TBMI26 – Computer Assignment Reports  
Reinforcement Learning

Deadline – March 12th 2018

Authors:

Jakob Dalsgaard, [jakda997@student.liu.se](mailto:jakda997@student.liu.se)

Erik Sandlund, [erisa920@student.liu.se](mailto:erisa920@student.liu.se)

In order to pass the assignment you will need to answer the following questions and upload the document to LISAM. You will also need to upload all code in .m-file format. If you meet the deadline we correct the report within one week after the deadline. Otherwise we give no guarantees when we have time.

1. **Define the V- and Q-function given an arbitrary policy as well as a given optimal policy (See lectures/classes).**

The V-function for the state *st*, given an arbitrary policy *p*, is in its simplest form defined as

where *rt+k* is the reward in the future states *st+k*. One can say that the value function is the expectation value of all future rewards. Thus the value function *V\** given an optimal policy becomes

Given the state *sk*, the Q-function is defined as

where in this case

So the Q-function indirectly encodes the optimal policy and its value function.

1. **Define a learning rule for the Q-function (Theory, see lectures/classes).**

One learning rule for the Q-function, using an iteration that is similar to the Temporal Difference update, is that the updated estimation of Q can be expressed as

where α is the learning rate, which is there to make the robot balance between putting emphasis on already learned experience or exploration of new policies, and the estimates of Q on the right-hand side are the previous ones.

1. **Describe your implementation, especially how you hinder the robot from exiting through the borders of a world.**

We first choose what world we want to use, initialize the parameters α and γ and set the number of iterations *max\_iter* we want to perform. Then we save the size of the world in x and y and initialize the Q-matrix as a x × y × 4 matrix containing random values between 0 and 1. We then set the Q-values for traversing across the borders of the world (for instance, going left when y = 0) to -∞.

From there we enter the main loop, in which we set *ε* to 1 in the first fifth of the iterations and then decrease it afterwards. After thatinitialize a new world for each iteration and then start moving around the world, updating the Q-function on the way, until we reach the terminal state. Depending on epsilon, we either set the action as the hitherto determined optimal action or set the variable *action* as a completely random action (that is, it has a 25 % possibility of being each of the four possible actions), which we then perform. If this takes us to an invalid state, we do not update the Q-function for the state, but simply perform another action. If, however, the action is valid, we check whether we are in the terminal state. If we are, we simply update the Q-function, break the loop and go to the next iteration. If not, we update the Q-function and perform the next action.

When we have reached the terminal state the number of times we have specified (in the code, this is 100), we print the Q-value of each action for all states in four different figures. We then plot what, according to our algorithm, is the optimal action by drawing arrows in that action’s direction for each state in a fifth figure.

1. **Describe the differences between the worlds explored by the robot. Any surprises?**

The first world contains an area with relatively large negative rewards, see the blue area in figure 1. When the robot explores the world, the optimal actions after *max\_iter* iterations of the algorithm described in section 3 can be seen in the feedback map in figure 1.

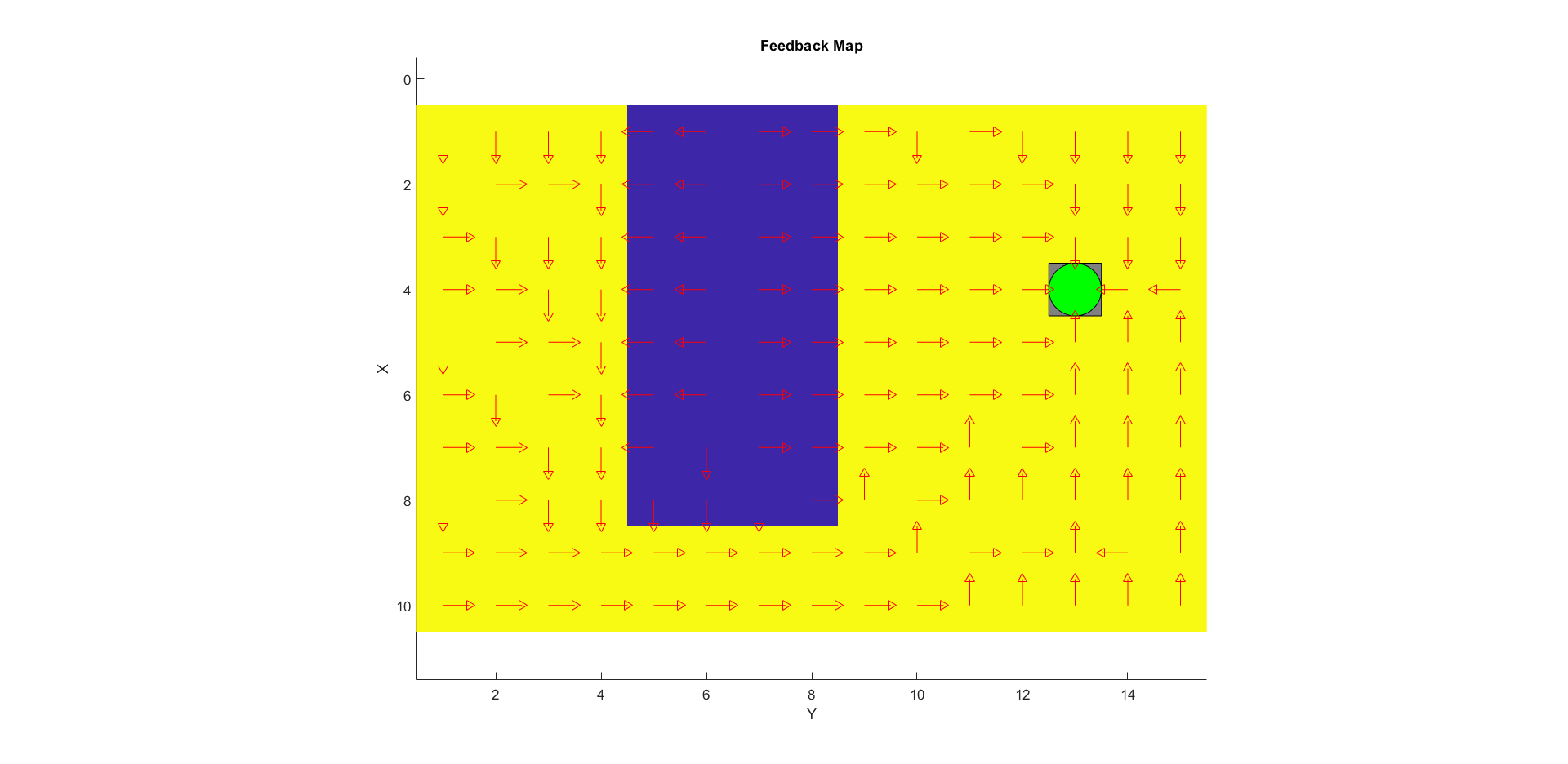
The second world is similar to the first one. However, the area with large negative rewards is here stochastic in the sense that there is a 20 % chance that the states in that area has a negative reward. Increasing the number of iterations would yield a more stochastic Q-matrix in the sense that the effective reward would be the mean value of all rewards, that is, 0.8 times the positive reward plus 0.2 times the negative reward.

The third world has no stochastic areas with negative reward. However, it instead has two such areas, which the robot will have to avoid.

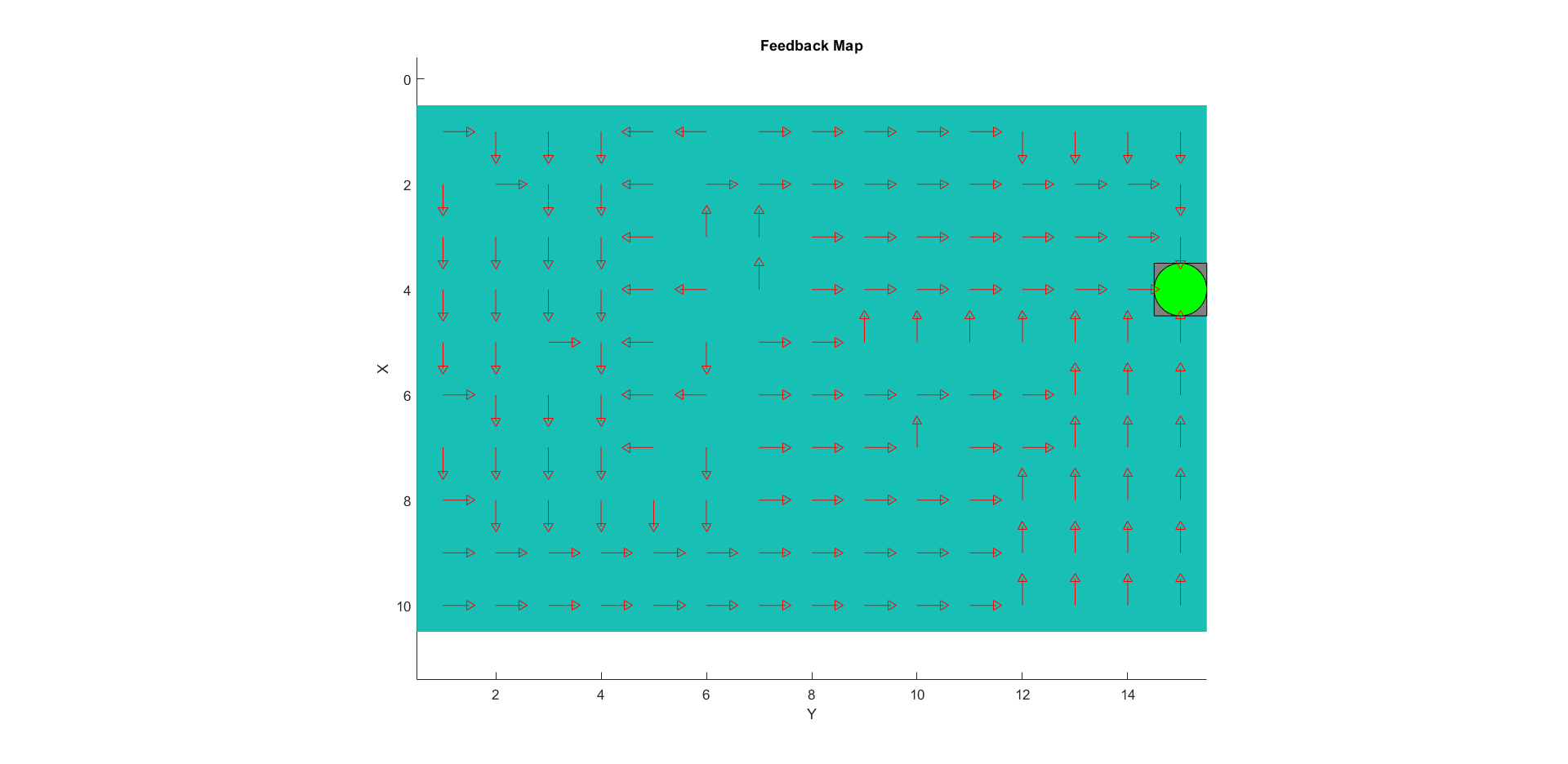
The fourth world is the exact same world as the third but with two differences. First, the terminal state moved. Second, in *gwaction.m*, there is a now a 30 % that the robot will take a completely random action. This means that it will be much more difficult for the Q-matrix to converge.

1. **For each world: Plot the V-function, i.e. how do you get to the goal from each position.**

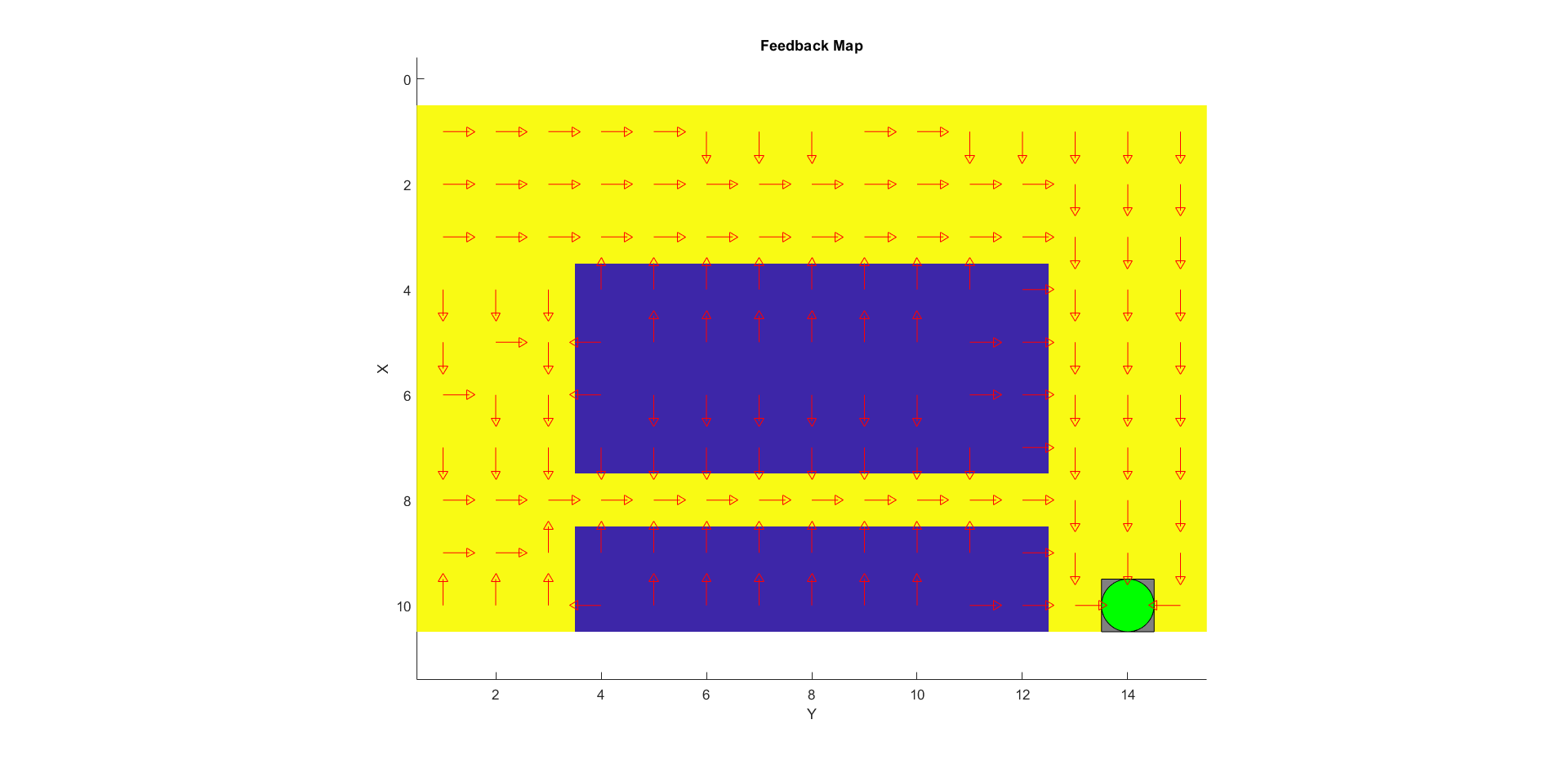
The V-functions for each world can be found in figures 1-4 below.



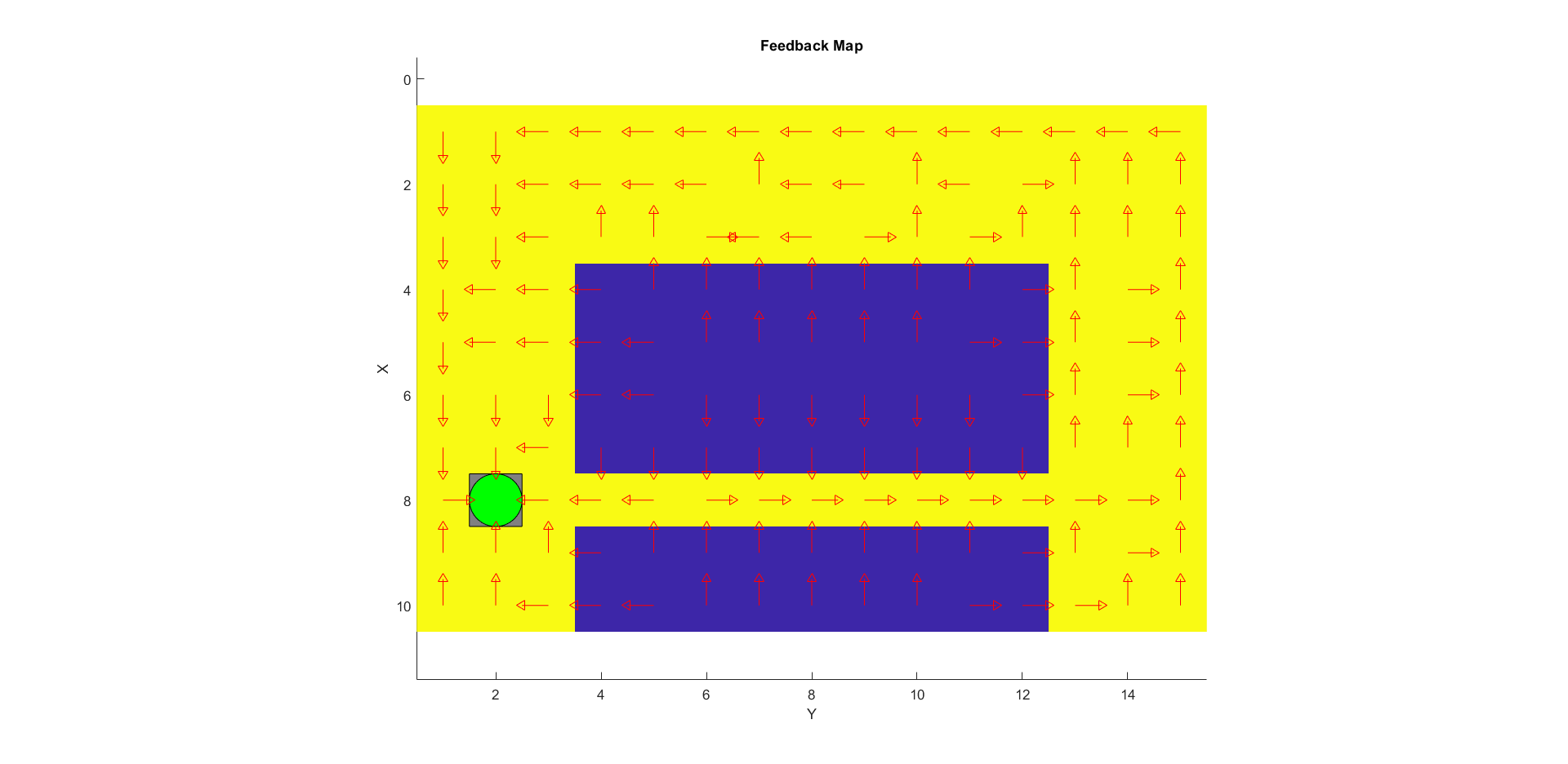
*Figure 1: The feedback map for world 1 after 100 iterations, γ = 0.7, α = 0.3.*



*Figure 2: The feedback map for world 2 after 1000 iterations, γ = 0.9, α = 0.1.*

****

*Figure 3: The feedback map for world 3 after 1000 iterations, γ = 0.7, α = 0.3.*

****

*Figure 4: The feedback map for world 4 after 1200 iterations, γ = 0.9, α = 0.1.*

1. **For each world: describe the key observations you have made with respect to parameter choices. Provide documentation of the parameters you have used for each figure! A good rule is to provide each figure with a caption. Plot policies and the V-function for appropriate worlds to the extent you find appropriate in order to explain what you have done and learned during the assignment.**

To get out of negative areas quickly, a high γ (close to 1) is preferred, especially for. For figure 4, to find the path around the negative areas, it is preferable with a high ε (close to or equal to 1), especially for the earlier iterations. See section 5 for figures with settings for parameters in the caption. As we see in figure 4, we choose a low alpha (close to 0) for world 4, which is preferable for reasons we will explore further in section 8.

1. **What would happen if we were to only use Dijkstra's shortest path finding algorithm in the ''Suddenly Irritating blob'' world? What about in the static ''Irritating blob'' world?**

It would perform reasonably well in the static world, it would find the shortest path and use it. In the stochastic world however, the only variable of use would be a stochastic expectation value, which would make it much harder for it to find a shortest path.

1. **Include an in-depth description of the to/from HG worlds (world 3 and 4). What happens on the way from HG? How and why can this problem be solved with Q-learning? Which path does the robot prefer, and why?**

World 3 and 4 are very similar in the respect that they have two large areas with negative rewards with a corridor of positive rewards between them. In the case of world 3, this is the path preferred by the robot. In world 4, however, there is a 30 % possibility that the robot takes a random step, which means that it statistically will go into the areas with negative rewards, which means that an optimal path would in that case rather be to traverse around the large negative areas. This is reflected in figure 4, when you compare it to figure 3. In figure 3, the robot goes along the corridor, while it in figure 4 prefers to walk around the negative areas. The effect of this random step can be remedied by setting a high epsilon, that is, depending more on previous knowledge instead of exploring new possibilities. Then the robot will more quickly decide to walk around the negative areas instead of attempting to go through the corridor.

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

Applications for reinforcement learning could, for instance, be autonomous vehicles and other areas involving the problem of optimal control or for creating computers capable of playing games such as Chess or Go.

1. **How does the different parameters () influence learning and appearance of the Q- and V-functions?**

The learning rate α (or γ) lies between 0 and 1. A lower value means that the learning algorithm will put more emphasis on already learned experience (that is, use the V-functions when updating the Q-functions) and a higher value will overwrite previous experience with new information.

The discount factor γ lies between 0 and 1. A lower value means that the learning algorithm will seek to maximize short-term rewards. A higher value means that it will instead focus on long-term rewards.

The exploration factor ε lies between 0 and 1. A higher value means that the system is more likely to take random actions and explore new policies. A lower value means that the system takes actions based on experience it has already learned.