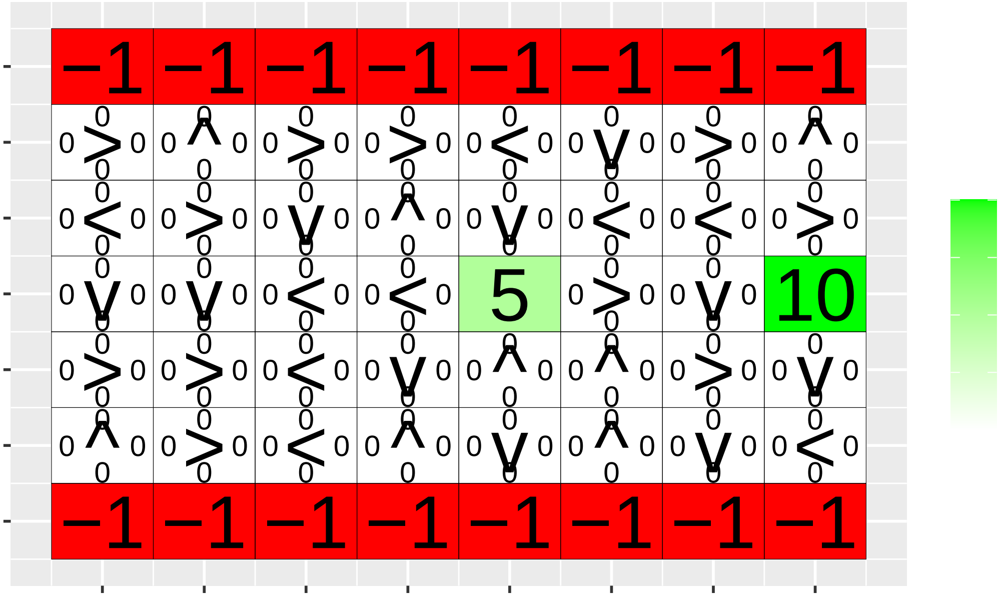


What has the agent learned after the first 10 episodes ? Is the final greedy policy (after 10000 episodes) optimal for all states, i.e. not only for the initial state ? Why / Why not ? Do the learned values in the Q-table reflect the fact that there are multiple paths (above and below the negative rewards) to get to the positive reward ? If not, what could be done to make it happen ?

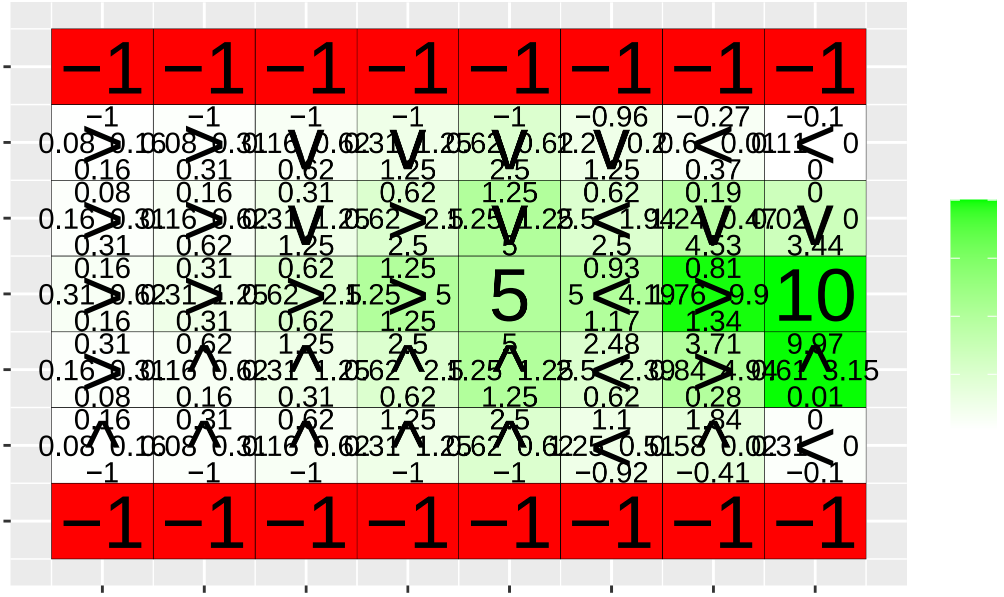
The agent doesn't learn much after 10 iterations. It only learned the some parts of the obstacles, which has -1 value in the reward map. After 10,000 iterations, even though the agent learns to find the reward regardless of the starting state, not with the smallest number of moves in each state. It can be observed that it tends to take a longer path when it starts below the obstacle, specifically from the state (1,3). The left side of the grid world did not get updated as often as the right side, probably because the agent quickly learnt to get out of there and explore the right side of the map. Additionally the rightmost column has been explored slightly less, and has good policy but could go to goal faster. The agent has learnt that the only way to go around the negative block is by going down, even when it starts at the very top. It could be interesting to change epsilon value, decreasing it more in time could give the agent explore more in the beginning and exploit more in the end. Currently it seems that it has learnt a path that works and keeps to that.Also we could try training the agent for more episodes or lower the discount factor γ somewhat, in order to penalize rewards far in the future, which might help the agent identify the shorter path to the 10 reward.

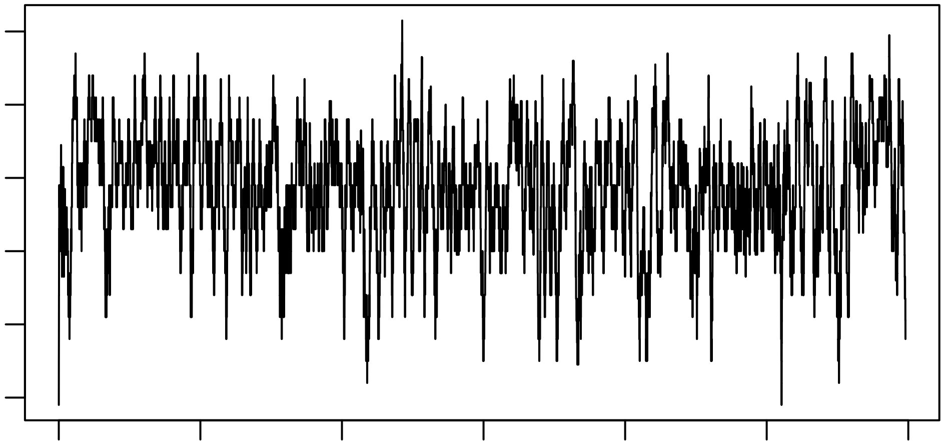
Question 2.3 - Environment B

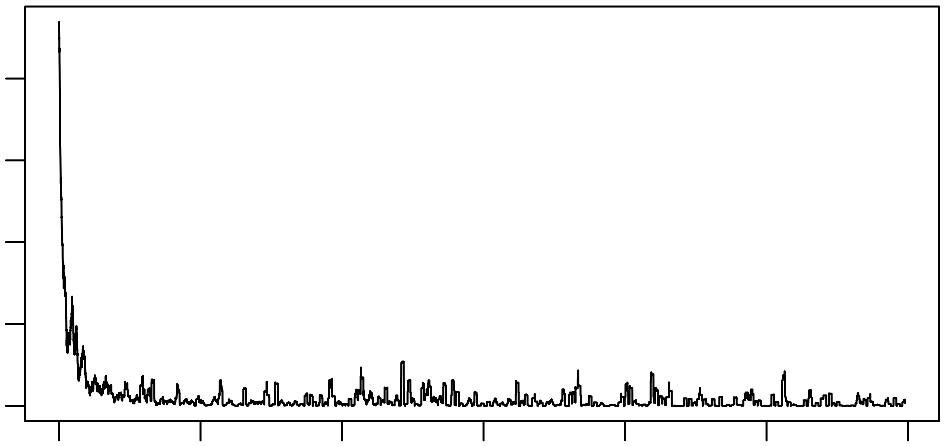
```
H <- 7
W <- 8

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()
```

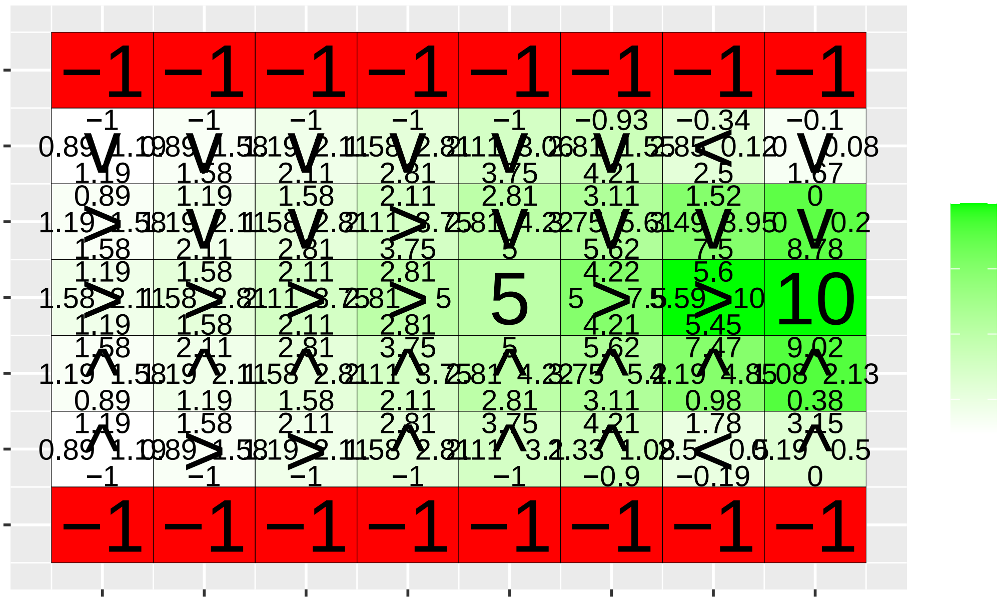


```
MovingAverage <- function(x, n){

  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)
  cat("\n\n")
  plot(MovingAverage(reward,100),type = "l")
  cat("\n\n")
  plot(MovingAverage(correction,100),type = "l")
  cat("\n\n")
}
```

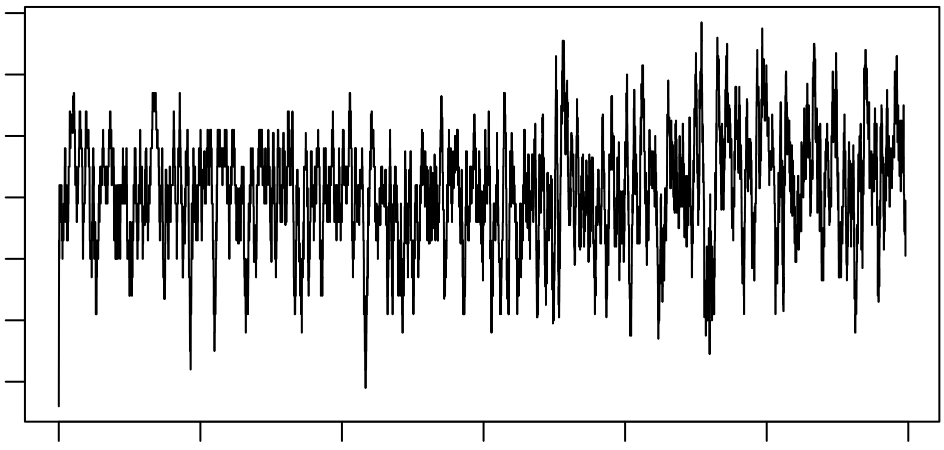


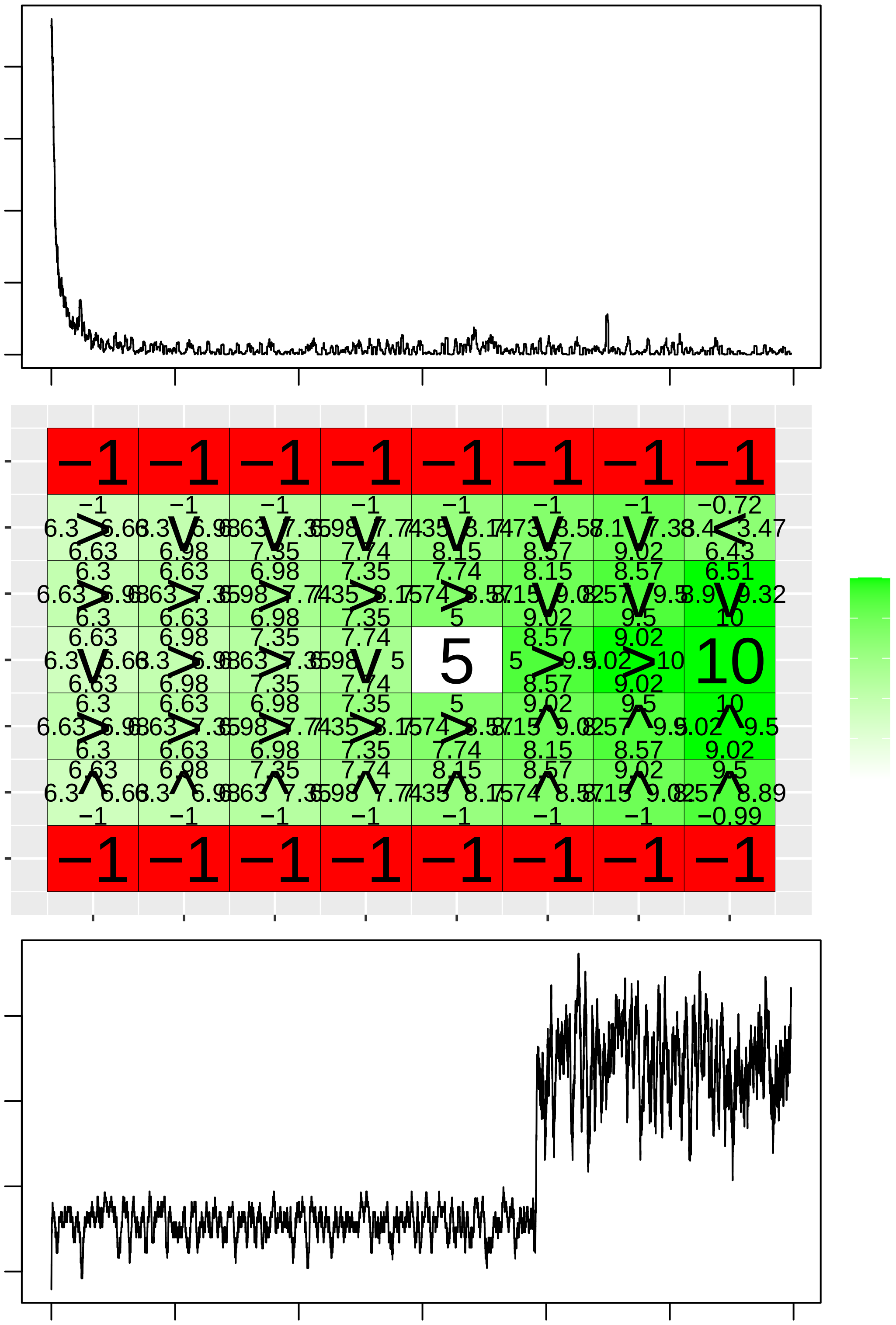


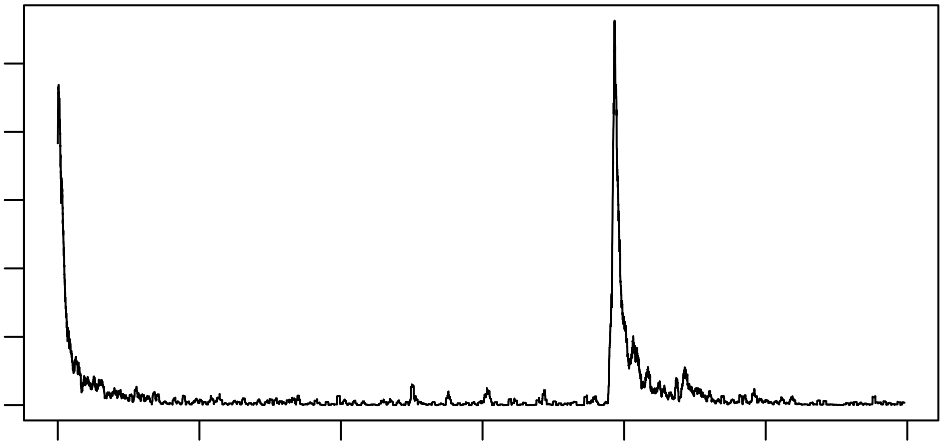




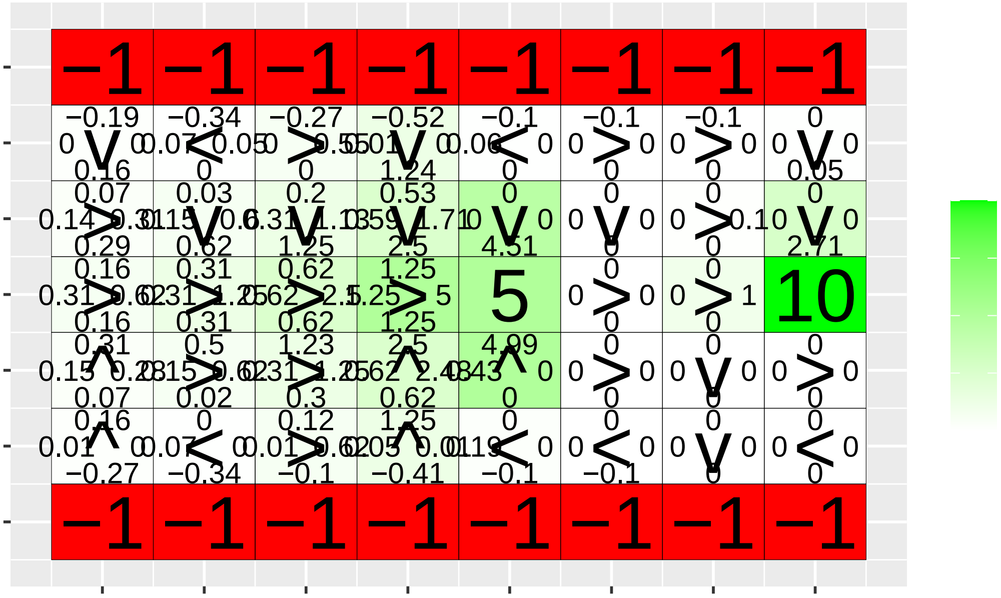
−1 −1 −1 −1 −1 −1 −1 −1
−1 −1 −1 −1 −1 −0.93 −0.34 −0.1
1.19 1.58 2.11 2.81 3.75 4.21 2.5 1.67
0.89 1.19 1.58 2.11 2.81 3.11 1.52 0
1.58 2.11 2.81 3.75 5 5.62 7.5 8.78
1.19 1.58 2.11 2.81 5 4.22 5.6 10
1.19 1.58 2.11 2.81 4.21 5.45
1.58 2.11 2.81 3.75 5 5.62 7.47 9.02
0.89 1.19 1.58 2.11 2.81 3.11 0.98 0.38
1.19 1.58 2.11 2.81 3.75 4.21 1.78 3.15
−1 −1 −1 −1 −1 −0.9 −0.19 0
−1 −1 −1 −1 −1 −1 −1 −1

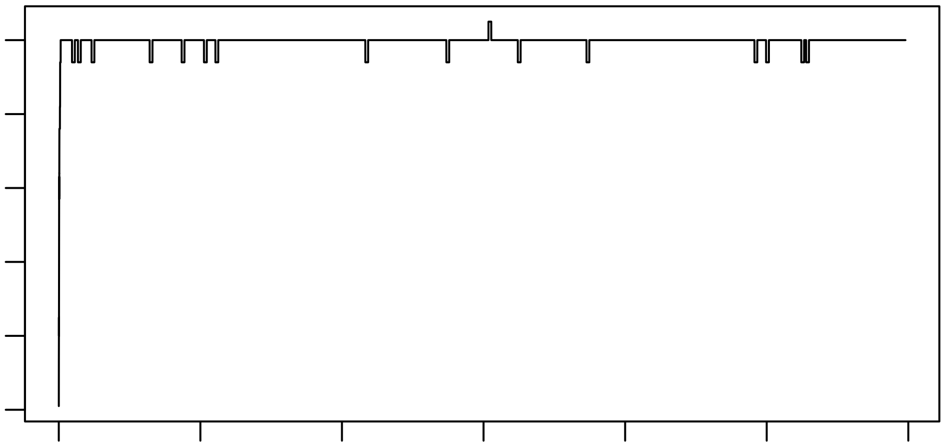


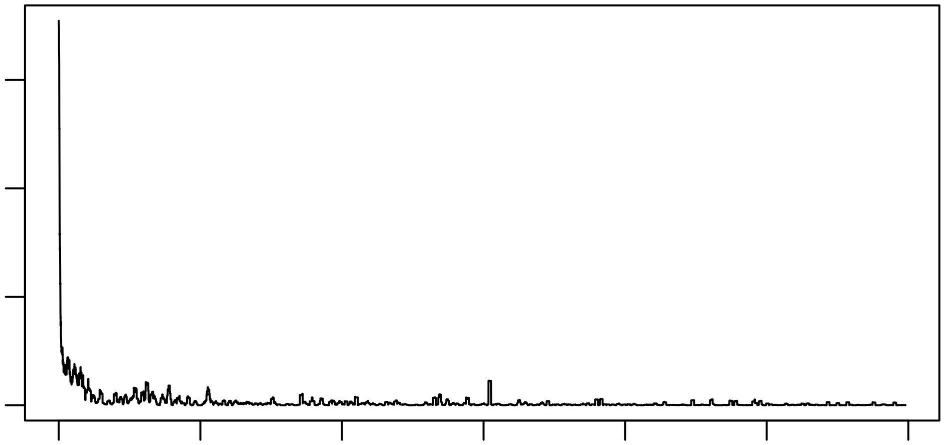

−1 −1 −1 −1 −1 −1 −1 −1
> V V V V V V <
> > > > > V V V
V > > V 5 > > 10
> > > > > ^ ^ ^
^ ^ ^ ^ ^ ^ ^ ^
−1 −1 −1 −1 −1 −1 −1 −1

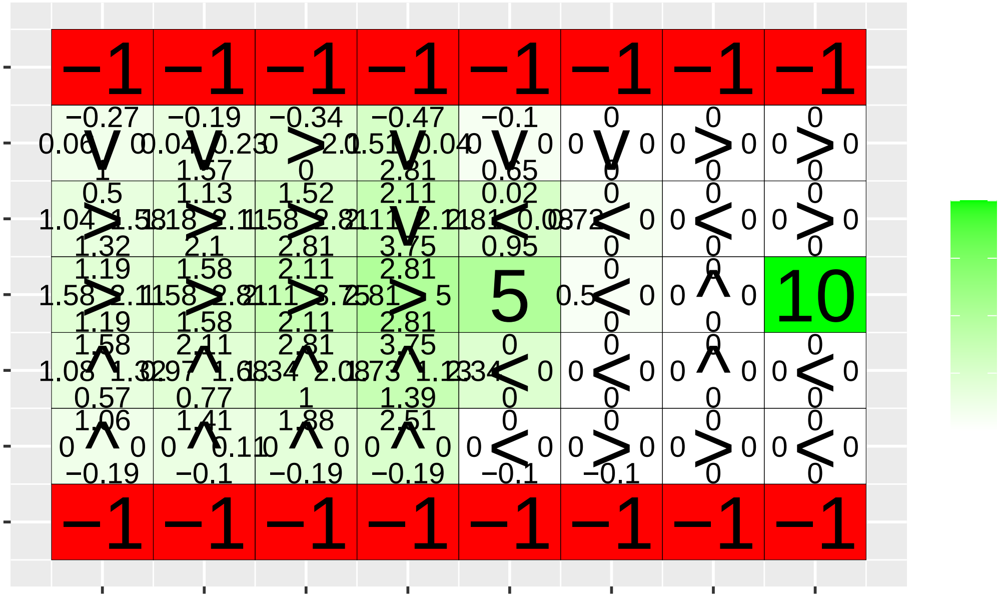


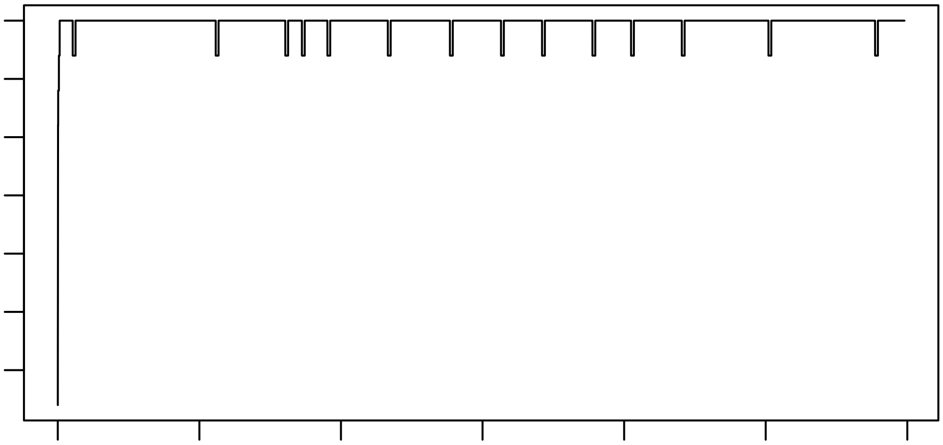
```
for(j in c(0.5,0.75,0.95)){
  q_table <- array(0,dim = c(H,W,4))
  reward <- NULL
  correction <- NULL

  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)
  cat("\n\n")
  plot(MovingAverage(reward,100),type = "l")
  cat("\n\n")
  plot(MovingAverage(correction,100),type = "l")
  cat("\n\n")
}
```

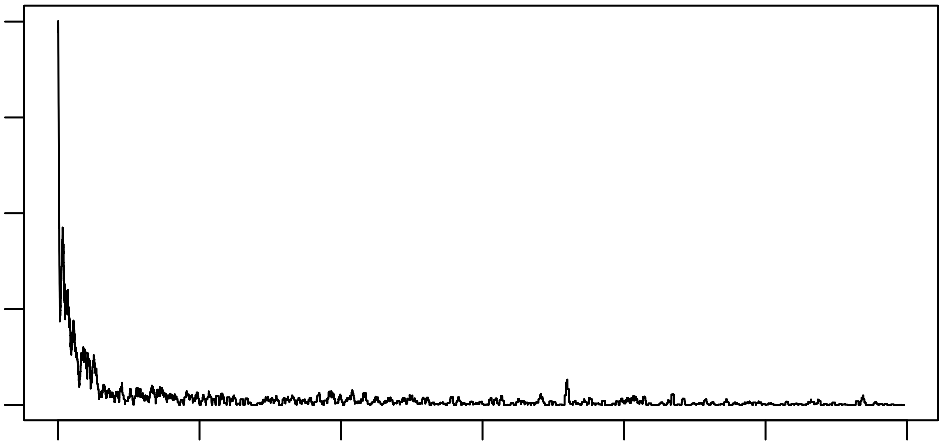


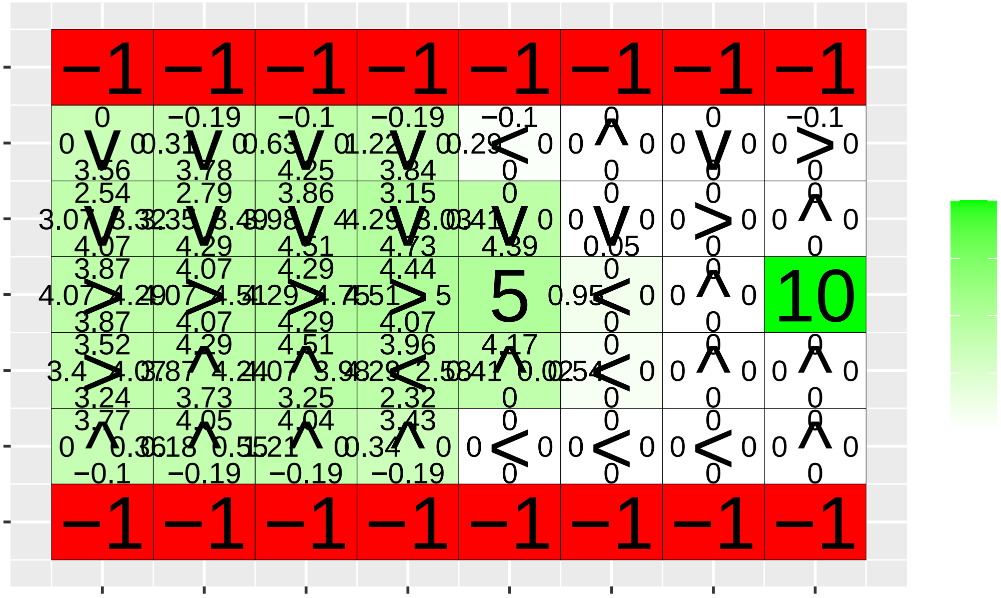


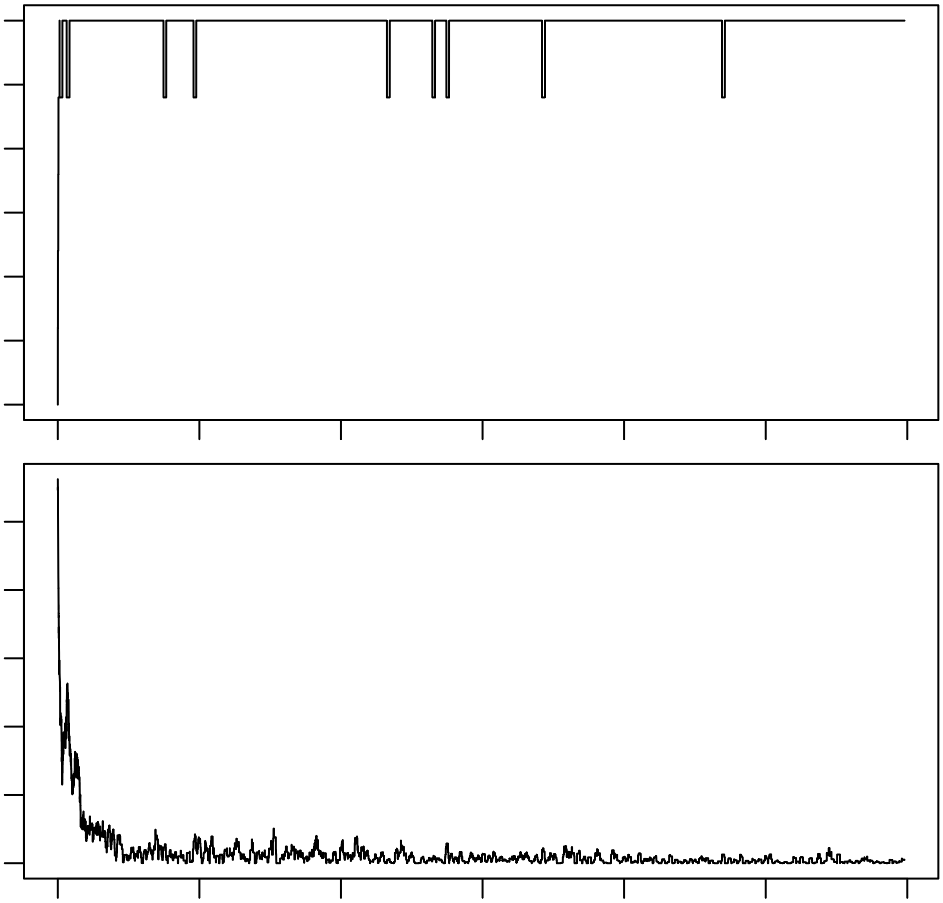



−1 −1 −1 −1 −1 −1 −1 −1
5
10
−1 −1 −1 −1 −1 −1 −1 −1






−1 −1 −1 −1 −1 −1 −1 −1
0 −0.19 −0.1 −0.19 −0.1 0 0 −0.1
3.56 3.78 4.25 3.84 0 0 0 0
2.54 2.79 3.86 3.15 0 0 0 0
4.07 4.29 4.51 4.73 4.39 0.05 0 0
3.87 4.07 4.29 4.44 5 0.95 0 10
3.87 4.07 4.29 4.07 0 0
3.52 4.29 4.51 3.96 4.17 0 0 0
3.24 3.73 3.25 2.32 0 0 0 0
3.77 4.05 4.04 3.43 0 0 0 0
−0.1 −0.19 −0.19 −0.19 0 0 0 0
−1 −1 −1 −1 −1 −1 −1 −1



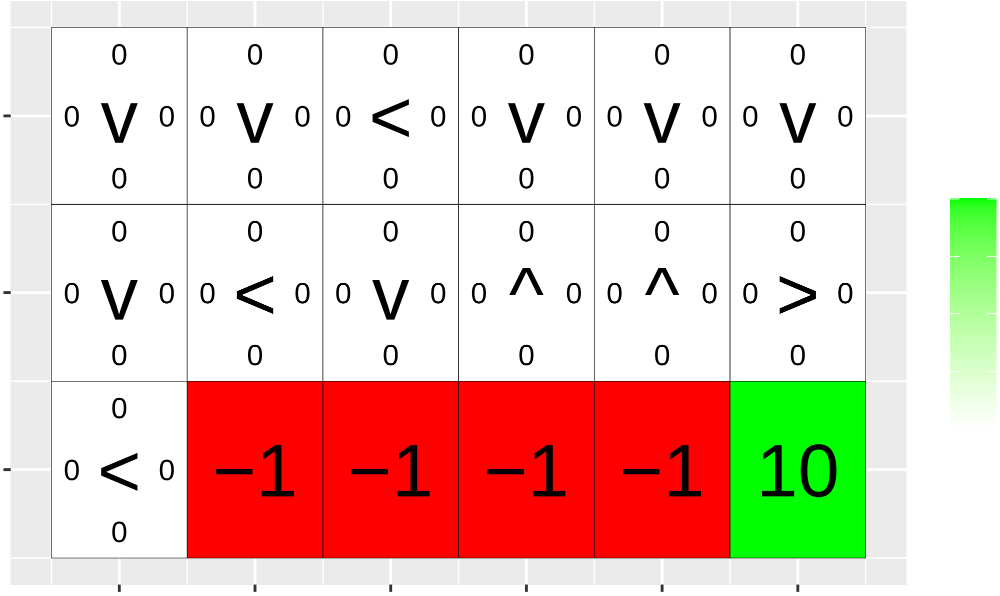
Investigate how ϵ and γ parameters affect the learned policy by running 30000 episodes of Q-learning with $\epsilon = 0.1, 0.5$, $\gamma = 0.5, 0.75, 0.95$, $\beta = 0$ and and $\alpha = 0.1$.

ϵ parameter determines how often the agent explores new actions versus exploiting. When it is lower, the agent explores less. When epsilon is low, the policy is focused around getting to 5 and sticks to that as the optimal policy because it never gets to explore beyond and realize there is a greater reward behind it. When epsilon is higher, however, the agent learns there is a 10 reward behind the 5, and gets to explore that region of the map. When $\epsilon = 0.5$ and $\gamma = 0.75$, the agent learns the path for the reward 5, it can only learn the path to reward 10 when it is near reward 10. In some states, even though reward 10 is closer, the path leads to reward 5, such as in state (2,7).

γ parameter determines how much future rewards are taken into account. When γ is lower, the agent values immediate rewards higher. But even for $\gamma = 0.75$ the agent still chooses the 5-reward state when starting in a y-position equal to or lower (left of) the 5-reward state which makes up a majority of all starting states. For $\gamma = 0.95$ however, the agent has learned to fully avoid the 5-reward state, and instead chooses to go around it, heading for the 10-reward state. Furthermore keeping ϵ constant, it can be seen that the average corrections get a bit bigger as γ increases. This is because, as γ grows, a bigger portion of the future reward is being added to the correction. This is especially striking when ϵ is low, because when γ is very high the agent still eventually realizes there is a greater goal beyond the first, and trains its policy to go there instead. These moments of realization can be seen from the moving average plots when there is a sudden jump in the reward and correction.

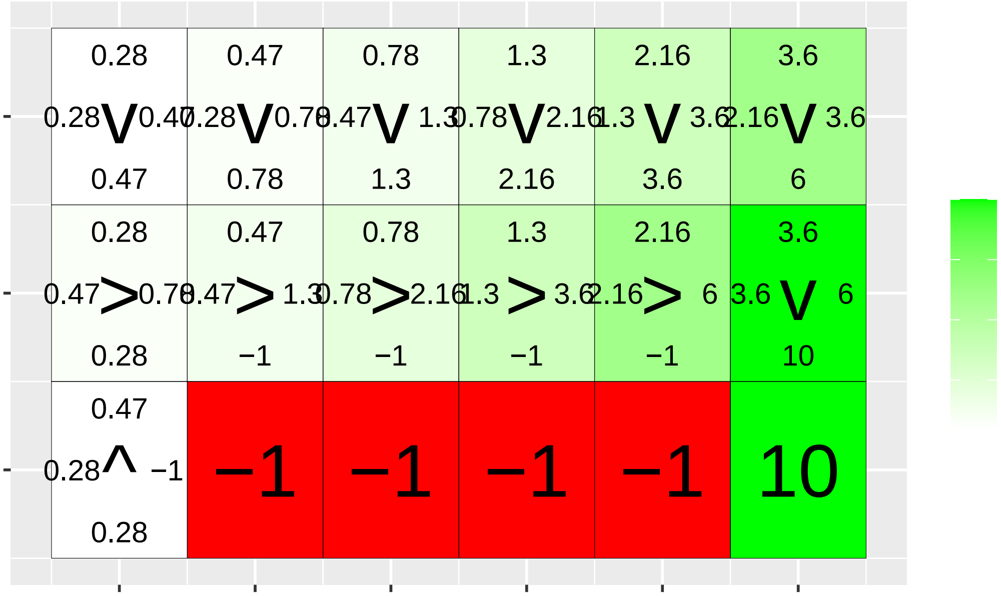
Question 2.4 - Environment C

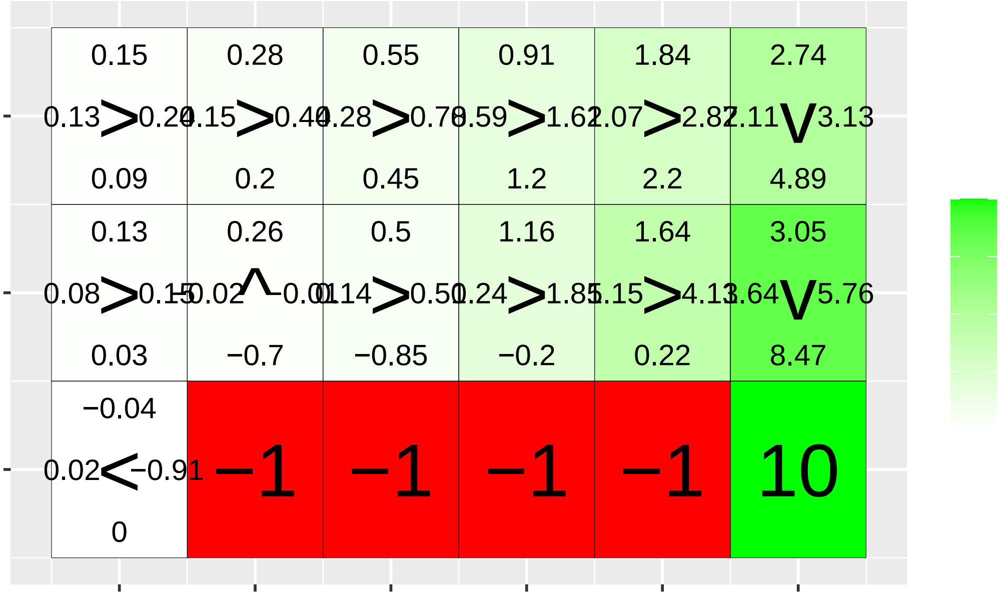
```
H <- 3
W <- 6

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()
```

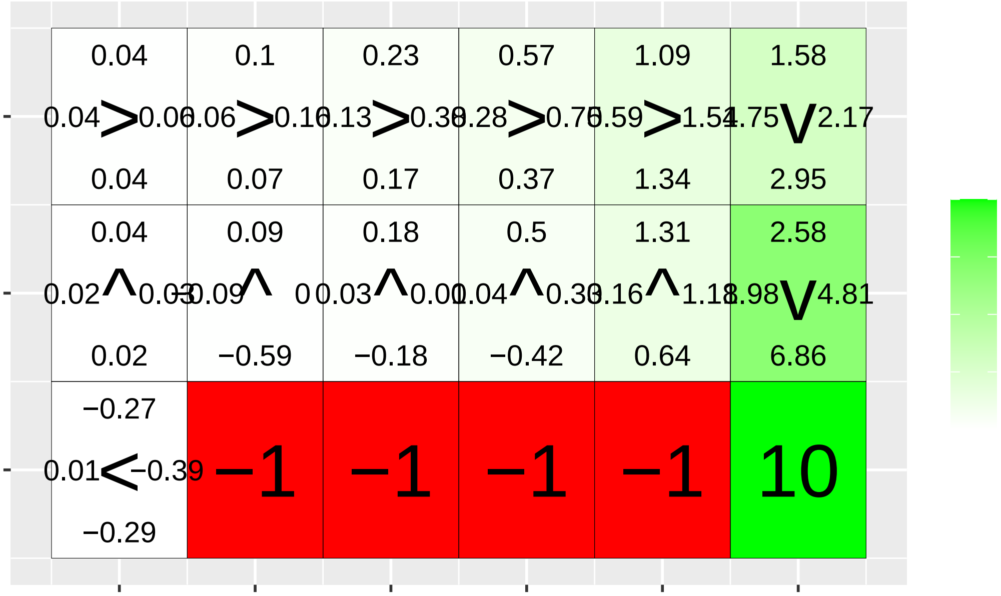

0 0 0 0 0 0
0 V 0 0 V 0 0 < 0 0 V 0 0 V 0 0 V 0
0 0 0 0 0 0
0 0 0 0 0 0
0 V 0 0 < 0 0 V 0 0 ^ 0 0 ^ 0 0 > 0
0 0 0 0 0 0
0
0 < 0 −1 −1 −1 −1 10
0

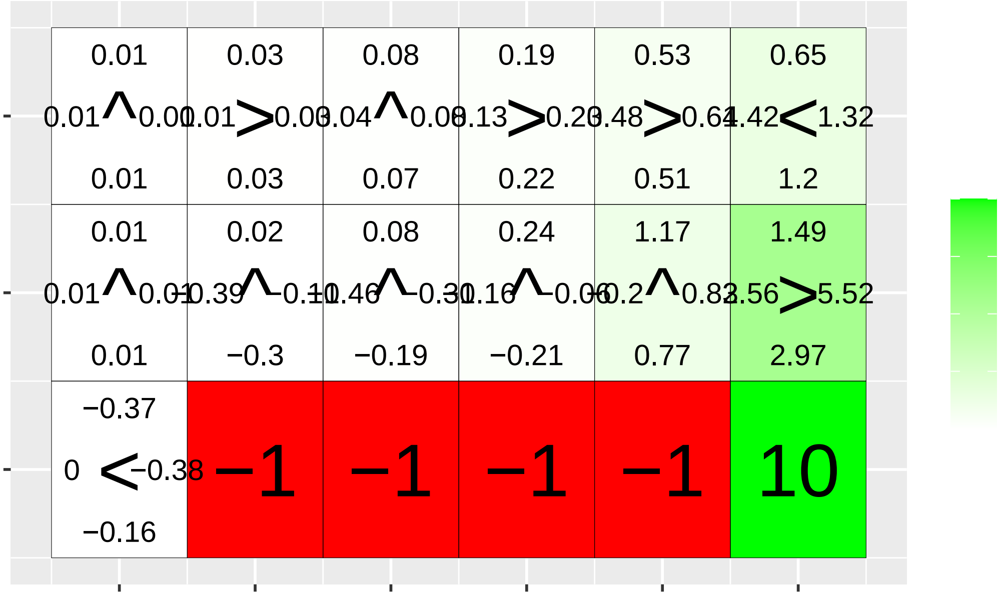
```
cat("\n\n")
```

```
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)
  cat("\n\n")
}
```


0.28
0.28 V 0.47
0.47
0.47
0.28 V 0.78
0.78
0.78
0.47 V 1.3
1.3
1.3
0.78 V 2.16
2.16
2.16
1.3 V 3.6
3.6
3.6
2.16 V 3.6
6
0.28
0.47 > 0.78
0.28
0.47
0.47 > 1.3
−1
0.78
0.78 > 2.16
−1
1.3
1.3 > 3.6
−1
2.16
2.16 > 6
−1
3.6
3.6 V 6
10
0.47
0.28 ^ −1
0.28
−1
−1
−1
−1
10


0.15
0.13 > 0.24
0.09
0.28
0.15 > 0.44
0.2
0.55
0.28 > 0.78
0.45
0.91
0.59 > 1.62
1.2
1.84
2.07 > 2.87
2.2
2.74
2.11 V 3.13
4.89
0.13
0.08 > 0.15
0.03
0.26
−0.02 ^ −0.01
−0.7
0.5
0.14 > 0.51
−0.85
1.16
0.24 > 1.85
−0.2
1.64
5.15 > 4.13
0.22
3.05
3.64 V 5.76
8.47
−0.04
0.02 < −0.91
0
−1
−1
−1
−1
10





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 $\alpha = 0.1$.

The agent follows the intended action with probability $(1 - \beta)$. As beta increases, the probability of the agent slipping becomes higher. This means it thinks it has performed an action a, but has actually moved to the left or right of that chosen action will probability $\beta/2$, leading to incorrect updates of the Q-table. The agent has learnt to take the shortest path to the 10-reward state, since the probability of accidentally "slipping" into the negative reward states is only 0.1 (for each step in a "danger zone" state). However, when β is taken to be larger than this (0.4, and 0.66), the agent learns that there is a high probability of accidentally slipping into one of the negative reward states.

Question 2.6 - Environment D

Has the agent learned a good policy? Why / Why not ? Could you have used the Q-learning algorithm to solve this task ?

We can say that the agent learned a good policy because almost all initial states lead to goal, except for the map in validation data 1 state (4,1). Also, in the first and third map there is a position in which the agent would end up in a loop if it were to strictly follow the most likely action, but other than that the policies are perfect.

According to Marijn, this task could not have been solved using Q-learning, as in Q-learning the goal is not known to the agent and future rewards would become irrelevant in a different episode. According to Simon the problem is not very difficult, and could probably be solved with Q-learning, since it is very similar to the problem in the previous problems (Environments A-C). Simge thinks that this problem can be solved using Q-learning since it's not too big but it requires a new reward map each time the goal changes its location and it is not memory efficient.

Question 2.7 - Environment E

Has the agent learned a good policy? Why / Why not ? If the results obtained for environments D and E differ, explain why.

This time the agent does not perform as well, even the states near the goal cannot lead to the goal. It appears as though the training goals, and the validation goals are different enough that the learned policy does not help the agent to find the validation goals. The agent just learns to go straight up which is far from optimal when the validation goals are below the top row. The results here are different from environment D, because in environment D the agent was trained on random goals across the board. Here, however, the agent has learnt that the goal will likely be in some section of it, so it doesn't want to go in a completely different direction.