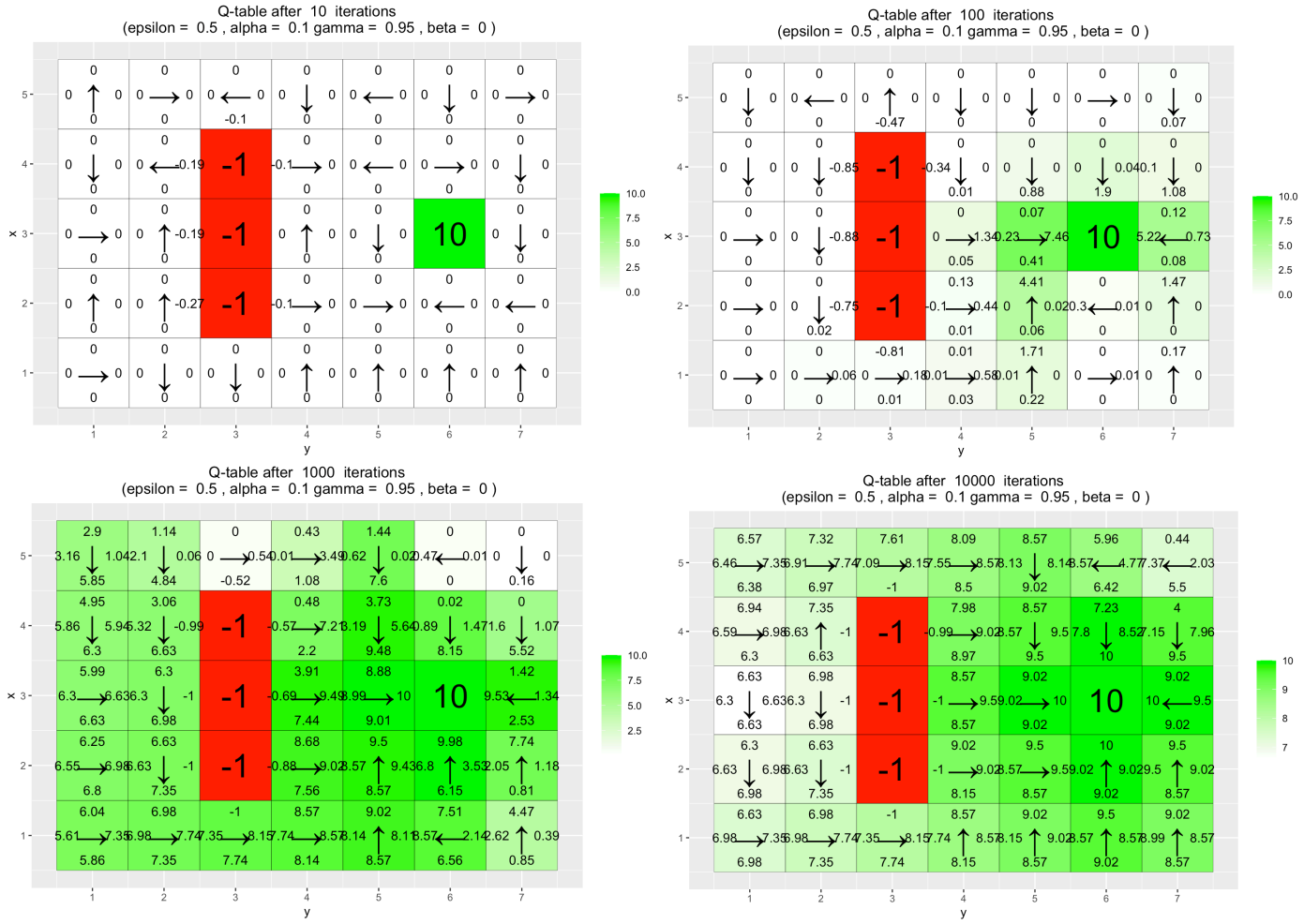
Lab 3 – Reinforcement Learning

2.2) Environment A



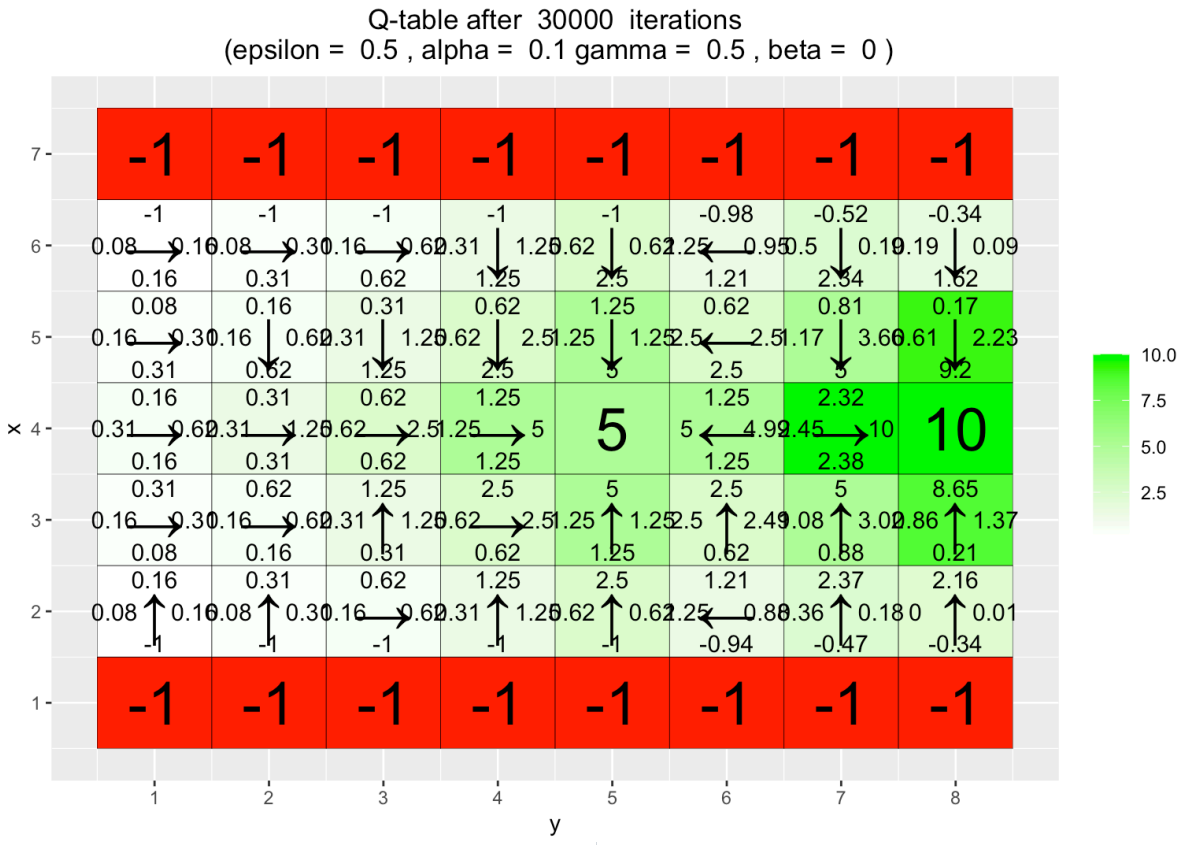
During the first 10 episodes, the agent seems to have found the negative rewards only, since there's only values for those in the q-table. After 1000 iteration, the agent has confidently found a way to the reward. However, going above the negative rewards is not a recognized path. After 10000 iterations, the agent has built paths both over and under the negative rewards.

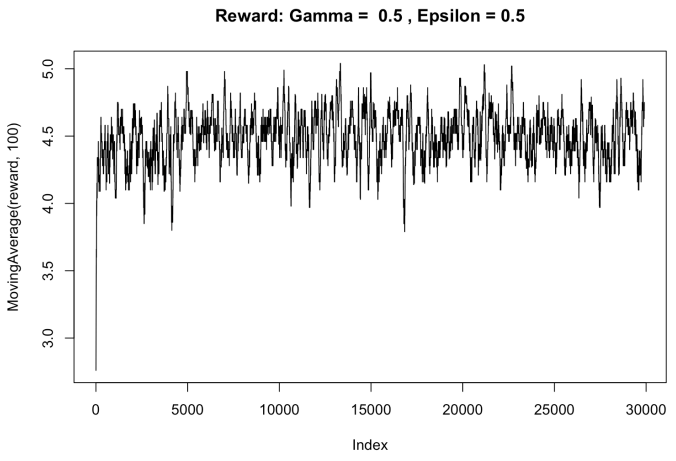
Is it optimal: All the edges going to rewards (negative and positive) has the same value as the reward itself. However, the rest of the q-table does not have the 'optimal' values. They should 'theoretically' follow along the lines of: q = 10*gamma^(# steps from 10). For example, the top right box has rarely been visited by the agent. It probably reaches 10 before having the chance to explore the top right box.

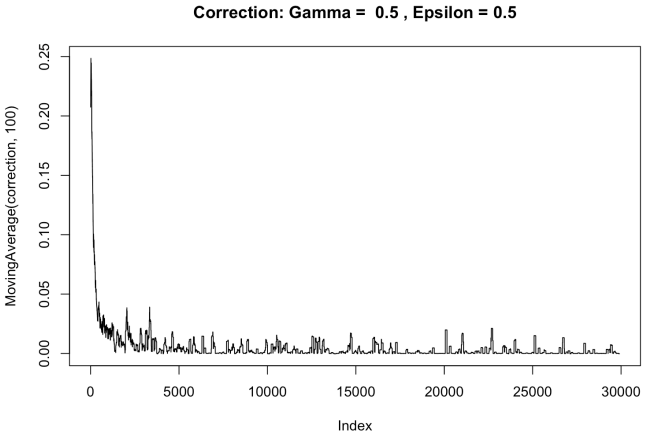
Looking from the starting point (3,1), the agent doesn't care where it goes. We also see that states next to the starting point has the same q-values. From this point of view, the agent has learned multiple paths.

2.3) Environment B – Effect on Epsilon and Gamma

Gamma = 0.5 & Epsilon = 0.5

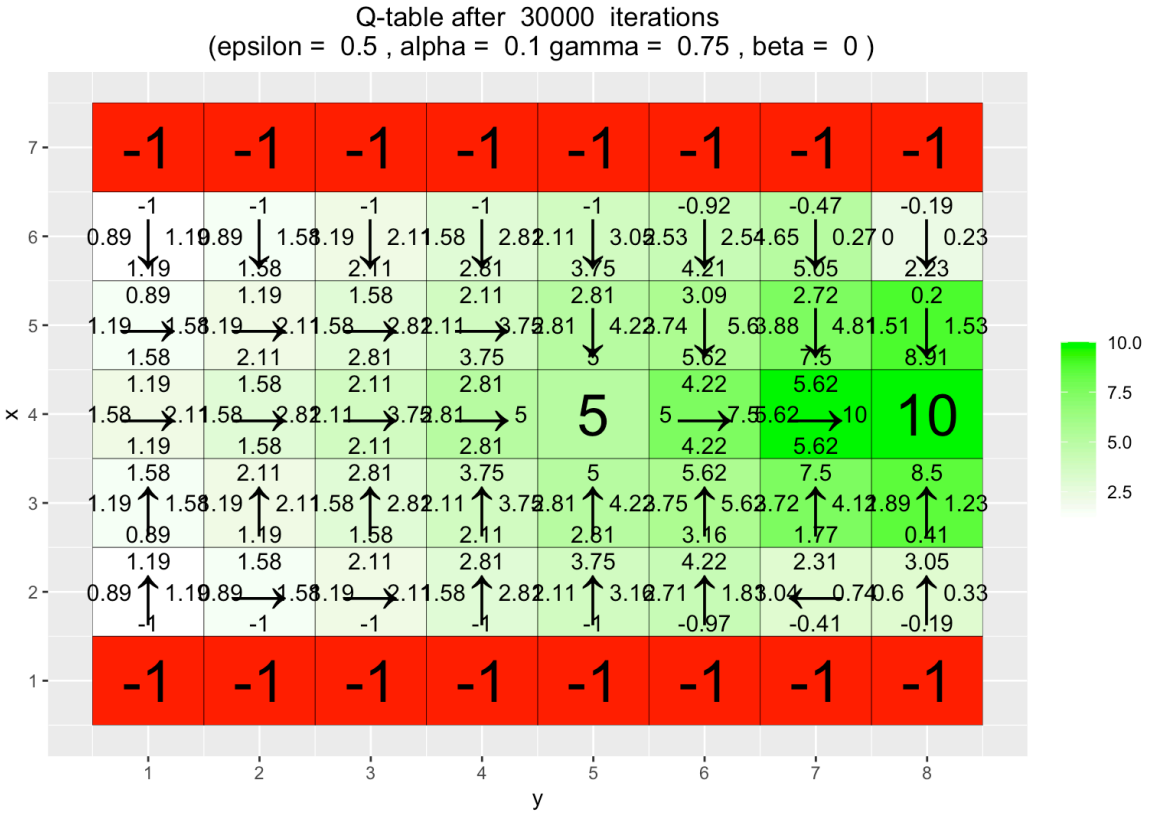


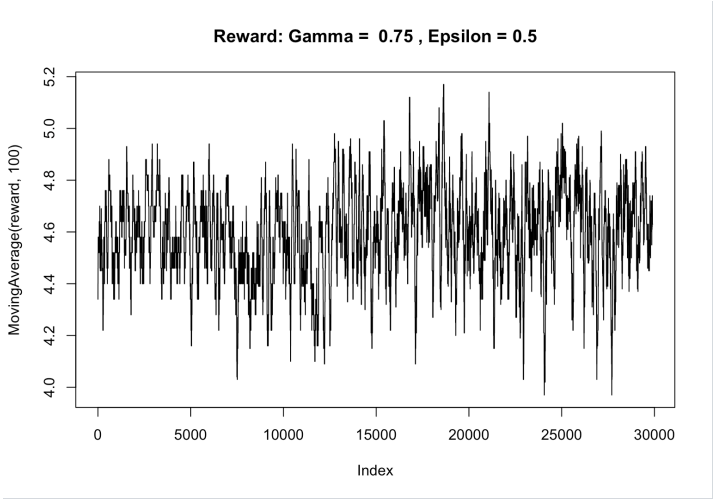


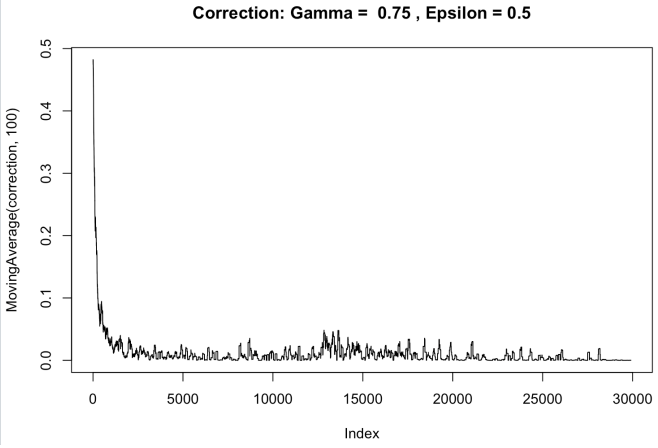


Number of visits to reward 10: 119

Gamma = 0.75 & Epsilon = 0.5

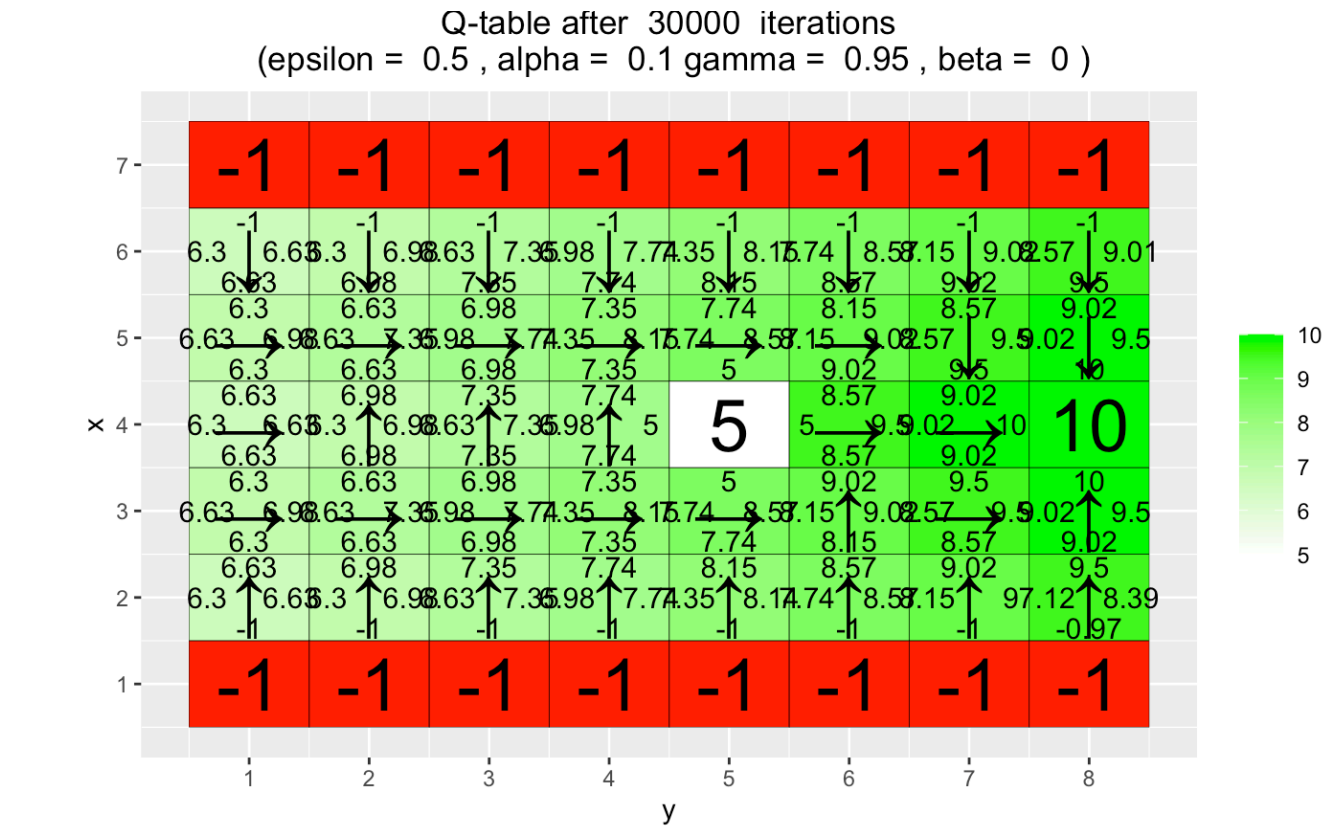


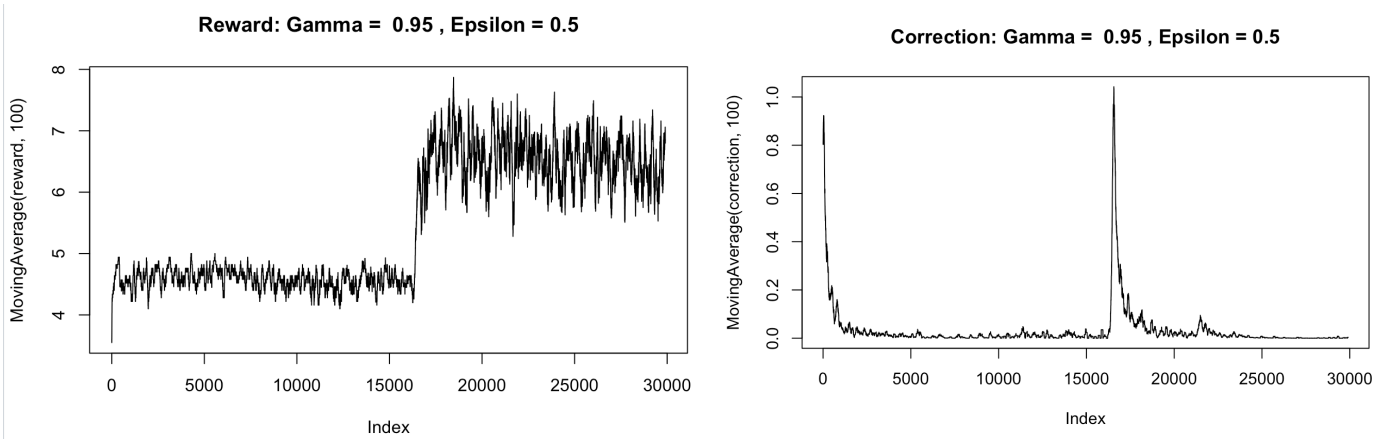




Number of episodes ending in reward 10: 133

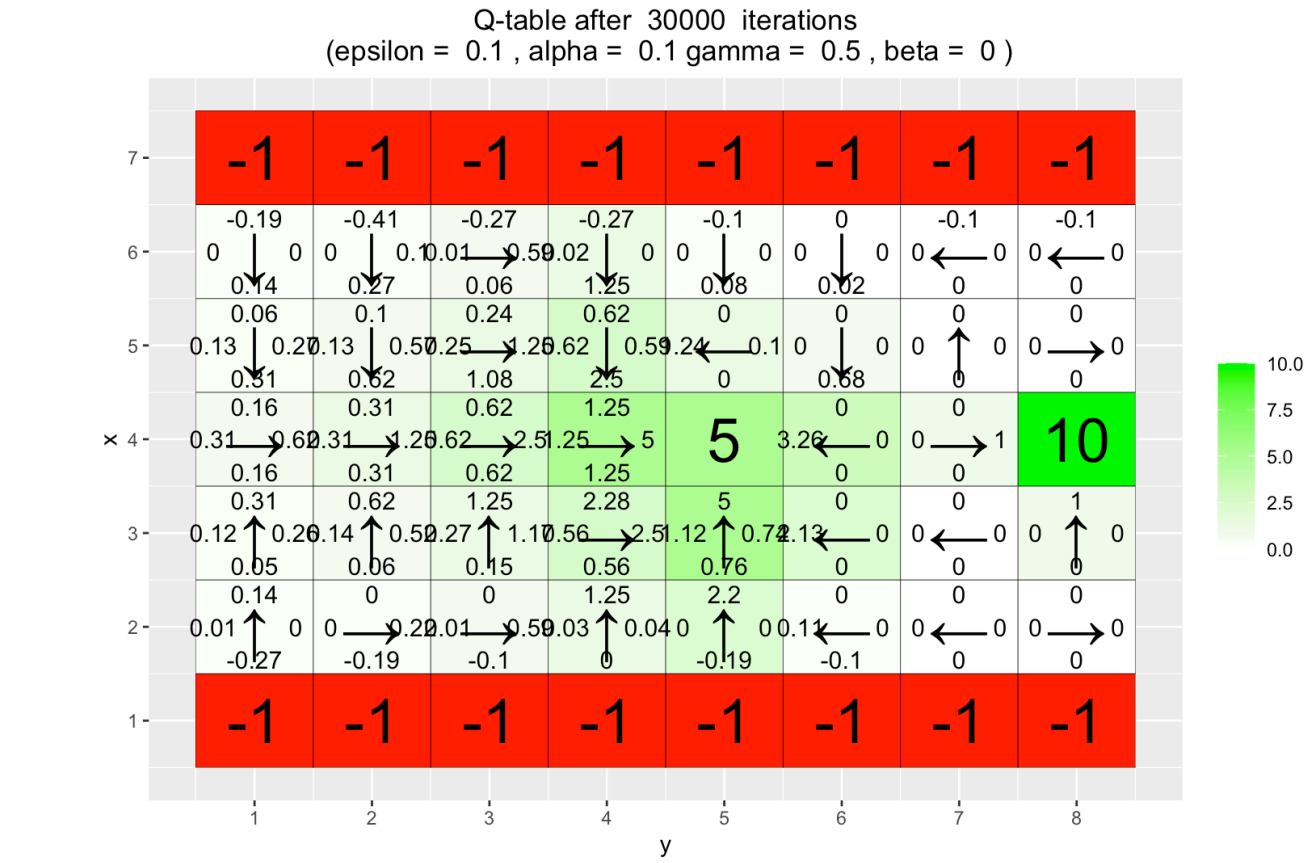
Gamma = 0.95 & Epsilon = 0.5

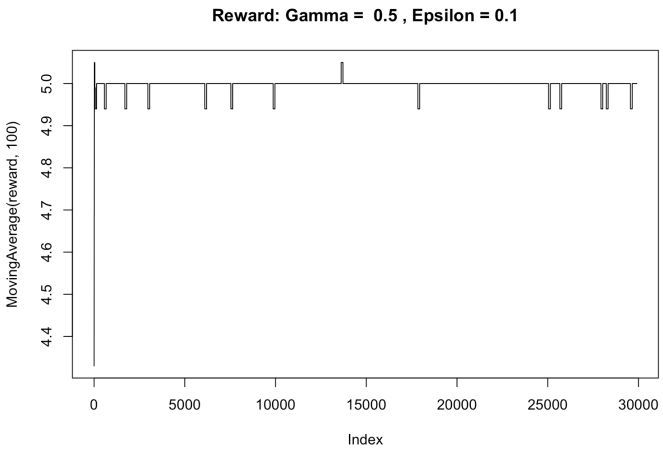


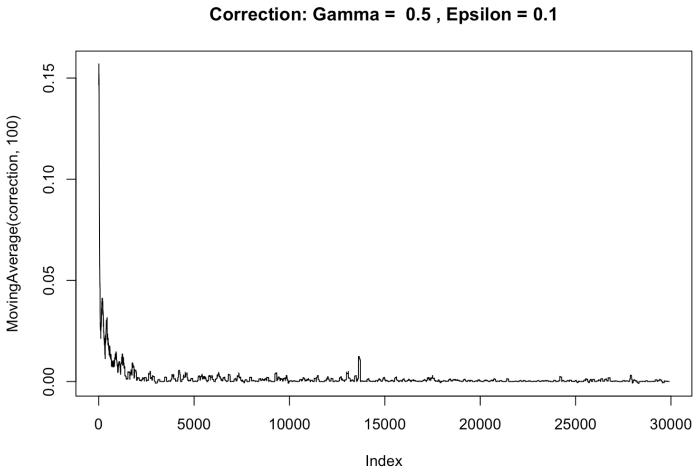


Number of episodes ending in reward 10: 8959

Gamma = 0.5 & Epsilon = 0.1

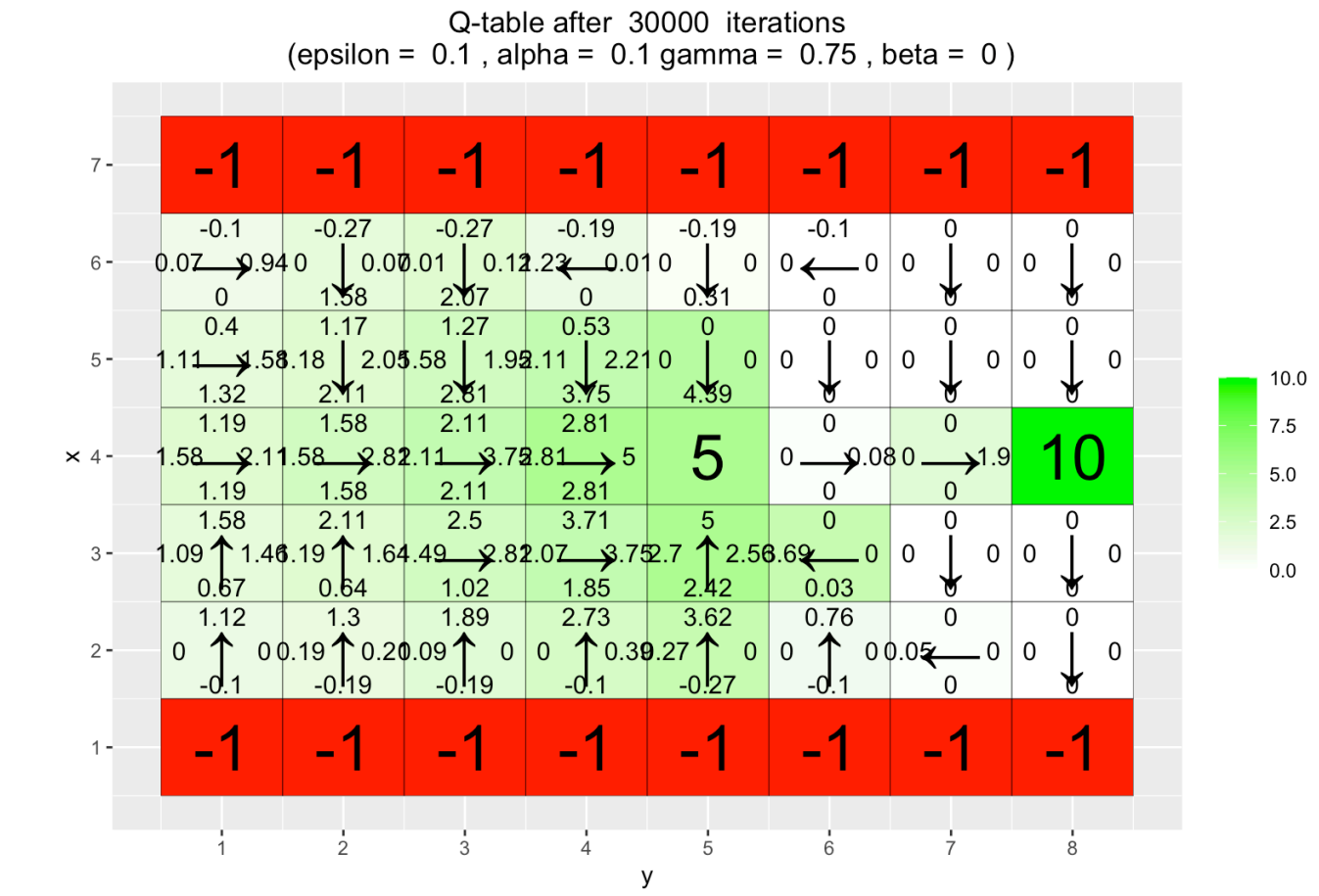


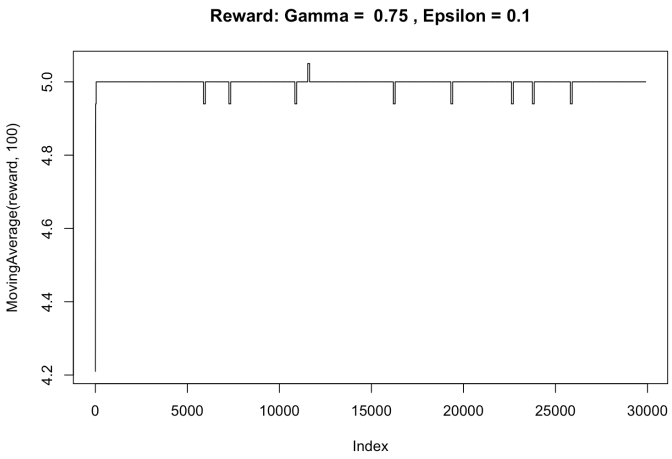


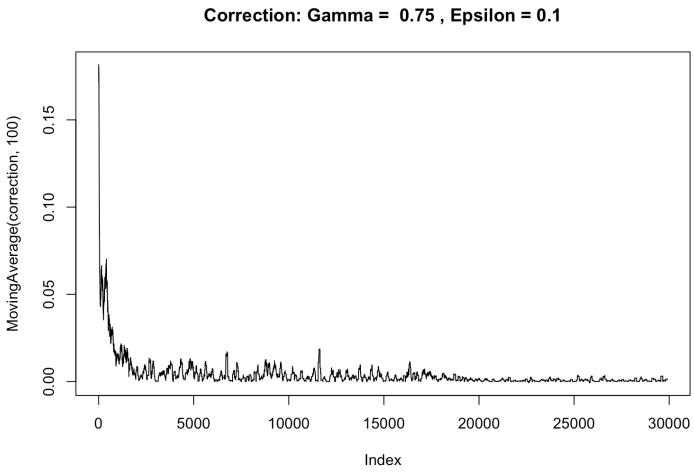


Number of episodes ending in reward 10: 1

Gamma = 0.75 & Epsilon = 0.1

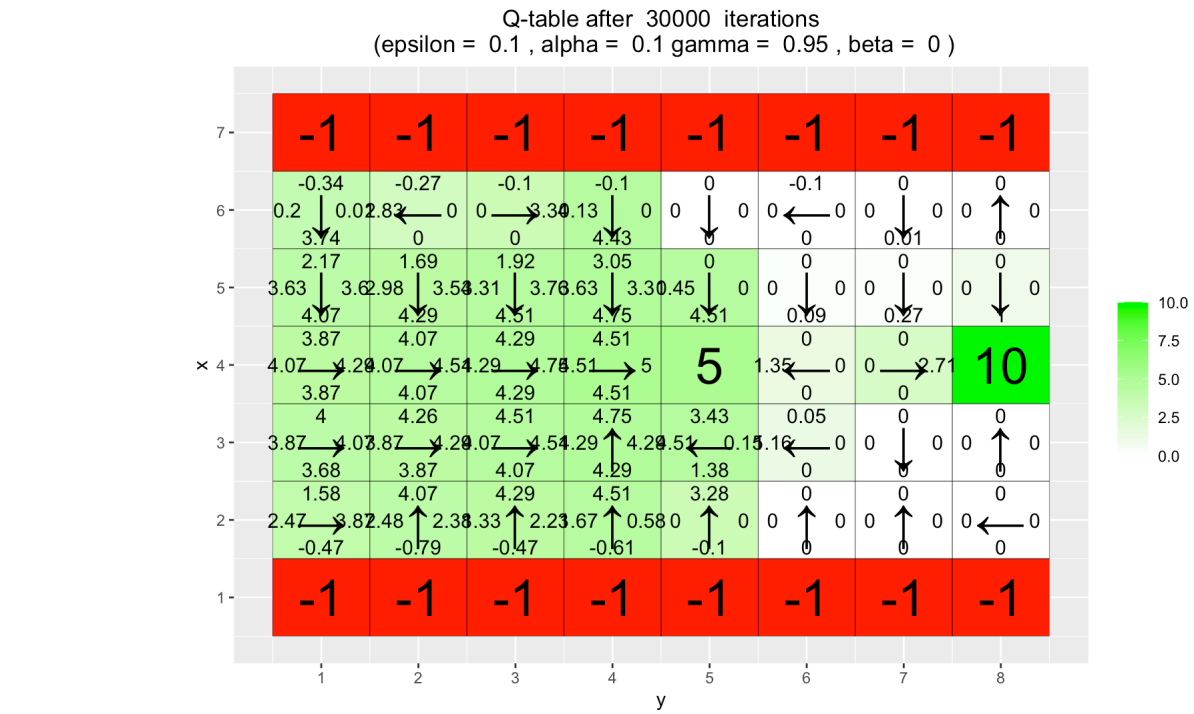


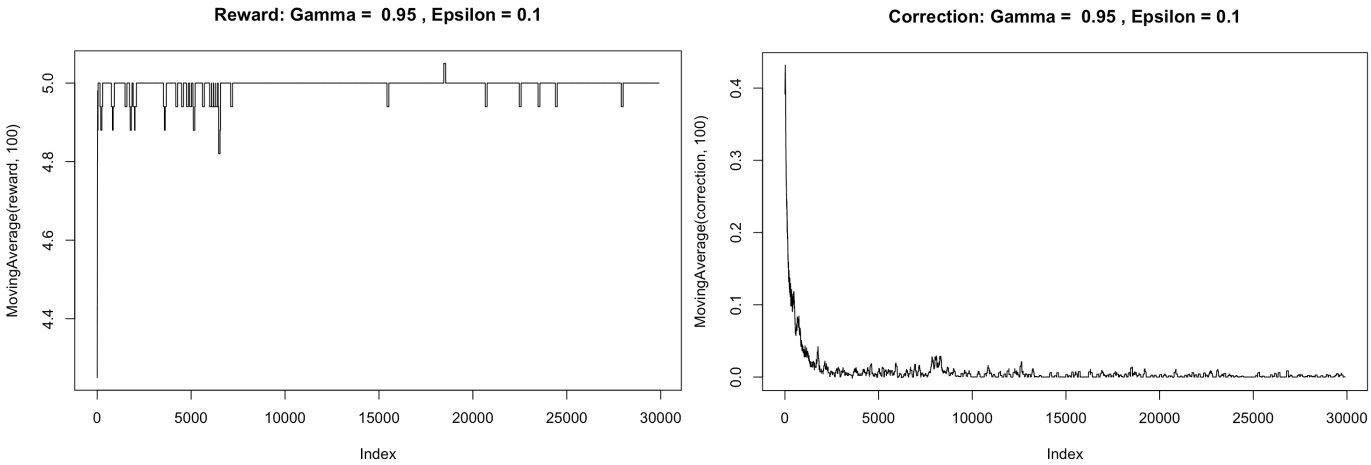




Number of episodes ending in reward 10: 1

Gamma = 0.95 & Epsilon = 0.1





Number of episodes ending in reward 10: 1

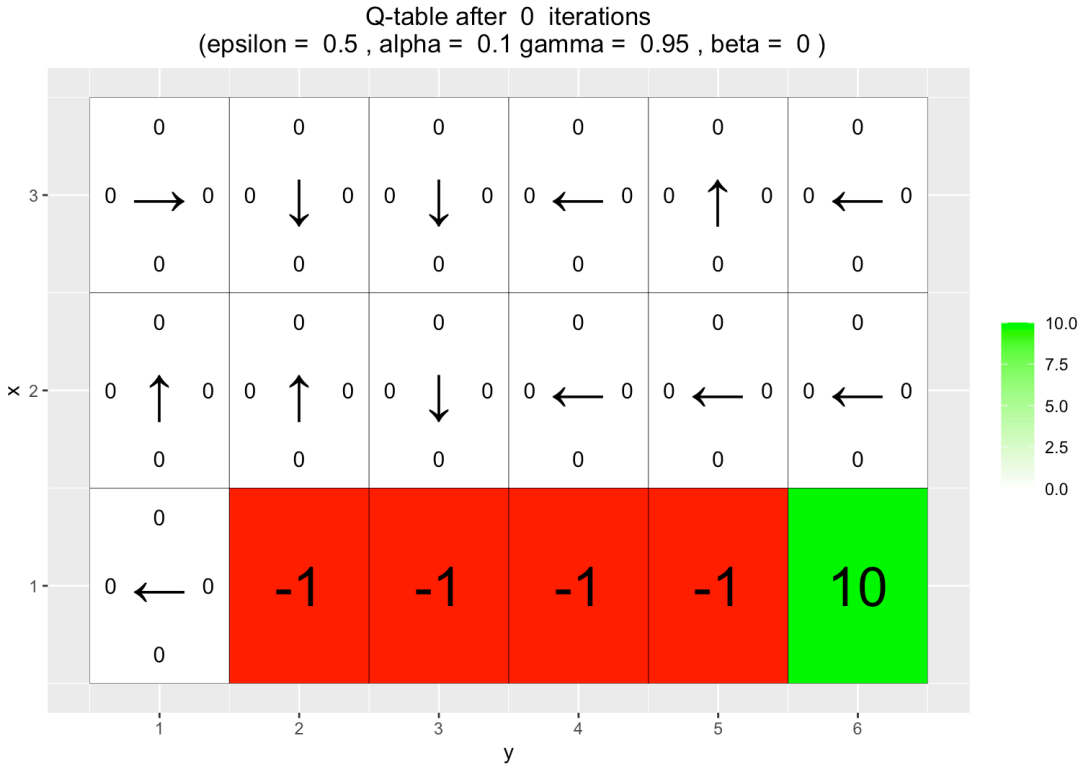
Gamma tells us show much the agent values future rewards. A large value on gamma puts higher values on actions that in a couple of moves will give a reward. A low value will not care as much if the agent is on it's way towards the reward.
Epsilon is the probability to choose a random action instead of choosing the action that maximizes our next step from the q-table. This is good because it promotes the agent to pick actions that it normally would not do, making it possible to find even better paths to the reward.

The reward plot shows the average reward from the last 100 episodes. When using Epsilon = 0,1 the agent very rarely takes a random step instead of greedy-picking the best from the q-table. From the reward plots we see that because of this, the agent never really finds the 10-point reward. Instead, the agent always reaches the 5-point reward. This can be seen from the 'straight' reward plots.

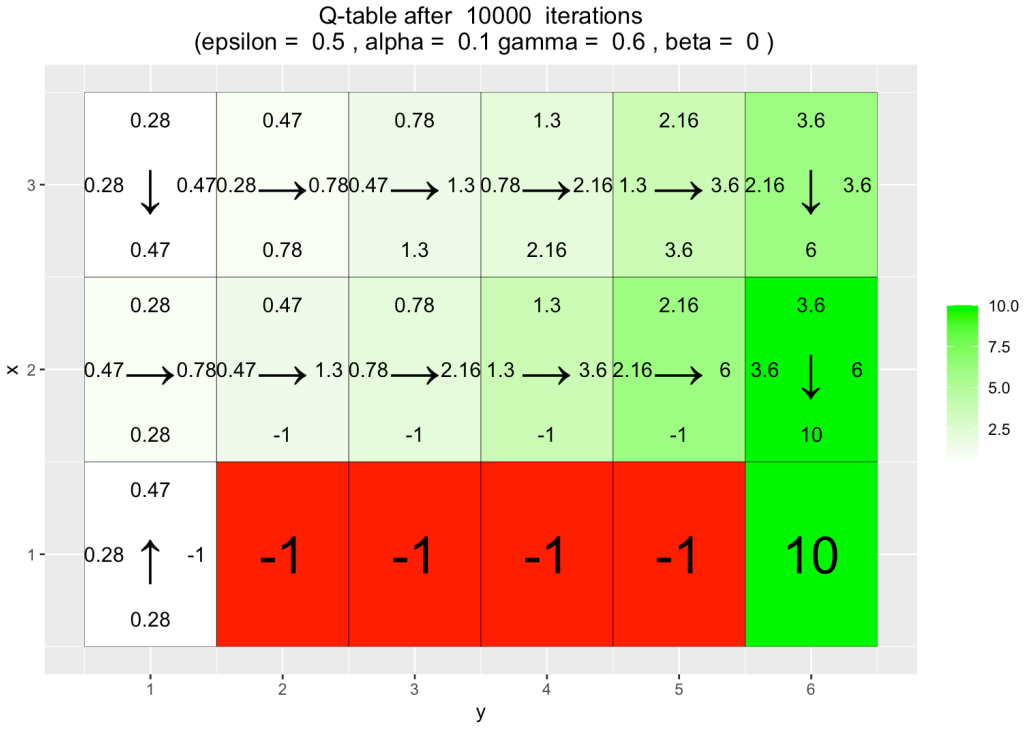
When using a higher gamma, or 'discount rate', the q-values are more spread out throughout the array. This means that q-values far away from the high score gets higher values. What happens at gamma=0,95 is that the agent realized that it can skip 5 point and reach the higher score of 10.

2.4) Environment C – Effect on Beta

Empty environment

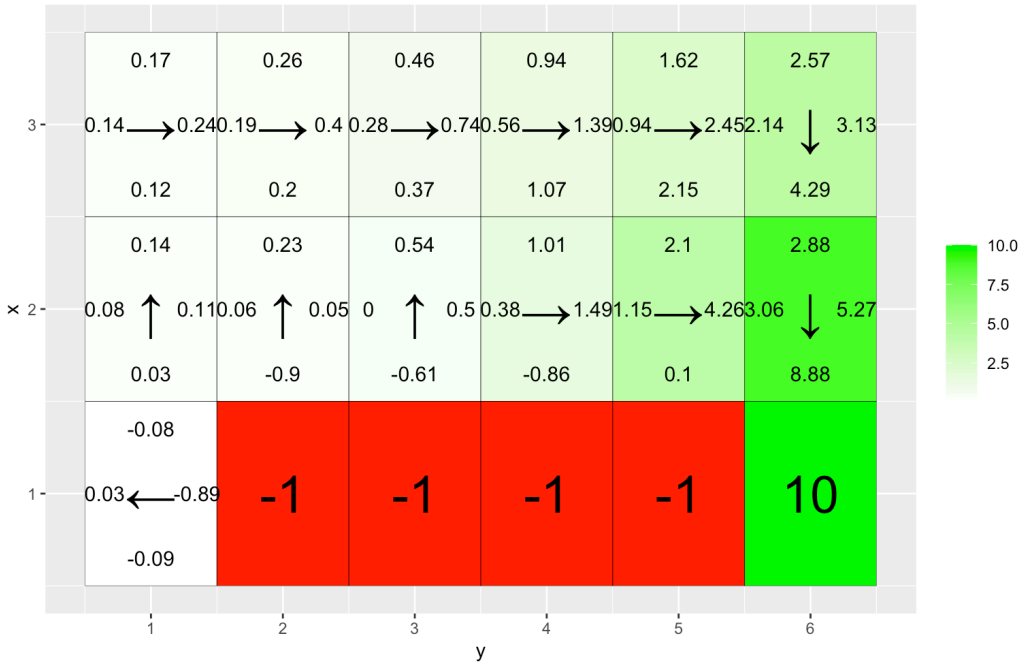


Beta = 0.0



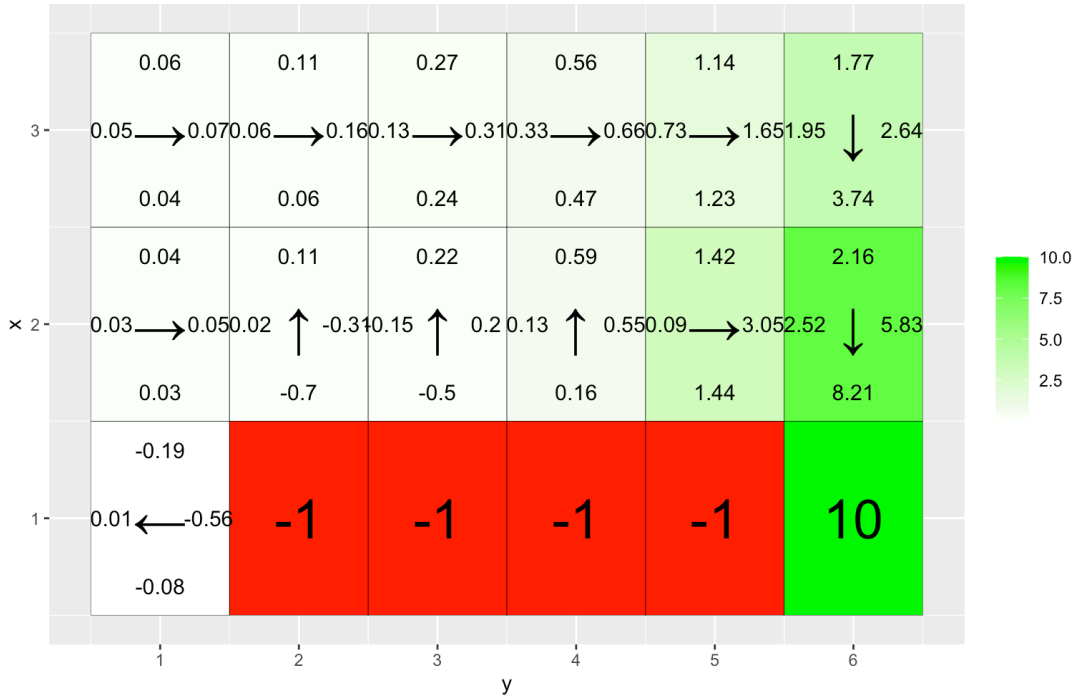
Beta = 0.2

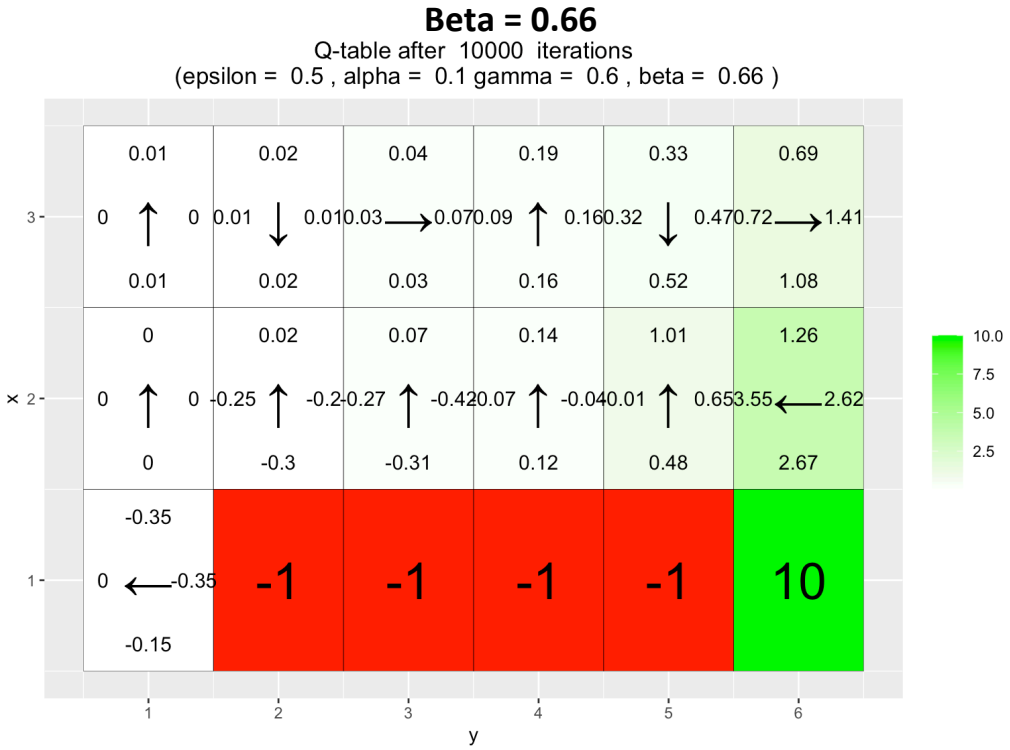
Q-table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.2)



Beta = 0.4

Q-table after 10000 iterations
(epsilon = 0.5 , alpha = 0.1 gamma = 0.6 , beta = 0.4)





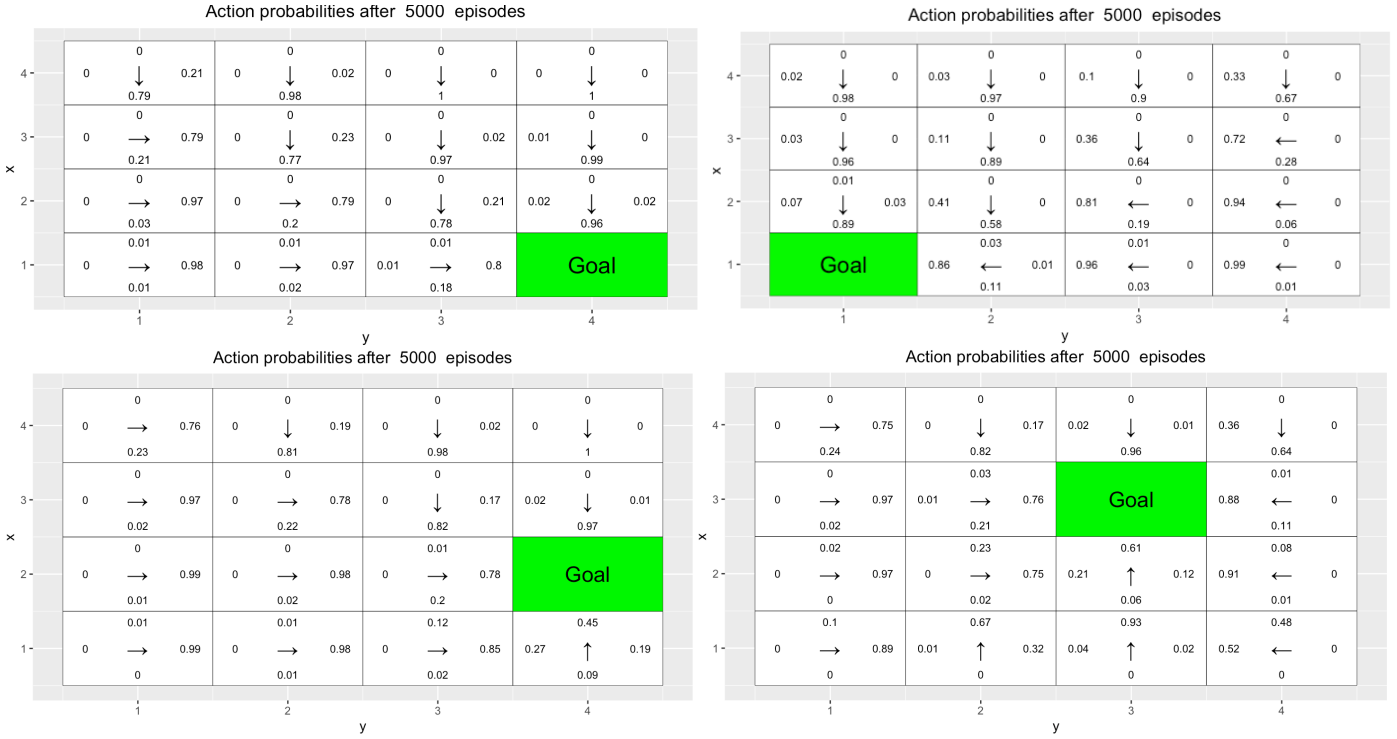
Beta tells us about slippage. Meaning the probability that the agent will make an incorrect move, going left or right of the desired more by equal chance beta/2. This gives a similar effect to **epsilon**, but the advantage that the agent cannot go backwards from the best move according to the q-table. With increasing beta, the agent slips more, and therefore explores more than it otherwise would.

The best result is achieved by using beta = 0. The q-values leads more securely to the goal. The agent gets more confused for higher values of beta. For example when beta = 0,66 the agent would rather move away from the reward than moving towards it. If these instructions were given to a person, they would not arrive at reward 10.

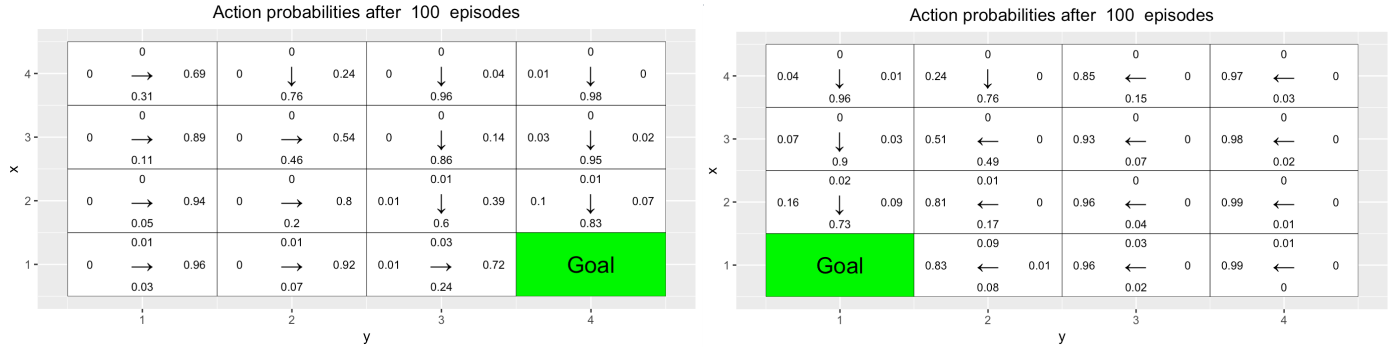
However,
Beta I built into the transition model, which means it's our model of how reality works. When beta > 0.66, there is a larger change to slip than to move in the greedy way. If this behavior is true in real life, the best way of action **IS** according to the model.

2.6) Environment D

I only included 4 different plots for comparison.



Comparing to what some goals looked like after 100 iterations.



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

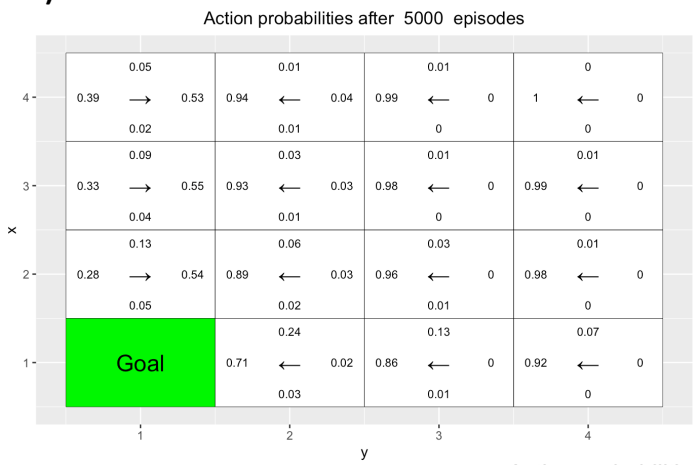
It can be said that the agent successfully reaches the goal, no matter the position of the goal. Meaning that the arrows are pointing towards the goal. One difference from q-learning is that we now get a distribution of actions that sum up to 1, instead of a score.

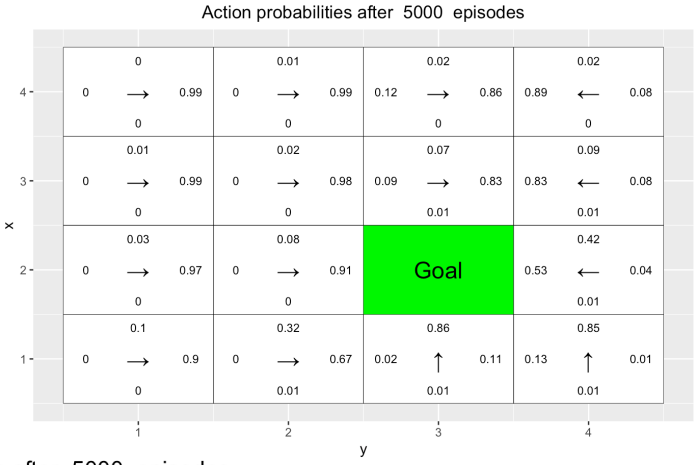
- **Could you have used the Q-learning algorithm to solve this task ?**

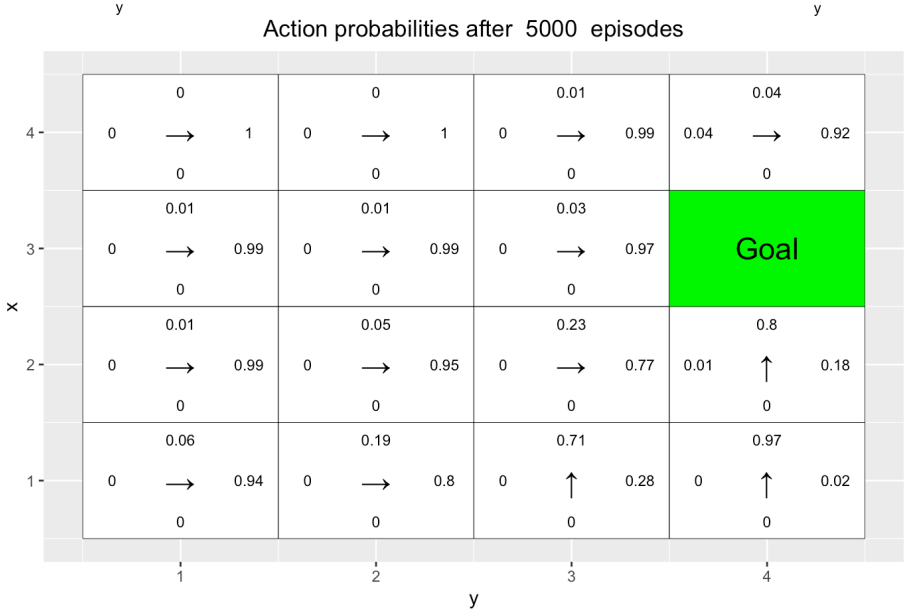
Q-learning would not be good for trying to reach a goal that can have different positions. It only learns the optimal weight for a fixed environment.

Here, the goal is known to the agent before trying to solve the problem. They are trying to solve different problems.

2.7) Environment D







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

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

The policy learned is not very good. Since the training nodes only is on the top, the agent will learn that going down is never a good idea. Meaning that it will never help the agent to reach the goal.

When evaluating the model on a target which is a top node, it performs well better than for nodes toward the bottom. Here, the training objective is different from the learning objective.

One learning here is that the training data needs to be diverse, in this case it is not.

Attached code:

```
GreedyPolicy <- function(x, y){

     return(which.max(rank(q_table[x,y,], ties.method = "random")))

}


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

  # Your code here.
  if(epsilon > runif(1,0,1)){
    rand = sample(1:4, 1)
    #return(q_table[x, y, rand])
    return(rand)
  }else{
    return(GreedyPolicy(x, y))
  }
}

q_learning <- function(start_state, epsilon = 0.5, alpha = 0.1, gamma =
0.95, beta = 0){

  # Your code here.
  state = start_state
  episode_correction = 0
  repeat{
    # Follow policy, execute action, get reward.
    action = EpsilonGreedyPolicy(state[1], state[2], epsilon) # Getting
action
    new_state = transition_model(state[1], state[2], action, beta) #
Updating state
    reward = reward_map[new_state[1], new_state[2]]

    # Q-table update.
    correction = alpha*(reward+gamma*max(q_table[new_state[1],
new_state[2],]) - q_table[state[1], state[2],action])
    q_table[state[1], state[2],action] <<- q_table[state[1],
state[2],action] + correction
    state = new_state
    episode_correction = episode_correction + correction

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