Assignment 1 - Report

CSE2530 Computational Intelligence

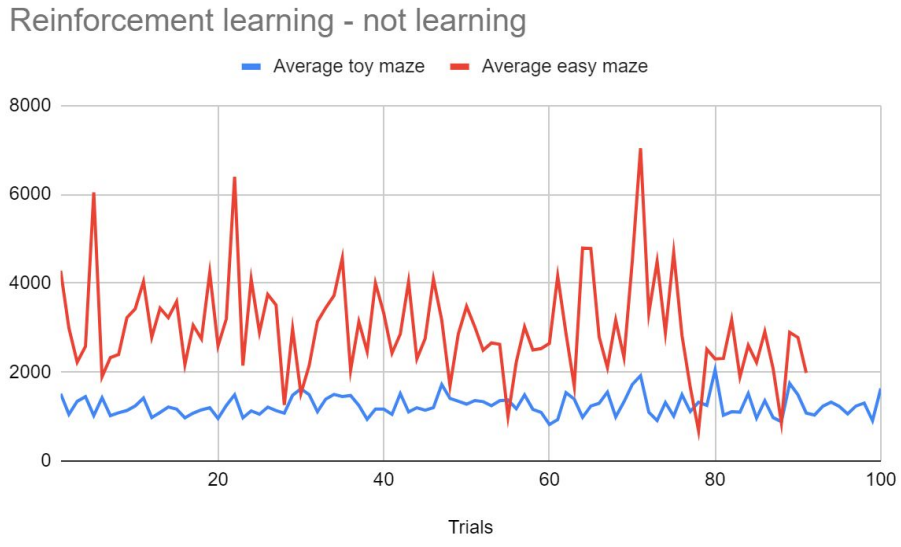
2.1 Development

1. How do you ensure that the agent does not have a bias for selecting the same action over and over in getBestAction() if it did not learn something yet?
If it didn't learn anything, then the *maxVal* variable will be equal to 0, and stay that way when comparing to the values of the possible actions. These actions are added to the ArrayList *listMax* and chosen at random, ensuring there is no bias for a particular action.

2. Explain your agent's cycle in RunMe()
(selecting actions, executing them, calling the updateQ etc...)

1. Select an action based on the ϵ-greedy Algorithm, taking either the best action or a random one, depending on the epsilon value.
2. The agent performs the action.
3. Update the Q values, as per the Q-learning algorithm.
4. Check if the agent is at the goal, and if he is, reset the agent back to the starting position.

4. Run your agent, it should print to the console a list of numbers, representing the steps taken between resets to its starting position. The period between two resets is called a trial. Make a plot of the average of 10 runs (one run is one launch of RunMe) to show that the agent indeed does not learn. The x-axis in your plot refers to the trial in your runs, the y-axis refers to the average number of steps needed in that trial. Make this plot for both mazes given. Explain your plots.



As seen in the plot above, the agent does not learn. The average amount of steps per trial might oscillate between the trials, but the amount of steps doesn't decrease in successive trials indicating that it does not learn from past experiences.
Note that since the average amount of steps per trial is higher for the easy maze, we can see that the easy maze is harder to solve than the toy maze.

5. Implement updateQ() in MyQLearning. Set α, γ and ϵ to 0.7, 0.9 and 0.1 respectively. This should be a straightforward implementation of Q-Learning. Explain your method in your report.

We followed the formula given in the assignment:

$$Q(s,a)_{new} = Q(s,a)_{old} + \alpha(r + \gamma Q_{max}(s', a_{max}) - Q(s,a)_{old})$$

The implementation can be summarized by the following:

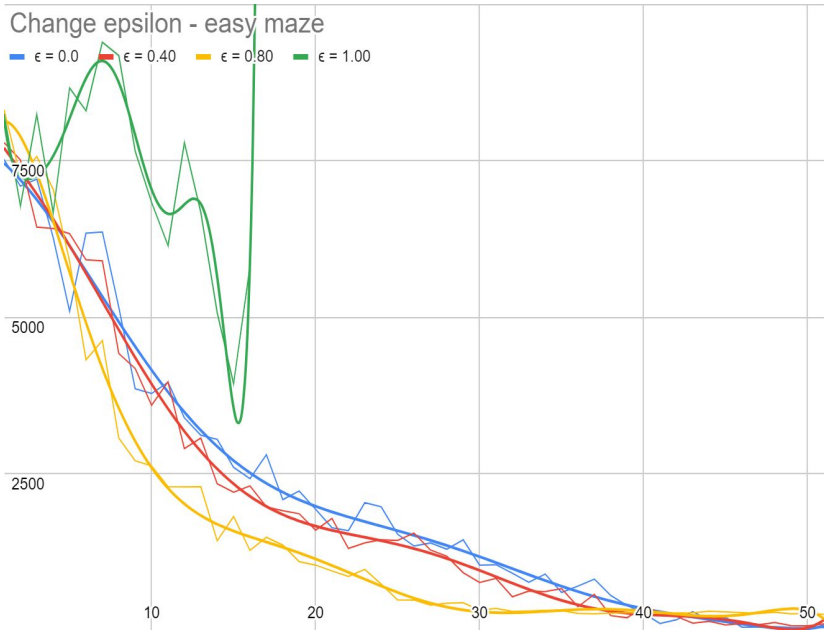
1. Find *Qmax* by going through the possible actions, pick the one with the biggest Q and return it.
2. Get *Qold* by calling *getQ()*.
3. Using the formula above to calculate *Qnew.*
4. Call *setQ* with the newly obtained *Qnew* as one of its parameters.

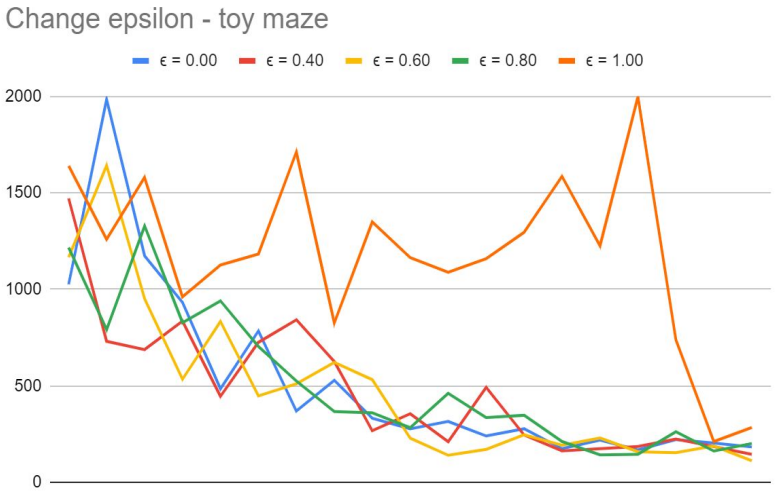
6. Now run your agent again on the mazes. You should be able to see that the numbers decrease over time during a run. Again, make a plot of the average of 10 runs to show that the agent indeed does learn. Make this plot for both mazes given. Explain your plots.

As we can see in the plot below, the agent does indeed learn this time. The average amount of steps per trial starts high but we can see that over time, i.e. over the amount of trials, the amount of steps per trial slowly decreases till the agent finally converges to around 25 steps for both mazes. This is a major improvement considering that the agent started with 1337 and 3051 steps for the toy and easy maze respectively.

2.2 Training

1. Play around with the ϵ parameter of your action selection algorithm. Make a couple of plots of 10 run averages (same plots as above) to study the effect of varying epsilon between 0 and 1. Explain your plots.





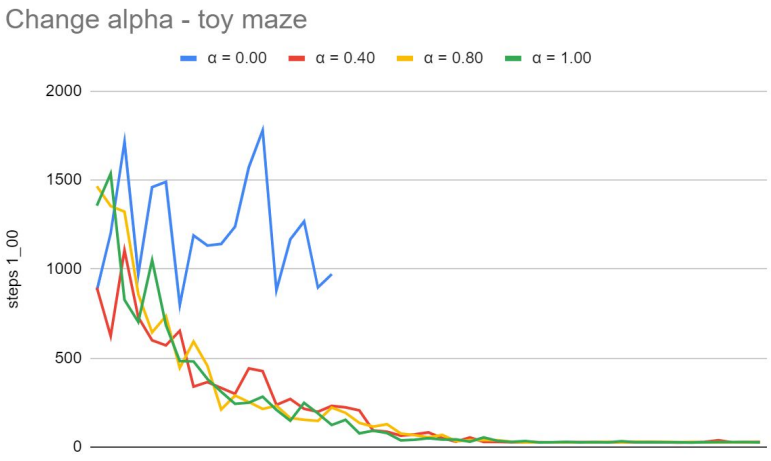
For each maze, we picked 4 different values of ϵ and from this comparison, we can see that the algorithm performed best for values 0.4 and 0.8 and worse for a value of 1.0

This is because, at ϵ = 1, the actions are selected at random, without consideration of rewards and action-value estimates.

For values 0.4 and 0.8, the algorithm behaves greedily for a probability of 1 - ϵ and strikes a balance between exploration and exploitation, causing it to complete trials in the optimal amount of steps.

This is similar for the toy maze, where we see that it takes the most amount of steps per trial initially with values 0 and 1. We also note that the greedy method (ϵ = 0) improves to less than 500 steps per trial by the 5th trial, faster than ϵ = 0.6, 0.8 and 1.0, but plateaus afterwards.

2. Play around with the learning rate α of your QLearning algorithm. Again make a couple of plots to study the effect of varying α between 0 and 1. Explain your plots.



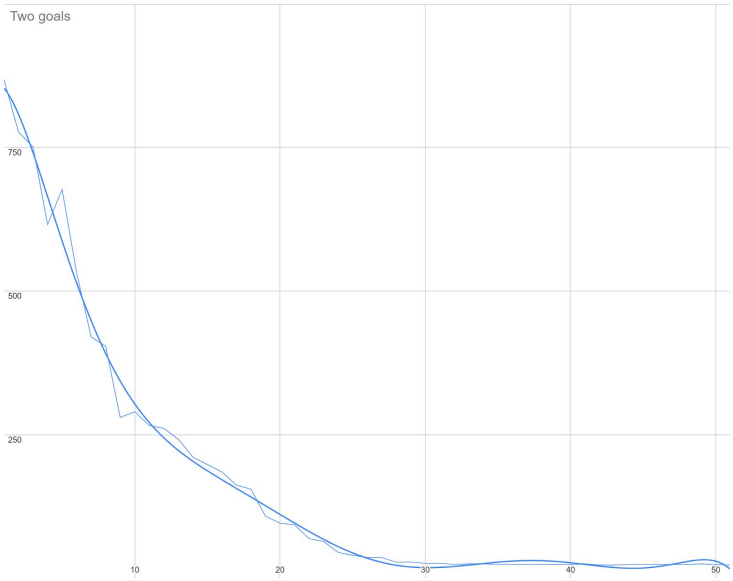
The alpha value denotes the learning rate, or the *step size*, which signifies the extent to which an algorithm evolves by processing the reward in each iteration.

Similar to the above, different values of **α** were picked and plotted, with a constant ϵ = 1.0. For **α** = 0, the algorithm only took prior knowledge into account while an alpha value **α** of 1 causes the algorithm to always 'learn' with each iteration.

We can see that for non-zero values of **α**, the closer it is to 1 the lower the amount of steps per trial. Given the learning rate, this is due to the higher expected return on an action, i.e. the value of taking that action in a current state.

3. What are the trade-offs between a high and a low ϵ ?
A high ϵ results in the algorithm selecting a random action as opposed to a greedy action more often, improving its chances of recognizing the optimal action and leading to more exploration. With a high epsilon value, even though the expected reward on a step might be lower, the total reward in the long term is greater.

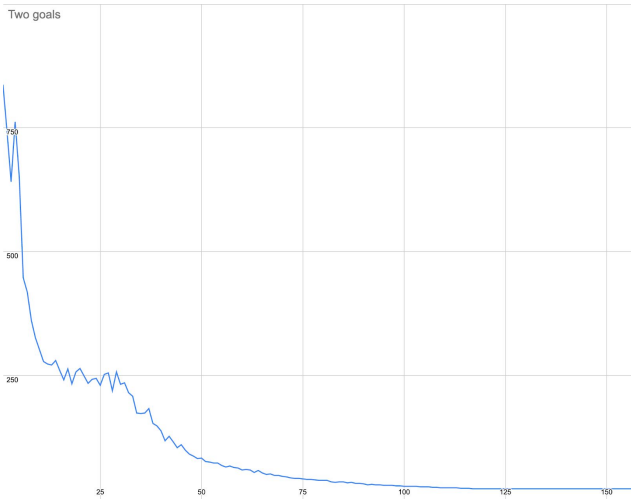
However, if we know the rewards to be similar to one another, a low epsilon value or fully greedy methods could be more beneficial as they would find the optimal action in less time and then further exploration is no longer necessary. Thus, fixing the value of ϵ involves finding the right balance between exploration and exploitation when maintaining estimates of action values.



2.3 Optimization

1. Add a second reward, sized 5, at the location (9,0) (so at the top right). Now run your program again a couple of times. What do you observe, and how can you explain this?
As we can see in the plot below, the agent still learns from experience but now he oscillates between the two goals, resulting in different amounts of steps in the trials and it isn't converging correctly.

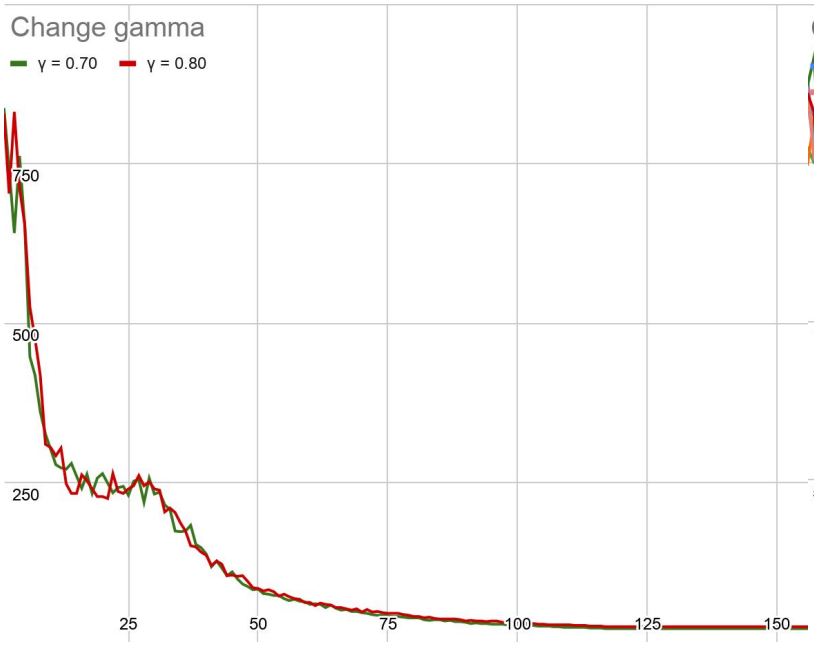
2. Invent a way to mitigate the previous problem using an ϵ that can change with each trial. Explain your method and show with a plot that the agent now indeed converges to the optimal solution.

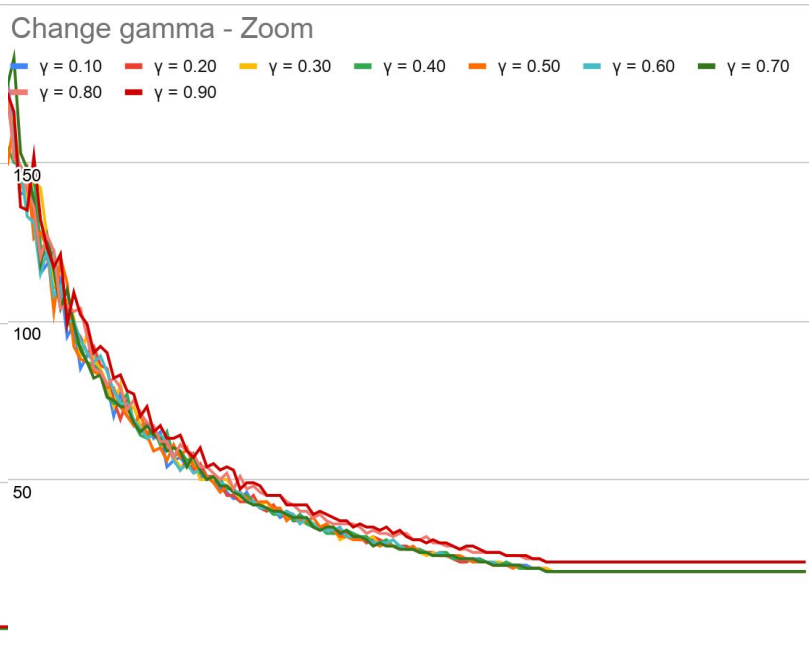


In order to ensure convergence to the optimal reward, we have maintained a constant value $\epsilon = 0.9$ for the first 30 trials, following which it is decreased by 0.01 per trial, till ϵ equals 0.

As a result, the algorithm converges to the optimal solution of 24 steps per trial by the end.

3. Can you, using your previous solution, experimentally find a γ for which the optimal solution is in fact to go up and get the smaller reward, rather than going down for the larger? Explain and show plots for the different values of γ you try.





If γ is smaller than 0.7 then the agent will go to the smaller goal, and if γ is larger than he will go to the bigger goal.