TDDE15 – Lab 3: Reinforcement Learning

hanma409 – Hannes Bengtsson

Exercise 1 – Q-Learning

Description summary:

- The environment consist of a HxW dimensional grid
- An agent acts by moving up, down, left or right in the grid-world
- We assume the state space to be fully observable
- The reward function is deterministic and independent of the agents actions
- We are given a template of the Q-Learning algorithm

Exercise 2 – Environment A

Implementation of the *GreedyPolicy, EpsilonGreedyPolicy* and *Q-Learning* functions. And do an analysis of the Q-Learning results.

GreedyPolicy

The GreedyPolicy function takes state coordinates as input and returns a greedy action for that state. In our Q-table we can find the expected reward of all possible actions for the given state, i.e., the expected reward of moving up, right, down or left. Using a GreedyPolicy we want to return the action that maximise the expected reward. Therefore we compare the expected reward of moving up, right, down or left and return the action (moving direction) with the highest expected reward. If two or more actions maximise the expected reward we sample one action uniformly from that set of actions, and return it.

```
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.
# The action that maximize the expected reward in the given state
# i.e., locate x,y in our q_table and check which direction that gives the highest expecte
d reward
exp_reward <- q_table[x,y,] # Save expected reward for possible actions at given state
max_value <- max(exp_reward) # Get the maximum expected reward
max_index <- which(exp_reward == max_value) # Get the positions in exp_reward that contain
s the max value
# Sample the action/direction with the highest exp_reward at random if we have more than 1
max
ifelse(length(max_index) == 1, action <- max_index,action <- sample(x=c(max_index),size=1)
)
return(action) # Return a greedy action
}
```

EpsilonGreedyPolicy

The *EpsilonGreedyPolicy* function takes state coordinates and epsilon as input and returns an epsilon greedy action for that state. Epsilon, ε, is a probability and the function returns a random action with probability ε and a greedy action with probability (1-ε). A random action is obtained by uniformly sampling an action from all possible actions, meaning at random we obtain an action move up, right, down, or left with equal probability. To obtain a greedy action we simply use the *GreedyPolicy* function described above.

```
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.
# Generates a random action
random_action <- sample(c(1,2,3,4),size=1)
# Extracts the action that maximize the expected return
greedy_action <- GreedyPolicy(x,y)
```

```
# Generate a random number between 0 and 1
random_number <- runif(1)
# Decision based on epsilon
# If the randomly generated number is less than or equal to epsilon we take a random actio
n
# otherwise we take a greedy action
ifelse(random_number <= epsilon, return(random_action),return(greedy_action))
}
```

Q-Learning

The *Q-Learning* function takes the coordinates of a state *S* and a bunch of constants (e.g., epsilon) as input. The function does the following:

- First, the current state, *S*, is set to input-state for which the reward is 0
- After that we complete an episode, which means that we move between states until we stumble upon a state with reward ≠ 0. Moving between states is done by:
    - Finding an action for the current state, *S,* by using our ε-greedy policy
    - The action from our policy is then plugged into our transition model which returns the next state, *S'*
    - By using *S'*:s coordinates we obtain its reward from our *reward map*
    - After this we calculate the *episode correction* (see details below) and use it to update the *Q-table* for state *S* given the action we got from our policy
    - *S* is set to *S'*
    - If the reward from S' was ≠ 0 we end the episode, otherwise start over finding an action for the current state *S* (previously *S'*)
- When the episode ends we return the most recently calculated reward and episode correction

In this exercise, *Exercise 2*, we perform 10 000 episodes of Q-Learning, i.e., we call the *Q-Learning* function 10 000 times.

The Q-table for state S given an action A is updated according to below:

$$Q(S,A) = Q(S,A) + EpisodeCorrection$$

Where the episode correction is calculated as below, *α* and *γ* are constants:

$$EpisodeCorrection = \alpha[Reward + \gamma max_a Q(S',a) - Q(S,A)]$$

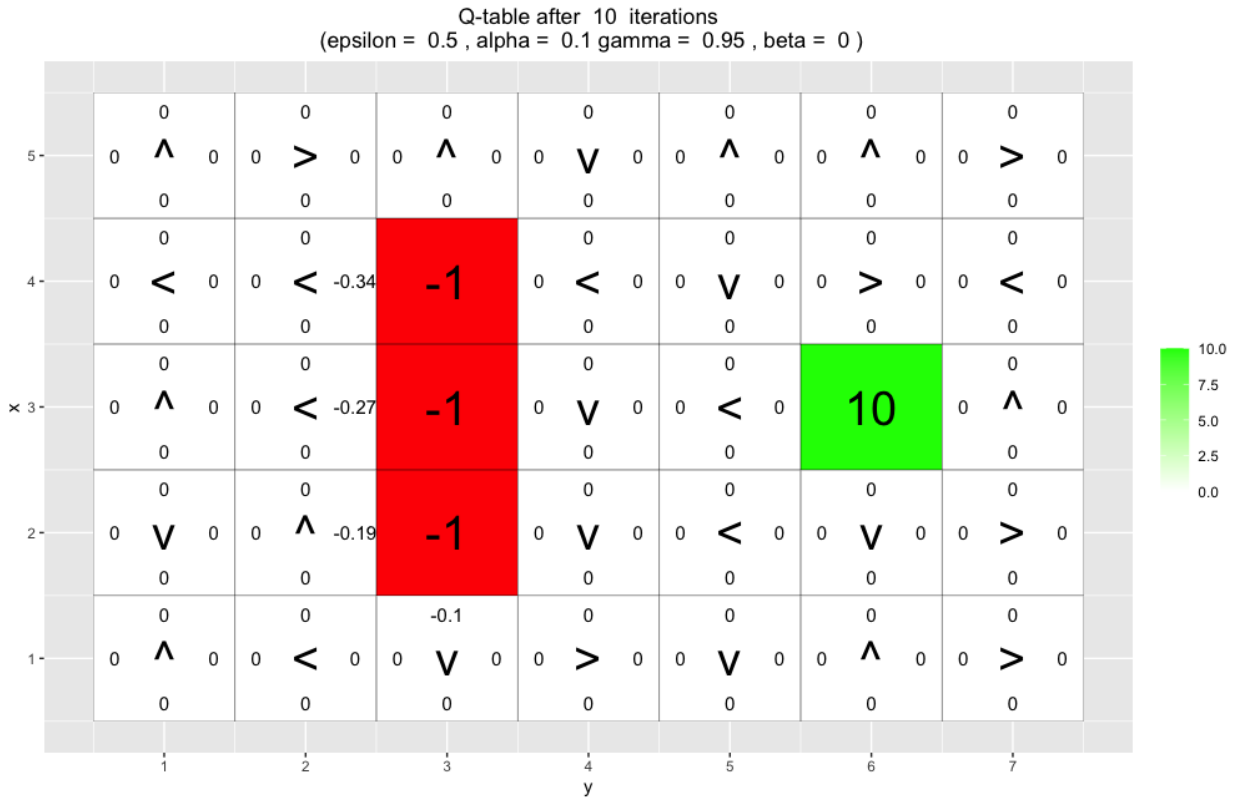
The transition model returns the next state, *S'.* With probability (1-β) the transition model use the action retrieved from our policy and returns the next state. But with probability β the transition model use the direction right or left of the action retrieved from our policy and returns the next state. Right or left is with equal probability, β/2.

```
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.
s <- start_state
episode_correction <- 0
repeat{
# Follow policy, execute action, get reward.
s.x <- s[1] # Save x coordinate for s
s.y <- s[2] # Save y coordinate for s
action <- EpsilonGreedyPolicy(s.x,s.y,epsilon)
sp <- transition_model(s.x,s.y,action,beta)
reward <- reward_map[sp[1],sp[2]]
# Q-table update.
sp.x <- sp[1] # Save x coordinate for s-prime
sp.y <- sp[2] # Save y coordinate for s-prime
sp.ga <- GreedyPolicy(sp.x,sp.y) # Get action that max reward for s-prime
```

```
sp.max_q <- q_table[sp.x,sp.y,sp.ga] # Get current reward for s-prime given greedy action
correction <- alpha*(reward+gamma*sp.max_q-q_table[s.x,s.y,action])
q_table[s.x,s.y,action] <<- q_table[s.x,s.y,action] + correction # Update Q
episode_correction <- episode_correction + correction # Increase episode correction
s <- sp # Assign s to s-prime
if(reward!=0)
# End episode.
return (c(reward,episode_correction))
}
}
```

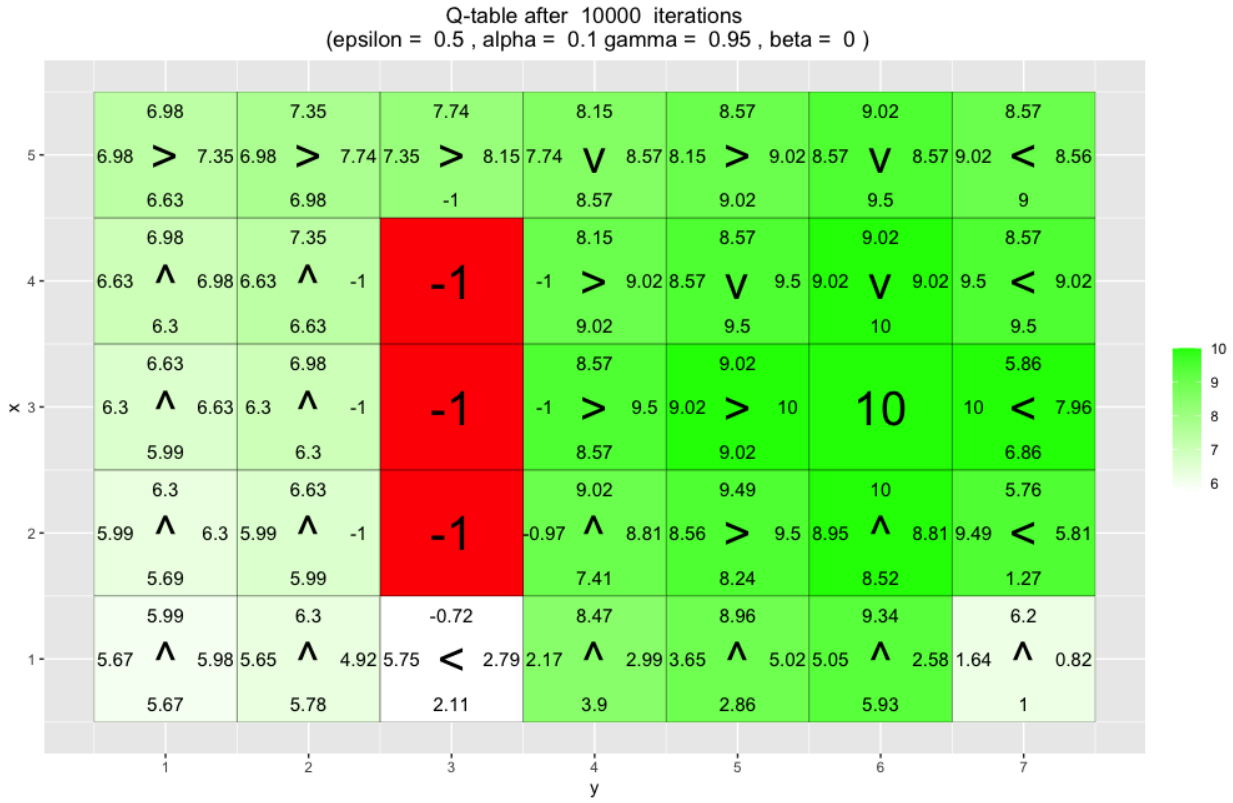
Q-Learning analysis

After the first 10 episodes the agent has learned that it's not beneficial to move to the states with negative reward.



After 10 000 episodes the greedy policy is not optimal for all states. If we consider the bottom left corner we note that in state (1, 3) for instance the greedy policy tells our agent to move left and that's away from the maximum reward of 10. Looking at it more generally the bottom part of our environment seems to be less explored. Our epsilon

value determines how probable it is that our agents explore new possibilities. Meaning if we have a higher epsilon value it's more likely for our agent to explore the bottom part of the environment. However, the greedy policy for states like (4, 5) might not have been optimal after 10 000 episodes like in this case. The values in the Q-table does not reflect the fact that there are paths above and below the the negative rewards to get the positive reward. To make this happen we can increase epsilon and thus the probability for our agent to explore new possibilities.



Exercise 3 – Environment B

Looking at the tuning of epsilon and gamma we note that:

- Higher gammas makes the agent value a greater reward later higher than a smaller reward right away
- With an epsilon equal to 0.5 the agent finds the reward of 10 and choose it over the reward of 5 and for higher gammas it choose the reward of 10 more frequently than for lower

- With an epsilon equal to 0.1 the agent choose the greedy action almost every time and doesn't explore new possibilities. Since the agent starts at (4, 1) it find the reward 5 first and with an epsilon of 0.1 it doesn't explore new possibilities after that

Below follow a more detailed analysis for the different values of epsilon and gamma

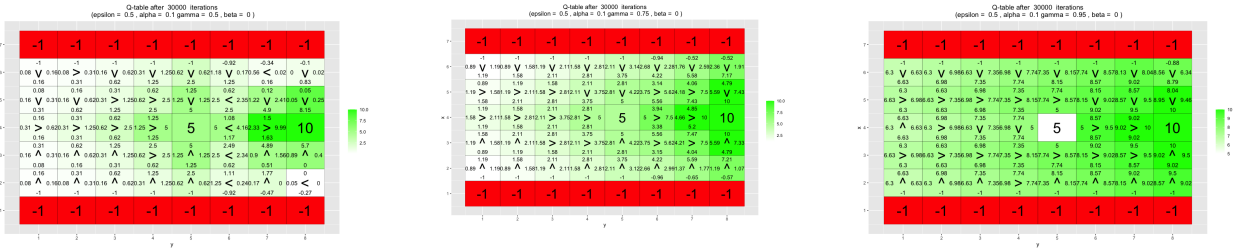
Epsilon equal to 0.5 and gamma equal to 0.5, 0.75, or 0.95

Using these settings we note:

- When gamma equals 0.95 the reward of 5 becomes completely uninteresting for the agent which makes sense because gamma represents the trade-off between immediate reward and reward later. As gamma increases the agents prefers a greater reward later in time and thus are more willing to explore a larger part of the environment in detail to find it
- The moving average for the reward stabilise around 4.5 for gamma equal to 0.5 and 0.75 but for gamma equal to 0.95 it initially stabilise around 4.5 but moves up to about 6.5 after 15 000 episodes which makes sense given our observations of the environment and greedy actions in it
- As we iterate trough episodes the moving average for the episode correction decreases. For gamma = 0.95 we observe a spike for it after 15 000 episodes which is when the moving average for the reward increases. This makes sense since the agent changes behaviour and starts favouring the 10 reward over the 5 we need to update the Q-table a lot more

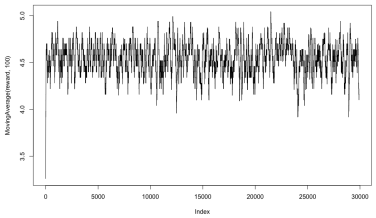
<u>Environment for different gammas</u>

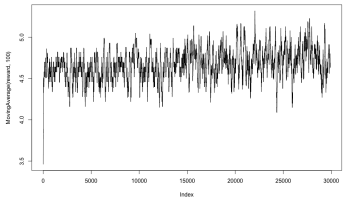
Gamma = 0.5 Gamma = 0.75 Gamma = 0.95

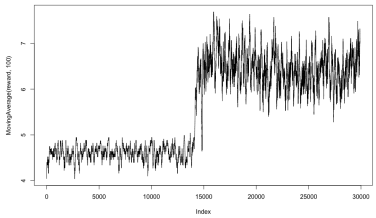


<u>Moving average for the reward with different gammas</u>

Gamma = 0.5 Gamma = 0.75 Gamma = 0.95

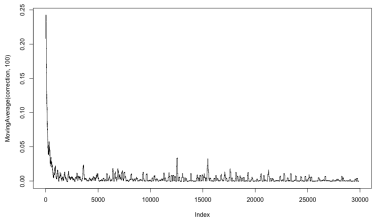


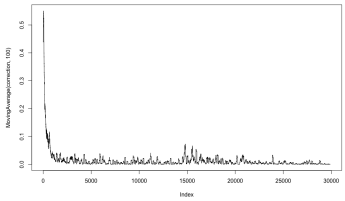


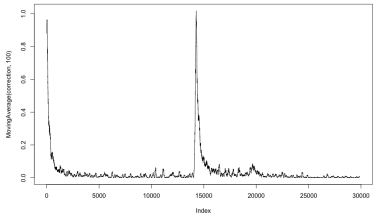


Moving average for the episode correction term with different gammas

Gamma = 0.5 Gamma = 0.75 Gamma = 0.95







Epsilon equal to 0.1 and gamma equal to 0.5, 0.75, or 0.95

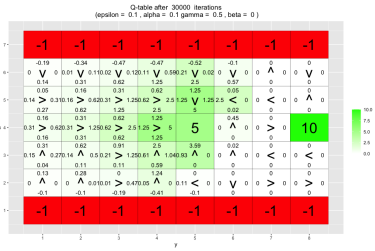
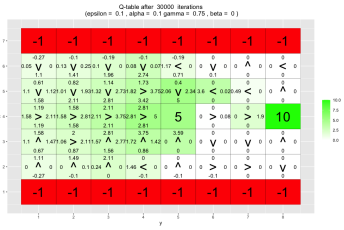
Using these settings we note:

- Using an epsilon of 0.1 the agent isn't exploring new possibilities in the same extent as with a higher epsilon. The agent starts in state (4, 1) and the first positive reward it stumbles upon is the 5 reward. Since we are using a low epsilon, greedy actions very more frequent and the agent are having a hard time finding the reward of 10 for all gammas. The reason for why the agent was able to find the reward of 10 for gamma equal to 0.75 and not for the higher gamma, 0.95, are probability just a coincidence
- The moving average reward is equal to 5 for (almost) every episode for the different gammas. This makes sense if we look at the environments where we note that the states around the reward of 10 is pretty much unexplored
- The moving average correction term is slightly higher for larger gammas but stabilise close to 0 after about 2 500 episodes for all gammas. One exception is for

gamma = 0.5 where we observe a minor spike in the moving average correction term after roughly 4 000 episodes. This likely due to the robot finding a new “much” quicker path to the 5 reward than previously

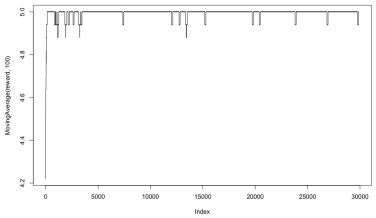
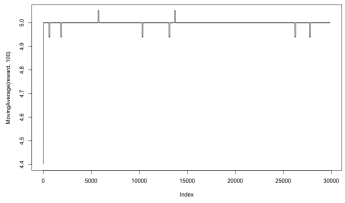
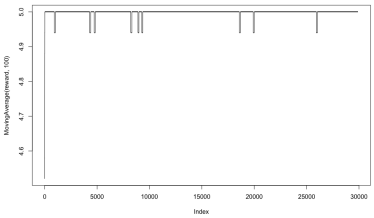
Environment for different gammas

Gamma = 0.5 Gamma = 0.75 Gamma = 0.95

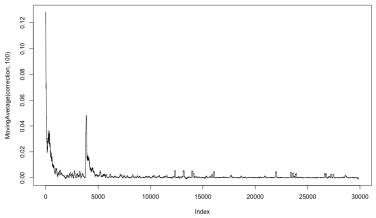
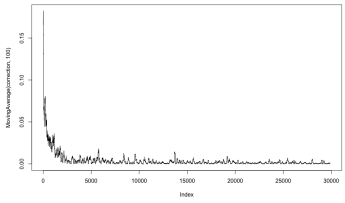
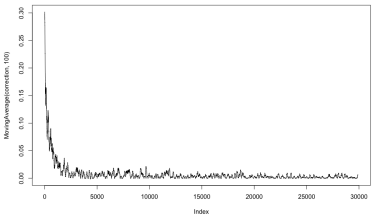

Moving average for the reward with different gammas

Gamma = 0.5 Gamma = 0.75 Gamma = 0.95

Moving average for the episode correction term with different gammas

Gamma = 0.5 Gamma = 0.75 Gamma = 0.95

Exercise 4 – Environment C

Beta is the probability that the agent slips and moves to the right or to the left of the chosen action given by our policy. The probability of left or right is equal, β/2. So, the higher the beta the higher the probability is that the agent slips and moves right or left of the chosen action. If we for instance compare the greediest action give by our policy in the initial state we note that for:

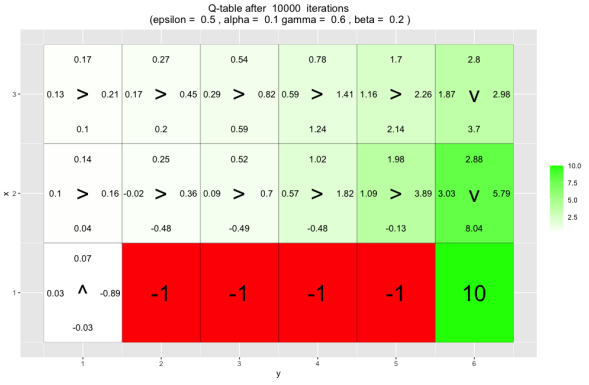
- β = 0.2 we chose to move up. The probability of moving right into a state with negative reward is thus 0.1, which is pretty low
- β = 0.4 and β = 0.66 we chose to move left. The probability of moving right into a state with negative reward is therefore 0. However, if we would chose to move up it would be 0.2 and 0.33 respectively, which is significantly higher than for β = 0.2. For these β:s the agent therefore tries to move left until it slips up and move to the right of the attended action, i.e., moving up.

Environments for different betas

When comparing the 3 environments for different betas we note that.

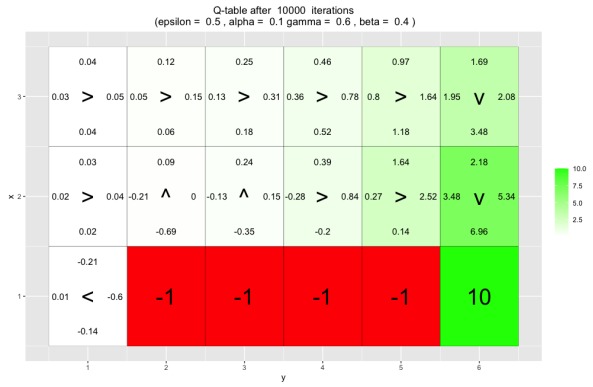
- For beta = 0.2 the greediest action in each state moves the agent closer to the positive reward
- The higher beta gets the more keen the agent is to stay away from the states with negative rewards

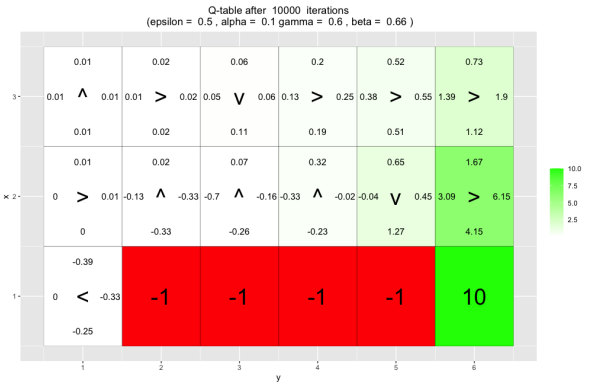
Beta = 0.2



Beta = 0.4

Beta = 0.66





Exercise 5 – REINFORCE

Description summary:

- The agent now starts at a random position
- The environment consist of a $4x4$ dimensional grid
- The goal position is random and its coordinates will be given to the agent, expanding the state to consist of 4 coordinates
- Initially the agent doesn't know what the additional coordinates corresponds to the coordinates of the goal position
- Actions of the agent can still only affects its own position
- The policy depends on the goal position and its coordinates are taken as input to the function
- We are given an implementation of the REINFORCE algorithm

Exercise 6 – Environment D

8 goal positions are used for training and the other 8 are used for validation ($4x4$ grid). The pictures presented below corresponds to the environment running the REINFORCE algorithm for 0 and 5 000 episodes for the validation positions with β = 0 and γ = 0.95. We note that the goal position for the validation and training data is pretty evenly distributed in the environment.

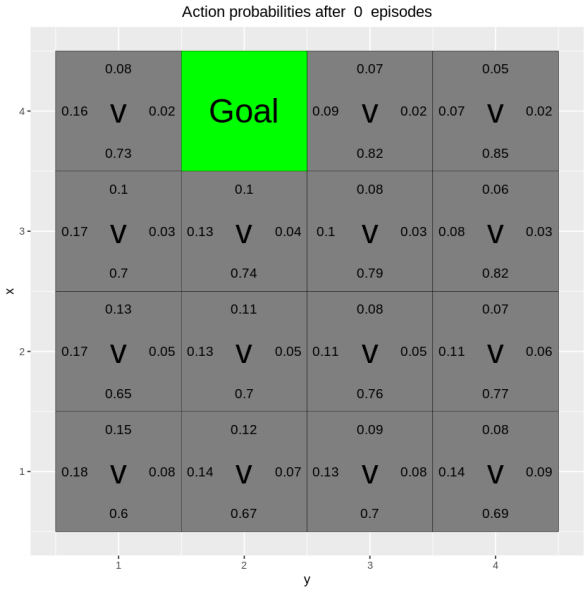
Without running any episodes the most probable action (the arrow direction for each state) given by our policy is down for all states, no matter where the goal position is. After running 5 000 episodes the agent seem to have figured out what the additional coordinates in the state corresponds to (the goal positions). We note this by looking at the arrows, where we observe that the arrows more often than not points in the direction of the goal position. Further the agent has figured out that trying to move outside the environment result in no reward, e.g., moving to the right while being in a state where *y = 4*. The agent seems to have learned a good policy based on the training data, because it's able to identify the goal position in the validation data.

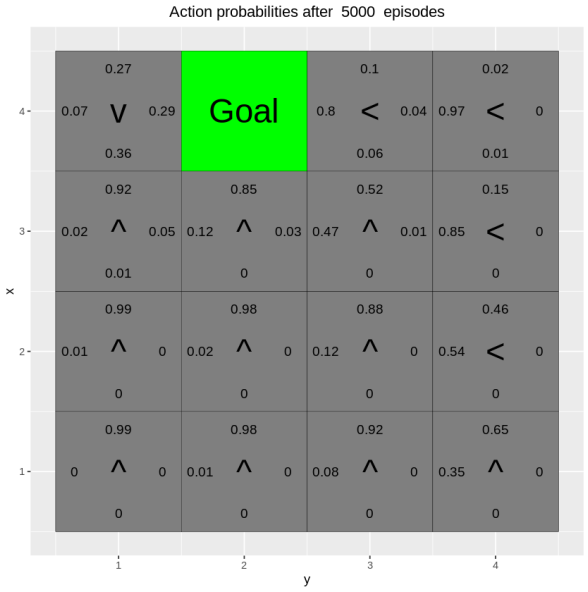
Using regular Q-Learning to solve this task might be possible but it would be better to use Deep Q-Learning because:

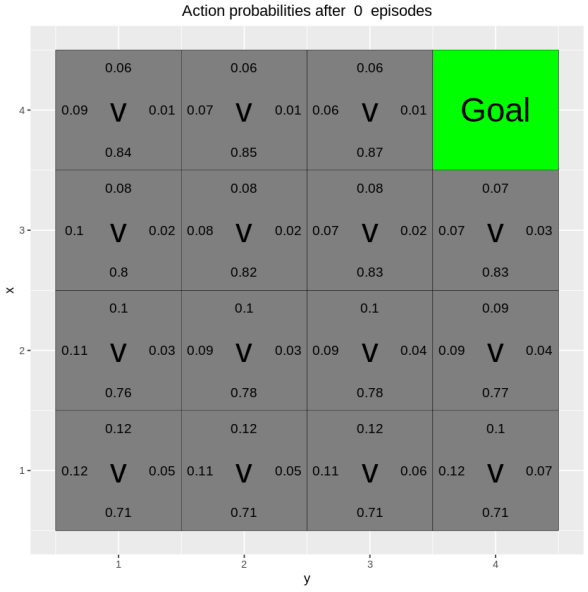
- Both the agents' starting position and the goals' position are randomised. Given our $4x4$ grid this means $16x16 = 256$ combinations for the state space. Multiplying this with our 4 possible actions for each state we get a Q-table with 1 024 which might not be feasible to look-up values in due to storage space and time to convergence
- Using Deep Q-Learning we can represent the state/action space as a parameterised function instead of a table which brings several advantages:
    - Fewer parameters and therefore easier to learn and reach convergence
    - Less storage space
    - Generalisation to unvisited state-action pairs, something that regular Q-Learning isn't great at since the state/action space is represented as discrete values using a Q-table
- To represent the state/action space as a parameterised function, a Neural Network is typically used

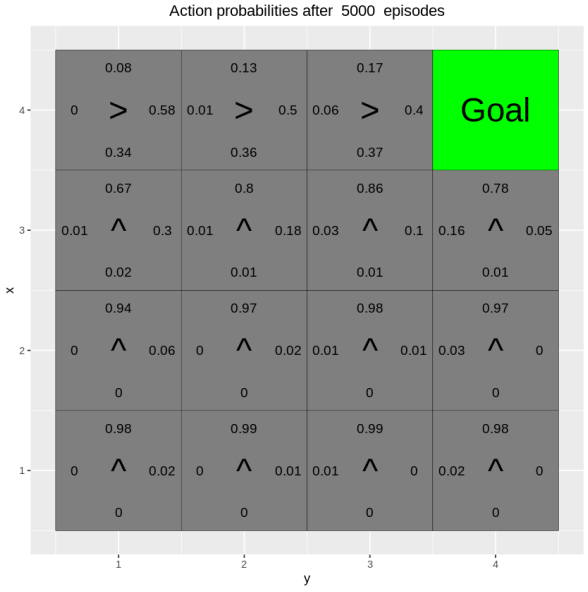
Validation goal positions: 0 episodes

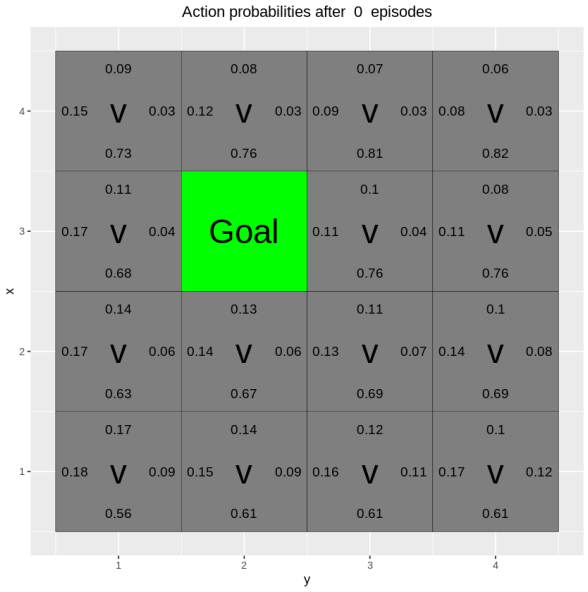
Validation goal positions: 5 000 episodes

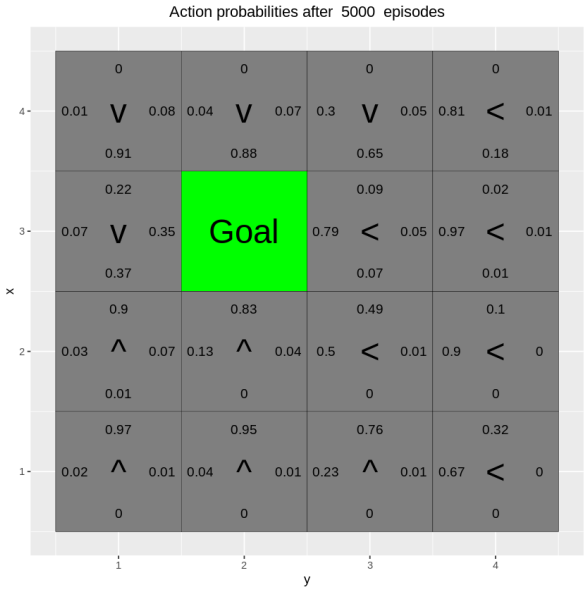

Action probabilities after 0 episodes
0.08 0.16 V 0.02 0.73
Goal
0.07 0.09 V 0.02 0.82
0.05 0.07 V 0.02 0.85
0.1 0.17 V 0.03 0.7
0.1 0.13 V 0.04 0.74
0.08 0.1 V 0.03 0.79
0.06 0.08 V 0.03 0.82
0.13 0.17 V 0.05 0.65
0.11 0.13 V 0.05 0.7
0.08 0.11 V 0.05 0.76
0.07 0.11 V 0.06 0.77
0.15 0.18 V 0.08 0.6
0.12 0.14 V 0.07 0.67
0.09 0.13 V 0.08 0.7
0.08 0.14 V 0.09 0.69
x
y
1
2
3
4

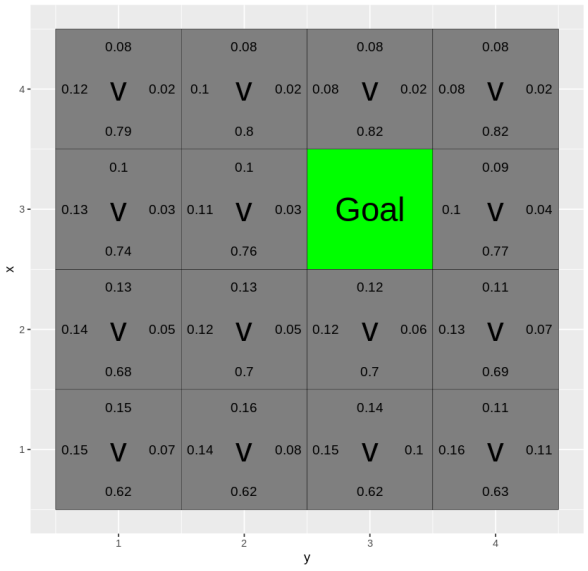

Action probabilities after 5000 episodes
0.27 0.07 V 0.29 0.36
Goal
0.1 0.8 < 0.04 0.06
0.02 0.97 < 0 0.01
0.92 0.02 ^ 0.05 0.01
0.85 0.12 ^ 0.03 0
0.52 0.47 ^ 0.01 0
0.15 0.85 < 0 0
0.99 0.01 ^ 0 0
0.98 0.02 ^ 0 0
0.88 0.12 ^ 0 0
0.46 0.54 < 0 0
0.99 0 ^ 0 0
0.98 0.01 ^ 0 0
0.92 0.08 ^ 0 0
0.65 0.35 ^ 0 0
x
y
1
2
3
4

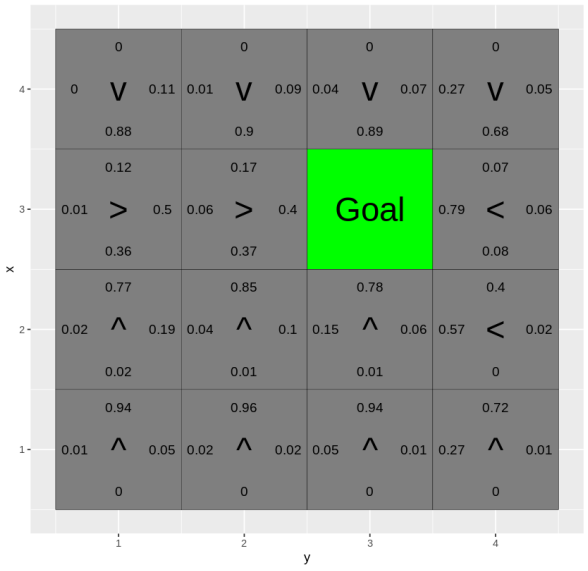

Action probabilities after 0 episodes
0.06 0.09 V 0.01 0.84
0.06 0.07 V 0.01 0.85
0.06 0.06 V 0.01 0.87
Goal
0.08 0.1 V 0.02 0.8
0.08 0.08 V 0.02 0.82
0.08 0.07 V 0.02 0.83
0.07 0.07 V 0.03 0.83
0.1 0.11 V 0.03 0.76
0.1 0.09 V 0.03 0.78
0.1 0.09 V 0.04 0.78
0.09 0.09 V 0.04 0.77
0.12 0.12 V 0.05 0.71
0.12 0.11 V 0.05 0.71
0.12 0.11 V 0.06 0.71
0.1 0.12 V 0.07 0.71
x
y
1
2
3
4

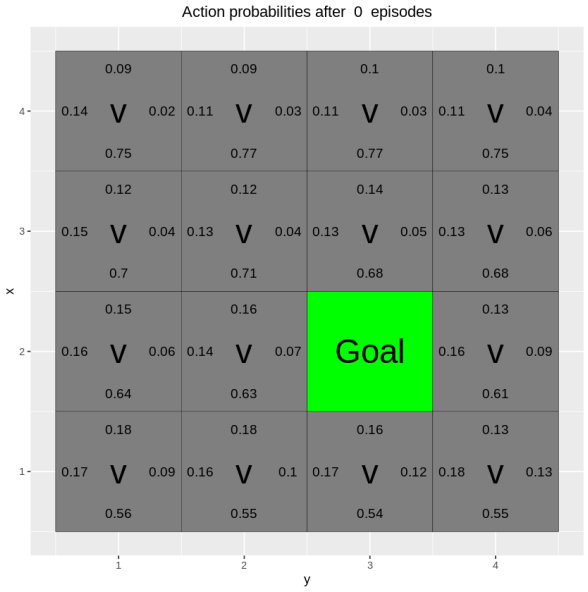

Action probabilities after 5000 episodes
0.08 0 > 0.58 0.34
0.13 0.01 > 0.5 0.36
0.17 0.06 > 0.4 0.37
Goal
0.67 0.01 ^ 0.3 0.02
0.8 0.01 ^ 0.18 0.01
0.86 0.03 ^ 0.1 0.01
0.78 0.16 ^ 0.05 0.01
0.94 0 ^ 0.06 0
0.97 0 ^ 0.02 0
0.98 0.01 ^ 0.01 0
0.97 0.03 ^ 0 0
0.98 0 ^ 0.02 0
0.99 0 ^ 0.01 0
0.99 0.01 ^ 0 0
0.98 0.02 ^ 0 0
x
y
1
2
3
4

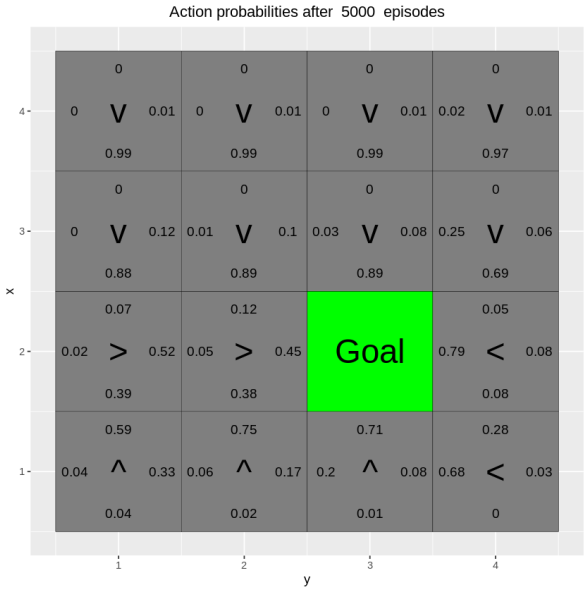

Action probabilities after 0 episodes
0.09 0.08 0.07 0.06
0.15 V 0.03 0.12 V 0.03 0.09 V 0.03 0.08 V 0.03
0.73 0.76 0.81 0.82
0.11 0.1 0.08
0.17 V 0.04 Goal 0.11 V 0.04 0.11 V 0.05
0.68 0.76 0.76
0.14 0.13 0.11 0.1
0.17 V 0.06 0.14 V 0.06 0.13 V 0.07 0.14 V 0.08
0.63 0.67 0.69 0.69
0.17 0.14 0.12 0.1
0.18 V 0.09 0.15 V 0.09 0.16 V 0.11 0.17 V 0.12
0.56 0.61 0.61 0.61
x
y
1 2 3 4

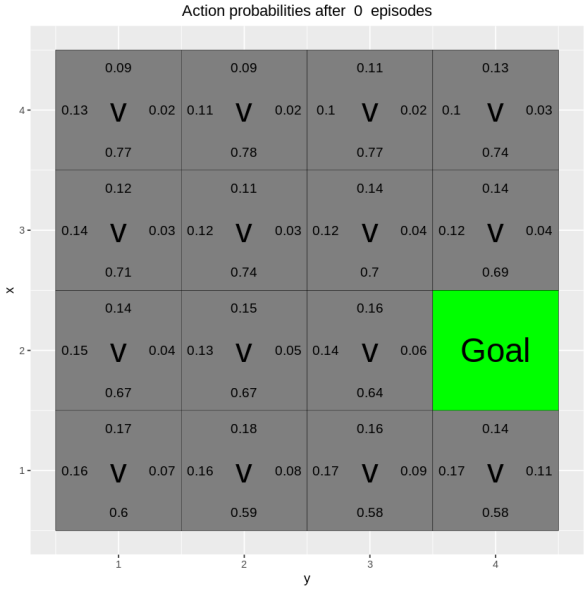

Action probabilities after 5000 episodes
0 0 0 0
0.01 V 0.08 0.04 V 0.07 0.3 V 0.05 0.81 < 0.01
0.91 0.88 0.65 0.18
0.22 0.09 0.02
0.07 V 0.35 Goal 0.79 < 0.05 0.97 < 0.01
0.37 0.07 0.01
0.9 0.83 0.49 0.1
0.03 ^ 0.07 0.13 ^ 0.04 0.5 < 0.01 0.9 < 0
0.01 0 0 0
0.97 0.95 0.76 0.32
0.02 ^ 0.01 0.04 ^ 0.01 0.23 ^ 0.01 0.67 < 0
0 0 0 0
x
y
1 2 3 4

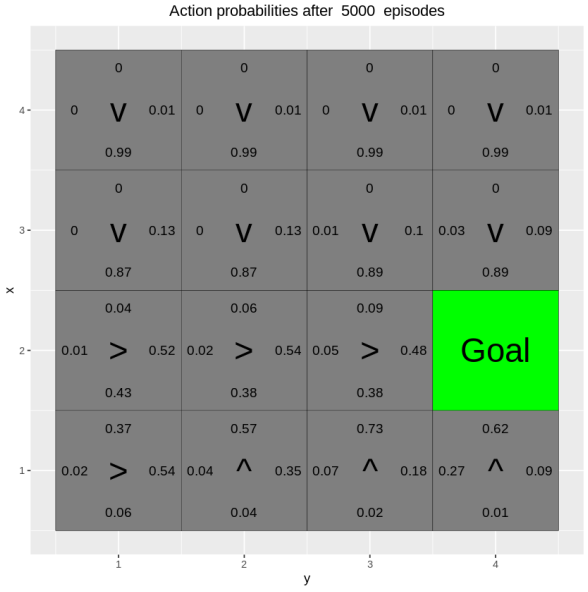

Action probabilities after 0 episodes
0.08 0.08 0.08 0.08
0.12 V 0.02 0.1 V 0.02 0.08 V 0.02 0.08 V 0.02
0.79 0.8 0.82 0.82
0.1 0.1 0.09
0.13 V 0.03 0.11 V 0.03 Goal 0.1 V 0.04
0.74 0.76 0.77
0.13 0.13 0.12 0.11
0.14 V 0.05 0.12 V 0.05 0.12 V 0.06 0.13 V 0.07
0.68 0.7 0.7 0.69
0.15 0.16 0.14 0.11
0.15 V 0.07 0.14 V 0.08 0.15 V 0.1 0.16 V 0.11
0.62 0.62 0.62 0.63
x
y
1 2 3 4

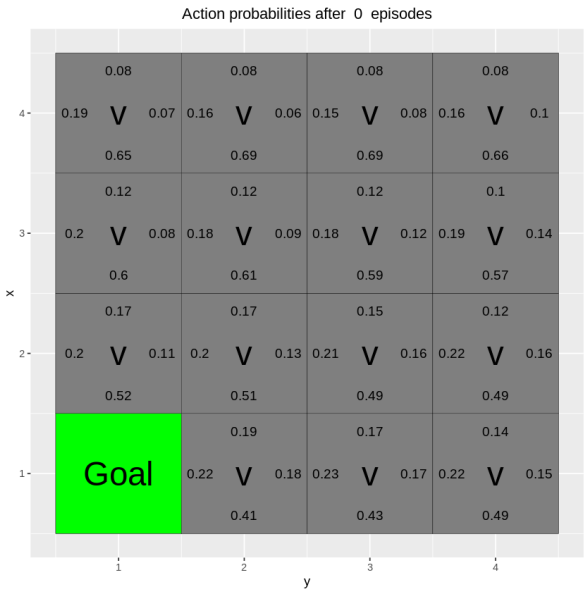

Action probabilities after 5000 episodes
0 0 0 0
0 V 0.11 0.01 V 0.09 0.04 V 0.07 0.27 V 0.05
0.88 0.9 0.89 0.68
0.12 0.17 0.07
0.01 > 0.5 0.06 > 0.4 Goal 0.79 < 0.06
0.36 0.37 0.08
0.77 0.85 0.78 0.4
0.02 ^ 0.19 0.04 ^ 0.1 0.15 ^ 0.06 0.57 < 0.02
0.02 0.01 0.01 0
0.94 0.96 0.94 0.72
0.01 ^ 0.05 0.02 ^ 0.02 0.05 ^ 0.01 0.27 ^ 0.01
0 0 0 0
x
y
1 2 3 4

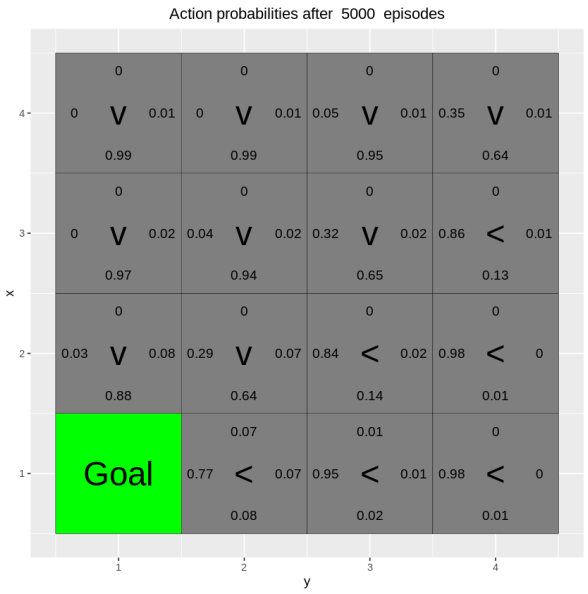

Action probabilities after 0 episodes
0.09 0.09 0.1 0.1
0.14 V 0.02 0.11 V 0.03 0.11 V 0.03 0.11 V 0.04
0.75 0.77 0.77 0.75
0.12 0.12 0.14 0.13
0.15 V 0.04 0.13 V 0.04 0.13 V 0.05 0.13 V 0.06
0.7 0.71 0.68 0.68
0.15 0.16 0.13
0.16 V 0.06 0.14 V 0.07 Goal 0.16 V 0.09
0.64 0.63 0.61
0.18 0.18 0.16 0.13
0.17 V 0.09 0.16 V 0.1 0.17 V 0.12 0.18 V 0.13
0.56 0.55 0.54 0.55
x
y
1 2 3 4

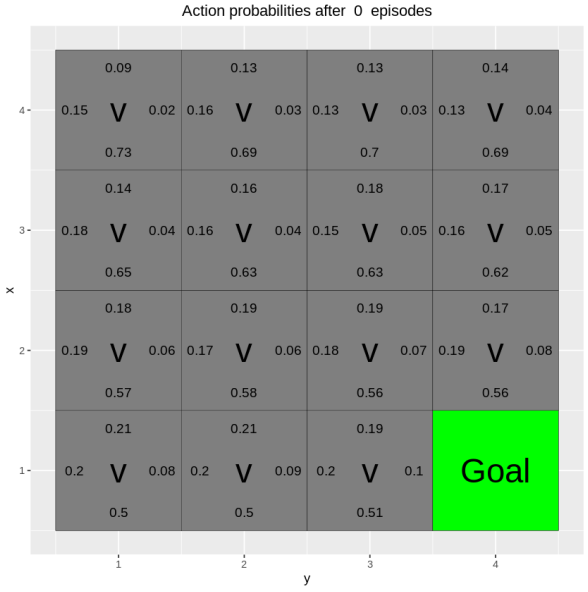

Action probabilities after 5000 episodes
0 0 0 0
0 V 0.01 0 V 0.01 0 V 0.01 0.02 V 0.01
0.99 0.99 0.99 0.97
0 0 0 0
0 V 0.12 0.01 V 0.1 0.03 V 0.08 0.25 V 0.06
0.88 0.89 0.89 0.69
0.07 0.12 0.05
0.02 > 0.52 0.05 > 0.45 Goal 0.79 < 0.08
0.39 0.38 0.08
0.59 0.75 0.71 0.28
0.04 ^ 0.33 0.06 ^ 0.17 0.2 ^ 0.08 0.68 < 0.03
0.04 0.02 0.01 0
x
y
1 2 3 4

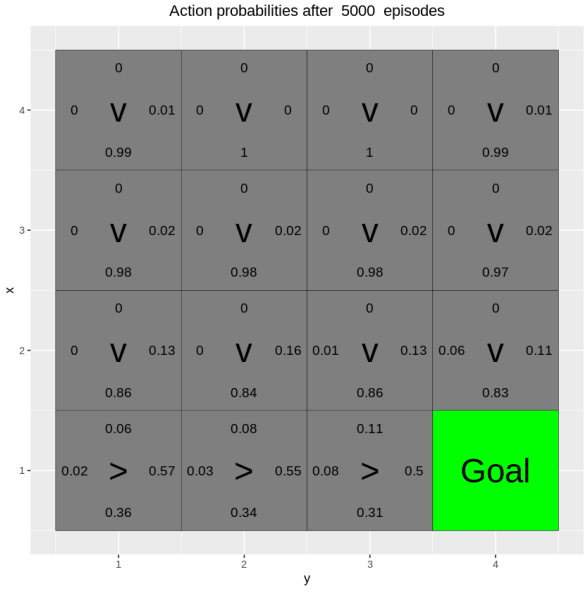

Action probabilities after 0 episodes
0.09 0.09 0.11 0.13
0.13 V 0.02 0.11 V 0.02 0.1 V 0.02 0.1 V 0.03
0.77 0.78 0.77 0.74
0.12 0.11 0.14 0.14
0.14 V 0.03 0.12 V 0.03 0.12 V 0.04 0.12 V 0.04
0.71 0.74 0.7 0.69
0.14 0.15 0.16
0.15 V 0.04 0.13 V 0.05 0.14 V 0.06 Goal
0.67 0.67 0.64
0.17 0.18 0.16 0.14
0.16 V 0.07 0.16 V 0.08 0.17 V 0.09 0.17 V 0.11
0.6 0.59 0.58 0.58
x
y
1 2 3 4


Action probabilities after 5000 episodes
0 0 0 0
0 V 0.01 0 V 0.01 0 V 0.01 0 V 0.01
0.99 0.99 0.99 0.99
0 0 0 0
0 V 0.13 0 V 0.13 0.01 V 0.1 0.03 V 0.09
0.87 0.87 0.89 0.89
0.04 0.06 0.09
0.01 > 0.52 0.02 > 0.54 0.05 > 0.48 Goal
0.43 0.38 0.38
0.37 0.57 0.73 0.62
0.02 > 0.54 0.04 ^ 0.35 0.07 ^ 0.18 0.27 ^ 0.09
0.06 0.04 0.02 0.01
x
y
1 2 3 4







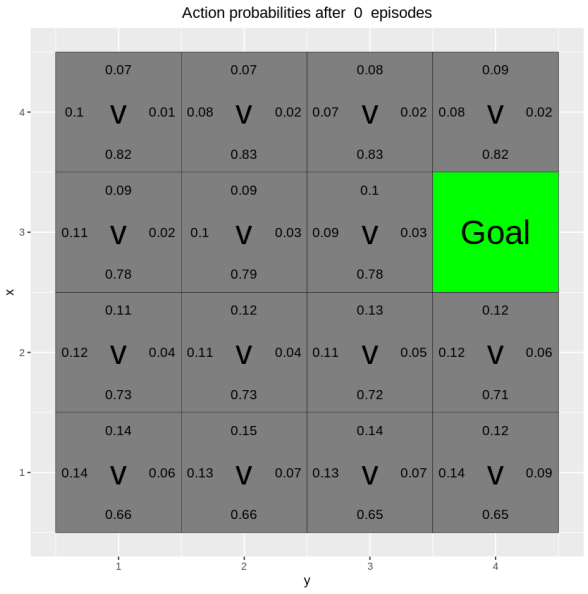


Exercise 7 – Environment E

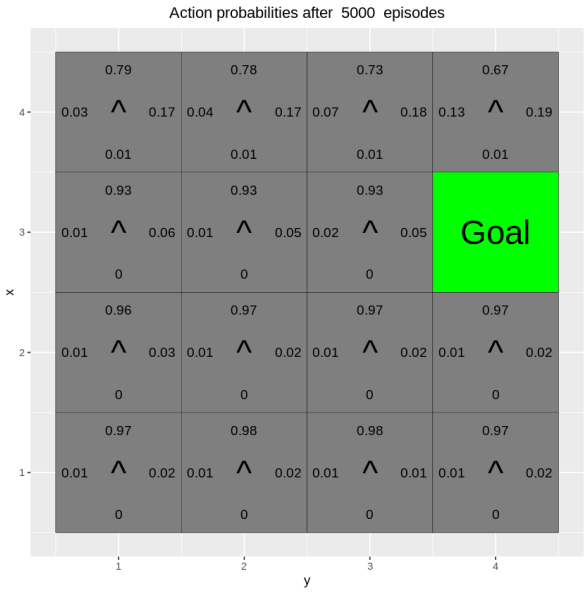
In this exercise 4 goal positions are used for training and the 3 are used for validation. The goal positions used for training are all from the top row of the grid. The pictures presented below corresponds to the environment running the REINFORCE algorithm for 0 and 5 000 episodes for the validation positions with β = 0 and γ = 0.95.

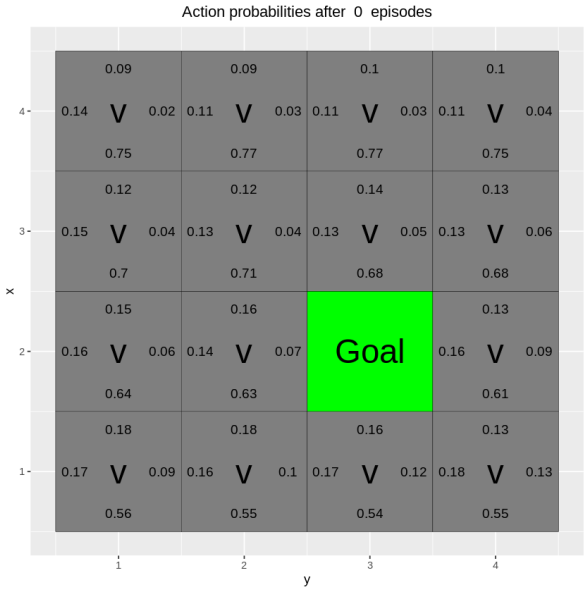
Similar to the previous exercise we note that, without running any episodes the most probable action (the arrow direction for each state) given by our policy is down for all states, no matter where the goal position is. After running 5 000 episodes the agent seem to have figured that moving up results in a reward, not a single arrow points down for our 3 validation goals. The agent isn't as great at adapting to the goal positions given by the validation data as in the previous exercise. This makes sense since the training goals in the previous exercise represents the validation goals better and contains a greater variation, covering more than the top row in the environment. The agent seems to have learned a bad policy based on the given training data and is biased towards the training data.

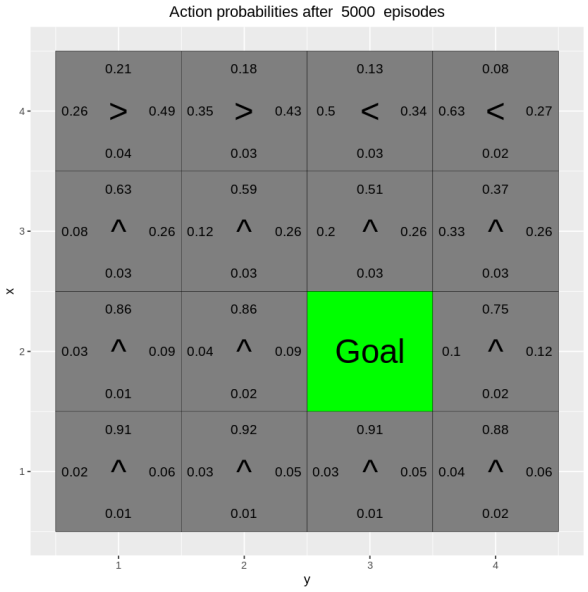
Validation goal positions: 0 episodes

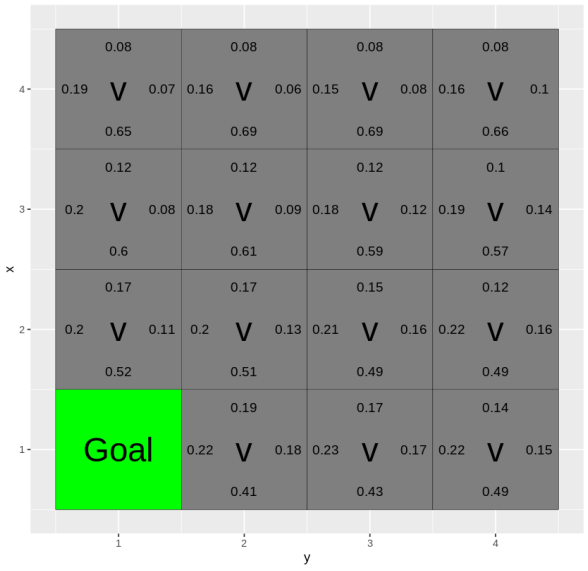


Validation goal positions: 5 000 episodes




Action probabilities after 0 episodes
0.09 0.09 0.1 0.1
0.14 V 0.02 0.11 V 0.03 0.11 V 0.03 0.11 V 0.04
0.75 0.77 0.77 0.75
0.12 0.12 0.14 0.13
0.15 V 0.04 0.13 V 0.04 0.13 V 0.05 0.13 V 0.06
0.7 0.71 0.68 0.68
0.15 0.16 0.13
0.16 V 0.06 0.14 V 0.07 Goal 0.16 V 0.09
0.64 0.63 0.61
0.18 0.18 0.16 0.13
0.17 V 0.09 0.16 V 0.1 0.17 V 0.12 0.18 V 0.13
0.56 0.55 0.54 0.55
x
y
1
2
3
4


Action probabilities after 5000 episodes
0.21 0.18 0.13 0.08
0.26 > 0.49 0.35 > 0.43 0.5 < 0.34 0.63 < 0.27
0.04 0.03 0.03 0.02
0.63 0.59 0.51 0.37
0.08 ^ 0.26 0.12 ^ 0.26 0.2 ^ 0.26 0.33 ^ 0.26
0.03 0.03 0.03 0.03
0.86 0.86 0.75
0.03 ^ 0.09 0.04 ^ 0.09 Goal 0.1 ^ 0.12
0.01 0.02 0.02
0.91 0.92 0.91 0.88
0.02 ^ 0.06 0.03 ^ 0.05 0.03 ^ 0.05 0.04 ^ 0.06
0.01 0.01 0.01 0.02
x
y
1
2
3
4


Action probabilities after 0 episodes
0.08 0.08 0.08 0.08
0.19 V 0.07 0.16 V 0.06 0.15 V 0.08 0.16 V 0.1
0.65 0.69 0.69 0.66
0.12 0.12 0.12 0.1
0.2 V 0.08 0.18 V 0.09 0.18 V 0.12 0.19 V 0.14
0.6 0.61 0.59 0.57
0.17 0.17 0.15 0.12
0.2 V 0.11 0.2 V 0.13 0.21 V 0.16 0.22 V 0.16
0.52 0.51 0.49 0.49
0.19 0.17 0.14
Goal 0.22 V 0.18 0.23 V 0.17 0.22 V 0.15
0.41 0.43 0.49
x
y
1
2
3
4


Action probabilities after 5000 episodes
0.01 0.01 0 0
0.68 < 0.28 0.81 < 0.17 0.86 < 0.13 0.89 < 0.1
0.03 0.02 0.01 0.01
0.04 0.03 0.02 0.01
0.6 < 0.31 0.72 < 0.21 0.78 < 0.17 0.83 < 0.14
0.05 0.04 0.03 0.02
0.17 0.11 0.06 0.04
0.43 < 0.32 0.57 < 0.26 0.67 < 0.21 0.71 < 0.18
0.08 0.07 0.06 0.07
0.41 0.28 0.21
Goal 0.27 ^ 0.23 0.36 < 0.22 0.43 < 0.2
0.09 0.14 0.17
x
y
1
2
3
4