TBMI26 – Computer Assignment Reports  
Reinforcement Learning

Deadline – March 15 2019

Author/-s:

Isak Johansson-Åkhe

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**. We will correct the reports continuously so feel free to send them as soon as possible. If you meet the deadline you will have the lab part of the course reported in LADOK together with the exam. If not, you’ll get the lab part reported during the re-exam period.

1. **Define the V- and Q-function given an optimal policy. Use equations and describe what they represent. (See lectures/classes)**
2. **Define a learning rule (equation) for the Q-function and describe how it works. (Theory, see lectures/classes)**

The updated Q-value of a given combination of state and action (left) is proportional to a fraction of the previous estimate of the Q-value (blue) plus the estimate of the reward that will come of continuing with the best policy from the new state (red). The learning rate sets how much new information will affect the Q-value. The discount factor determines how “far ahead” the function will look for future rewards.

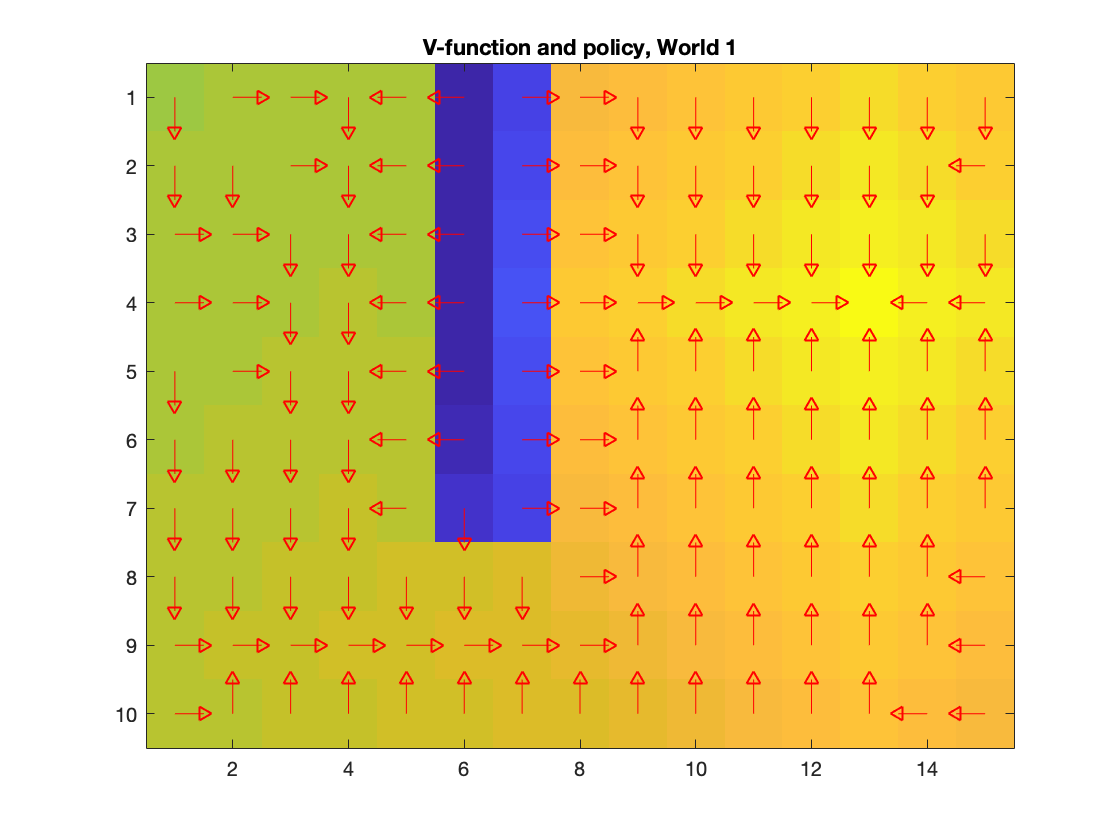
1. **Briefly describe your implementation, especially how you hinder the robot from exiting through the borders of a world.**
2. The world is initialized. The Q-table is initialized as a world-ysize\*world-xsize\*4 matrix (there are 4 possible actions) with random values. *Q-value for all actions which would lead to the robot falling off the world are set to -inf.*
3. Training begins. Repeat the following a number of times (episodes):
   1. The world is reset
   2. The exploration factor is set depending on how many episodes of the total have been run.
   3. While the state of the system is not terminal:
      1. Choose a new action depending on Q and exploration factor.
      2. Fetch the state of the new action
      3. If the new state is valid, update the Q-value of the given position and set the current state to the new state. (if the new state is invalid, the position of the robot will not have changed)
   4. If the robot reached the goal, set all Q-values for current state to 0.
4. The optimal policy can be found by taking the argmax of the Q-matrix along the 3rd dimension (action).
5. **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 is simple, with one block of punishing terrain close to the center and otherwise free pastures, where the robot needs to find its way around the “blob” to the goal. Given the uncomplicated nature, learning rate can afford to be high.

Learning Rate: 0.5

Discount Factor: 0.9

Exploration Factor: 0.9 to 0.045 (diminishes with number of episodes)

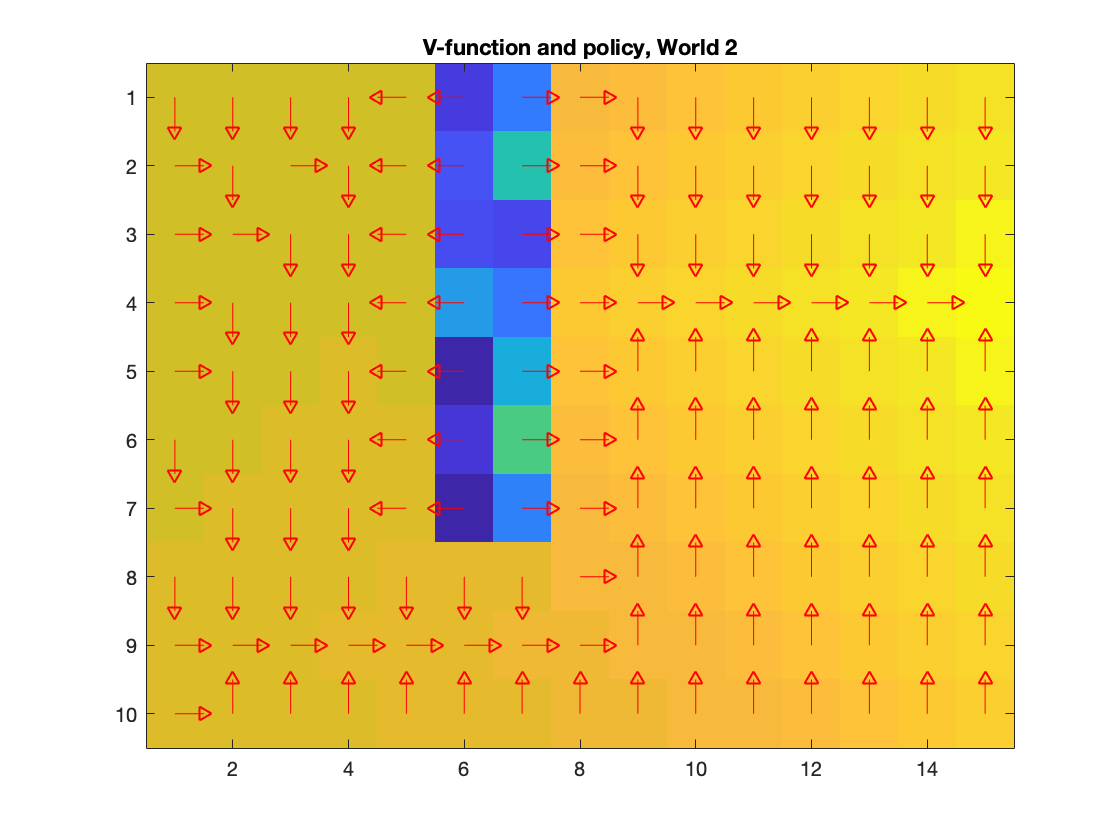
Episodes: 2000 ****

1. **Describe World 2. 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 2 is the same as world 1, except that the “blob” only appears in 1 in 5 episodes, and when it does it gives a very steep penalty. The learning rate should be kept low, so the method will remember the blob even when it isn’t there. Because of the decreased learning rate, the number of episodes will need to increase proportionally.

Learning Rate**:** 0.05

Discount Factor: 0.9

Exploration Factor: 0.9 to 0.045 (diminishes with number of episodes)

Episodes: 20 000****

1. **Describe World 3. 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.**

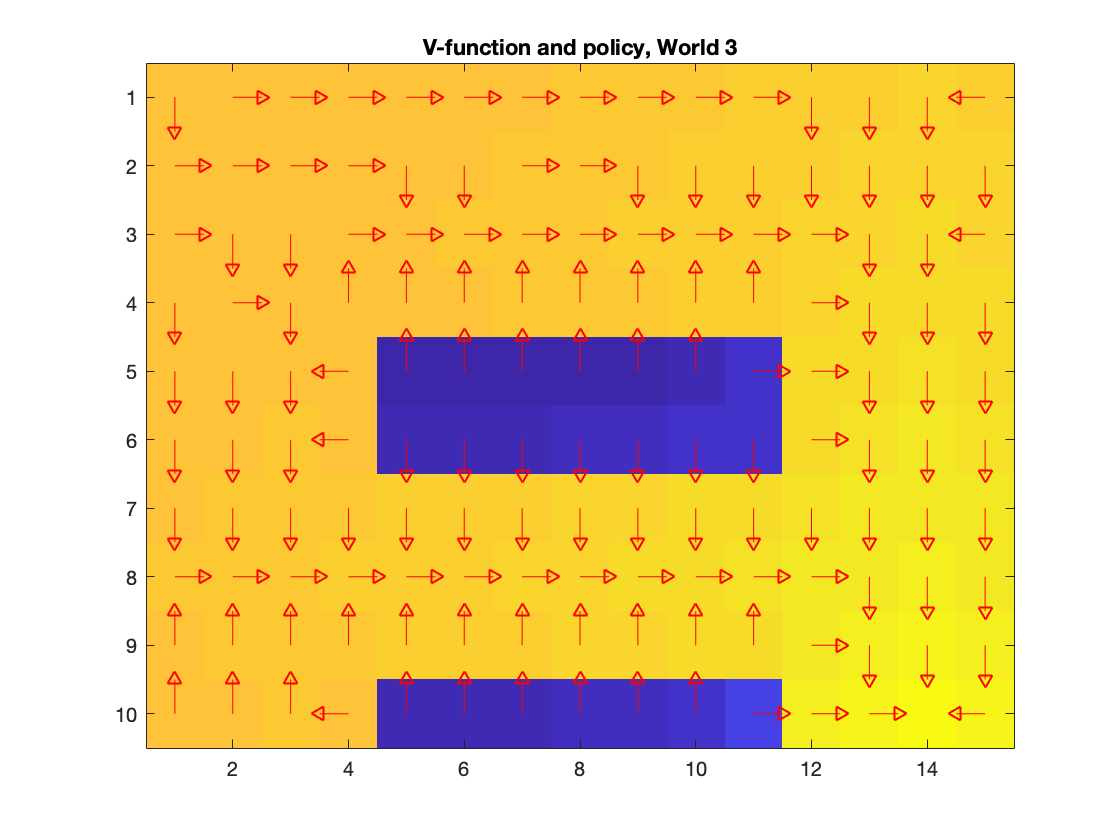
In world 3, most of the world is covered by a “blob” of punishing terrain. However, the “blob” is bisected by a thin path which is a faster way to the goal than walking around it. In this situation, a constantly high Exploration factor would lead to the robot deeming the thin path too dangerous to traverse. With the combination of a linearly decreasing exploration factor and a high learning rate, the path will probably end up the best alternative, as early episodes will find the path and deem it slightly less dangerous than surrounding areas, later episodes will refine it (realizing its even less dangerous) and overwrite the previous Q-values. Exploration factor still needs to start high to explore a meaningful section of the board. An alternative would be to keep exploration factor constant but to make the discount-factor more “far-seeeing”.

Learning Rate: 0.5

Discount Factor: 0.9

Exploration Factor: 0.9 to 0.045 (diminishes with number of episodes)

Episodes: 2000



1. **Describe World 4. What is the goal of the reinforcement learning in this world? How is this world 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.**

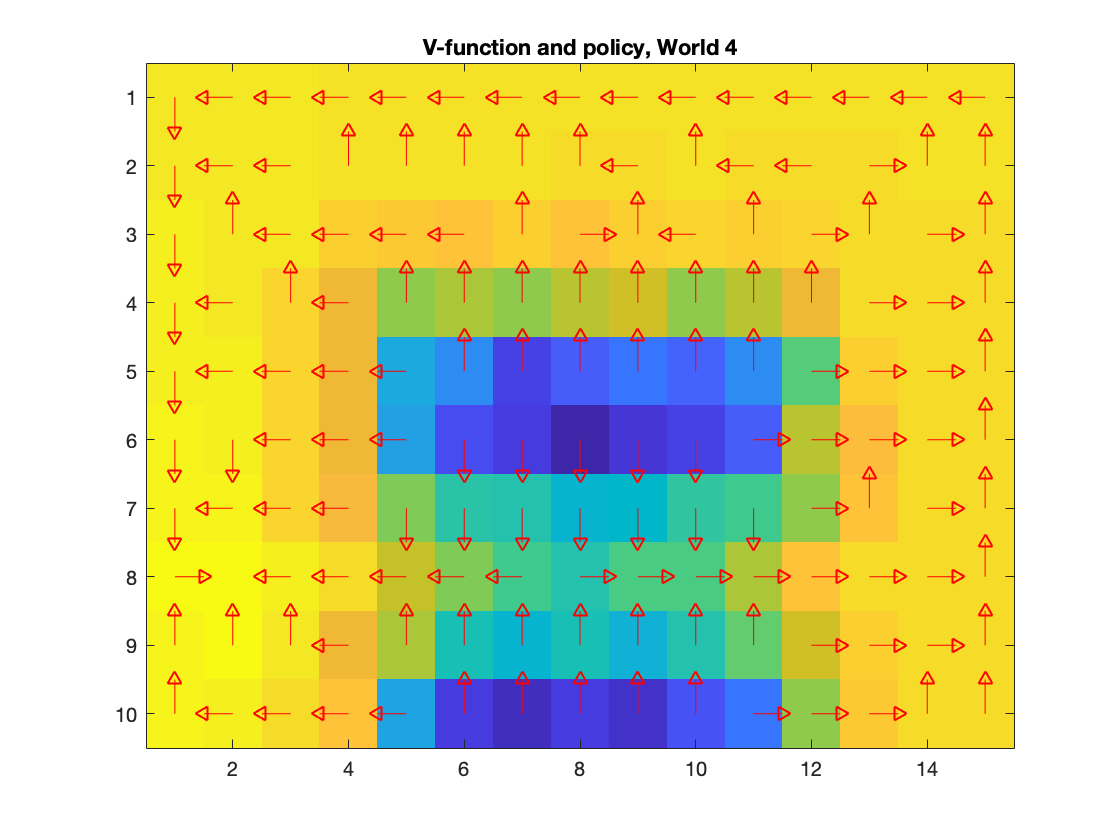
World 4 is very similar to world 3, but with the huge unseen difference that 30% of the time, the robot will step in a random direction rather than what you tell it to. The problem is still solvable by reinforcement learning as in reinforcement learning you don’t make decisions based only on singular runs, but by the accumulated experience acquired through many runs. Basically, the system will “learn” that most often it will step in the correct direction, but in the future it will add risk even if you’re just close to high-penalty areas. In a way, this world is similar to world 2, in that there is an element of chance whether some states will be punished more severely or not. As such, using a low learning rate (to make sure a lot of previous runs are remembered) combined with high number of episodes will help.

Learning Rate: 0.1

Discount Factor: 0.9

Exploration Factor: 0.9 to 0.045 (diminishes with number of episodes)

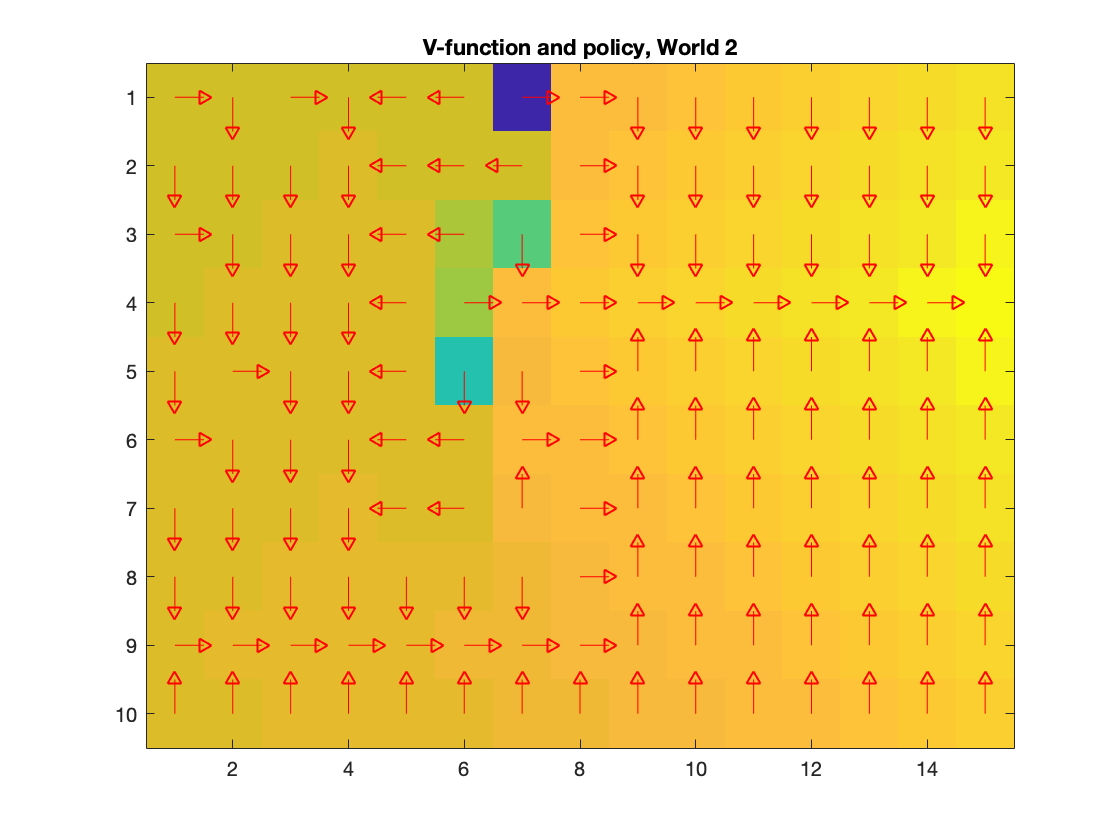
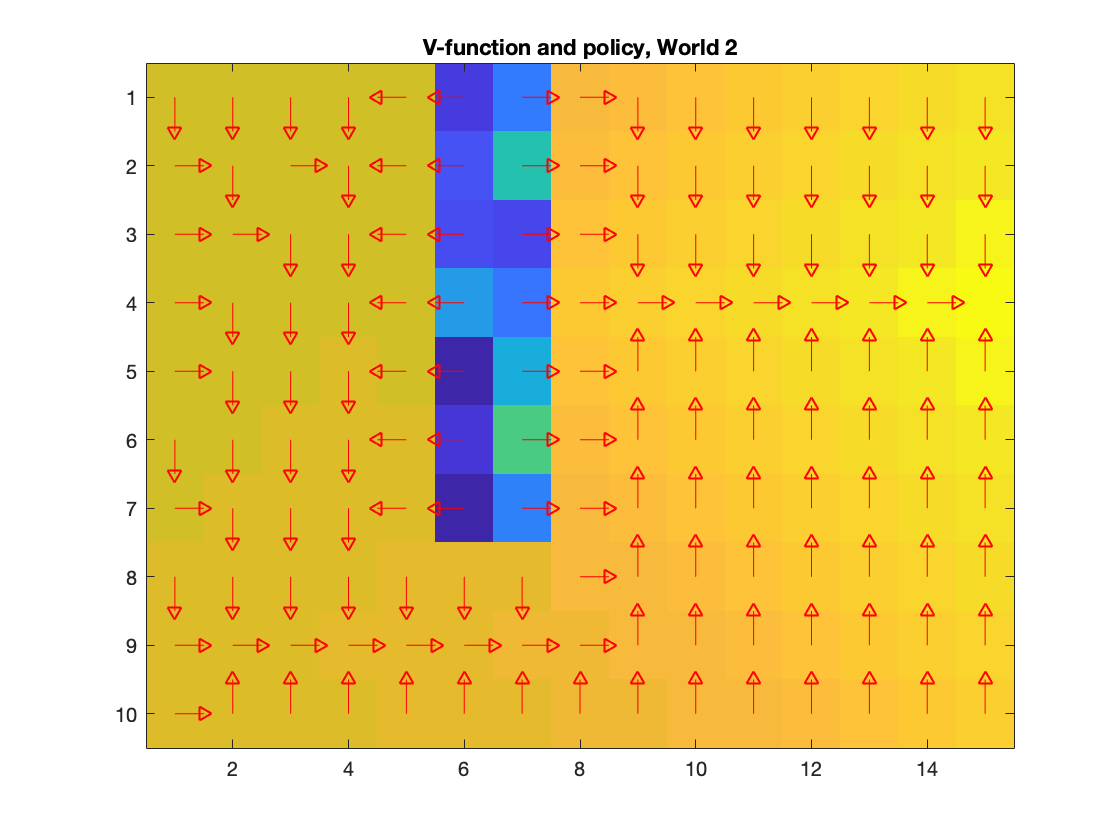
Episodes: 10 000

****

1. **Explain how the learning rate α influences the policy and V-function in each world. Use figures to make your point.**

In worlds without an element of chance, such as worlds 1 and 3, the learning rate does not influence much, but rather just makes the method slower at learning the optimal route (as long as its set to sensible values).

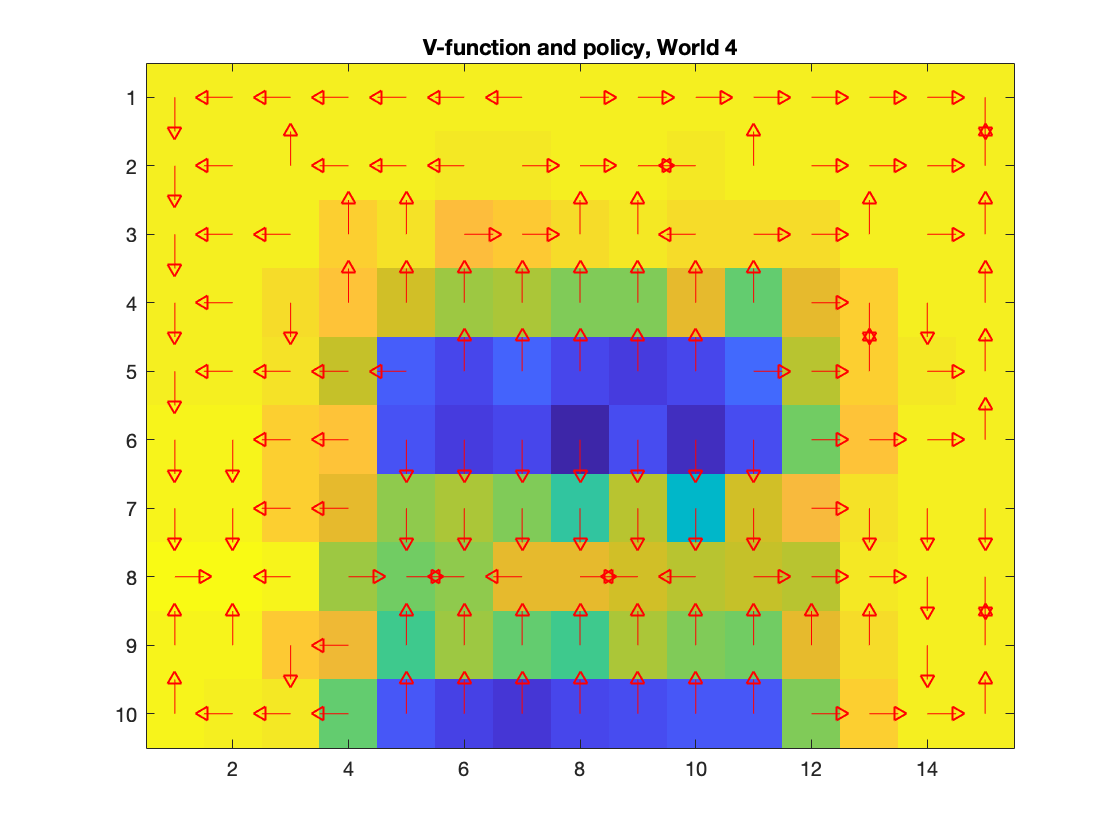
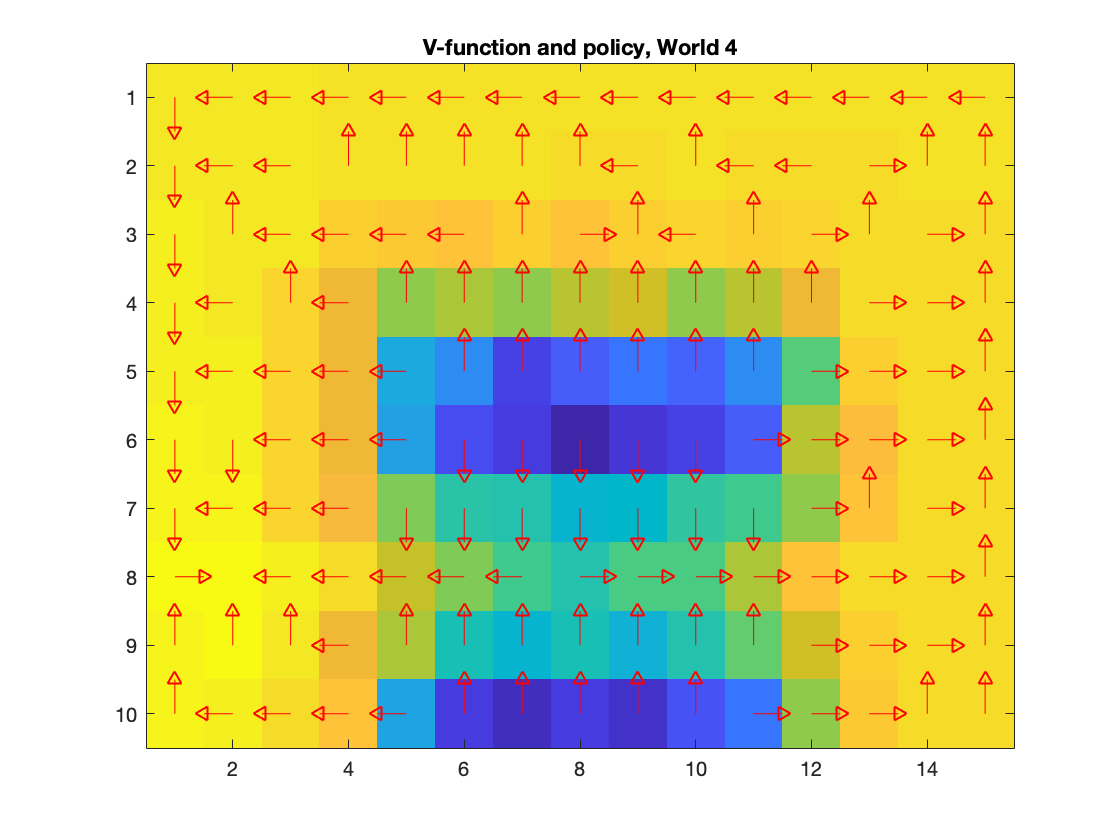
In worlds where chance plays a role, such as worlds 2 and 4, the learning rate becomes more important. A high learning rate in these worlds would mean that the method is prone to forget what it has previously learned. Say for instance that the learning rate is set to 0.5. In just 4 episodes of passing through the same state, only 6.25% of the Qvalue in that state will be determined by what happened before these 4 episodes. In a world such as 2, where there is a terrible negative phenomena which only occurs in 1 in 5 episodes, there is a very real possibility that the signal from this phenomena will be completely drowned out if chance makes it so the last few episodes saw particularly few examples of the phenomena. Similarly, if the random movement directions in world 4 comes into play often in the last episodes of training, the method will immediately switch policies to a convoluted way of moving to avoid the effectively invisible potholes introduced by chance.



Compare for instance the policy on the left (learning rate 0.05) with the one on the right (0.7). In the policy on the right, it becomes obvious that some states did not see much of the “blob” in later episodes, which majorly changed them, resulting in poor decisions such as moving around inside the potentially dangerous area, and walking the wrong direction to get out of it.

1. **Explain how the discount factor γ influences the policy and V-function in each world. Use figures to make your point.**

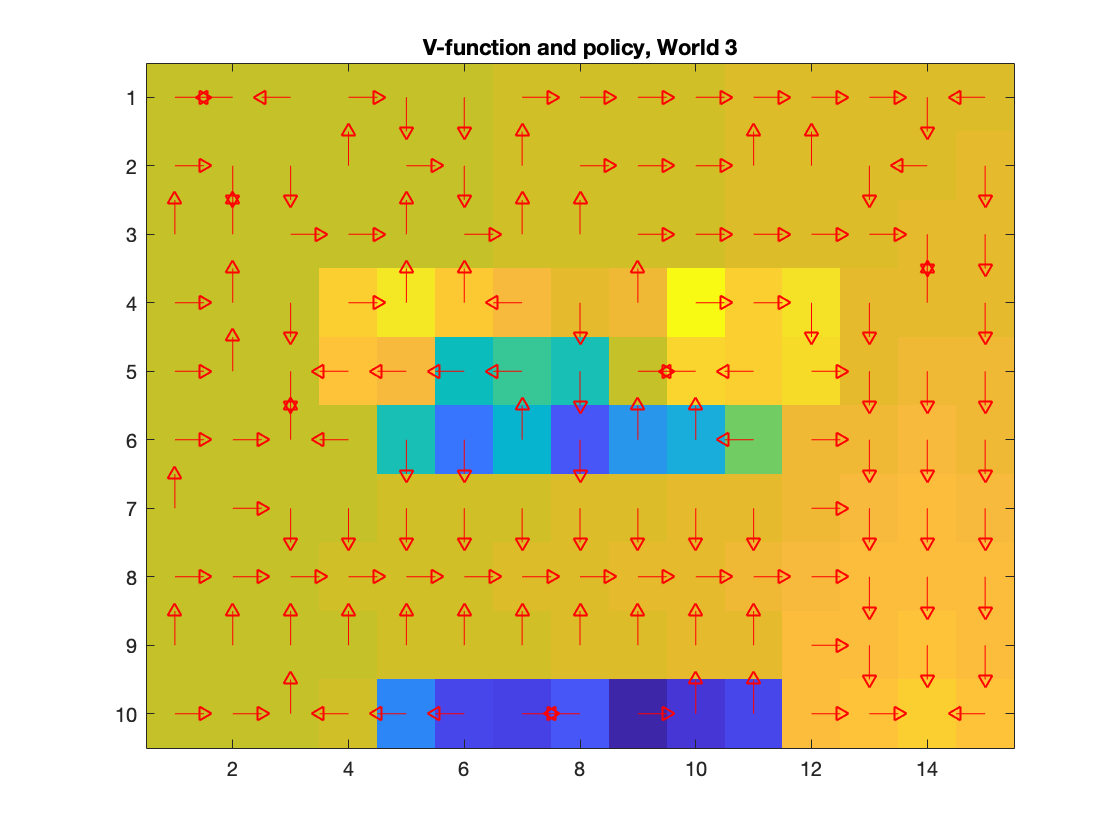
A lower discount factor will look more to the short term, making more “rash” decisions. Changing the discount factor does not make the robot objectively better or worse, it simply reflects that the user has told it that optimal behavior is “long term” or “short term”. For instance, if the discount factor is set lower, the method will be more likely to immediately dash out of bad areas, seeking lower immediate penalty, while a high discount factor might make the method consider walking through some higher-penalty zones. Setting the discount factor low will however massively slow down training, as looking ahead towards the goal will not be as valued, so there will be more excursions into the wild non-penalizing squares, and there might even be risk of “getting stuck” if training does not progress enough, example below:

****

The left shows using a discount factor of 0.9 (suitable), and the right shows a discount factor of 0.7. Notice how the path through the “blob” in the right case is not punished as heavily as in the left case, and notice especially how there are lots of loops and “traps” in the right case, as the method does not look far enough ahead to actually find a goal.

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

If the exploration rate is too high, training will be considerably slowed down. If the exploration rate is too low on the other hand, many parts of the world might go completely unexplored, potentially missing an optimal path.

****

Above is an example of world 3 when exploration is set way too low, resulting in too little of the search space being scored properly.

I used a linearly diminishing exploration rate, which starts high so that much of the world will initially be explored, and then gradually lowers so the method can focus more on optimizing the paths found.

1. **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?**

Dijkstra’s cheapest path would not work on the “suddenly irritating blob” if implemented without some extra term for handling stochastic phenomena (which would also require knowledge of exactly how the world works). Otherwise it would base its path completely on the observed state of the world when the algorithm is applied, which can be either with or without blob.

On world 1 there is nothing that prevents it from working perfectly fine. You could just set each grid-point as a node with 4 vertices, one to each adjacent grid-point, and put the “distances” of these as the feedback values.

1. **Can you think of any application where reinforcement learning could be of practical use? A hint is to use the Internet.**Teaching an automatic vacuum-cleaner how to clean a room (given random placement of chairs), or similarly, teaching an automatic grass-cutter how to best navigate the lawn.
2. **(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?**In world nr 10 and 12, the method did not take into account the possibility of impassable terrain existing within the training boundaries. A suggested solution would be to either search for all possible impossible actions at the start (time consuming) or simply impose a penalty each time the robot collides with an object (attempts to move in a non-allowed way).

In world 11, the trap quite easily catches the robot. Perhaps it could be solved by: creating a copy of the Q-function for each episode, limiting the number of steps the robot may take, and finally only updating the original Q with the copy if the robot reached the goal.