TBMI26 - Assignment 4

Ludvig Knast - ludkn080
Erik Jareman - erija971

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

The Q-function is defined as: $Q(s_k, a) = r(s_k, a) + \gamma V^*(s_{k+1})$

The optimal V-function is defined as: $V^*(s) = max_a Q(s, a)$

The Q-function describes what values each action in each state gives, whereas the V-function shows the action with the best value in each state.

Q2. Define a learning rule (equation) for the Q-function and describe how it works. (Theory, see lectures/classes)

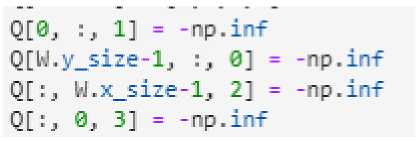
We train (update Q) by using the following formula:

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

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 added borders around the world by the following code


```
Q[0, :, 1] = -np.inf
Q[W.y_size-1, :, 0] = -np.inf
Q[:, W.x_size-1, 2] = -np.inf
Q[:, 0, 3] = -np.inf
```

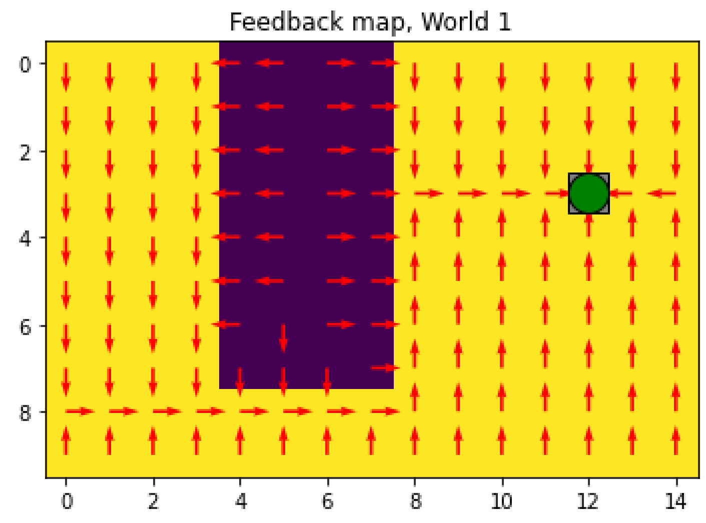
where the value for taking an action over the border is -inf which means it always will be the worst action to take in those states. The action function already had a validation check so we did not have to worry about random actions over the border. The implementation is kind of simple, in the beginning the agent gets to explore a lot i.e taking a lot of random actions and updating the Q table after each action. After each iteration we lower epsilon a bit so the agent more frequently is just following the action with best value. This is done until all iterations are done and epsilon goes to 0. At this point the agent has finished training and the policy can be tested.

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.

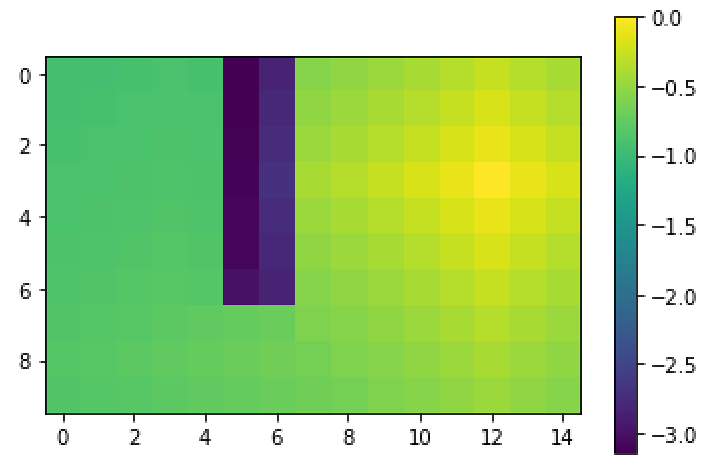
epsilon = 0.9 (going to 0 linearly)
gamma = 0.9

learning rate = 1
training iterations = 1000
The goal position is static but the spawn is random, this makes us want to explore the entire world so wherever the agent spawn an optimal policy to the goal can be used. We start with a big epsilon to force a lot of exploring. We can also use the fact that the world is static (the whole world is always the same, constant blob) in the form of using a learning rate of 1 as all information gained from training will remain true for each state in the world.

Optimal policy:



V-function:



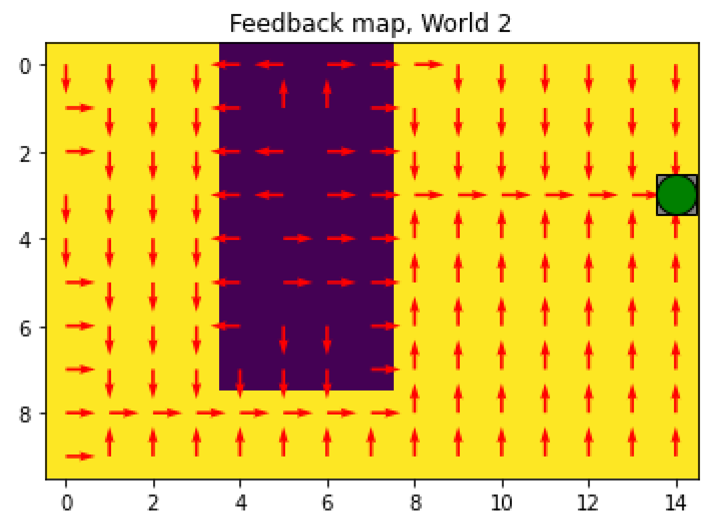
Q5. Describe World 2. What is the goal of the reinforcement learning in this world? This world has a hidden trick. Describe the trick and why this can be solved with reinforcement learning. What parameters did you use to solve this world? Plot the policy and the V-function.
epsilon = 0.9 (going to 0 linearly)
gamma = 0.9
learning rate = 0.1
training iterations = 15000

Similar to world 1 with the difference that the irritating blob only occurs 20% of the time. As we want to make sure to always be on the safe side and dodge the blob if it occurs (when done training) we lower the learning rate and therefore “keeping” the knowledge of the blob as we are not doing too big of a Q update when the blob is not present. We still start on a high epsilon as we want to explore the whole world due to the random spawn.

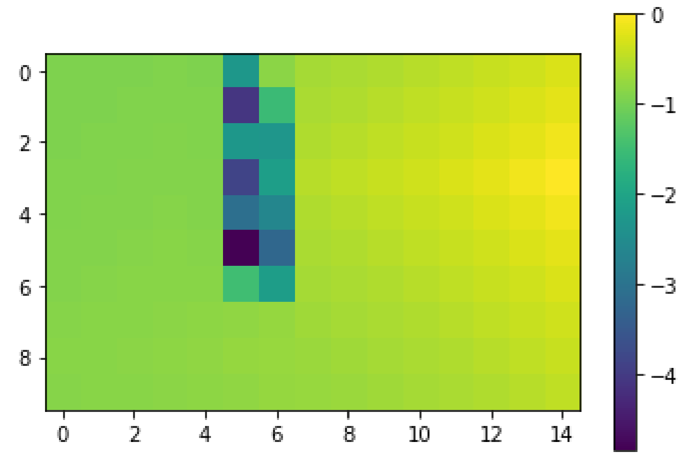
One could have different approaches to this problem, depending on risk management etc but we took the stand that we wanted the agent to act almost exactly like in world 1, i.e always assume the blob would be there. To accomplish this we choose a very low learning rate and a lot of iterations. Safety first! :D

We can see that we almost manage to get the same arrowmap, which is a positive result when having the approach (goal) as we did.

Optimal policy:



V-function:

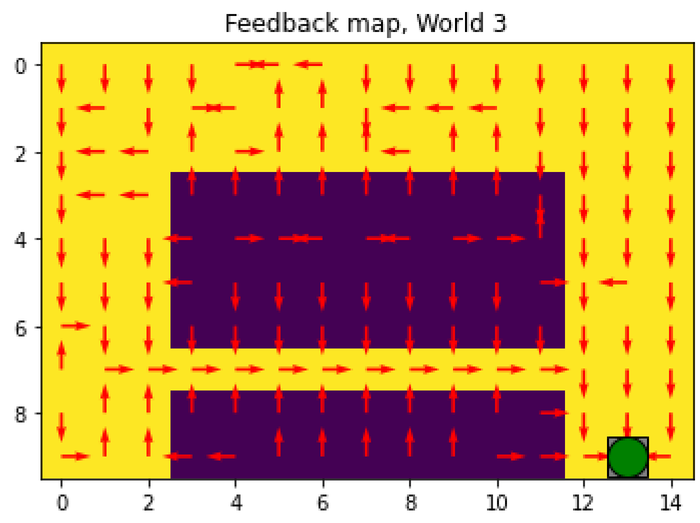


Q6. Describe World 3. What is the goal of the reinforcement learning in this world? Is it possible to get a good policy from every state in this world, and if so how? What parameters did you use to solve this world? Plot the policy and the V-function.

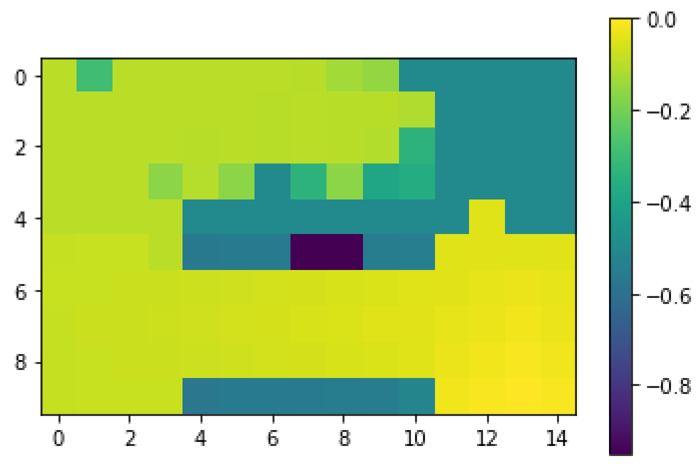
epsilon = 0.5 (going to 0 linearly)
gamma = 0.9
learning rate = 1
training iterations = 200
This world has a static starting point, goal, and world which basically means we don't really have to explore as much as we previously have done. We can also use a learning rate of 1 again. This parameter tuning, combined with the nature of the task will mean we will need a low number of iterations as well as short training time. However much of the world will be unexplored and have non useful state/action information, but this doesn’t matter as we are just solving the same problem.

If you want to get an optimal policy from all states you could go full on explore mode (static epsilon around 0.9) for many iteration and just explore as much as possible, however it would be statistically more difficult to get a good policy for states far away the spawn and goal, i.e row 0 in our example.

Optimal policy:



V-function:

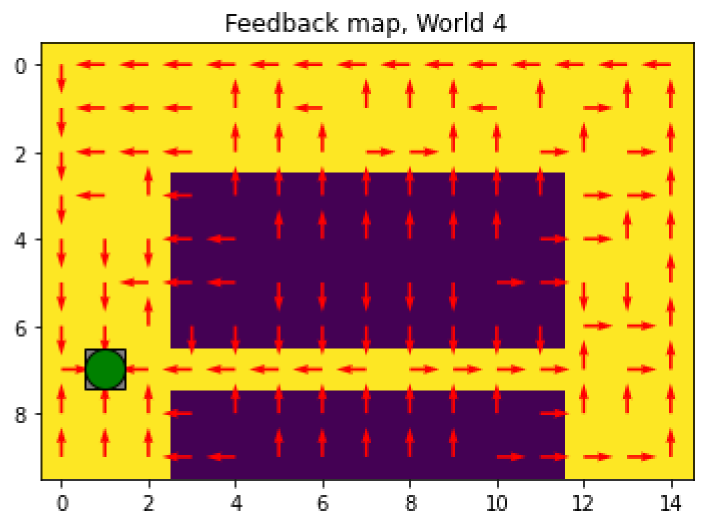


Q7. Describe World 4. What is the goal of the reinforcement learning in this world? This world has a hidden trick. How is it different from world 3, and why can this be solved using reinforcement learning? What parameters did you use to solve this world? Plot the policy and the V-function.

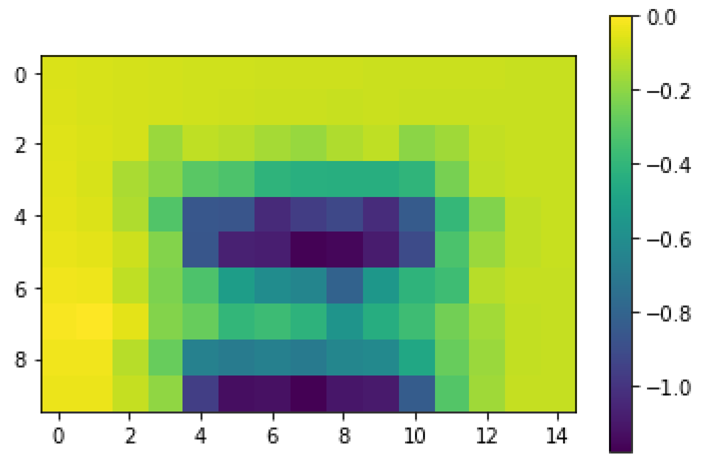
epsilon = 0.9 (going to 0 linearly)
gamma = 0.9
learning rate = 0.1
training iterations = 10000

After some beers at HG we now have to head home, the only problem is that we take random actions 30% of the time which means our agent must learn a policy where he is able to take a random action without stepping on to the "bad" areas, which is also why the narrow path that is the optimal policy in world 3 wont work here. What we then can expect/look for is a result where the best policy is one that goes alongside the borders all the way home.

Optimal policy:



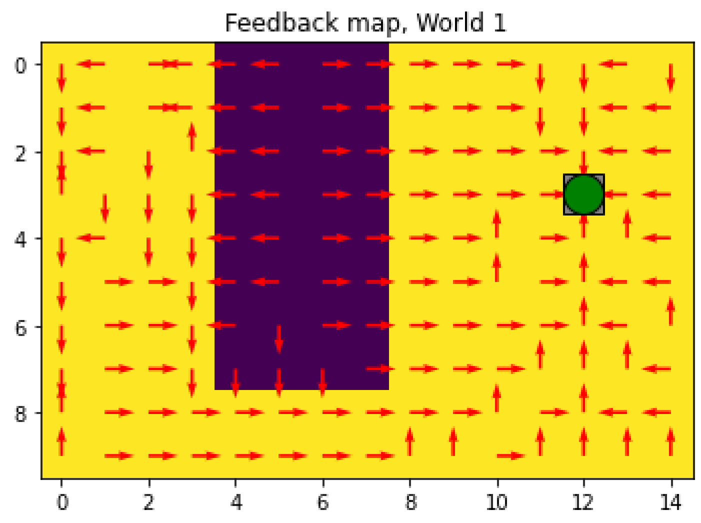
V-function:



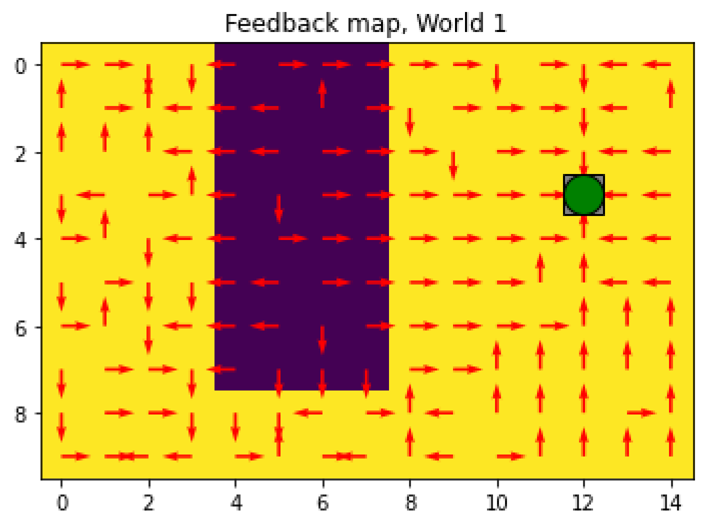
Q8. Explain how the learning rate α influences the policy and V-function. Use figures to make your point.

The learning rate is basically the amount of new information we choose to update the policy and values with. A higher learning rate uses more of the new information and vice versa. In a static context like world 1 it means a higher learning rate converges faster vs a lower learning rate which can be seen here below.

High learning rate:



Low learning rate:

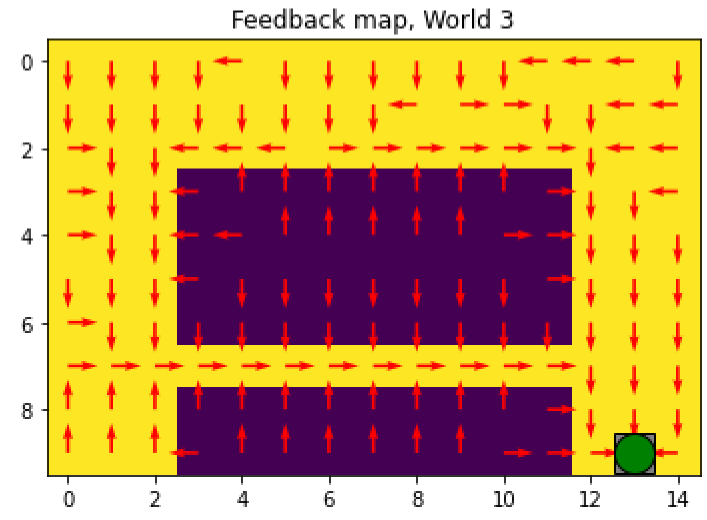


Q9. Explain how the discount factor γ influences the policy and V-function. Use figures to make your point.

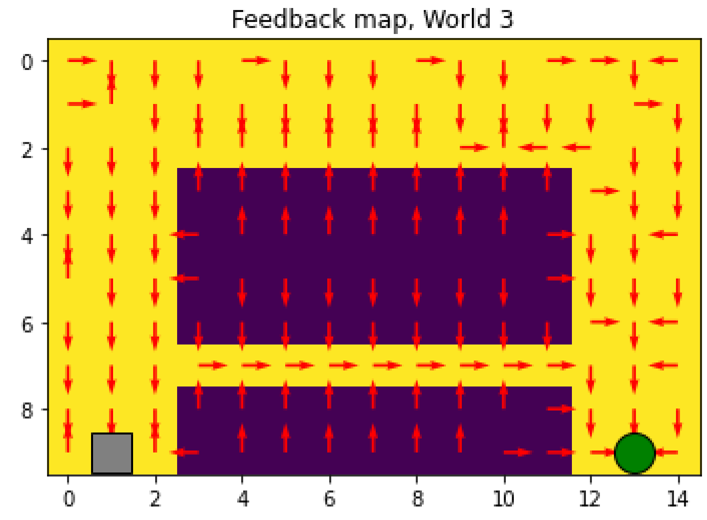
The discount factor controls how much we value close vs far rewards. A low gamma will result in a policy where the focus is on close rewards i.e dodge an obstruction or similar while a high gamma looks for the long term rewards i.e finding the goal. In the task of finding the optimal path to the goal a high gamma is good as we want to look for and also highly encourage finding the far reward, in this case the goal.

Below we can see world 3 with both high and low gamma. With high gamma we can see the arrows going towards the goal while with low gamma there are many instances where the best action is just away from the “bad” areas which results in oscillating positions as the policy basically says “the best action is always to dodge the nearest obstacle and look for close rewards”.

High gamma:

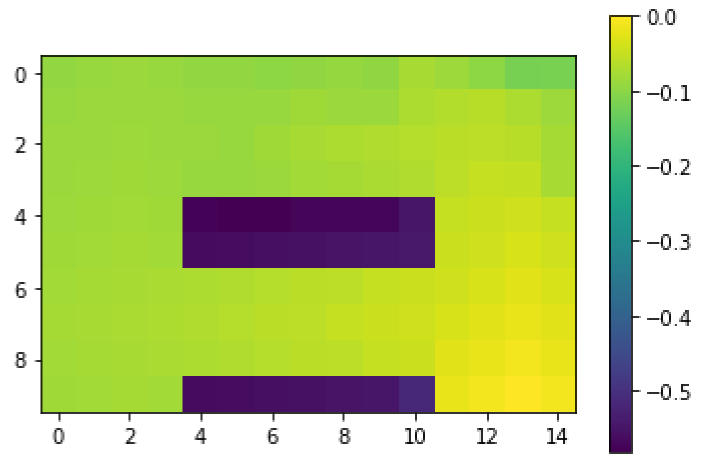


Low gamma:

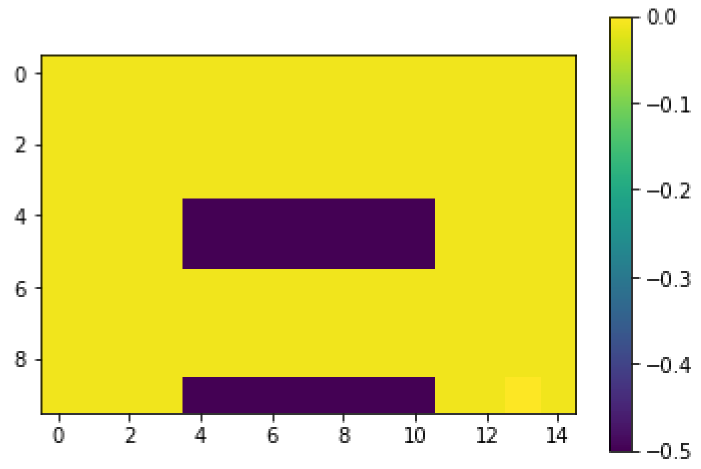


We can also see that the V-function shows that the end goal has the most value for high gamma while low gamma makes each state which isn’t an obstacle basically the same value i.e it is as good to move to a non obstacle as it is to move to the goal.

High gamma:



Low gamma:

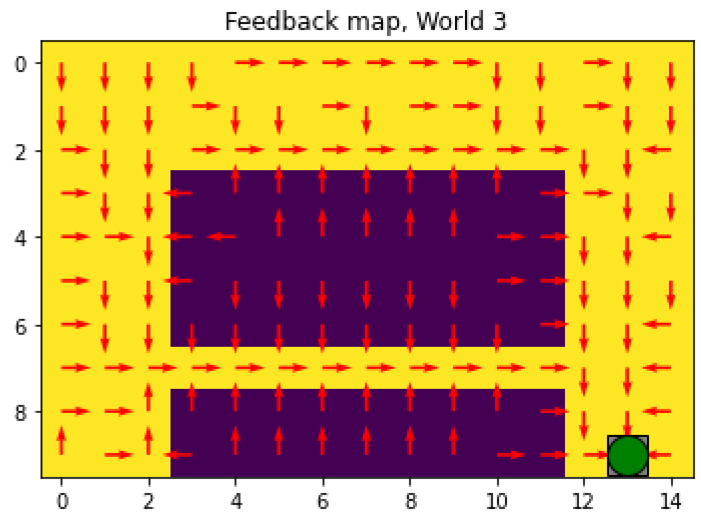


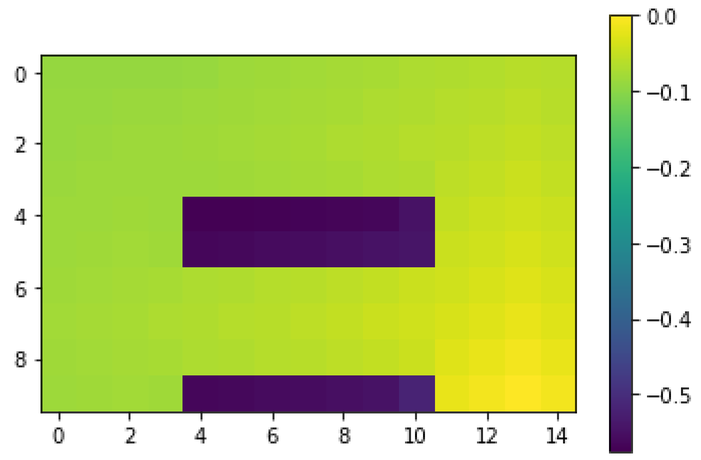
Q10. Explain how the exploration rate ε influences the policy and V-function. Use figures to make your point. Did you use any strategy for changing the exploration rate during training?

Epsilon decides the amount we should explore vs exploit. A high epsilon makes us explore more which means taking random actions and see what happens while a low epsilon makes us more likely to exploit by taking the so far best known action in a specific state. What we did was starting off with a high epsilon to explore the worlds and then linearly lower it until we reach almost full exploitation. This made sure we both got to explore around the map while still using the knowledge we accumulated during training.

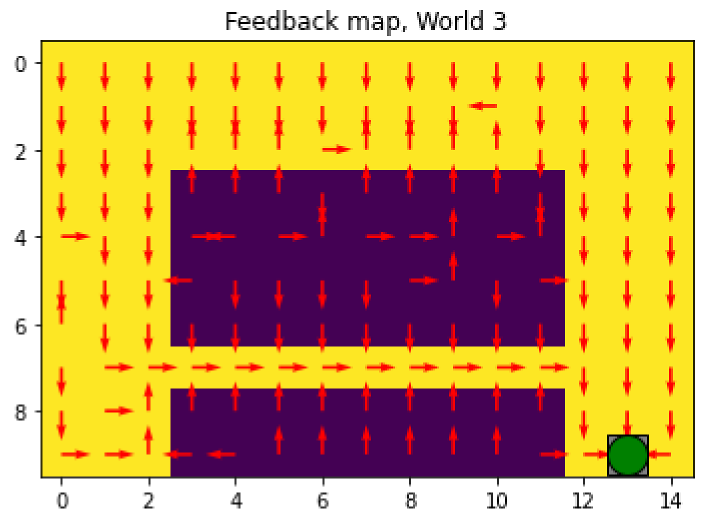
A good way to see the effect of epsilon is by looking at world 3 with both a high and low epsilon from the start. The high epsilon made sure the agent got to explore the whole world which in the end showed in the form of a good policy all over the world, despite the optimal path only being at the bottom each time. In contrast, the low epsilon made basically no exploration and only found the optimal path from the spawn, thus having really bad knowledge about the northern part of the world.

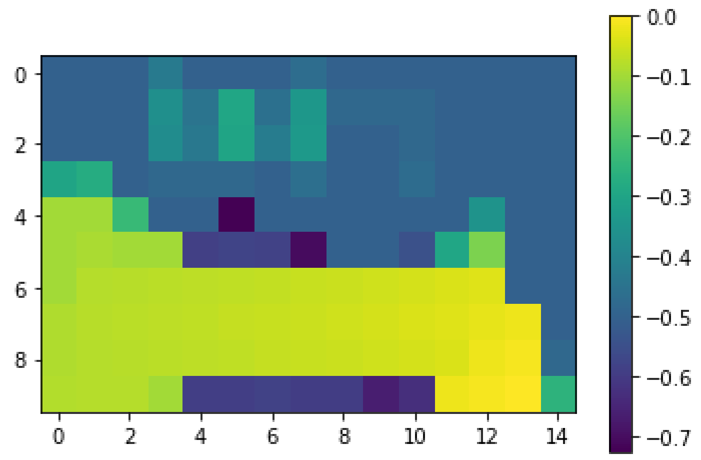
High epsilon from start:





Low epsilon from start:





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?

For the suddenly irritating blob we would have to run Dijkstra's algorithm for each new world we encounter. For the Irritating blob which is static we could run Dijkstra's one time to find the optimal route and then use that every time.

Q12. Can you think of any application where reinforcement learning could be of practical use?

Finding the optimal way to play a video game, i.e finding the optimal path through a level or trying to speedrun levels.