TBMI26 – Computer Assignment Reports
Reinforcement Learning

Deadline – March 15 2019

Sridhar Adhikarla, Naveen Gabriel:

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 <u>and</u> describe what they represent. (See lectures/classes)**

   Value functions are functions of states, or of state-action pairs, that estimate how good it is for an agent to be in a given state, or how good it is for the agent to perform a given action in a given state.
   There are two types of value functions –

   - State Value Function (V-function)

     The *state-value function* for policy π, denoted as v_π, tells us how good any given state is for an agent following policy π. In other words, it gives us *the value of a state* under π.

     $$v_\pi(s) = E_\pi[\infty\sum_{k=0} \gamma^k * R_{t+k+1} | S_t=s]$$

   - Action Value Function(q-function)

     The *action-value function* for policy π, denoted as q_π, tells us how good it is for the agent to take any given action from a given state while following following policy π. In other words, it gives us *the value of an action* under π.

     $$q_\pi(s,a) = E_\pi[{}^{\infty}\sum_{k=0} \gamma^k * R_{t+k+1} | S_t=s, A_t=a]$$

Optimal state-value function: The optimal policy has an associated *optimal* state-value function which is denoted as v_* and define as:

$$v_*(s)=\max_\pi\ v_\pi(s)$$

Optimal action-value function: The optimal policy has an *optimal* action-value function, or *optimal* Q-function, which we denote as q_* and define as:

$$q_*(s, a)=\max_\pi q_\pi(s, a)$$

Notations –

- R_t : reward at time step t
- γ^k : discount rate. Ranges between 0-1. It determines the value of a reward over time. Some rewards value depricate with time, and this determines that.
- S_t : This is the state at time srep t
- E_π : Notation for expected value

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

The Q-Value for a corresponding state action pair is updated using the **Bellman's Equation**.

$$q_*(s,a) = E[\ R_{t+1} + \gamma * \max_{a'}\ q_*(s',a')\ ]$$

This is called the *Bellman optimality equation*. It states that, for any state-action pair (s,a) at time t, the expected return from starting in state s, selecting action a and following the optimal policy thereafter is going to be the expected reward we get from taking action a in state s, which is R_{t+1}, plus the *maximum* expected discounted return that can be achieved from any possible next state-action pair (s′,a′).

The update euation for a state action pair is :

$$q_{new}(s,a)=\underbrace{(1-\alpha) * q(s,a)}_{\text{[old value]}} + \underbrace{\alpha * (R_{t+1} + \gamma * \max_{a'}\ q(s',a'))}_{\text{[new learned value]}}$$

Our learned value is the reward the agent receives from taking that action (a) from the starting state (s) plus the discounted estimate of the optimal future Q-value for the next state-action pair (s′,a′) at time t+1. The α in this equation is the learning rate, that controls how much do we have to remember from the old q-value and how much weight we must give to the new learned value. Its value is in the range (0-1).

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

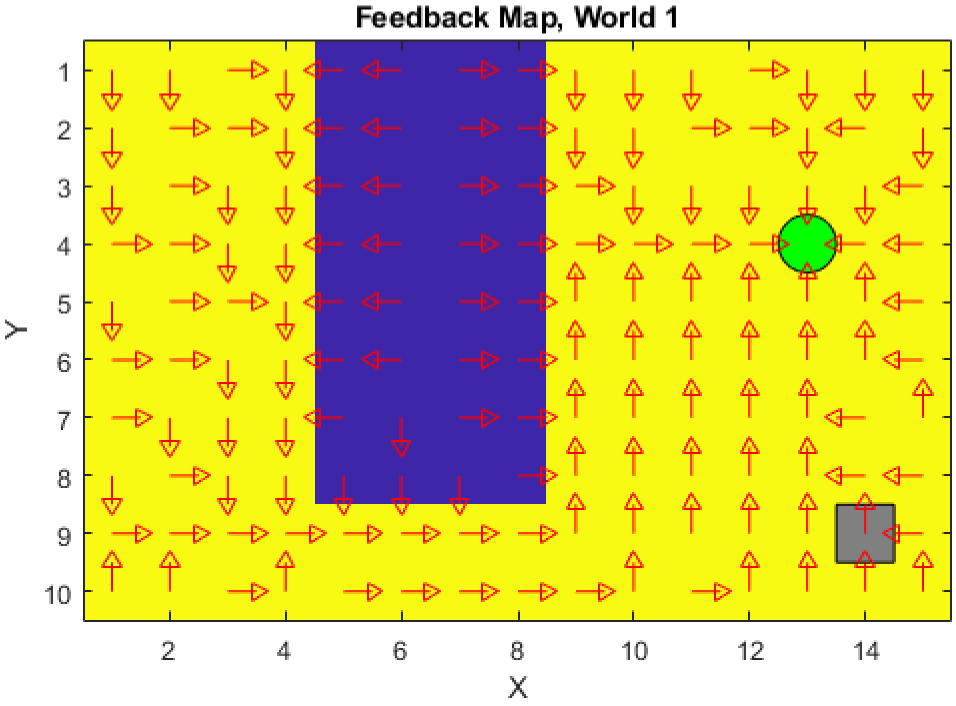
Whenever the agent tries to make an invalid move, I assign the corresponding (state, action) pair in the q table to a large negative value (-1000). This assures that this action pair would never be picked by an optimal action function as that takes the max of all the possible actions for that state.
For all other valid moves I am updating the q table according to the learning rate and the feedback received for that state using the equation stated above. This worked well for the first three world, where the actions of the robot were not random.
In the forth world, the actions of the robot were random, so setting a large negative value for an invalid move failed because, there was stochasticity in the actions of the robot. With 0.3 probability it would not do the action we intend. So, I had to remove the setting of large negative weight for the forth world. It turned out that the algorithm, when trained for a longer time, figures out on its own the invalid moves and does not pick them in its optimal policy. This was something Interesting I got to learn from the forth world.
I even set a limit on the max number of steps allowed in each episode, so that the algorithm does not run forever. I did this because in some worlds (world 11), the way to maximize rewards is not to terminate but to go to a region where you get better(positive) rewards, instead of the termination state. So, you need a max number of steps so that the algorithm does not run forever.

I chose the parameters for the world by using the optimal q-function obtained after training, on 1000 simulations on that world, and chose the parameters that reached the goal all thousand times. Then I looked at the plot for the optimal q function and decided on the parameters for the world.

4. 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.

In this world we used random initialization of the robot. The world has a patch colored blue that has higher negative rewards. The goal of the Reinforcement Learning algorithm is to maximize rewards. The only way this is possible is if the robot terminates the world fast by reaching the terminal point, marked with a green circle. The robot figures that with time and calculates its optimal policy for the world, which tries to avoid the blue patch with higher negative reward.
The parameter values used for this world are:

- `num_episodes = 2000; (2000 simulations were run on this world to get this optimal policy)`
- `max_steps_each_episode = 500; (the max number of time steps that were allowed in a episode, so that the robot does not get stuck in a simulation forever)`
- `learning_rate = 0.2; (A higher learning rate value was chosen for this world as this was a very easy world to learn for the robot.)`
- `discount_rate = 0.9;(discount rate of 0.9 was chosen)`
- `exploration_rate = 1; (At the start the exploration rate was set to 1 and then it kept decaying over time.)`
- `exploration_rate_decay = 0.001; (This is the deacy rate for exploration)`



The optimal policy learned by the algorithm for the first world.

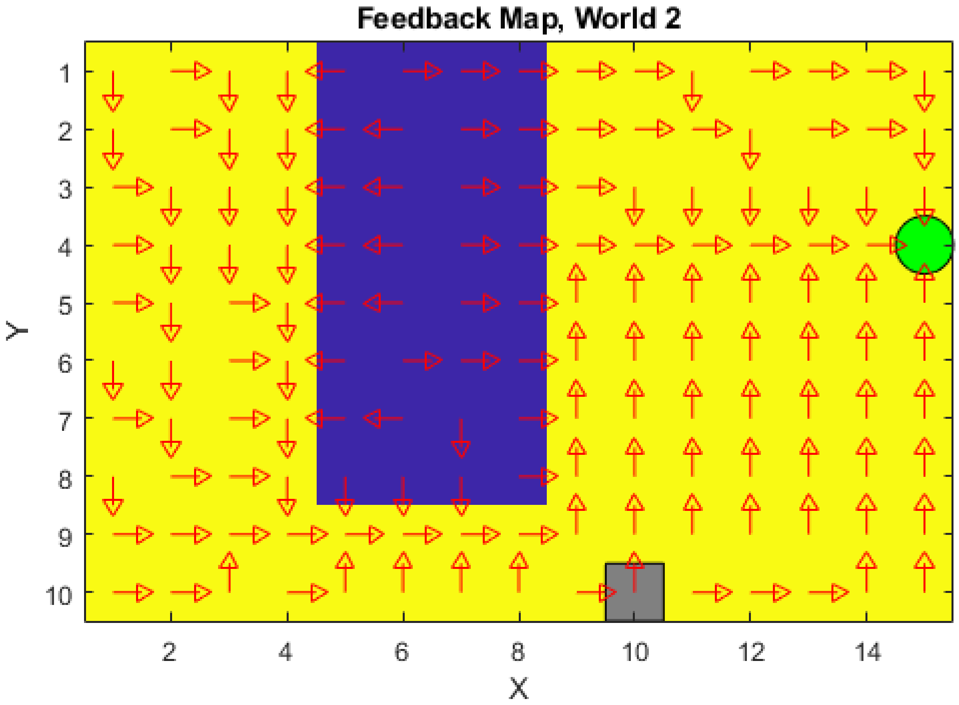
5. Describe World 2. What is the goal of the reinforcement learning in this world? What parameters did you use to solve this world? Plot the policy and the V-function.

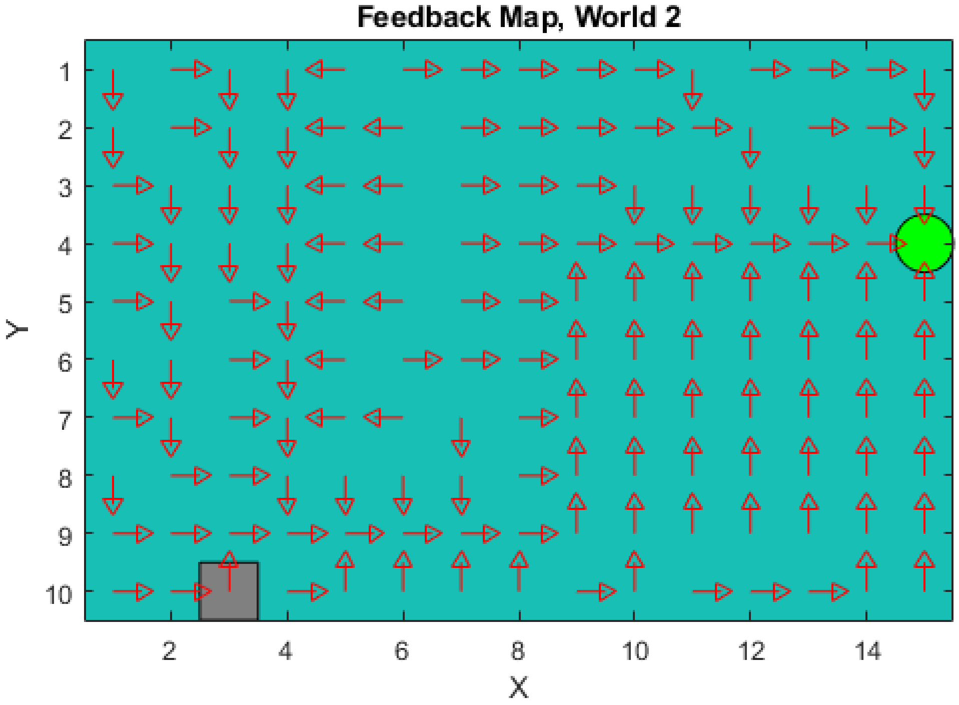
In this world we used random initialization of the robot, and the world is also random. With 0.2 probability we get the first world, else we get a uniform world where each state has a similar reward. The goal of the Reinforcement Learning algorithm is to maximize rewards. The only way this is possible is if the robot terminates the world fast by reaching the terminal point, marked with a green circle. The robot figures that with time and calculates its optimal policy for the world, which tries to avoid the blue patch with higher negative reward, in all the worlds. Even if it is not the first world the algorithm tries to play safe and assumes that the patch is there and goes around it. It does so because the negative reward associated to each time step in the normal state is much lower than the negative reward in the blue patch in the first world(Which has a 23 times larger negative reward). So, the algorithm tries to play safe and goes around the patch in both the worlds. It cannot figure out which world is going to come up as it is a random pick, so the optimal path for this algorithm in this world is going around the patch in both the worlds.

The parameter values used for this world are:

- num_episodes = 3000; (3000 simulations were run on this world to get this optimal policy)
- max_steps_each_episode = 500; (the max number of time steps that were allowed in a episode, so that the robot does not get stuck in a simulation forever)
- learning_rate = 0.2; (A higher learning rate value was chosen for this world as this was a very easy world to learn for the robot.)
- discount_rate = 0.9;(discount rate of 0.9 was chosen)
- exploration_rate = 1; (At the start the exploration rate was set to 1 and then it kept decaying over time.)
- exploration_rate_decay = 0.001; (This is the deacy rate for exploration)

Almost the same parameters worked for this world also. I just had to change(increase) the number of episodes required to get the optimal policy.

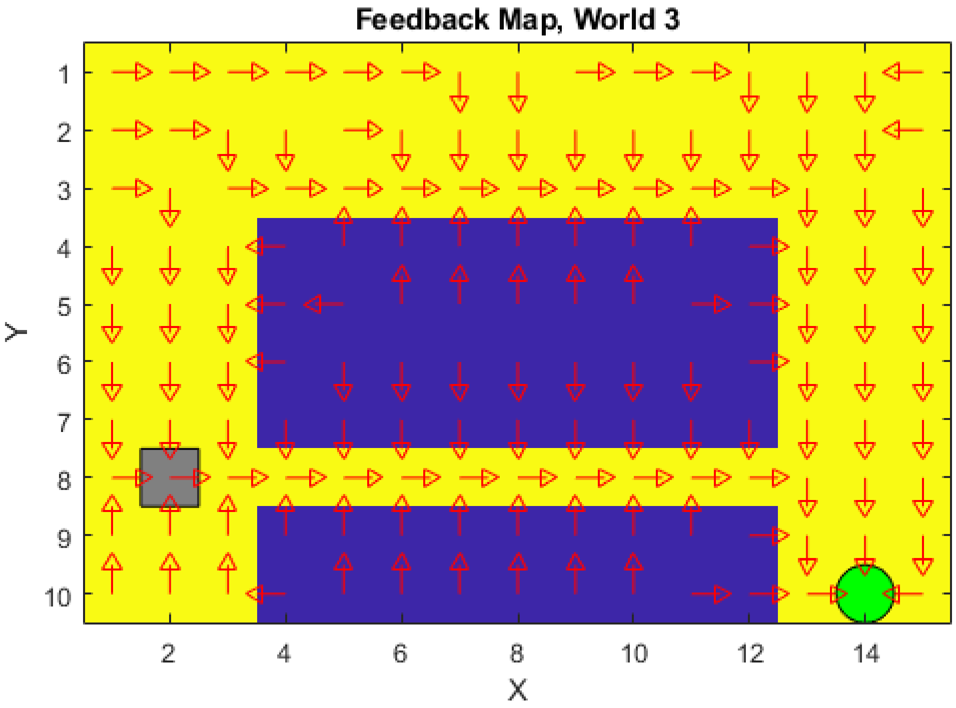




6. **Describe World 3. What is the goal of the reinforcement learning in this world? What parameters did you use to solve this world? Plot the policy and the V-function.**

The third world has no randomness in it. The robot is initialized at the same point every time and the world remains unchanged, it has some areas with higher negative rewards, and some with low negative rewards. The algorithm is still able to figure out the optimal path from every point in the world to the terminal point, and this is all because of the exploration of the robot in the initial simulations. The exploration rate keeps decreasing as the number of episodes keep increasing. We used a leger learning rate so that the new learned q-value gets higher weightage. This helps the algorithm to get the optimal q-value for states that are rarely visited easy. I thing increasing the learning rate in this world will just help in improving the optimal q-values as there is no randomness involved in it. The parameters used for this are:

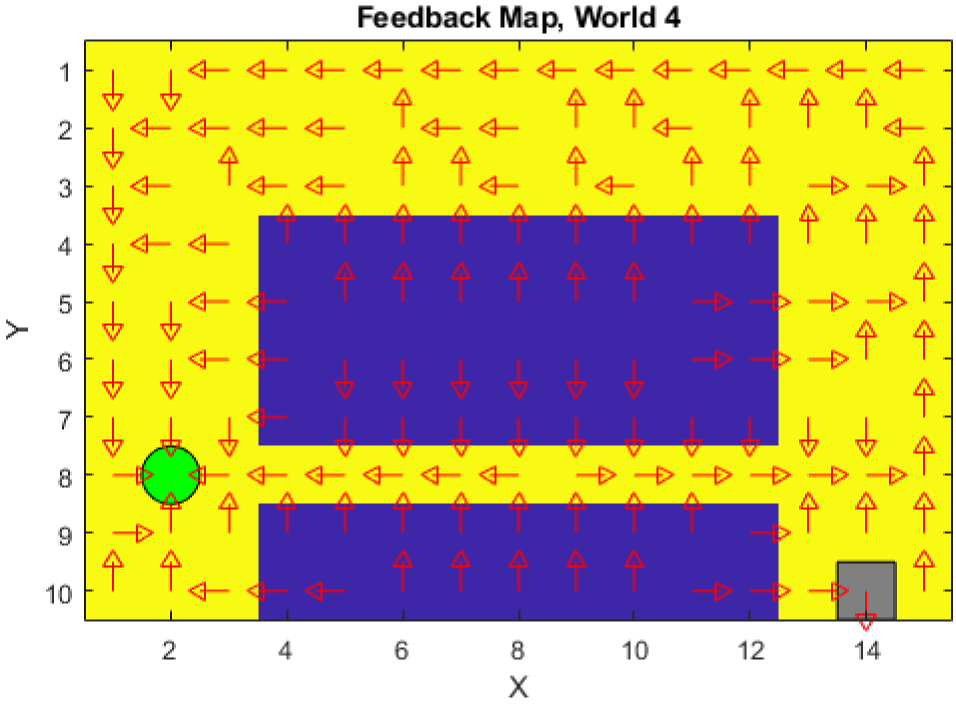
- `num_episodes = 3000; (3000 simulations were run on this world to get this optimal policy)`
- `max_steps_each_episode = 1000; (the max number of time steps that were allowed in a episode, so that the robot does not get stuck in a simulation forever. I increased the max number of steps each episode, so that the robot could explore all the states as it is initialized at the same point everytime.)`
- `learning_rate = 0.4; (A higher learning rate value was chosen for this world, as the robot is initialized at the same point every time, only randomness can get it to some states. Having a larger learning rate will help it update the q-values with the learned new value for those states that were rarely explored easy.)`
- `discount_rate = 0.9;(discount rate of 0.9 was chosen)`
- `exploration_rate = 1; (At the start the exploration rate was set to 1 and then it kept decaying over time.)`
- `exploration_rate_decay = 0.001; (This is the deacy rate for exploration)`



7. 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.

This world is like the third world, but with stochasticity in the actions. With a 0.3 probability the action won't do what its asked for. For example, an up action can go up with 0.7 probability, and can go in any other direction with a probability of 0.3. The approach we had been using for the first three worlds wont work in this one. We had been setting the q-value of invalid moves to -1000(a large negative number), so that it is not picked the next time. But now due to random actions even a valid action may get a q-value of -1000. So, I removed the part where we checked for invalid moves, and trained the algorithm with this. I had to run it for more number of iterations but the algorithm learned not to pick up invalid moves in its optimal policy. It found the safest path that goes around the corners avoiding the area with larger negative rewards. The parameters for this world were:

- `num_episodes = 10000; (10000 simulations were run on this world to get this optimal policy)`
- `max_steps_each_episode = 500; (the max number of time steps that were allowed in a episode, so that the robot does not get stuck in a simulation forever.)`
- `learning_rate = 0.1; (A lower learning rate value was chosen for this world.)`
- `discount_rate = 0.9;(discount rate of 0.9 was chosen)`
- `exploration_rate = 1; (At the start the exploration rate was set to 1 and then it kept decaying over time.)`
- `exploration_rate_decay = 0.001; (This is the deacy rate for exploration)`



The optimal path for this world is to stick to the walls and go along the walls so that even if a move goes wrong we still don't end up in the area with higher negative rewards.

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

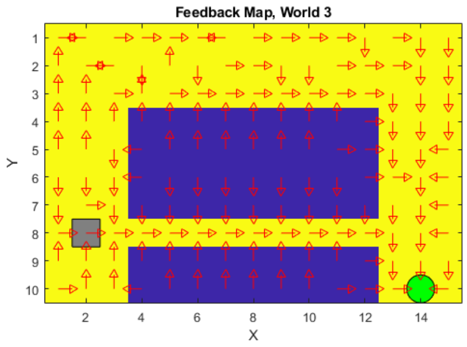
The learning rate is a number between 0 and 1, which can be thought of as how quickly the agent abandons the previous Q-value in the Q-table for a given state-action pair for the new Q-value. We don't want to just overwrite the old Q-value, but rather, we use the learning rate as a tool to determine how much information we keep about the previously computed Q-value for the given state-action pair versus the new Q-value calculated for the same state-action pair at a later time step. The higher the learning rate, the more quickly the agent will adopt the new Q-value.

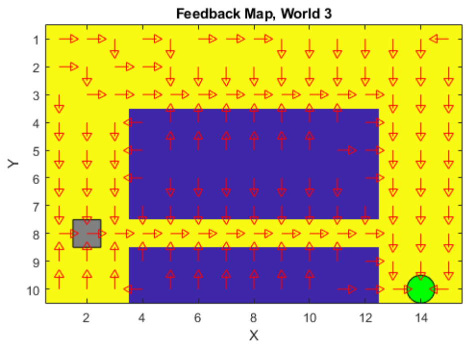
As we increase the learning rate it gives rise to more parallel lines in the optimal policy, and I think it is because as we increase the dependence of the optimal policy on the latest timestep in which that state occurred increases. The value for this is extracted from the one before that. This is the reason all the lines get parallel and keep following towards the goal state. This worked well for the third world as there was no randomness in it.

On increasing the learning rate for the 4th world, where the actions are random, the optimal policy starts getting messy. It needs the previous knowledge to find an optimal policy through the randomness in the world.

So, higher learning rates are good for worlds with no randomness, but for worlds with randomness the learning rate needs to be small in order to tackle the randomness.

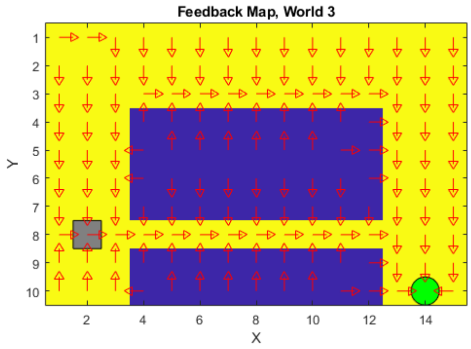
WORLD 3

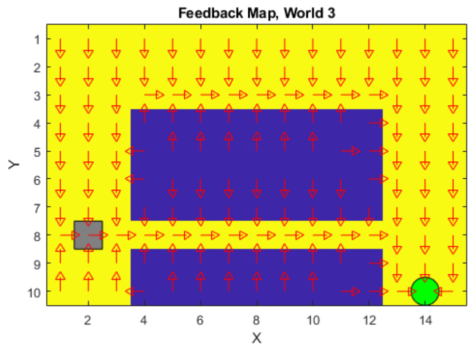




Learning Rate: 0.1

Learning Rate : 0.5



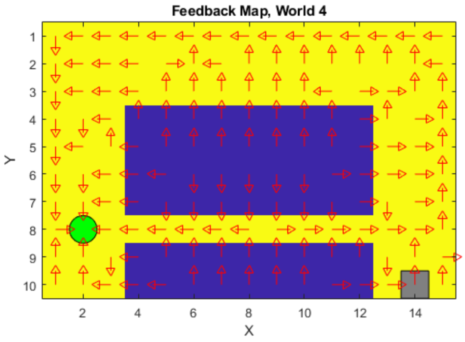


Learning Rate: 0.7

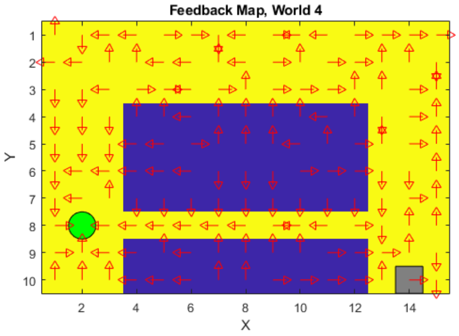
Learning Rate : 1.0

As you can see from the plots, the optimal policy keeps getting better as the learning rate increases. This is because this world has no randomness in it.

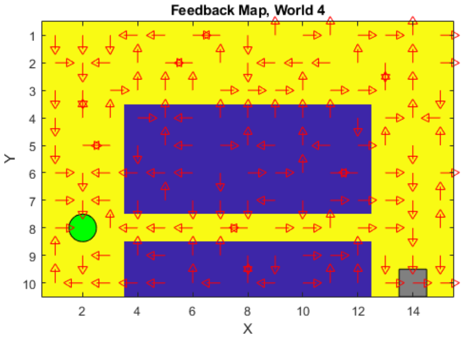
WORLD 4



Learning Rate: 0.1



Learning Rate : 0.5



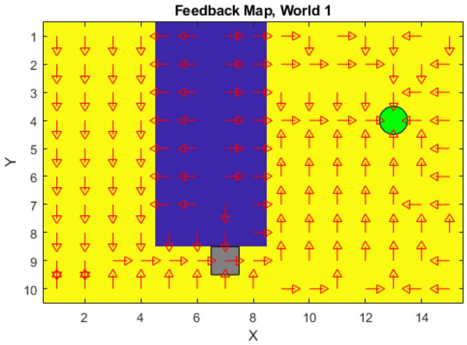
Learning Rate: 0.9

As you can see from these plots, increasing the learning is not working well for this world. The optimal policy keeps getting bad as we keep increasing the learning rate. This is because of the randomness involved in this world.

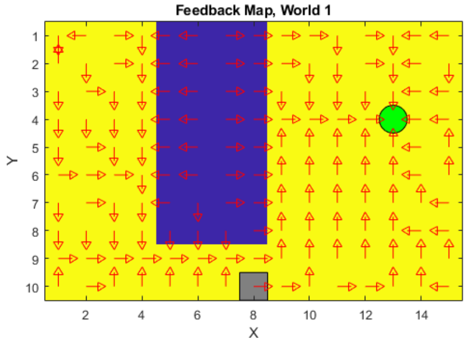
9. **Explain how the discount factor γ influences the policy and V-function in each world. Use figures to make your point.**

We use discount when, sooner rewards probably have higher utility than later rewards. This also helps the algorithm converge as now the algorithm can differentiate between two sequence of rewards. Discount of 1 means there is not discount, the rewards remain same over time. A small value of discount as 0.1, means that the rewards decay very fast, like a reward of 1 in the first timestep will be 0.1 * 1 in the second time step, and $0.1^2 * 1$ in the third timestep and so on. This world works well even without discount (ie. Discount=1), but if we use a very small value for discount, paths further away from the termination point, find it hard to connect the dots and reach the terminal point as, the reward is completely decayed by the time it reaches them. We can see some cycles formed in the plots corresponding to the discount values of 0.1 and 0.5, that are not able to connect the path to the terminal. We can also see that the number of cycles decrease from the plot of 0.1 discount to 0.5, and reduces to no cycles in discount 0.99.

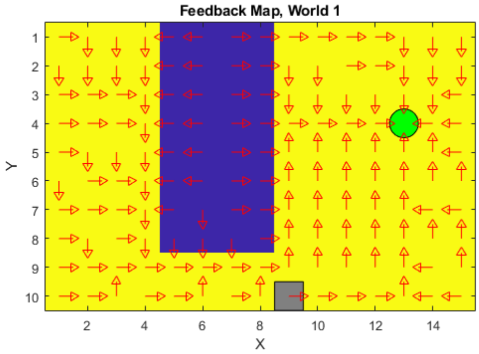
I would say using a value of 0.9 to 0.99 for discount would be reasonable for these worlds.



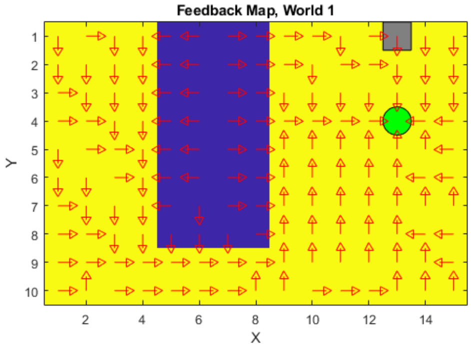
Discount: 0.1



Discount: 0.5



Discount: 0.99



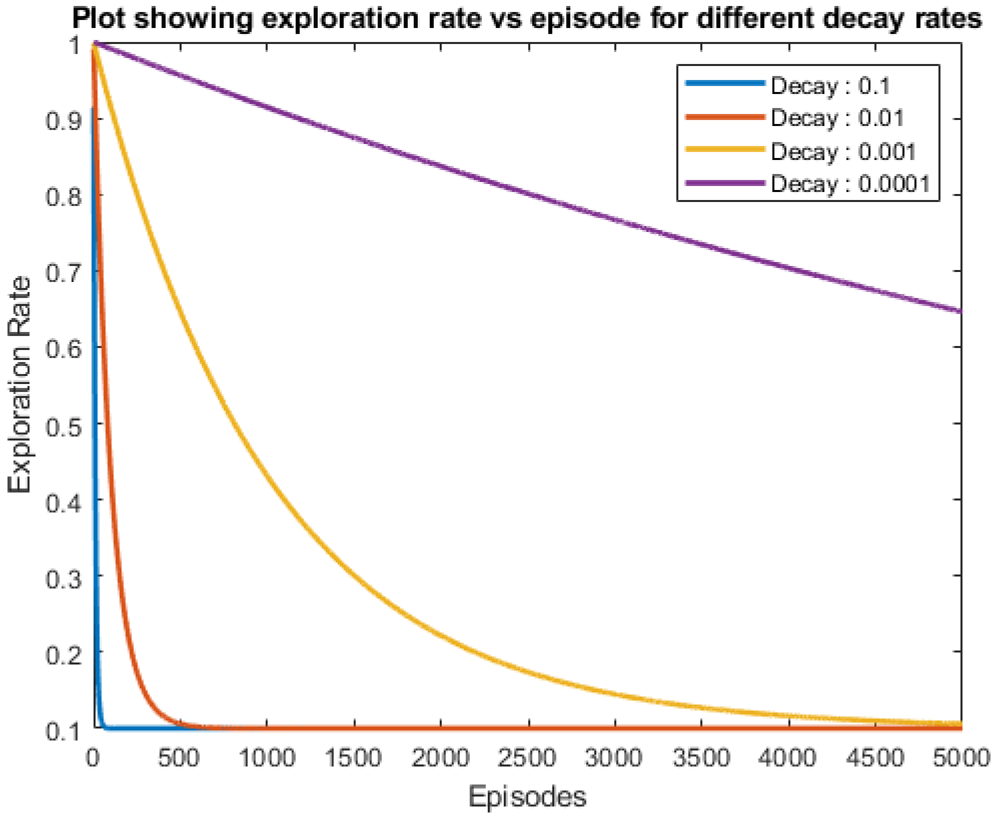
Discount: 1.0

10. **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?**

Exploration rate is a very important parameter that helps the algorithm improve over time. Exploration rate is a value between 0 and 1. We generate a random number between 0 and 1 and if the random value is greater than exploration rate we exploit, ie. Use the optimal value from the q table to choose and action, and if the random value if smaller than the exploration rate we choose a random direction and move that way. This helps us explore random moves and set their corresponding q-values in the q-table. We start with exploration rate set to 1 and keep decaying it as the episodes increase. Setting exploration rate to 1 at the start

is very important as we start with the q-table values all zeros. We need to explore different states and update their Q-values so that we can exploit it later. When we start we know nothing about the environment/world so setting the exploration rate to 1 is very important. As we keep learning about the world we decay the learning rate.
I set the upper limit and lower limit for this parameter as 1 and 0.1 correspondingly. I then update the exploration rate at the end of every episode using decay rate of 0.001 in the formula:

```
Exploration_rate = min_exploration_rate + (max_exploration_rate -
min_exploration_rate)*exp(-exploration_rate_decay*episode);
```



11. 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?

When we start with a world in Reinforcement learning, we know nothing about the world. We just have the goal, “Maximize the Rewards”. On training it explores and learns that going to the termination state is the way we can maximize our reward and changes its optimal policy accordingly.

Dijkstra’s algorithm would require a defined goal state, and we don’t have that before training. So, the algorithm would not be able to converge. We would have to change the goal of the task for this algorithm to work. We would have to keep the goal as finding the cheapest path to the termination state. That would make this algorithm work in both the worlds. It would find the cheapest path better than the reinforcement learning optimal policy, but it would take a long time to find that.

The optimal policy of a reinforcement learning bot has all optimal values calculated during training, so it does not have to do much when testing and works quickly. But Dijkstra’s algorithm is just the opposite, it requires no training, but take a long time when testing, and is bad as the world keeps getting larger.

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

The way we used to find the optimal path in a simulated environment, we can make a robot explore a real-world environment and learn it, so that if it is asked to go to the closest goal it knows where to go once it has localized itself.
Even in a self-driving car we can make the reinforcement algorithm learn to keep the car in between the lanes. This is already been used in many of the implementations of self-driving cars.

13. (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?

This algorithm with the same parameters worked for almost all the worlds except for the World 11. For rest of the worlds I got 100% termination for the trained model while testing. I have saved all the trained q-tables in the same location.
But for world 11, I got 0% termination for the trained model. The model learned not to terminate as there were some states the had a high positive reward, and the algorithm learn to spent all the time in that area to maximize its reward instead to termination. The goal of the algorithm remained the same, maximize the reward, but the way the algorithm achieves it was changed.
So, I can conclude that the implementation worked on all the worlds. The q-table I got was not the optimal for worlds 5-12, it can be improved further by changing the parameters, but the implementation worked on all the worlds.