Multiagent Systems 2010
(Multiagent) Reinforcement Learning

Peter Lubell-Doughtie, 6095445 and Davide Modolo, 6209033

University of Amsterdam

1 Introduction

In this assignment we address the problem of how autonomous agents that sense and act in their environment can learn to choose optimal actions to achieve their goals. We will use the Q-learning algorithm, which learns optimal control strategies from delayed rewards, even when agents have no prior knowledge of the effects of their actions on the environment. We will experiment with two different strategies for choosing actions, an ϵ-greedy strategy that randomly chooses actions and an ϵ-greedy strategy that chooses actions weighted by their estimated value.

The environment consists of a grid world in which two types of agents, two predators and one prey, move around. In this world the goal of the predators is to capture the prey as quickly as possible. In the first exercise the task is for each predator to learn an individual policy for capturing the prey. In this exercise and prey is captured when it is surrounded by two predators, therefore although the predators develop individual policies they must coordinate their actions.

In the second exercise the task is for both predators to learn a joint policy for capturing the prey. In this exercise a prey is captured when it is surrounded by the predators for two time steps. This is a harder problem and we expect it to take a greater number of moves for the predators to capture the prey.

In the next section, we describe the implementation details for both of these learning policies, and we explore a way to represent the state space such that we can use Q-learning on larger state spaces more efficiently. In Section 3, we describe our experiments, and in Section 4 we present and discuss the results. Finally, in Section 5, we conclude.

2 Implementation Details

We implement an algorithm in which the predators attempt to learn an optimal policy for capturing prey by using a *temporal difference* learning method called Q-learning. The pseudo-code of our implementation for the Q-learning algorithm applied to this problem is presented in Algorithm 1. Below we will present preliminaries and provide a detailed description of the different components of the algorithm.

The first step in both the individual and joint policies is for each predator to determine a unique fixed label for itself and the other predator. This is done

by assigning an ID of 1 to the predator that is to the East or, if both predators are on the same longitude, to the predator that is to the North, and an ID of 2 to the other predator. Below we will refer to an arbitrary predator as $pred_i$ and to the predators with ID 1 and 2 as $pred_1$ and $pred_2$ respectively. We note that any scheme assigning deterministic IDs can be used.

2.1 Moves

Given an arbitrary location on the grid, a predator can either move in one of the four cardinal directions or remain in the same location. We denote these moves as $\{m_1, m_2, m_3, m_4, m_5\}$ where each m_i respectively represents the movement $\{North, South, East, West, None\}$.

2.2 States

At every time step in the simulated world a predator is presented with sensory information about all the objects in its environment, this includes the locations of the prey and the other predator relative to the current predator. In order to determine the global state both predators translate their coordinates and the coordinates of the other predator to a new coordinate system. In the new coordinate system all coordinates are with respect to the prey being positioned at $(0, 0)$. This computation is performed as follows:

$$x_{pred_1} = -x_{prey} \tag{1}$$

$$y_{pred_1} = -y_{prey} \tag{2}$$

$$x_{pred_2} = x_{pred_2} - x_{prey} \tag{3}$$

$$y_{pred_2} = y_{pred_2} - y_{prey} \tag{4}$$

In this manner both predators observe the same state at every time step. This is necessary for the predators to learn and represent a global policy. A state is then defined by the tuple $\{x_{pred_1}, y_{pred_1}, x_{pred_2}, x_{pred_2}\}$.

Size Issues and Solutions Using the complete grid makes the state space very large. Given a grid of 15x15 = 225 cells, $pred_1$ can occupy one of the 224 locations not occupied by the prey, and $pred_2$ can occupy one of the 223 locations not occupied by the prey and the other predator. Therefore the total state space based on the full grid consists of 224*223 = 49,952 states.

During the learning phase of the Q-learning algorithm all state-action pairs are visited infinitely often and a discount factor determining the value of delayed rewards, γ, decreases slowly with time. Both of these conditions are required to ensure the algorithm converges to the optimal control strategy, Q^* [5], but the first becomes increasingly challenging to achieve as the size of the state space grows. If one uses the complete grid the algorithm will converge very slowly and, because a Q-value is stored for each state, the total number of stored Q-values will be very large.

Algorithm 1 Pseudo code of our implementation of the Q-Learning algorithm

$Q_1(s,a) \leftarrow arbitrary\,value$
$Q_2(s,a) \leftarrow arbitrary\,value$
for each episode **do**
  $s \leftarrow undefined$
  $a \leftarrow undefined$
  repeat
    observe the new state s'
    receive immediate rewards r_1 and r_2
    if s' is a non-terminal state **then**
      $V_1(s') \leftarrow max_a(Q_1(s',a))$
      $V_2(s') \leftarrow max_a(Q_1(s',a))$
    else
      $V_1(s') \leftarrow 0$
      $V_2(s') \leftarrow 0$
    end if
    if predator 1 **then**
      $Q \leftarrow Q_1, \quad V \leftarrow V_1, \quad r \leftarrow r_1$
    else
      $Q \leftarrow Q_2, \quad V \leftarrow V_2, \quad r \leftarrow r_2$
    end if
    if s is defined & a is defined **then**
      if capture_case equals 3 **then**
        $Q(s,a) \leftarrow Q(s,a) * (1-\alpha) + \alpha * (r + \gamma * V(s'))$
      else
        $Q_1(s,a) \leftarrow Q_1(s,a) * (1-\alpha) + \alpha * (r_1 + \gamma * V_1(s'))$
        $Q_2(s,a) \leftarrow Q_2(s,a) * (1-\alpha) + \alpha * (r_2 + \gamma * V_2(s'))$
      end if
    end if
    compute the exploration function e
    if e equals true **then**
      $a' \leftarrow random\,action$
    else
      choose optimal action a' from state s' using policy derived from Q
    end if
    $s \leftarrow s'$
    $a \leftarrow a'$
    make the move m corresponding to action a'
  until s is terminal state
end for

To address this problem we note that many locations on an NxN grid are symmetric and actions being taken in symmetric locations can be represented by a single action that is appropriately reflected through the lines of symmetry. Furthermore, any symmetrically related locations can share the same Q-value and thereby significantly decrease the total state space. A simple example is given by the states $\{-3, -1, -5, -2\}$ and $\{3, 1, 5, 2\}$ in which the two predators are in different locations, but are positioned in a rotationally equivalent manner with respect to the prey. These two states can therefore share the same Q-value and the direction of movement can be rotated to produce the appropriate action.

In order to reduce the size of the state space we use this property of the grid and reposition both predators by reflecting their coordinates with respect to the axes x, y and $x = y$. This symmetry is visually depicted in Fig. 1. The procedure accomplishing this reflection is shown in Algorithm 2.

While shifting the predators positions in order to determine the state, we also need to keep track of how their available moves $\{m_1, m_2, m_3, m_4, m_5\}$ are reoriented by these reflective operations. In order to do this we create a 4-tuple $d = \{d_1, d_2, d_3, d_4\}$, initially associated with the directions $\{North, South, East, West\}$. As we apply reflections, we reorganize the elements in d so that making the move d_i, in its position as indicated by the state, is symmetrical to making the move m_i given the predator's actual position on the board.

Using this state space reduction technique we are only required to represent the location of $pred_1$ relative to the grid cells in the first quadrant which are on or south of the line $x = y$. In a 15x15 grid a predator can only occupy the 7 cells of the line y, the 7 cells of the line $y = x$ (the 8th cell of both these lines is occupied by the prey) and the 21 cells between these two lines. Using this strategy the total state space is significantly smaller that the original space of 49,952 states.

2.3 Actions *a*

So far we have discussed which moves are available for the predators at any location on the grid. Now we will introduce the notion of an action and describe what actions are available to the predator at every state of the world. Without the state space reduction described in the previous subsection each move would correspond to a unique action, but thanks to the state space reduction, symmetrical board positions map to the same state and therefore we can project symmetrical moves to the same action.

For an independent learning policy, each predator has five actions available to it. These actions form a 5-tuple $\{a_1, a_2, a_3, a_4, a_5\}$ respectively representing the movements $\{North, South, East, West, None\}$. At any state each of the predators will take an action independently from the action of the other predator.

For the joint learning policy each predator still has five actions available to it. However, since we are learning a joint policy, we consider 25 possible joint actions which form the 25-tuple $\{a_{11}, ..., a_{15}, a_{21}, ..., a_{51}\}$ where a_{ij} represents the joint action of $pred_1$ taking individual action i and $pred_2$ taking individual

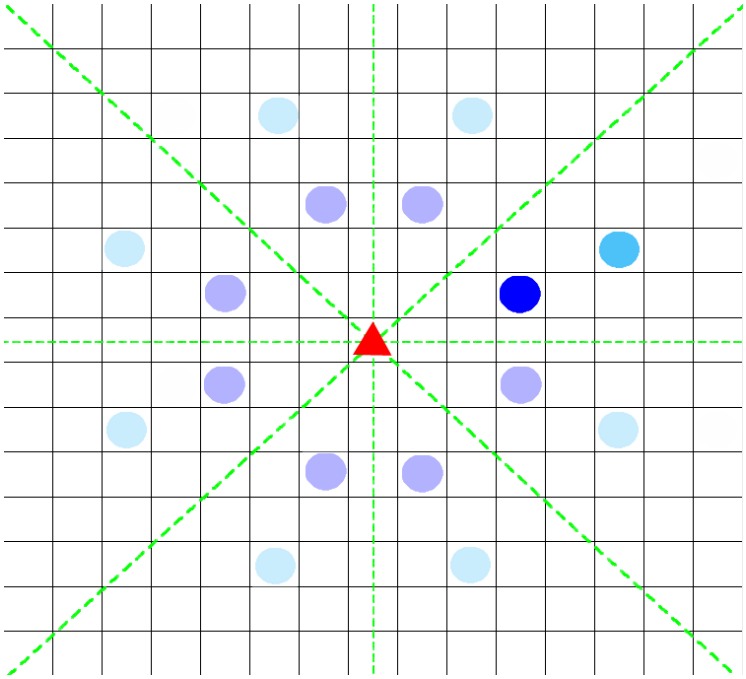


Fig. 1: Symmetry in the state space. The strategies of the ghosted predators' locations are symmetrically homomorphic to the strategy of the solid predators' locations below the $x = y$ axis in the first quadrant.

Algorithm 2 State Space Reduction

translate (x_{pred1}, y_{pred1}) and (x_{pred2}, y_{pred2}) following Eq. 1, 2, 3, 4
$(d_1, d_2, d_3, d_4, d_5) \leftarrow (N, S, E, W, None)$
if $(x_{pred1} < 0)$ or $(x_{pred1} == 0$ and $x_{pred2} < 0)$ **then**
  $x_{pred1} \leftarrow -x_{pred1}$
  $x_{pred2} \leftarrow -x_{pred2}$
  swap d_3 and d_4
end if
if $(y_{pred1} < 0)$ or $(y_{pred1} == 0$ and $y_{pred2} < 0)$ **then**
  $y_{pred1} \leftarrow -y_{pred1}$
  $y_{pred2} \leftarrow -y_{pred2}$
  swap d_1 and d_2
end if
if $(x_{pred1} < y_{pred1})$ or $(x_{pred1} == y_{pred1}$ and $x_{pred2} < y_{pred2})$ **then**
  swap x_{pred1} and y_{pred1}
  swap x_{pred2} and y_{pred2}
  swap d_1 and d_3
  swap d_2 and d_4
end if

action j. At any state each of the predators will take the same joint action possibly representing different individual actions.

2.4 Q-learning

For an independent learning policy, each predator maintains a function $Q(s, a)$, which provides a mapping from states to 5-tuples $\{v_1, v_2, v_3, v_4, v_5\}$, where v_i is the $Q(s, a)$ value for taking action a_i in state s. The map at the ith iteration of the algorithm is defined by:

$$Q_i \leftarrow (1 - \alpha_i)Q_{i-1}(s, a) + \alpha_i(r + \gamma * V_{i-1}(s')) \tag{5}$$

where α is the learning rate – described below in subsection 2.6, γ is the discounted factor – described below in subsection 2.7, and $V(s)$ is the value of state s defined by:

$$V_i = \max_a Q_i(s, a) \tag{6}$$

which is equivalent to the value of the best action available to $pred_i$ in state s.

For the joint learning policy, each predator maintains two different maps, $Q_1(s, a)$ and $Q_2(s, a)$, which correspond to the Q-value maps for $pred_1$ and $pred_2$, respectively. $Q_1(s, a)$ and $Q_2(s, a)$ provide a mapping from states to 25-tuples $\{v_{k11}, ..., v_{k15}, v_{k21}, ..., v_{k25}, ..., ..., v_{k51}, ..., v_{k55}\}$, where v_{kij} is the $Q_k(s, a)$ value for $pred_k$ taking the joint action a_{ij} in state s. Given this representation $pred_k$ computes the value of a state s as:

$$V = \max_a Q(s, a^*) \tag{7}$$

where

$$a^* = \operatorname*{argmax}_a \sum_{k=1}^{2} Q_k(s, a) \tag{8}$$

which is equivalent to the best joint action available to both predators in state s.

To illustrate how the Q-learning algorithm works in our game, consider a single action taken by $pred_1$. Let's say the predator moves $North$ and it receives an immediate reward for this transition. It then applies the training rule of Equation 5 to refine its estimate of Q for the state-action transition it has just executed. According to the training rule, the new Q estimate for this transition is the α weighted sum of received reward and the highest Q associated with the resulting state, where this resulting state reward is discounted by γ. Each time the predator moves forward from an old state to a new one, Q-learning propagates Q estimates backward from the new state to the old one. At the same time, the immediate reward received by the predator for the transition is used to augment these propagated values of Q.

Consider applying this algorithm to the grid world where catching the prey gives the greatest reward and can therefore be seen as the goal state. Since this world contains an absorbing goal state, we will assume that training consists of

a series of *episodes*. During each episode, the predator begins at some randomly chosen state and is allowed to execute actions until it reaches the absorbing goal state. When it does, the episode ends and both prey and predators are transported to a new randomly chosen initial state for the next episode.

2.5 Reward Function r

After experiment with various different reward schemes we found that *Occam's Razor* held: simpler is better. We reward predators by combing a base reward, r_b with a reward shaping function to calculate the final reward for $pred_i$, r_i. At every time step the predators receive a base reward of -1 unless they capture the prey, in which case they receive a base reward of 0, as indicated in Table 1. It may seem intuitive to explicitly penalize the predators for colliding with each other or colliding with the prey but this is unnecessary and can potential harm performance. We can see this by noting that the goal of the predators is only to minimize the time taken to capture the prey. When the two predators are far from the prey, but close to each other, they can decrease capture time by colliding with each other and subsequently being randomly placed closer to the prey.

Table 1: Predator rewards at each time step.

Action	Reward r_b
Capture	0
¬Capture	-1

Reward Shaping Because the state, even when reduced, is quite large, using these base rewards alone would result in very slow convergence. Fortunately, the distance between the predators and prey provides additional structure in the state space that we can exploit to decrease convergence time. We use reward shaping based on the change in Manhattan distance between the predators and prey to shape the predators' policies.

In the independent policy each predator stores the previous distance between itself and the prey, for $pred_i$ let this be d_i. This is calculated using the Manhattan distance: $d_i = |x_{pred_i}| + |y_{pred_i}|$. In the joint policy each predator's reward is dependent upon the state of both predators. Therefore, the joint policy calculates the sum over all predators' Manhattan distances: $d_i = \sum_{k=1}^{2} |x_{pred_k}| + |y_{pred_k}|$. Based upon these distance calculations the final reward is then computed as:

$$r_i = d_i - d'_i - r_b \tag{9}$$

where d'_i is the current distance between the predator and the prey and d_i is the stored previous distance. When a predator has captured a prey $d'_i = 0$, $r_b = 0$ (as seen in Table 1), and $d_i = 1$, therefore Eq. 9 simplifies to $r_i = 1$.

2.6 Learning Rate α

The learning rate is a decreasing function of time that is defined in the range [0,1]. We use a linear alpha function defined by:

$$\alpha = \begin{cases} \alpha_i * \frac{(\tau_l - \tau_n)}{\tau_l}, & \text{if } \tau_n \leq \tau_l \\ 0, & \text{if } \tau_n > \tau_l \end{cases} \tag{10}$$

where α_i is the initial learning rate, τ_l is the total number of episodes during which the predator learns, and τ_n is the current episode number. From Eq. 5 we see that when $\alpha = 1$ the predator will only consider the most recent information obtained about the value of the state and when $\alpha = 0$ the predator will not learn anything new.

Through Eq. 10 we decrease the value of α as the number of episodes increases, so that the effect of new state information becomes smaller as training progresses. By reducing the value of α at an appropriate rate during training, we converge to the optimal Q-function.

2.7 Discount Factor γ

The discount factor is a constant that determines the relative value of delayed versus immediate rewards. In particular, rewards received i time steps into the future are discounted exponentially by a factor of γ^i. Where γ is defined in the range [0,1] and from Eq. 10 we see that if we set $\gamma = 0$, only the immediate rewards are considered. As we set γ closer to 1, future rewards are given greater emphasis relative to the immediate reward.

In our implementation, the reward function only returns non-negative rewards if the predator moves closer to the prey at every time-step, until it captures it. As a result, a predator that attempts to maximize the reward function is guaranteed to reach a terminal state in every episode. Therefore, we can set γ to 1 and still have a convergence guarantee.

2.8 Exploration Function e

As already introduced in Subsection 2.2, one of the conditions that guarantees the Q-learning algorithm will converge, is that all state-action pairs must be visited infinitely often during the learning task. To ensure that this happen we introduce an exploration function e. The exploration function was tested as an ϵ-greedy function and using a probabilistic exponential weighted by reward. This ensures that the predator will not always choose the optimal action a^*, and that all the states will be visited infinitely often. The function e returns $true$ or $false$, respectively indicating that we should explore and choose a non-optimal action or exploit and choose the optimal action.

ϵ-greedy This exploration function is computed at each time step and decides either to take a pseudo-random action from the possible actions or to take the optimal action. In the case e returns $false$ the predator will choose an action a randomly, while in case it returns $true$ it will choose the optimal action a^*.

The ϵ-greedy strategy computes the value of e with a steadily decreasing probability choosing a random action, as follows:

$$e = \begin{cases} true, & \text{if } (\tau_n < \tau_e) \wedge (n \leq \epsilon_i * \frac{\tau_e - \tau_n}{\tau_e}) \\ false, & \text{otherwise} \end{cases} \tag{11}$$

where n is a randomly generated real number in the range [0,1], ϵ_i is the initial exploration rate also in the range [0,1], τ_e is the total number of episodes during which the predator explores, and τ_n is the current episode number, as above. As more episodes pass the chances of exploring decrease relative to the value of ϵ_i. After all the learning episodes have passed the predator shifts to exploitation and no longer chooses non-optimal actions.

Probabilistically ϵ-greedy As an alternative we assigned probabilities to each action based on their reward and chose an action according to this probability distribution. We use the probability distribution suggested by Mitchell [2]. In case of the individual policy, for $pred_l$, we define the probability of action a_i in state s by:

$$P_l(a_i|s) = \frac{k^{Q_(s,a_i)}}{\sum_j k^{Q_l(s,a_j)}} \tag{12}$$

where k is set to constant plus a function of the number of episodes passed. We set $k = 1 + \tau_n/\tau_e$.

For the joint policy the probability of a joint action equal for both predators based on both of their $Q(s, a_i)$ values. It is computed as:

$$P(a_i|s) = \frac{k^{Q_1(s,a_i)+Q_2(s,a_i)}}{\sum_j k^{Q_1(s,a_i)+Q_2(s,a_i)}} \tag{13}$$

If e from Eq. 11 is equal to $true$ we choose an action according to Eq. 12 in the individual policy or according to Eq. 13 in the joint policy. If e is equal to $false$ we choose the best action.

3 Experiments

We performed experiments with a fixed parameter set for the individual and joint policies given their respective learning problems. We tested our implementation on a 15x15 and 10x10 grid.

3.1 Parameters

We ran different experiments with various values for the parameters [α_i, γ, ϵ_i, τ_l, τ_e]. We achieved good results for both policies using the parameters in Table 2. As shown in Table 2, we set α_i to a low value so that the predator will learn slowly and will not be considerably affected by unexpected behaviors. As described in Subsection 2.7, we set γ equal to 1, and can guarantee that the predator will reach a terminal state in every episode because we weight future rewards to be more important than immediate rewards. We set ϵ_i to 0.75 so that the predator has sufficient time to explore all states.

Table 2: Best tested parameter set

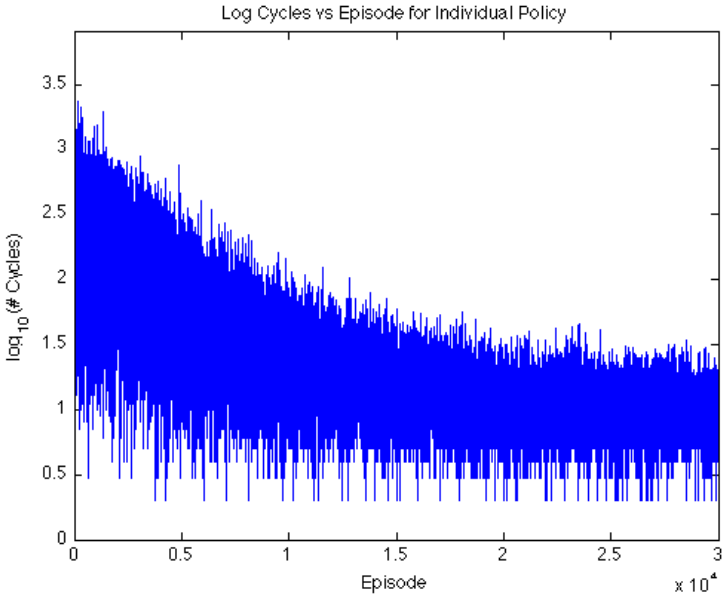
Parameter	Individual Policy	Joint Policy
α_i	0.15	0.15
γ	1	1
ϵ_i	0.75	0.75
τ_l	30,000	20,000
τ_e	15,000	5,000

As shown in Table 2, in the independent policy predators explore and learn for a greater number of episodes compared to in the joint policy. In the joint policy, we consider joint actions and therefore the number of explored state-action pairs will be five times greater than for the independent policy. However, because the joint policy is updating both predators' Q values on each cycle it should converge faster. We used different exploration and learning parameters in the probabilistically weighted action selection method, we defer discussion of those parameters until after presenting the results.

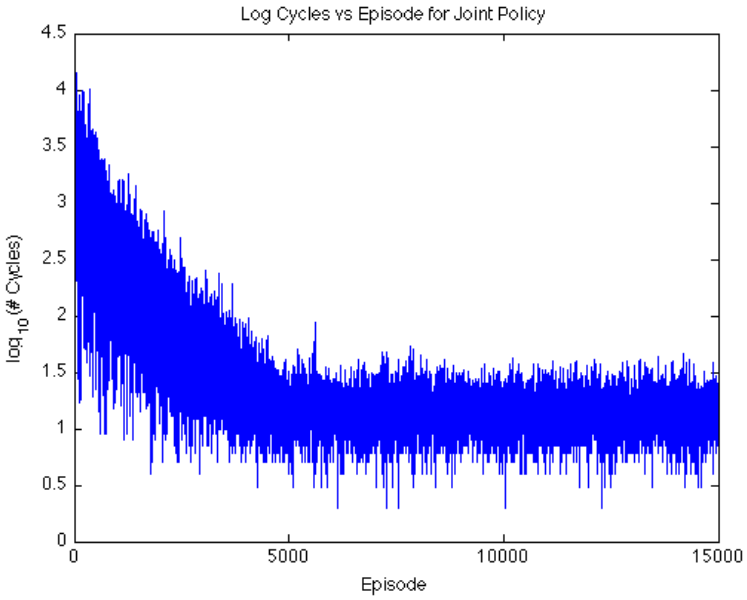
4 Discussion and Results

In Figure 2 we plot the learning curves (base 10 logarithm of the average capture time versus number of episodes passed) for the individual and joint policies using a 15x15 grid. These plots clearly show quicker convergence for the joint policy, as well as a larger maximum number of cycles before capture. Initially, the algorithm for the joint policy performs much worse than the individual policy because (i) the state space is larger and (ii) the problem is harder. In the independent policy the predators only need to surround the prey for 1 time step, while in the joint policy they must be next to the prey for 2 time steps.

As shown in Fig. 2a and 2b, for both policies we see a significant improvement in performance over the first τ_e episodes. This is because the predator is both learning the optimal policy, and will be choosing better actions, and because the exploration rate is decreasing, and the predator will therefore be more likely to choose the optimal action. The predators still continue to learn until τ_l episodes



(a) Independent Policy



(b) Joint Policy

Fig. 2: Learning curve obtained using the parameters described in Table 2 on a grid of 15x15.

have passed, and we see a slight improvement during this time, at least in the first 10,000 episodes in 2a and in the first 4,000 episodes in 2b. For the independent policy, we see that learning converges after a total of about 25,000 episodes. For the joint policy learning converges after about 9,000 episodes.

Figure 3 shows the learning curve for the individual and joint policy on a grid of size 10x10. As expected the number of cycles is lower over all the episodes, the learning curve is more or less shifted towards $y = 0$. Learning the policies on the smaller grid also results in faster convergence but only marginally so.

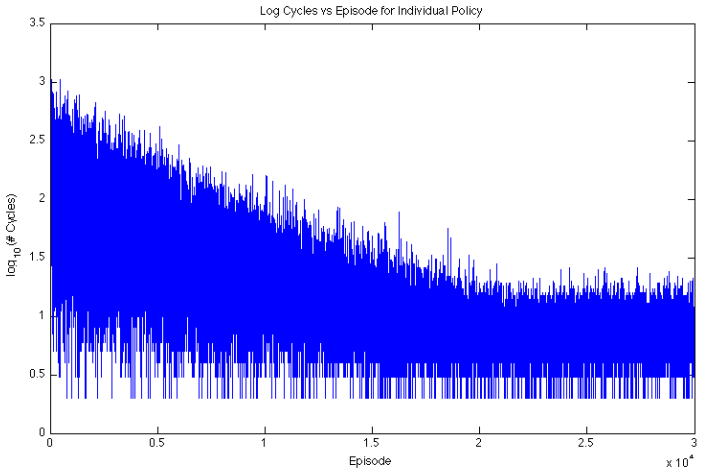
Table 3: Mean and Standard Deviation capture times for the final learned policy.

Grid Size	e	Policy	Mean Cycles	Standard Deviation
15x15	Prob	Individual	10.7742	3.9065
15x15	Prob	Joint	15.8492	7.5963
15x15	random	Individual	11.0281	4.3438
15x15	random	Joint	14.2913	5.5354
10x10	random	Individual	7.8267	3.2410
10x10	random	Joint	11.1455	5.1007

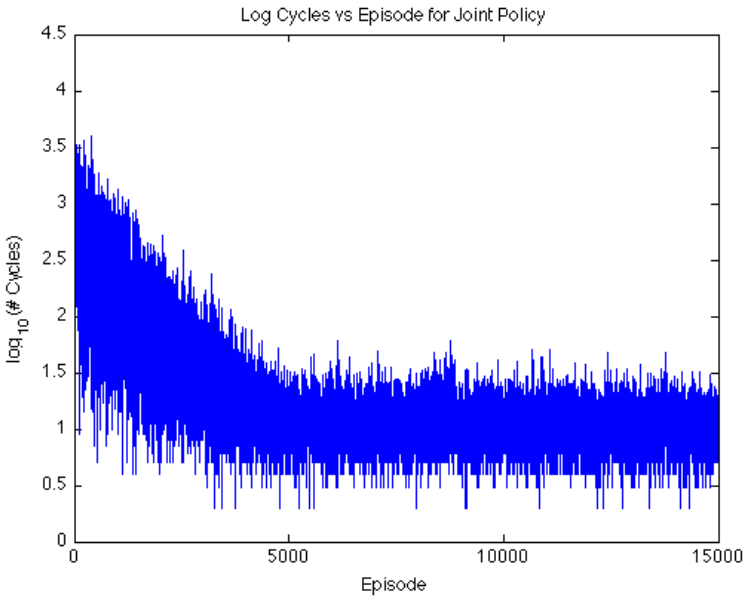
The capture times for the final learned policy, averaged over the last 10,000 episodes after exploration has finished, is shown for both grid sizes in Table 3. For the individual policy the means are 7.83 and 11.03 for the 10x10 and 15x15 grid sizes respectively. To put this into perspective, consider that if a predator and prey are placed as far as possible from each other on the grid, e.g. at [x,y] locations [0,0] and [7,7], their Manhattan distance equals 13. This distance equals one plus the total number of predator moves needed to reach the prey, assuming the prey doesn't move. In the best case this distance is 1. Based on this we estimate that on average the predator will need $(13 - 1 + 1 - 1)/2 = 6$ moves to be next to the prey, ignoring changes based on movement of the prey.

Based on this analysis an mean of 11 moves until capture is good and the predators have likely learned a near optimal policy. For the 10x10 grid the same analysis leads to an estimated 4 moves needed to be next to prey and the rounded 8 move strategy found by the our implementation again indicates it is likely that a near optimal policy has been learned. The standard deviation in capture time was 4.34 and 3.24 for the 15x15 and 10x10 grid respectively. We can interpret this as resulting from the random starting positions of the predators and prey, as well as the random movements of the prey.

For the 15x15 and 10x10 grid sizes the joint policy mean capture times are 14.29 and 11.15 respectively and the standard deviations are 5.54 and 5.10 respectively. In the joint policy an extra cycle will be needed to stay next to the prey for an additional time step. Therefore we'd expect these numbers to be higher in the optimal policies. We can conclude that a near optimal policy was learned for the joint action case.

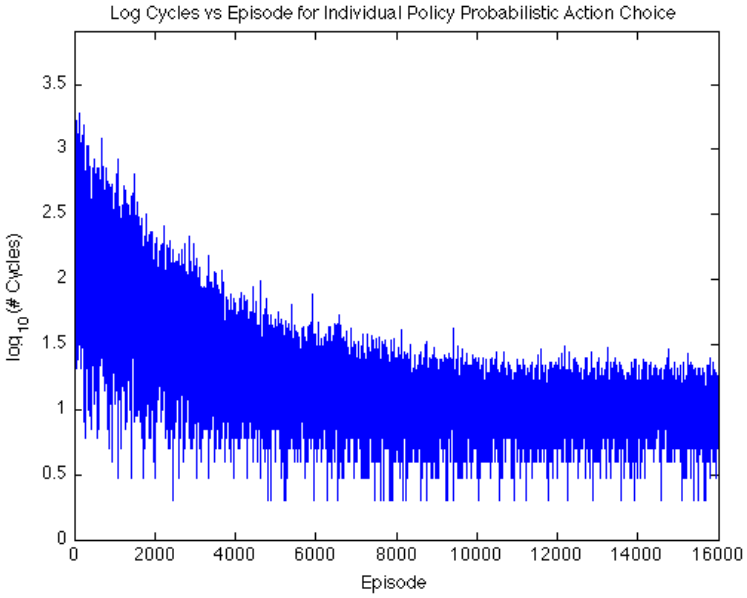


(a) Independent Policy

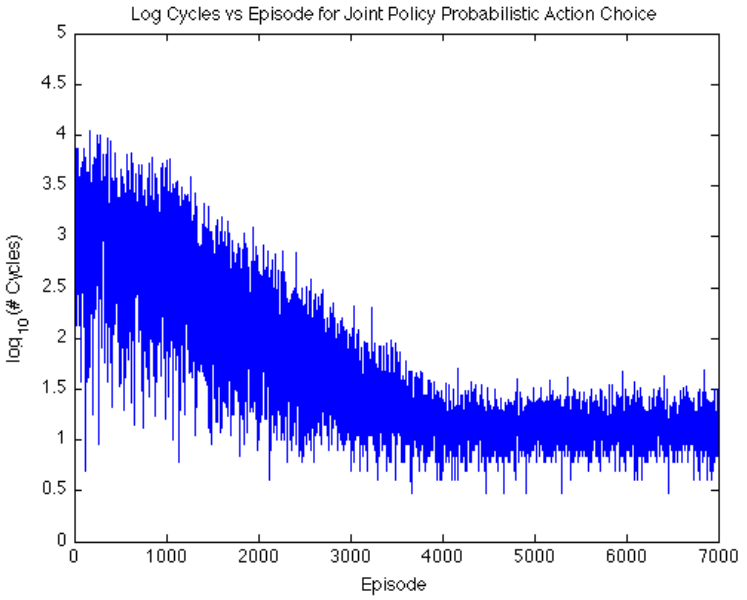


(b) Joint Policy

Fig. 3: Learning curve obtained using the parameters described in Table 2 on a grid 10x10.



(a) Independent Policy



(b) Joint Policy

Fig. 4: Learning curve obtained for probabilistic ϵ using the parameters described in Table 2 on a grid 15x15.

4.1 Probabilistic ϵ

The learning curves for the individual and joint policies using probabilistic ϵ learning are presented in Fig. 4. Note that in the individual probabilistic ϵ experiments we set the number of episodes before learning to $\tau_e = 10,000$ and $\tau_l = 15,000$. The statistics are computed for 1000 episodes from 15,000-16,000. In the joint policy $\tau_e = 4,000$ and $\tau_l = 6,000$.

We see in Fig. 4 that using probabilistically chosen actions results in significantly faster convergence than is obtained with random actions, depicted in Fig. 2. In other experiments with the individual policy, after only 1,000 cycles we were able to find policies as good as the policy found after 20,000 cycles with randomly chosen actions. This is not represented in Fig. 4b because of the large τ_e and τ_l values used. Setting τ_e to 1,000 produces quick convergence but the converged to value was greater then the value produced with $\tau_e = 10,000$, although still lower than or comparable to the converged mean cycles with randomly chosen actions. In an online learning setting, or when episode runs have a cost, quick convergence may be desirable. We also expect that the manner in which k changes over time could be adjusted to produce improved performance with respect to time to convergence and value converged to.

The probabilistic individual policy outperformed the random policy with a lower mean number of cycles until capture of 10.77 and a lower standard deviation of 3.91. As we can see by comparing Fig. 2b to Fig. 4b, the probabilistic joint policy converged faster. However, the probabilistic joint policy performed noticeably worse, with a mean number of cycles until capture of 15.85 and standard deviation of 7.60. The poor performance of the probabilistic action choice is most likely related to a suboptimal method for selecting the probability distribution over actions and a suboptimal method for changing this distribution over time. We would expect that with improved probability assignment, and longer run time, the mean cycle time could be competitive with or improve over the random action selection method.

5 Conclusion

In this second assignment we dealt with the problem of how autonomous agents that sense and act in their environment can learn to choose optimal actions to achieve their goals. We focused on temporal difference learning, and in particular on the Q-learning algorithm. Q-learning can acquire optimal control strategies from delayed rewards, even when agents have no prior knowledge of the effects of their actions on the environment. The Q-learning algorithm is proven to converge under certain conditions, as explained in the previous sections. However, this convergence can be very slow in the case of a large number of state-action pairs. To solve this problem we did not use a hard-coded policy when far from the prey, but instead implemented a powerful strategy which uses a set of geometric operations to reduce the number of possible actions available to the predators at each state.

The performances obtained by the independent and the joint policy were satisfactory. Significant improvement in convergence time and some improvement in mean capture time was seen using probabilistically chosen exploration instead of random exploration. However, improvements could be made to achieve an even better capture time and reduce the time taken to convergence. We could, for example, use more complex reward shaping, instead of the simple shaping technique based on the Manhattan distance. This may improve the performance.

Another problem that we would like to solve in future work is how to manage an environment with a larger number of predators. The independent policy we implemented will work fine with more predators, since each predator stores a single $Q(s, a)$ map with a set of possible actions $a = (a_1, a_2, a_3, a_4, a_5)$. However, the joint policy will suffer heavily, since every predator has to chose a joint action in a space of actions that will increase exponentially with the number of predators. This problem could be solved using *Sparse Cooperative Q-Learning* [1], which uses Q-learning to learn the coordinated joint actions of a group of cooperative agents, using a sparse representation of the joint state-action space of the agents.

References

1. Jelle R. Kok and Nikos Vlassis, *Sparse cooperative Q-learning*, Proceedings of the International Conference on Machine Learning (Banff, Canada) (Russ Greiner and Dale Schuurmans, eds.), ACM, July 2004, pp. 481–488.
2. Tom M. Mitchell, *Machine learning*, McGraw-Hill, New York, 1997.
3. Richard S. Sutton and Andrew G. Barto, *Reinforcement learning - an introduction*, Adaptive Computation and Machine Learning, MIT Press, 1998.
4. Nikos Vlassis, *A concise introduction to multiagent systems and distributed artificial intelligence*, Morgan and Claypool Publishers, 2007.
5. Christopher J. C. H. Watkins and Peter Dayan, *Technical note q-learning*, Machine Learning **8** (1992), 279–292.