# Revised - Lab 4 Reinforcement learning

See changes marked in red

## Q1. Define the V- and Q-function given an optimal policy. Use equations and describe what they represent. (See lectures/classes)

The Value function (V) describes the expected optimal amount of future rewards for each given state. The Q-function describes the optimal value (of transitions) between states given a certain action in a certain state.

## Q2. Define a learning rule (equation) for the Q-function and describe how it works.

In the training, we update Q by the formula from lecture slides:

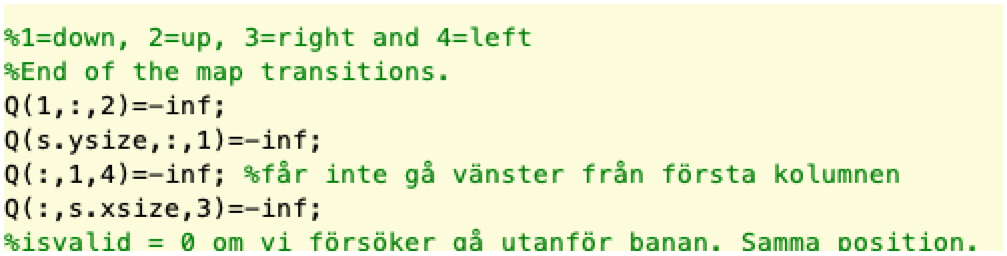
$$\hat{Q}(s_k, a_j) \leftarrow (1-\eta)\hat{Q}(s_k, a_j) + \eta\left(r + \gamma \max_a \hat{Q}(s_{k+1}, a)\right)$$

*Figure 1. Training formula*

We use the Q value in the old position and multiply with one minus the learning rate. This is then added to the learning rate multiplied with the reward and discount factor and the max Q value from all actions and Q values in the new position.

## Q3. Briefly describe your implementation, especially how you hinder the robot from exiting through the borders of a world.

We prevented the robot from going out of bounds by defining the Q value (transition) to negative infinity for the actions (column or row) that will lead us out of bounds.



```
%1=down, 2=up, 3=right and 4=left
%End of the map transitions.
Q(1,:,2)=-inf;
Q(s.ysize,:,1)=-inf;
Q(:,1,4)=-inf; %får inte gå vänster från första kolumnen
Q(:,s.xsize,3)=-inf;
%isvalid = 0 om vi försöker gå utanför banan. Samma position.
```

After this boundary the robot's Q-function was updated with random actions and the function get action this was done while the robot was not in goal. After this, a new iteration started with a little lower epsilon and otherwise the same initial parameters. After n iterations and updates of the Q-function. After all training has been done we update the optimal V-matrix and optimal policy (P) from the Q-matrix by using a max function in the main script.

[V,P] = max(Q,[],3), which returns two matrices. One with our max value in each state (V) from the Q matrix and also the optimal policy (The index) from Q. So P is a matrix with numbers between 1 and 4 and V is "the expected future reward in each state.

## Q4. Describe World 1. What is the goal of the reinforcement learning in this world? What parameters did you use to solve this world? Plot the policy and the V-function.

World 1 has the same "irritating blob" present at all times. The goal is static, but the starting point is random. The goal of the reinforcement learning is to train enough times to explore the entire world, to generate the optimal policy for the entire world. By having a large (close to one) epsilon, we will encourage the robot to explore the world and therefore find the optimal policy with fewer iterations, though every iteration might be longer. Since the world is static, a learning rate of 1 is optimal, since the knowledge we gain from each coordinate will remain.

We used the following parameters:
Learning rate = 1
epsilon = 0.9
gamma = 0.9
training iterations = 200.

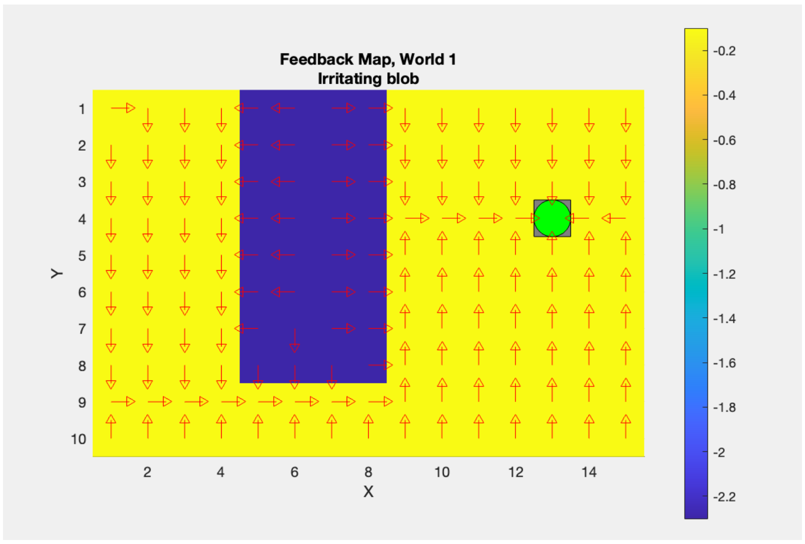


*Figure 2. Policy for world 1.*

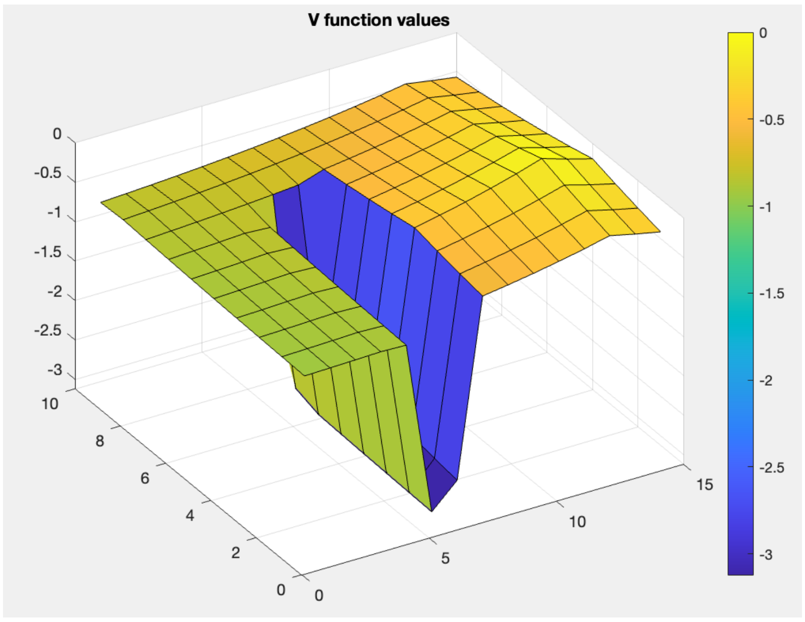


*Figure 3. V-function values for world 1.*

Q5. Describe World 2. What is the goal of the reinforcement learning in this world? This world has a hidden trick. Describe the trick and why this can be solved with reinforcement learning. What parameters did you use to solve this world? Plot the policy and the V-function.

World 2 has a static goal, random starting position and a chance of containing the “irritating blob” in 20% of the initializations. This leads to that the area of the irritating blob will affect the policy of the world. Depending on what risk you want to take, the algorithm can be tuned to always avoid this area, and not risk losing reward in 20 % of the cases. The other case would be to be greedy and always choose the shortest path and risk to lose reward by passing through the irritating blob in 20% of all cases.

We chose to try to avoid the area, by focusing on a small learning rate, keeping knowledge of the rare case of the irritating blob. We also introduce a dynamic exploration factor which explores a lot in the beginning and chooses the optimal action more often in the end of training. Gamma is high to focus on long time reward to include the occasional irritating blob.

Chosen parameters:
epsilon = 0.99 - (0.5 * i / training), i = current training iteration, training = 1..1100
gamma = 0.95
learning rate = 0.2
training iterations = 1100

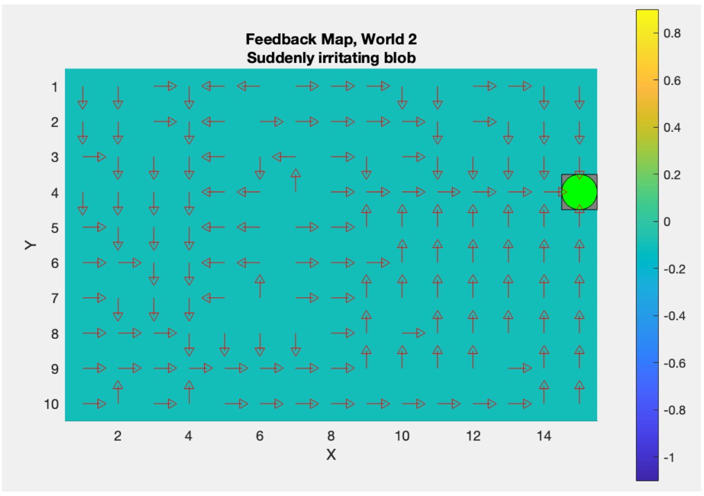


*Figure 4. Policy for world 2.*

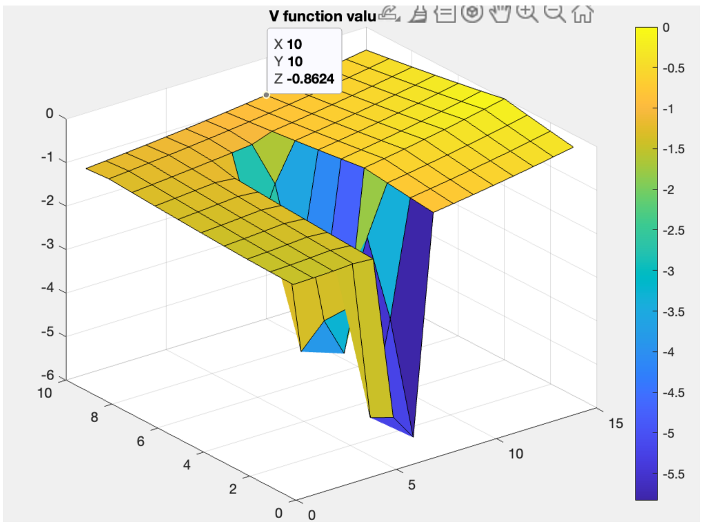


*Figure 5. V-function values for world 2.*

Q6. Describe World 3. What is the goal of the reinforcement learning in this world? Is it possible to get a good policy from every state in this world, and if so how? What parameters did you use to solve this world? Plot the policy and the V-function.

World 3 is static. The starting point and goal are always the same, as well as the purple area and the short cut through the purple area. Therefore, we can use a learning rate of 1 and just make sure that we find the shortcut. If we want to, we can have a high epsilon to find the optimal policy for every coordinate, but since the starting point and goal are fixed, we are really only interested in finding the shortest way to the goal (when avoiding the purple area). It can be compared to that I don't need to know the shortest path to LiU from every point of Linköping, but I should know it from my home. For our chosen parameters and policy, we are sure to choose the shortcut and always return to the shortcut if going into the blob. If we would want information about the entire world (there might be a more rewarding state somewhere) we would have to increase the exploration factor as well as the iterations to quite high numbers. In comparison with world 1, we would need higher exploration/training iterations since the starting point is static in world 3. See answer 4 for more explanation.

Chosen parameters:
epsilon: 0.3
gamma = 0.95
learning rate = 1
training iterations: 50

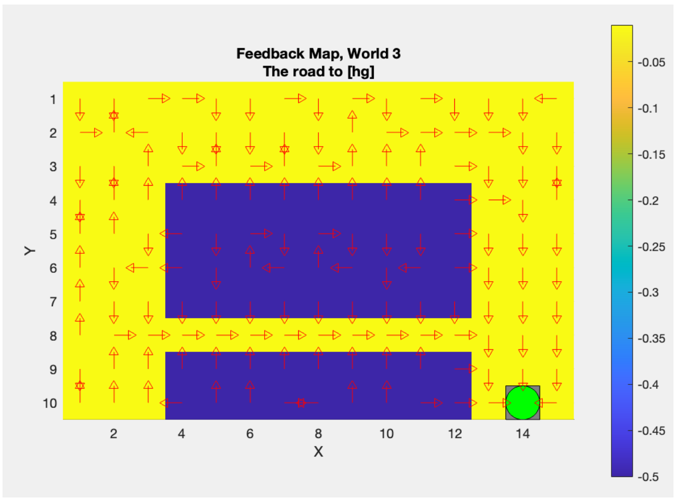


*Figure 6. Policy for world 3.*

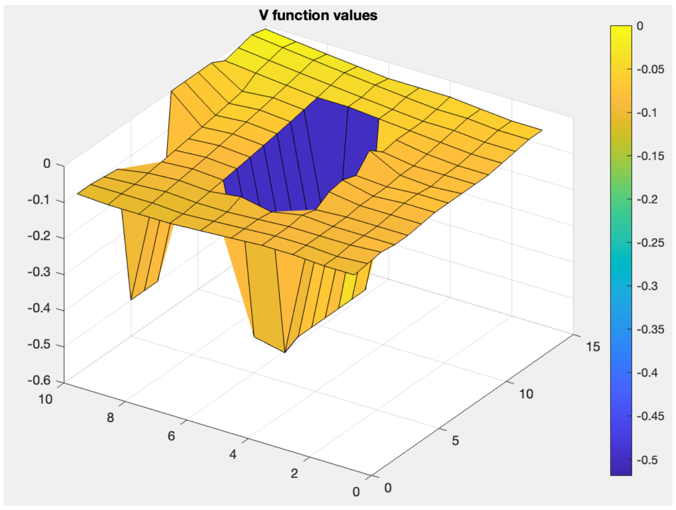


*Figure 7. V function values for world 3.*

Q7. Describe World 4. What is the goal of the reinforcement learning in this world? This world has a hidden trick. How is it different from world 3, and why can this be solved using reinforcement learning? What parameters did you use to solve this world? Plot the policy and the V-function.

World 4 looks just like world 3, with the exception that we want to get home from hg instead of to hg. The difference is that we now are "drunk", which makes us do random actions in 30% (22.5% will result in the wrong action since 1/4 of all random actions is the correct one) of all actions. This makes the policy not work as intended, since 22.5% of all actions will be incorrect. Therefore, it's very risky to choose the shortcut, since this can mean that we "fall into the water" and lower our reward. That's why we want to train the algorithm to understand that it is risky to be on the shortcut, and that it should be avoided. See figure 8 and figure 10. This can be done by having a low learning rate, taking into consideration all the times that we have failed to stay en route.

Chosen parameters:
epsilon: 0.5
gamma = 0.95
learning rate = 0.3
training iterations: 3000

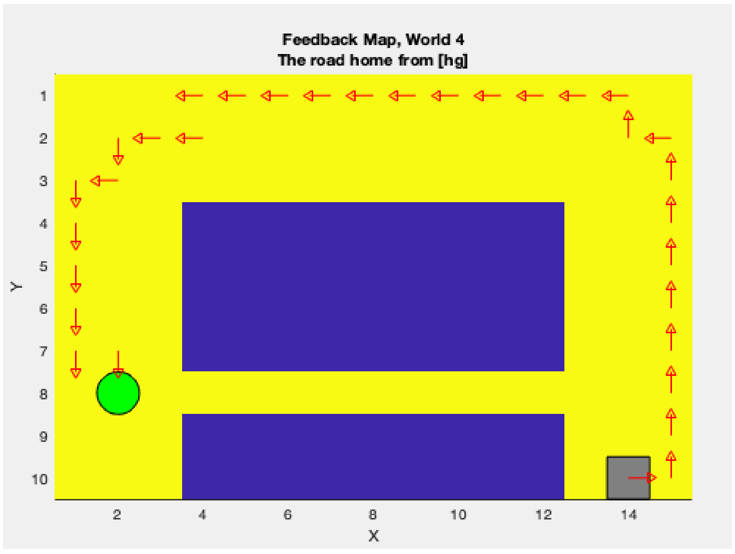


*Figure 8. Test run of world 4. Arrows represent "wanted optimal action".*

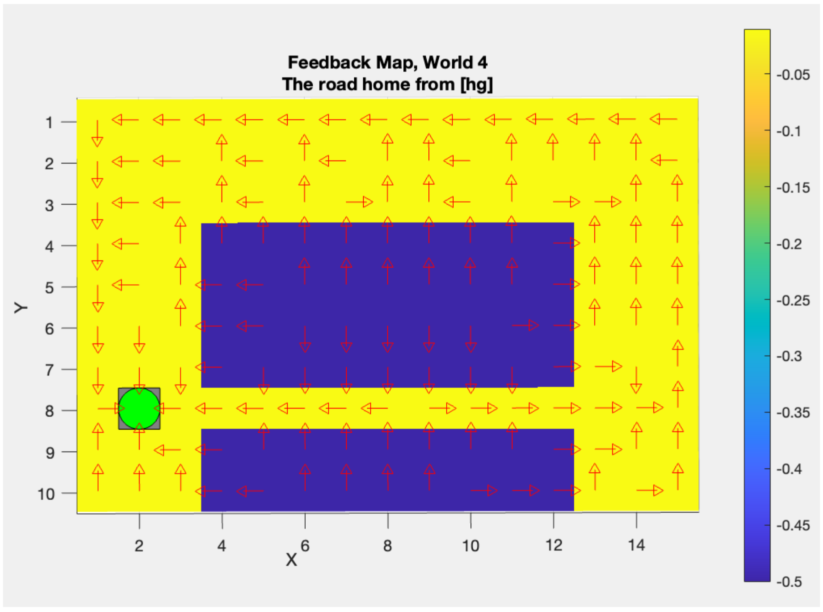


*Figure 9. Policy for world 4.*

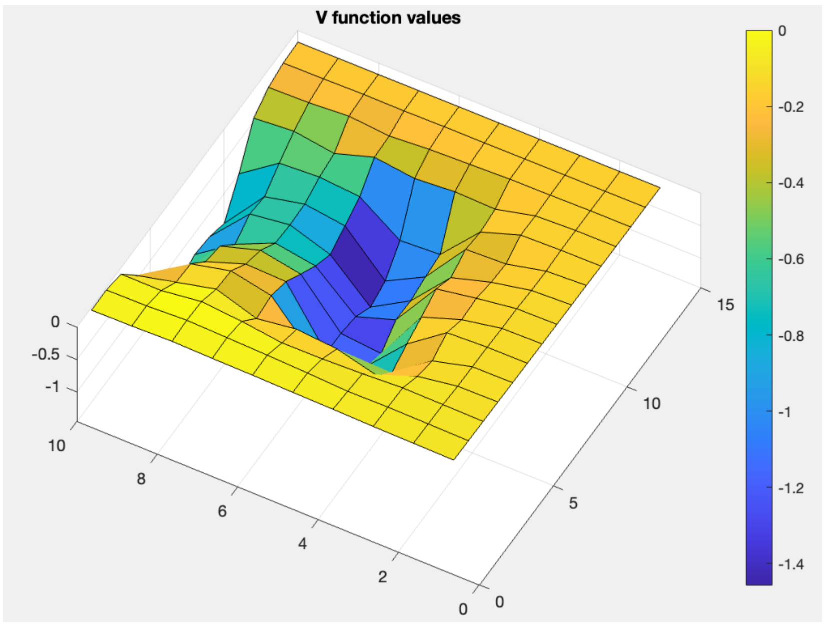


*Figure 10. V function values for world 4.*

Q8. Explain how the learning rate α influences the policy and V-function. Use figures to make your point

The learning rate n is between 0 and 1 and is used in our training to update the Q-function. In other words the learning rate describes how much the robot believes in its stored Q-values. In a static world a learning rate of 1 could be used because there is no need to adapt to changes in the environment. On the other side a low value of the learning rate helps the robot to adapt to new changes. Below you will find a comparison in world two with both a high and low learning rate. As visible in figure 11, the robot tries to avoid the suddenly irritating blob (even if the policy isn't as good as in figure 4 because of fewer training iterations), because it has a lower learning rate, and takes into consideration the loss in reward from earlier iterations. In figure 12, the robot doesn't take this into consideration, instead it uses the policy that worked best in the last training iteration, which results in going through the irritating blob and getting a lower reward.

Chosen parameters using a low learning rate (figure 11):
epsilon: 0.99 - (0.5 * i / training), i = current training iteration, training = 1..100
gamma = 0.95
learning rate = 0.3
training iterations: 100

Chosen parameters using a low learning rate (figure 12):
epsilon: 0.99 - (0.5 * i / training), i = current training iteration, training = 1..100
gamma = 0.95

learning rate = 1
training iterations: 100

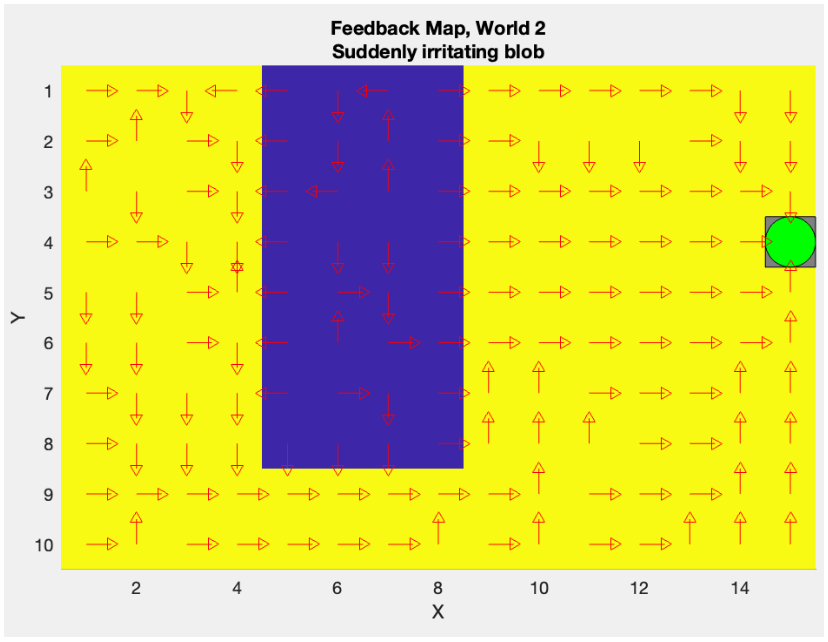


Figure 11. World 2 with 0.3 in learning rate

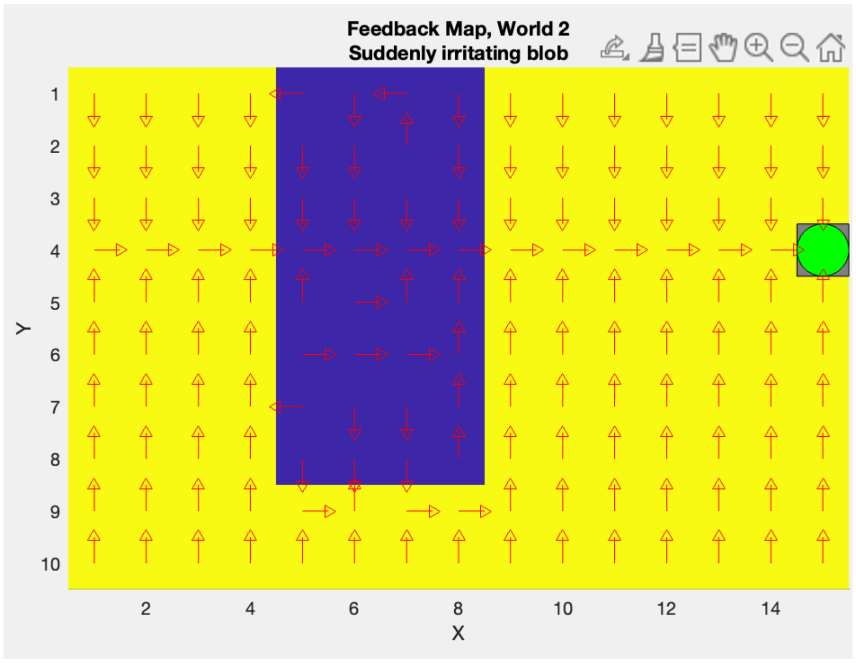


Figure 12. World 2 with 1.0 in learning rate.

Q9. Explain how the discount factor γ influences the policy and V-function. Use figures to make your point

The discount factor (gamma) controls the trade-off between immediate rewards and long term rewards. In other words should the robot focus on rewards that are close or far away. When training the robot, we need to think in the long run, because we need to find the end goal. There might be cases where focusing on short term rewards is more optimal, such as a game where you need to complete simple tasks in the beginning to be able to reach the end goal, and the quicker you gain the short term rewards the better.

When applying a low gamma in world 3 we focus on the short term reward. You can see in figure 14 that we get stuck in the down left corner. The agent tries to avoid "the water" and thinks in the short term and therefore newer reaches the goal. Compare this with figure 13 with a high gamma.

Chosen parameters using a low learning rate (figure 13):
epsilon: 0.3
gamma = 0.95
learning rate = 1
training iterations: 500

Chosen parameters using a low learning rate (figure 14):
epsilon: 0.3
gamma = 0.05
learning rate = 1
training iterations: 500

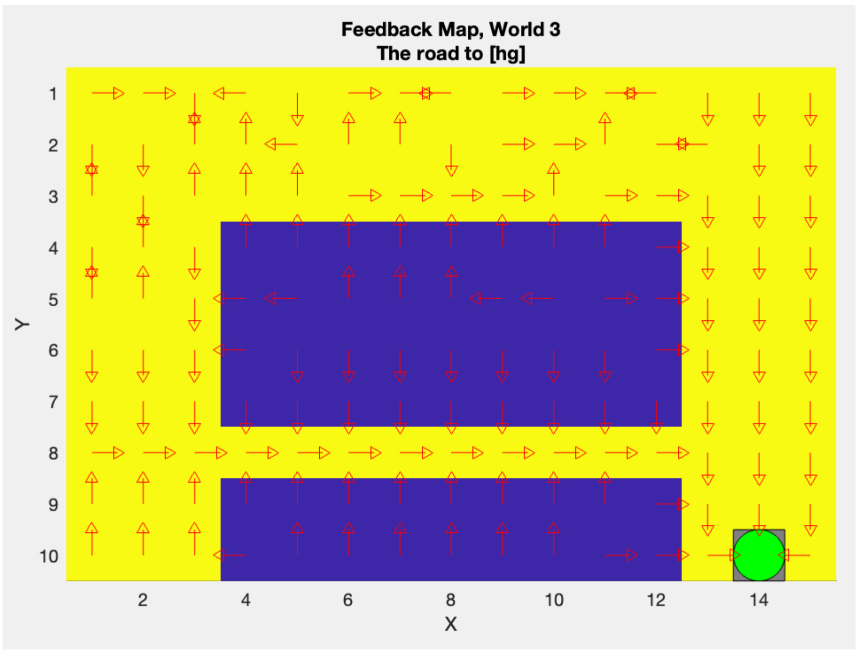


*Figure 13. World 3 with gamma = 0.95*

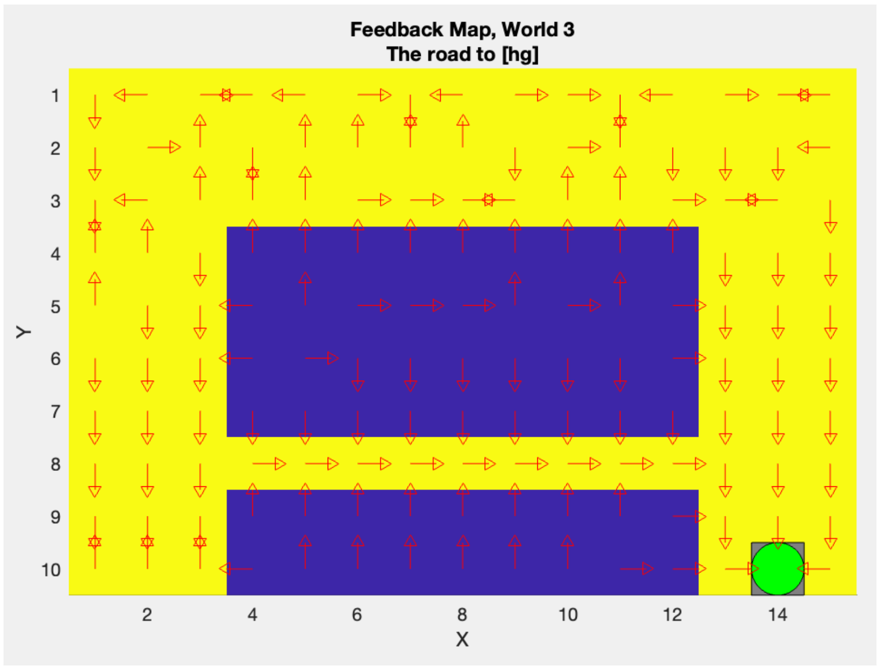


*Figure 14. World 3 with gamma = 0.05*

Q10. Explain how the exploration rate ε influences the policy and V-function. Use figures to make your point. Did you use any strategy for changing ε during training?

As stated earlier, the exploration rate influences the policy and function by raising the chance of choosing another action than the current most rewarding one in Q(x,y), which leads to that the robot explores the world and generates an optimal policy and true values for all positions. This might or might not be necessary, depending on the characteristics of the world we are training in. We tried to use different exploration rates for all worlds, since there is an optimization problem where we want to be effective with our training and not waste time exploring when there is no need. In some of the worlds, we used a static exploration rate, and in others we chose to have a decreasing rate, which focuses more on reaching the goal, the further the training has gone. In comparison we can look at figure 2 and figure 6. Both worlds are static, in figure 2 (world 1) we have generated an optimal policy for each point in the world. In that case we used an exploration rate of 0.9. In figure 6 (world 3), we haven't generated an optimal policy for all points of the world (see discussion of Q6). This is done to show that we can lower the exploration rate, have shorter iterations, and still find an optimal way to the reward. In that case we used an exploration rate of 0.3.

Q11. What would happen if we instead of reinforcement learning were to use Dijkstra's cheapest path finding algorithm in the "Suddenly irritating blob" world? What about in the static "Irritating blob" world?

In the first world using this optimization (network) algorithm which finds the cheapest way from one node to all other nodes will work. This is because everything is static (with the exception of the starting point). In the second world this will be problematic for this algorithm due to randomness. We have uncertainty which can't be modelled using Dijkstra's algorithm.

Q12. Can you think of any application where reinforcement learning could be of practical use? A hint is to use the Internet.

A common application is an agent which plays games and tries to maximize its score. A real world example would be self driving cars, which can do the main training virtually. Testing and evaluation is done in the real world under certain circumstances such as a supervising passenger, ready to interfere if needed. Another example could be in finance and the analysing of insurance and credit assign problems in assigning credit rates for companies.

Q13. (Optional) Try your implementation in the other available worlds 5-12. Does it work in all of them, or did you encounter any problems, and in that case how would you solve them?

Overall in the other worlds we had no problems with them (5-10). We did not analyse every world in detail and eventual troubles in multiple test runs. Also world 11 was pretty slow so it was not analysed with many training runs. In the last world with randomness some problems might arise and affect our solution.

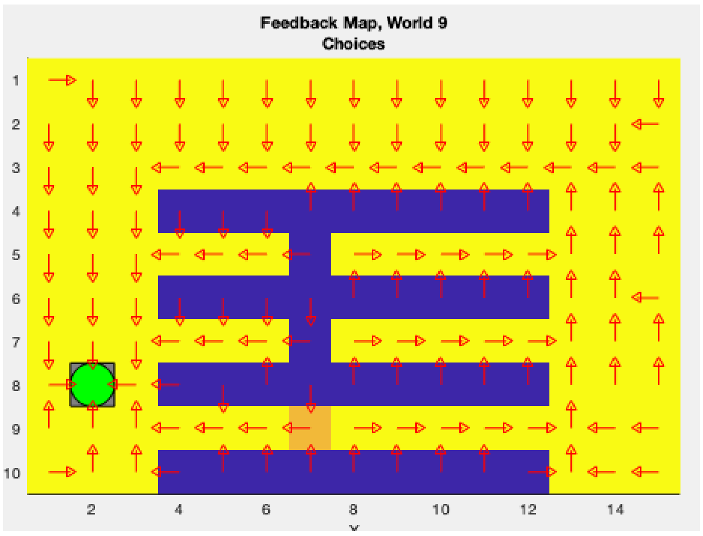


*Figure 11. One of the other worlds.*