Autonomous Agents - Assignment 2 Report

Nicolò Girardi, Jorge Sáez Gómez, Johan Sundin

October 4, 2013

1 Introduction

Something here...

2 Method

In order to conduct our experiments, we made use of the reduced state-space we designed for the previous assignment. A detailed explanation of this state-space can be found in our previous report. Even if the algorithms do not sweep through the state-space directly, because they sweep through state-action pairs within episodes, they indirectly benefit from having a reduced state-space. This is due to the fact that the state values will converge sooner to their true value, because more samples are averaged for any given state in the reduced state-space versus the original one. As a result, the algorithms converge faster to the optimal solution.

We ran all of our experiments until 3000 episodes were generated for each algorithm. These episodes were always started with the prey at the location $(5, 5)$ and the predator at the location $(0, 0)$. Whenever a measure of the steps needed to catch the prey is presented, we ran 1000 episodes to determine the average of its value. We evaluated all algorithms by using the corresponding greedy policy that was returned by them. We chose to evaluate greedy policies over ϵ-soft derivated policies of the output of the algorithms because the performance of the latter ones is usually worse, and thus we would be interested in the first ones in real-world applications. Since we were testing greedy policies here, there were some cases when their application resulted in never-ending games. This is represented in our graphs by the absense of points.

We first conducted some experiments for the *Q-Learning* algorithm. Figure 2 shows the average number of steps needed to catch the prey when the *Q-Learning* algorithm is run for different combinations of its parameters α and γ. We performed this initial experiment in order to determine which combinations of these parameters could be interesting for further experiments. Afterwards, we looked into the performance of the greedy policy returned by *Q-Learning* at different points in its execution process for different combinations of the parameters α and γ. Furthermore, and in order to obtain more consistent results, we averaged the performance results across the execution of 10 instances of the *Q-Learning* algorithm. These results are shown in figure 3.

Furthermore, we also experimented different combinations for ϵ and the initial values of the Q table. We used values of ϵ ranging from 0.1 to 1 in order to cover all possible cases, from using the almost-greedy policy to the random walk scenario, respectively. Note that it makes no sense to use the 100% greedy policy here since, as a result, for those cases when the Q table is initialized optimistically, the episodes will never end. For the initialization of the Q table we tried the values $\{30, 15, 0, -15\}$, since this set of values range from a very optimistic initialization to a very pessimistic one. We fixed α and γ to 0.1 for all these tests. Our results are summarized in figure 4.

We also compared the effect of using ϵ-greedy policies versus softmax action selection for the *Q-Learning* algorithm. We compared the evolution of the solutions when using the whole range of values for the parameters ϵ and τ of the corresponding method. We used $\alpha = \gamma = 0.1$, since this makes the convergence of the algorithm slower and allows us to better visualize the results. An optimistic initialization of $Q(s, a) = 15, \forall s, a$ was used. Figure 5 shows our results for this experiment.

We then implemented Monte-Carlo control methods, both on-policy and off-policy. For the off-policy case, we used a random walk as the policy π' used to generate the episodes. The reasons for this choice of ϵ-soft policy are twofold. Firstly, as we will see, this policy performed the best in our experiments. Secondly, and more importantly, this allows us to simplify the calculation of the variable w used within the algorithm. We have that:

$$w \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k, a_k)} = \prod_{k=t+1}^{T-1} |\mathcal{A}| = |\mathcal{A}|^{T-1-t}$$

Where $|\mathcal{A}|$ is the number of available actions for the agent, 5 the case of our predator. We then compared both algorithms for different values of γ and fixed values of $\epsilon = 1$ and $Q(s, a) = 0, \forall s, a$. Our results from this experiment are summarized in figure 6.

Something about figure 7.

3 Results

Random policy	Optimal policy
275.2661	10.061098

Figure 1: Average number of steps needed to catch the prey for different policies, shown for reference purposes. These averages were calculated by simulating one million games.

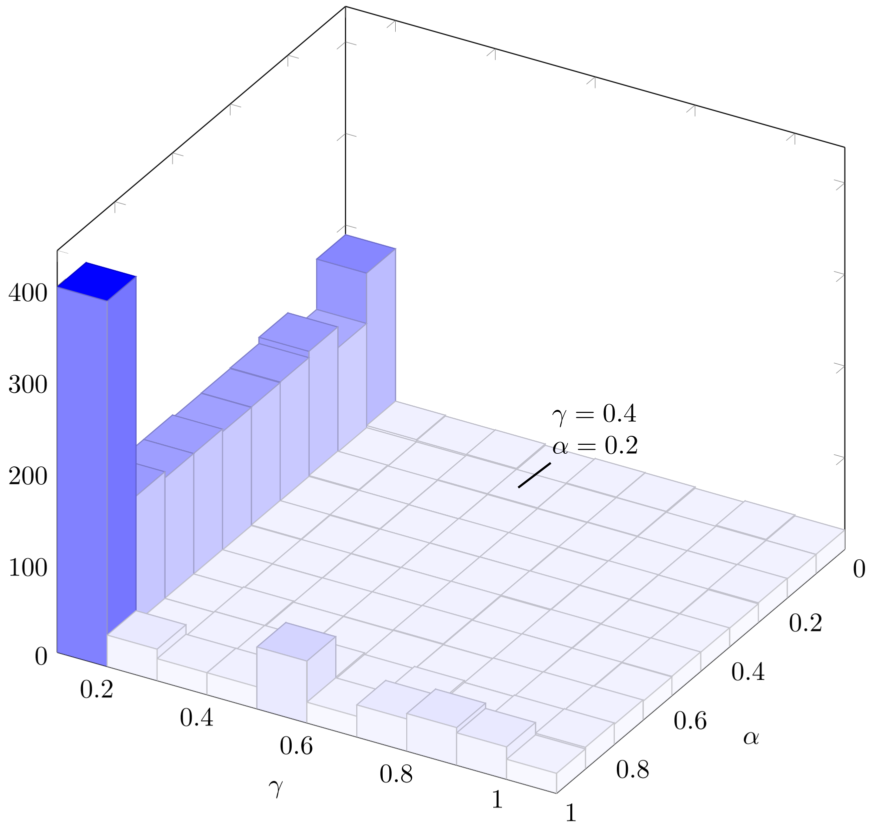

Figure 2: Average number of steps needed to catch the prey when the *Q-Learning* algorithm is run for different combinations of its parameters $\alpha \in [0.1, 1]$ and $\gamma \in [0, 0.9]$. We set $\epsilon = 0.1$ and $Q(s, a) = 15, \forall s, a$ for all instances of the algorithm. The best performance obtained, close to 20, corresponds to $\gamma = 0.4$, $\alpha = 0.2$.

4 Discussion

Figure 2 shows our first results for the *Q-Learning* algorithm. We can see that the algorithm converges to an almost-optimal policy for all combinations of its parameters α and γ, except for $\gamma = 0$. This almost-optimal policy has a performance of around 20, while the optimal policy is around 10 (see figure 1). We hence can see that *Q-Learning* manages to discover a relatively good policy, although it is still not completely optimal. For $\gamma = 0$ the predator does not look into future states at all, and only guides itself by the inmediate reward it receives. As figure 2 shows, this approach leads to poor results. Furthermore, for $\alpha = 1$ we can see how there is more noise in the performance obtained, which is also slightly worse than the one obtained for $\alpha < 1$. This suggests that such a high learning rate might negatively affect performance, and thus should be avoided as well.

Figure 3 shows our main results for the *Q-Learning* algorithm. As a general trend, we can observe here that higher values of either α or γ result in a faster converge of the algorithm. However, the final performace of the greedy policy seems to be, in any case, independent on the choice of parameters. This also matches our results of figure 2. Besides, the parameter γ has a smaller influence the larger the learning rate is. Finally, and more importantly, higher values of α or γ reduce the number of episodes *Q-Learning* needs to find greedy policies which result in games that always end. This problem arises from the optimistic initialization of the $Q(s, a)$ table. Since $15 > Q^*(s, a) \forall s, a$, the moment the action that catches the prey is explored, it receives a value of 10, making it a non-greedy action, and thus it never gets selected again, resulting in non-ending games. This issue is solved when the rest of the actions are eventually explored.

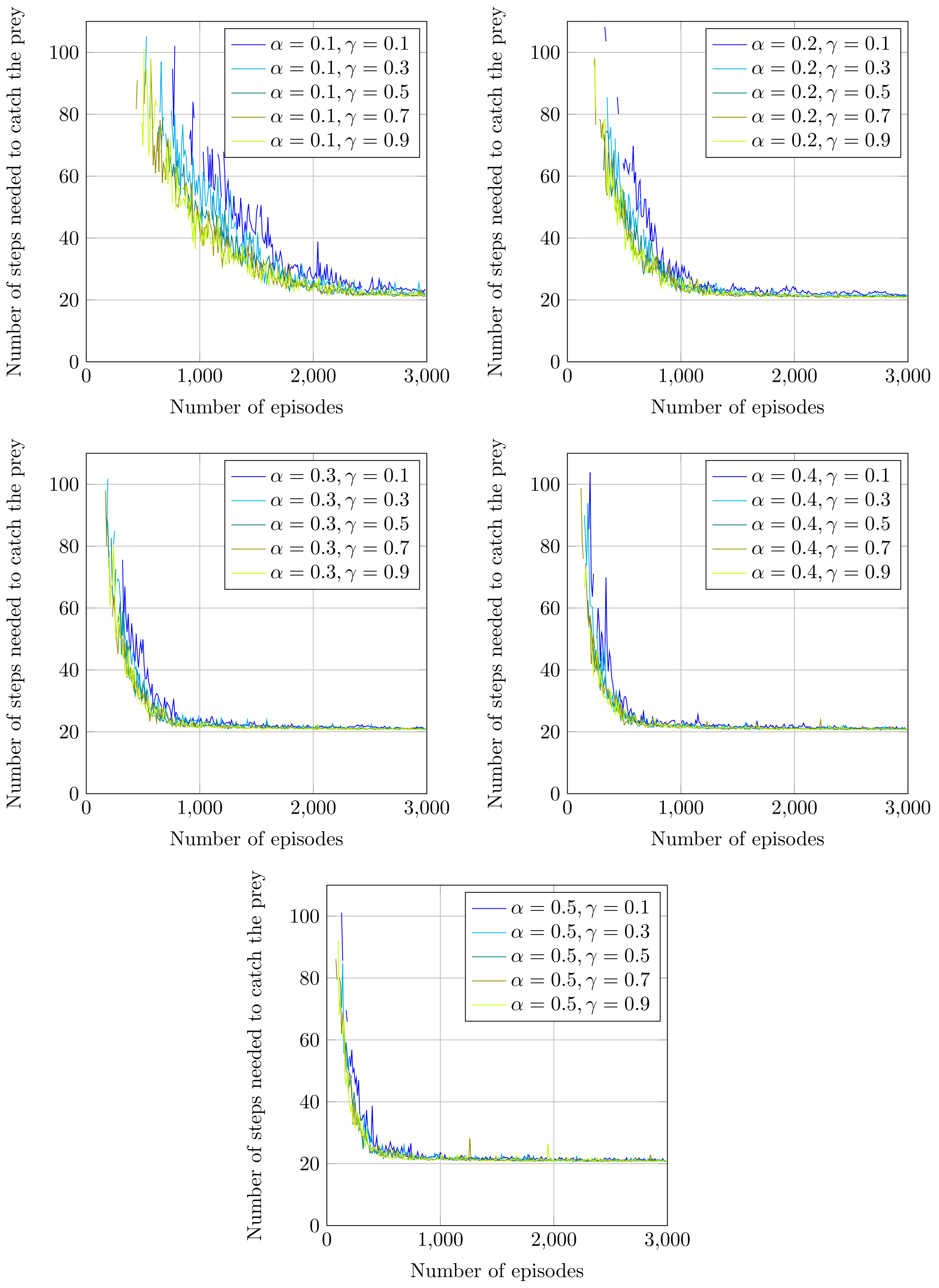

Figure 3: Evolution on the average number of steps needed to catch the prey for the *Q-Learning* algorithm for different combinations of its parameters α and γ. We set $\epsilon = 0.1$ and $Q(s, a) = 15, \forall s, a$ for all instances of the algorithm. The graphs are plotted against the number of episodes the algorithm was trained with.

Figure 4 shows the results of our comparison of different values of ϵ and initialization values for the Q table of the *Q-Learning* algorithm. First of all, we can see how no difference can be appreciated from different values

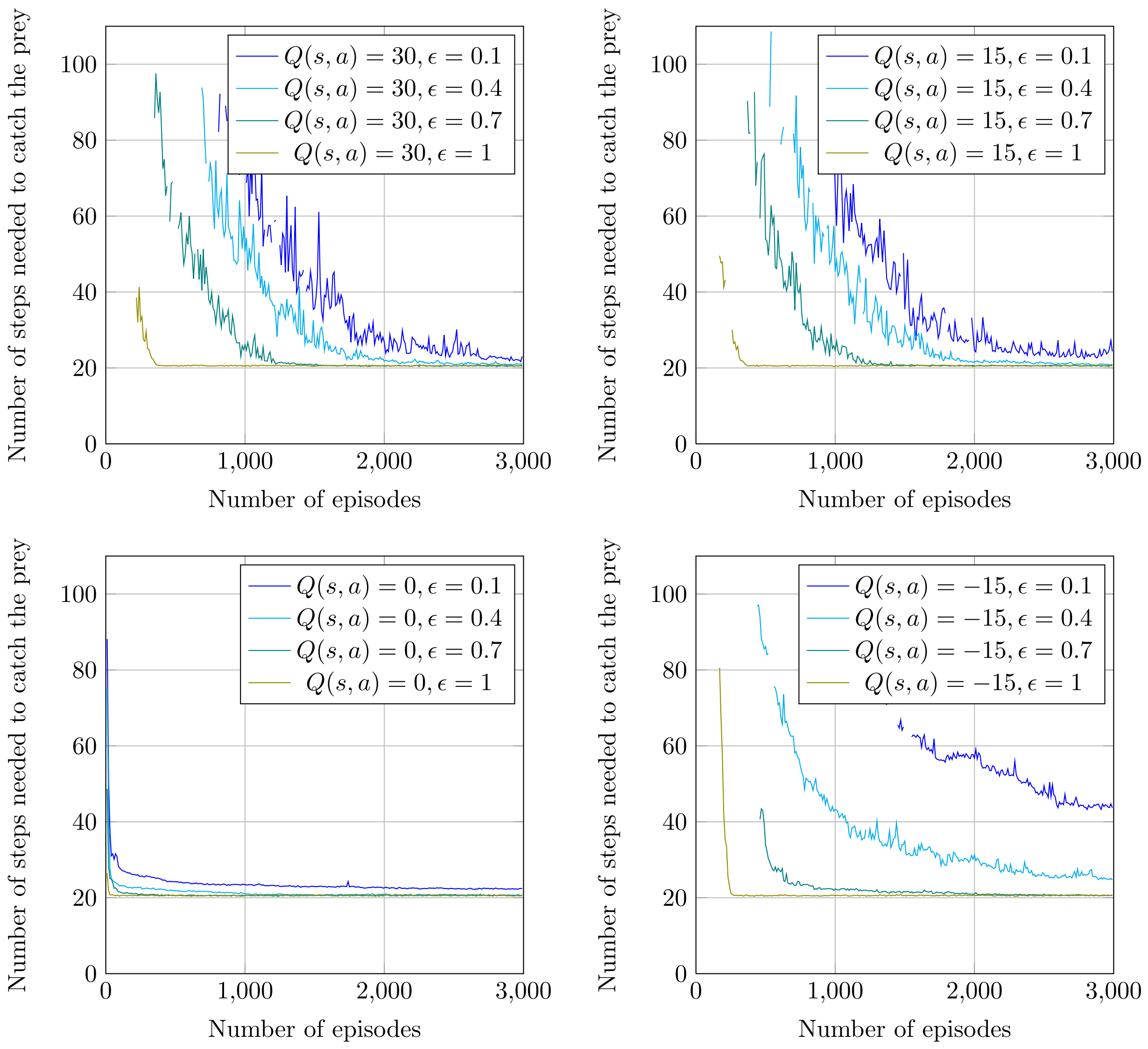

Figure 4: Evolution on the average number of steps needed to catch the prey for the *Q-Learning* algorithm for different combinations of ϵ and the initialization of the Q table. We set $\alpha = \gamma = 0.1$ for all tests.

of optimistic initializations. (TODO: Why?). Besides, we can see how an initial value of zero for the Q table outperforms both optimistic and more pessimistic approaches.

Figure 5 shows the results of our comparison between ϵ-greedy policies and softmax action selection for generating episodes within the *Q-Learning* algorithm. $\epsilon = 0$ corresponds to the case when the episodes are generated by always choosing a greedy action. As it can be seen in the plot, this approach leads to the worst results, due to the fact that exploration becomes almost non-existent. Larger values of ϵ seem to work better, since the convergence of the algorithm becomes faster. The extreme case corresponds to $\epsilon = 1$, which equals to a complete random walk. This choice of ϵ makes the returned greedy policy to perform the best, and it shows a very fast convergence, due to the fact that exploration is maximized when generating episodes. We can appreciate a similar behaviour for the softmax method. When $\tau \to 0$ this method prefers to select only greedy actions, and when $\tau \to +\infty$ it converges to a random walk. In conclusion, we can see from our results that a completely random walk performs as good as any softmax action selection method for this particular problem. Thus, it is advisable to use it, since it is faster in execution time than using softmax, where exponential functions need to be calculated.

Figure 6 ...

Figure 7 ...

5 Conclusion

Something here...

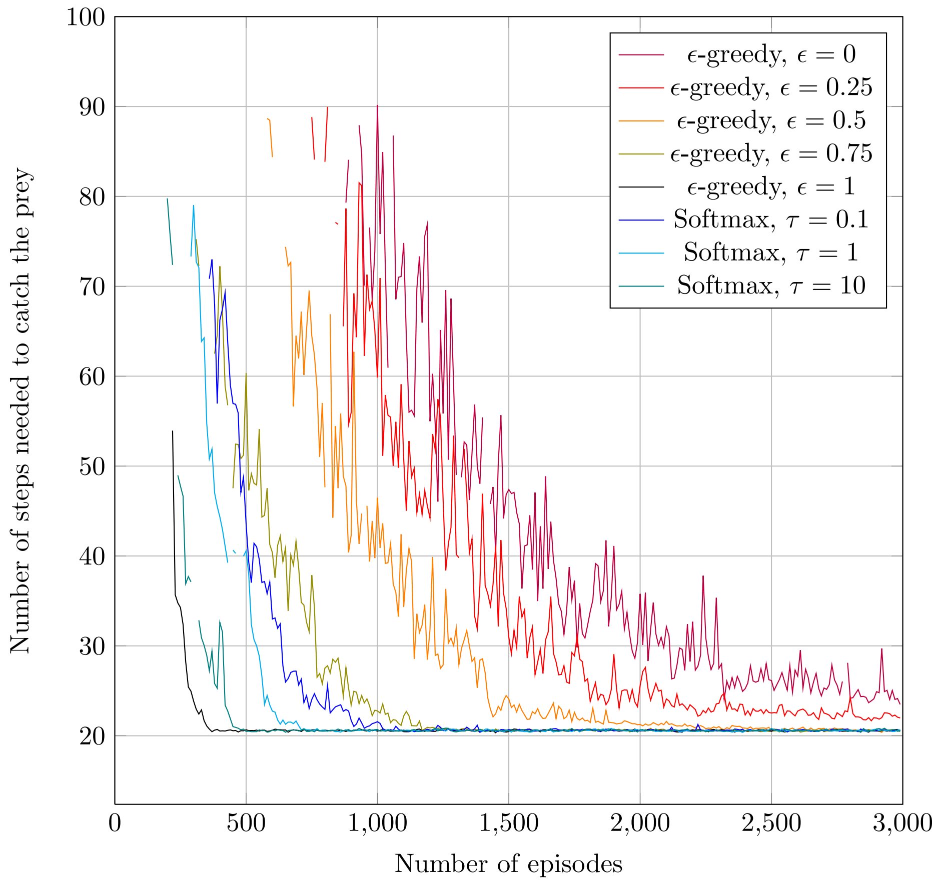

Figure 5: ϵ-greedy versus softmax action selection for the *Q-Learning* algorithm for various values of ϵ and τ. We fixed $\alpha = \gamma = 0.1$ and $Q(s, a) = 15, \forall s, a$.

References

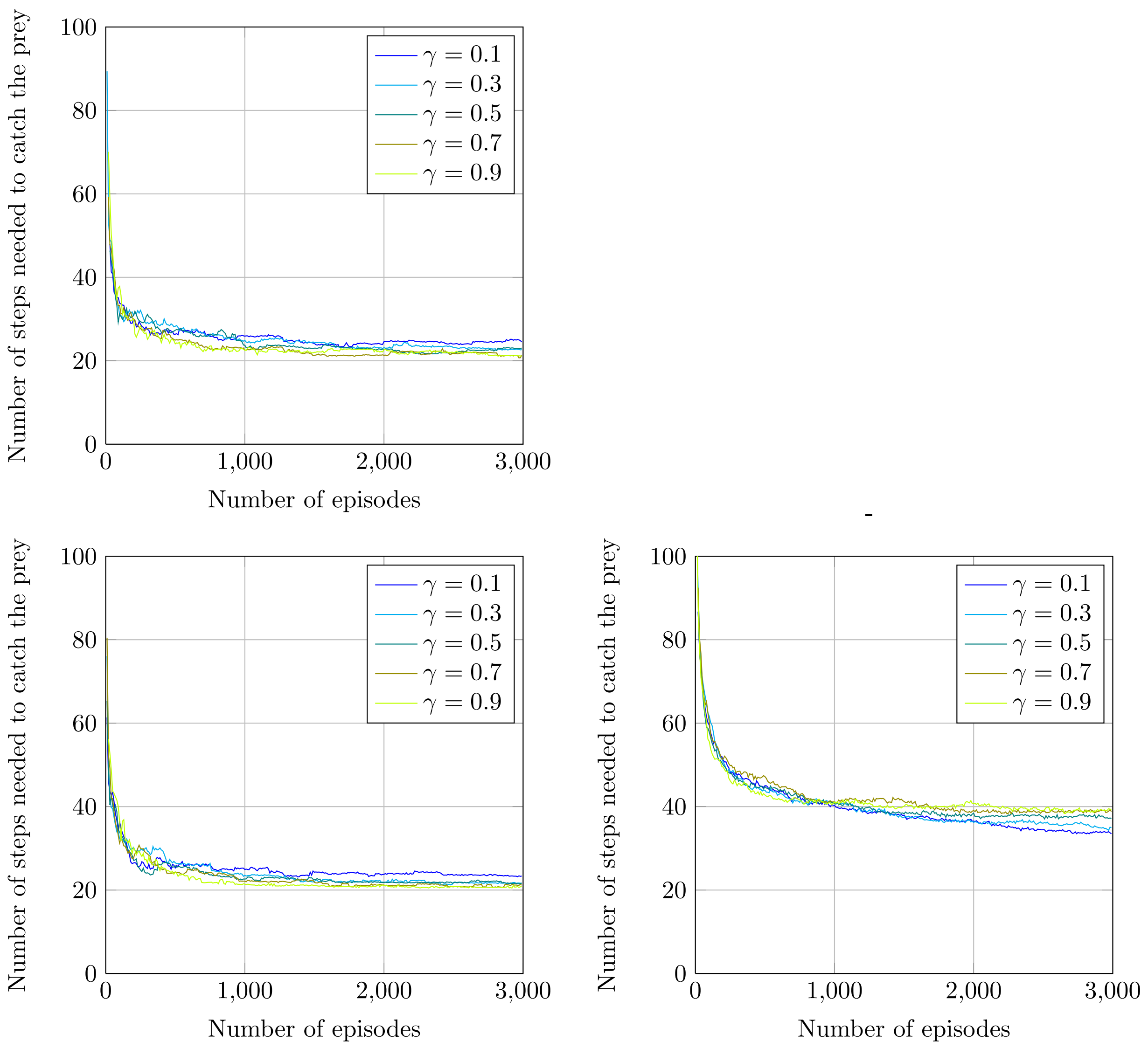

Figure 6: Performance of the implemented Monte-Carlo control methods for different values of the parameter γ. On-policy is shown on the left, while off-policy is shown on the right plots. $Q(s, a) = 15, \forall s, a$ is shown above and $Q(s, a) = 0, \forall s, a$ is shown in the second row of plots. We fixed $\epsilon = 1$ for all cases.

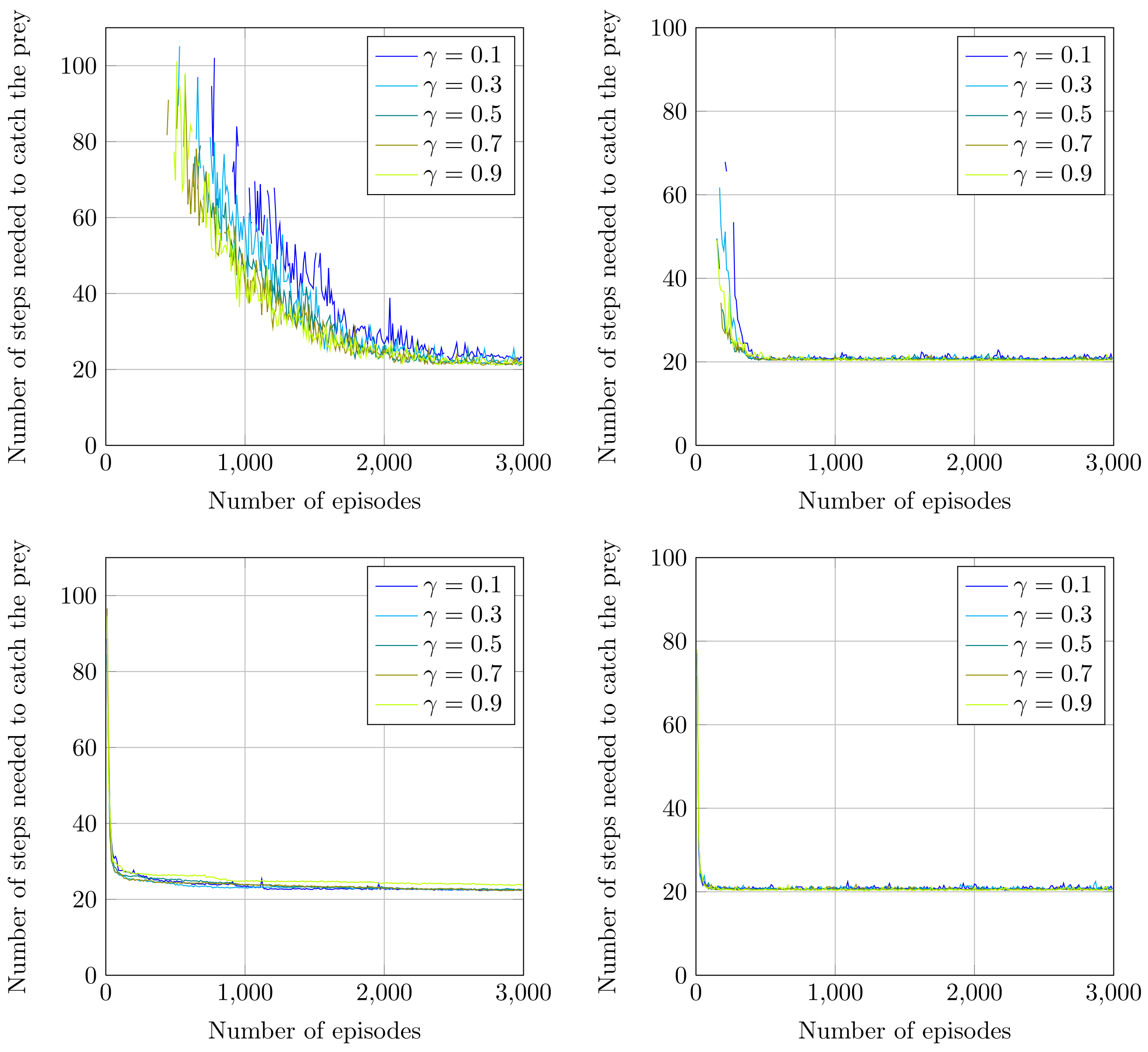

Figure 7: Comparison of the performance of *Q-Learning* versus *Sarsa*. *Q-Learning* is shown on the left, while *Sarsa* is shown on the right plots. $Q(s, a) = 15, \forall s, a$ is shown above and $Q(s, a) = 0, \forall s, a$ is shown in the second row of plots. We fixed $\epsilon = 1$ for all cases.