Autonomous Agents 1
Assignment 3

By Group 4: Gieske, Gornishka, Koster, Loor

December 12, 2014

Contents

1 Introduction

This report contains the analysis of different multi-agent reinforcement learning algorithms.
The algorithms were tested in an 11×11 grid-world with one prey-agent and $n \in \{1, 2, 3, 4\}$ predator-agents. The goal of the prey is to have the predators bump into each other, while the goal of the predators is to have one of them catch the prey. All agents learn, by receiving a reward of 10 if their team wins, and a reward of -10 if their team loses.
Three model-free learning algorithms were compared: Independent Q-learning, Independent Sarsa and Minimax-Q [2]. Contrary to earlier implementations, the problem at hand considers an intelligent prey, that learns alongside the predators. The effect of the different algorithms, with different parameter settings (learning rate α, discount factor γ, explore-rate ϵ), on different numbers of learning agents was analyzed and the results reported in this paper.

2 Theory

2.1 Independent Q-learning

In independent Q-learning, all the agents independently use Q-learning to maximize their expected reward.

Q-learning[1] itself is a temporal difference method that uses the update rule in equation 1. Because it retrieves the Q-value of the state-action pair where Q(s', a) is maximized, it is an *off-policy* method. The algorithm for Q-learning can be found in pseudocode in figure 1.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{1}$$

Initialize Q(s,a) arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s' using policy derived from Q (e.g., ϵ-greedy)
        Take action a, observe r, s'
        Q(s,a) $\leftarrow$ Q(s,a) + $\alpha[r + \gamma \max'_a Q(s', a') - Q(s, a)]$
        s $\leftarrow$ s';
    until s is terminal

Figure 1: The algorithm for one-step Q-learning[1]

2.2 Minimax Q-Learning

The problem with independent Q-learning is that it is based on a stationary environment, i.e. where the rules stay the same. However, in a Multi-agent environment where other agents learn as well, the environment is dynamic. This means that the guarantees that hold for single-agent learning do not hold in this setting. Littman [2] specifically considers two-player zero-sum games. In this type of Markov Game, it is possible to use a single reward function, that one player tries to *maximize*, and the other (the opponent) tries to *minimize*. For MDPs, there is a policy π that is optimal. However, for Markov Games, there is often no *undominated* policy[2]. The solution to this is to pick a policy and estimate its value by assuming the opponent will take the actions that are worst for the agents, with regards to this policy. In short, minimax picks the policy that maximizes the agent's reward in the worst case. This optimal policy can be stochastic, as seen in the policy for *rock, paper, scissors* (where the best policy is being unpredictable so you cannot be exploited). To find the optimal policy π^*, linear programming can be used, where the value of a state is

$$V(s) = \max_{\pi \in PD(A)} \min_{o \in O} \sum_{a \in A} Q(s, a, o) \pi_a \tag{2}$$

and the best action is selected using the Q-values, computed by

$$Q(s, a, o) = R(s, a, o) + \gamma \sum_{s'} T(s, a, o, s') V(s') \tag{3}$$

where T is the transition function for transitioning from state s to state s' if the agent picks action a and the opponent picks action o. However, since s followed by s' after actions a and o happens with a probability

[1] More specifically, this is *one-step Q-learning*, which is the algorithm evaluated in this paper.
[2] If a policy is dominated, that means a better policy exists

$T(s, a, o, s')$, this function can be left out of the equation.

Consequently, each agent uses the following update rule:

$$Q(s, a, o) \leftarrow Q(s, a, o) + \alpha(R + \gamma V(s') - Q(s, a, o)) \tag{4}$$

Then, linear programming is used to find a policy π so that

$$\pi(s,) \leftarrow \underset{\pi'(s,)}{\operatorname{argmax}} \left\{ \min_{o'} \left\{ \sum_{a'} \{\pi(s, a') \times Q(s, a', o')\} \right\} \right\} \tag{5}$$

Initialize Q(s,a) arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s' using policy derived from Q (e.g., ϵ-greedy)
        Take action a, observe reward R, s' and opponent's action o
        Q(s,a,o) $\leftarrow$ Q(s,a,o) + α(R + γ V(s') - Q(s,a,o))
        $\pi(s,) \leftarrow \underset{\pi'(s,)}{\operatorname{argmax}} \left\{ \min_{o'} \{\sum_{a'} \{\pi(s, a') \times Q(s, a', o')\}\} \right\}$
        $V(s) \leftarrow \min_{o'} \{\sum_{a'} \{\pi(s, a') \times Q(s, a', o')\}\}$
        $\alpha \leftarrow \alpha \times decay$

Figure 2: Minimax-Q learning[2]

2.3 Independent SARSA

In independent SARSA, all the agents independently use SARSA to maximize their expected reward.

SARSA is a temporal difference method, like Q-learning, that uses the update rule in equation 6. Because this algorithm retrieves the Q value after taking an ϵ-greedy action, it is an *on-policy* method. The algorithm for SARSA can be found in pseudocode in figure 3.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{6}$$

Initialize Q(s,a) arbitrarily
Repeat (for each episode):
    Initialize s
    Choose a from s' using policy derived from Q (e.g., ϵ-greedy)
    Repeat (for each step of episode):
        Take action a, observe r, s'
        Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
        Q(s,a) $\leftarrow$ Q(s,a) + $\alpha[r + \gamma Q(s', a') - Q(s, a)]$
        s $\leftarrow$ s';
    until s is terminal

Figure 3: The algorithm for Sarsa [1]

3 Implementation

3.1 New state space

In the previous assignments, two agents participated in the game. A predator chased a prey, which hardly ever moved and didn't learn. Now, both the predator and the prey move around on the grid. Both agents learn and it is possible to initialize the game with up to four predators, totaling five agents. Adding an agent to the game increases the state space exponentially, making computation of the algorithms very expensive. Combined with the fact that all algorithms need time to learn and that multiple experiments must be run for accuracy, the algorithms will be executed several thousand times. It is, therefore, essential to calculate results as quickly as possible. To reduce the state space complexity, it is encoded as follows: instead of using the locations of all agents as states, the distance to every other agent is calculated. Along these distances, the Q-values are evaluated and used to take an action. The Q-values will be updated for distances between agents, disregarding the absolute locations of agents and thus, taking advantage of the state space symmetry. Using this technique, the number of states to be updated per agents is reduced, resulting in reducing the state space from 11^{2n} to $11^{2(n-1)}$ where n is the number of agents in the grid.

3.2 Files

The implementation consists of the following files:

Agents_new
: This file contains implementions of the Agent class, the Prey class and the Predator class. Both the predator and the prey inherit functions of the Agent class. The Agent class contains functions any agent needs, such as a set of actions, a policy and other functions. As the predator is the agent is the agent this implementation focuses on, the predator class contains more functions than the predator class.

Helpers
: This file contains many helper functions. These functions aid in computation and decision making, but cannot (and need not) be part of a specific class.

Other_objects
: This file contains the Policy and Environment classes. The environment of the game as well as the rules are implemented in the Environment class. The Policy class contains the implementation of Q-Learning, Sarsa, ϵ-greedy, softmax action selection and more functions that help in determining and optimizing a policy as well as choosing an action of this policy.

Newstate
: This file contains the Game class as well as a demonstration function. The Game class instantiates the game, initialized the predator and the prey and assigns policies to these. The game is run N times and the result is printed. The demonstration function also performs independent Q-learning, minimax Q-learning and independent SARSA. It uses ϵ-greedy action selection. The results are printed in the command line and graphs are used for analysis.

4 Analysis

This section discusses the results of the implementations. In order to display and compare results, graphs are used. The title describes which parameters are analysed and the legend shows which color represents which setting of said parameters. Contrary to previous reports, tests were run 5 times with 2000 episodes. Eventually, the results were averaged and used for analysis. In order to analyse the results, several default parameters were established, based on the previous assignments. These are:

- 2000 runs
- 5 experiments
- 2 predators
- 0.9 discount factor (γ)
- 0.5 learning rate (α)
- 0.1 epsilon (ϵ)

When testing the implementation or parameters, the rest of the parameters remain as stated above.

4.1 Independent Q-Learning

This section analyses the effects of independent Q-learning, as well as different parameter settings for this learning method.

4.1.1 1 predator vs. 1 prey

To start off, a test was run to see if the new environment behaves as expected. Not much has changed, except allowing multiple predators to be initialized, different functions for Q-learning and policies (they need to be dynamic and change with the number of agents) and the caveat that the prey has to trip during 20% of its actions.

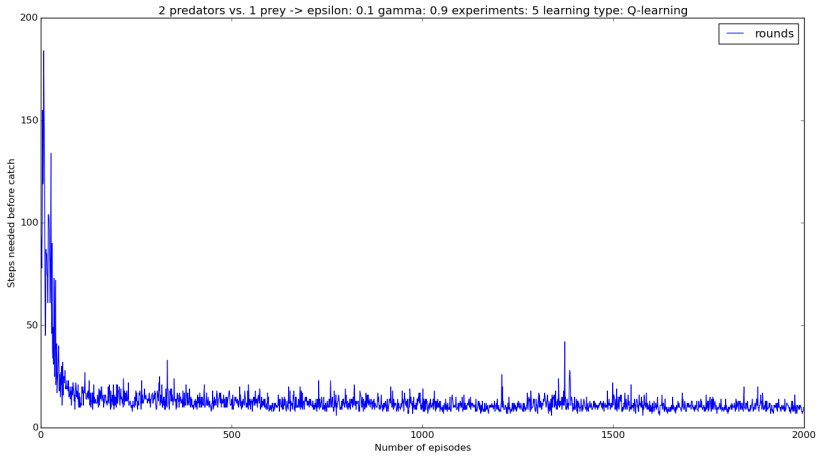

Figure 4: Independent Q-learning: 1 predator versus 1 prey

As the graph shows, the predator learns how to catch the prey more quickly after each time step. However, in the previous implementation, the number of time steps it took the predator to catch the prey became more stable. Though the number of time steps needed to catch the prey drops significantly (apparently, the 20% trip rate more than offsets the intelligence of the prey), there is more variance than before. Also, the algorithm learns more quickly than before. As the state space grows exponentially with every added predator,

the state space was encoded to reduce the amount of possible states and is now of size 11^2. Therefore, after the predator has spent enough time learning, each greedily chosen action is always in the direction of the prey. This leads to catching the prey quicker than before as there is a much smaller state space to be explored.

4.1.2 2 predators versus 1 prey

In this case, there are two predators hunting one prey. The state space is now larger than before (11^4 states) and leads to slower computations. Tests have shown that the amount of rounds it takes for any of the predators to catch the prey vary significantly, but more importantly, cannot be informatively represented in a graph. Therefore, the cumulative wins and losses for the predators have been graphed, rather than the amount of rounds it takes the predators to catch the prey. Unsurprisingly, the graph for wins becomes steeper, and the graph for losses becomes flat after learning. This indicates that the predators are winning more and losing less. Moreover, the numbers of rounds it takes does improve over time. These numbers have been tracked and logged in the table below.

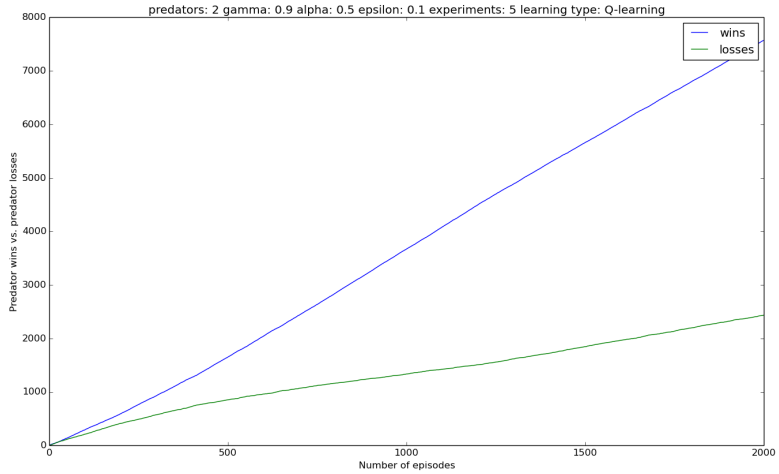

Figure 5: Independent Q-learning: 2 predators vs. 1 prey

The graph shows that the predators learn to cooperate and catch the prey. This number increases, as the number of predator losses increases more slowly over time, while the number of predator wins increases more quickly over time. As this is counted cumulatively, it can be expected that the number of wins by the prey will become almost steady. It is expected that, in the end, the predators will win almost each game. As their policies are still exploratory, it is possible for two predators to bump into one another and lose the game.

The table below shows the average number of rounds the predators need to catch the prey.

	Avg wins (first 100)	**Avg losses (first 100)**	**Avg wins (last 100)**	**Avg losses (last 100)**
Predators	58	42	76	23

Table 1: Average number of rounds two predators need to catch one prey

As the table shows, the predators learn to catch the prey quicker over time.

4.1.3 3 predators versus 1 prey

With four agents on the grid, the implementation became very slow. It was possible to run the implementation, but as it is very slow, the parameter changes have not been tested. However, figure -some-reference-

shows the results of running three predators versus one prey. Due to time constraints and the amount of time it takes for the algorithm to calculate the all the states, this was for 1000 episodes. Contrary to the other testing approaches, this test was run once and so not averaged over several experiments.

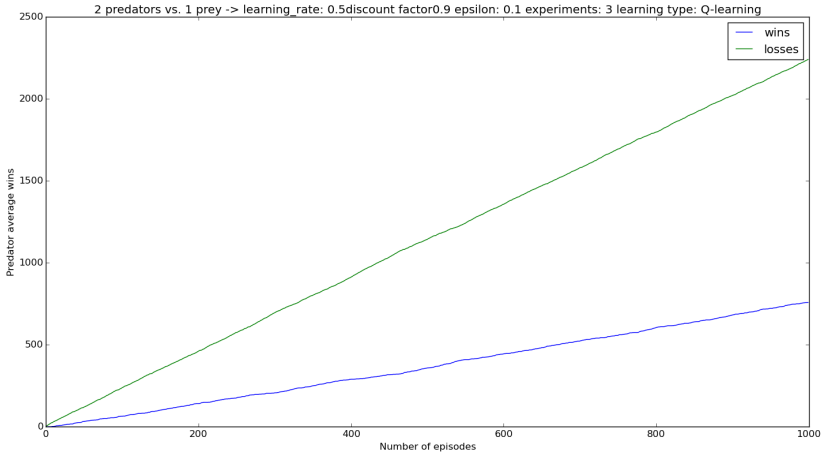

Figure 6: Independent Q-learning: 3 predators vs. 1 prey

This figure shows that the predators lose the game a lot. This is interesting as it is expected of the predators to learn not to bump into one another. However, the grid is both toroidal as well as small and the prey learns. It is possible that the prey learning, combined with a small, toroidal grid, leads to the predators bumping into one another. Perhaps the prey learns to trick the predators into bumping into one another. Therefore, it is interesting to see what happens if the prey learns slower than the predators. By making the prey learn slower, theoretically it is possible for the predators to learn not to bump into one another and catch the prey. In order to simulate this, the predators learn as stated in default, but the prey learns with a learning rate of 0.01. These results are shown in the next figure.

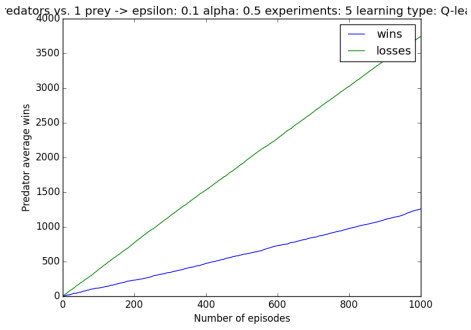

Figure 7: Independent Q-learning: 3 predators vs. 1 prey, slow learning prey

The following table shows the results of the first- and last 100 wins and losses of the predators.

	Avg wins (first 100)	**Avg losses (first 100)**	**Avg wins (last 100)**	**Avg losses (last 100)**
Default learning rate	21	78	25	73
Low learning rate	23	76	31	68

Table 2: Average number of wins and losses by the predators with varying learning rates

The results in table # show that the predators do learn to catch the prey. When the prey learns slower, the predators manage to catch the prey more often in the same amount of time. Therefore, it can be concluded that the algorithm does learn the predators to cooperate and catch the prey. This just takes many episodes.

4.1.4 4 predators versus 1 prey

Though it is implemented for four predators and one prey to be placed on the grid, this leads to implementations freezing. It is therefore conclusive to state that the program has become intractable. This could be solved by using function approximation, for example using Kanerva coding [3], where a number of *prototype* state-action pairs are selected, to be used for storing Q-values for similar state-action pairs. Another possible solution is to have the predators learn *together*, meaning they share a Q-value grid and learn simultaneously. However, this is not truly independent Q-learning.

4.1.5 Parameter settings

It is interesting to see what happens when the parameters of the learning methods change. As the effects of parameter settings have been researched in a 1 versus. 1 scenario, it is interesting to see what is different when there are more agents on the grid. Also, as all agents now learn, the effects of these learning methods should change.

4.1.6 Learning rate

First, the effect of the learning rate is researched. As the learning rate determines to what extent the newly acquired information will override the old information, it is interesting to see what happens.

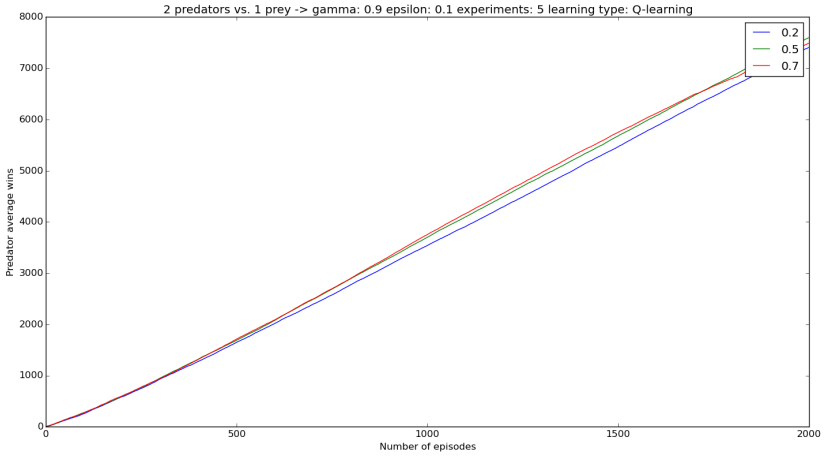

Figure 8: Independent Q-learning: 2 predators vs. 1 prey, learning rate

The graph shows that a low learning rate yields worst results. For a long time, a high learning rate yields good results, however, in the end a learning rate of 0.5 yields best results. This shows that for a long time, a lot of recent information is interesting. Later on, however, an even balance of new and old information leads to more wins for the predator.

Learning rate	Avg wins (first 100)	Avg losses (first 100)	Avg wins (last 100)	Avg losses (last 100)
0.2	50	49	74	24
0.5	54	45	72	27
0.7	54	45	63	35

Table 3: Average number of wins and losses by the predators with varying learning rates

The table shows that the lowest learning rate shows better and better results over time. This shows that in the beginning, a lot must be learned. As the game progresses, a low learning rate yields better results. This could indicate that the predators as well as the prey become predictable and so less new information has to be learned. As minimax Q-learning contains a decay in learning, perhaps this is the reason why.

4.1.7 Discount factor

The discount factor determines the importance of future rewards. In the previous assignment, the a high discount factor yielded best results. This means that the future reward was most important. Only the goal state yielded a reward, making reaching the goal state very important. Currently, there are two terminal states: the win state and the lose state. It is interesting to see what effect the negative rewards have on the importance of the immediate reward.

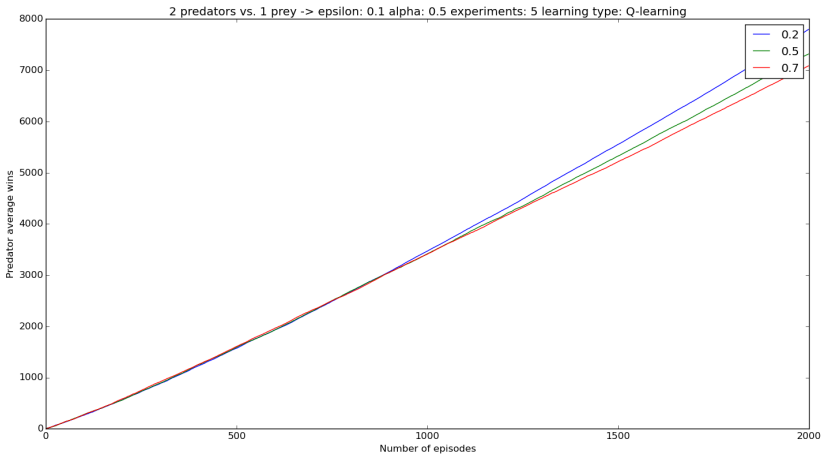

Figure 9: Independent Q-learning: 2 predators vs. 1 prey, discount factor

The graph shows that for a long time, it does not matter how important the future reward is. However, eventually the graph shows that a low discount factor yields best results. This might be caused by the fact that the predators will receive a negative reward when running into another predator. In order to avoid this, winning (or not losing) immediately is more important than expected future rewards.

Discount factor	Avg wins (first 100)	Avg losses (first 100)	Avg wins (last 100)	Avg losses (last 100)
0.2	52	47	98	4
0.5	55	44	78	21
0.7	55	45	74	24

Table 4: Average number of predator wins and losses for varying discount factors

The table shows that the discount factor has a huge impact on the success of the predators. By making sure that the predators do not run into each other, the game is not lost as often.

4.1.8 ϵ-greedy action selection

In the current implementation, ϵ-greedy action selection was used to find actions for the agents. For this action-selection mechanism, ϵ determines the percentage of greedy versus exploratory actions. An ϵ value of 0 selects only greedy actions. The closer this value is to 1, the more exploring actions are selected. The following figure shows the results of this test.

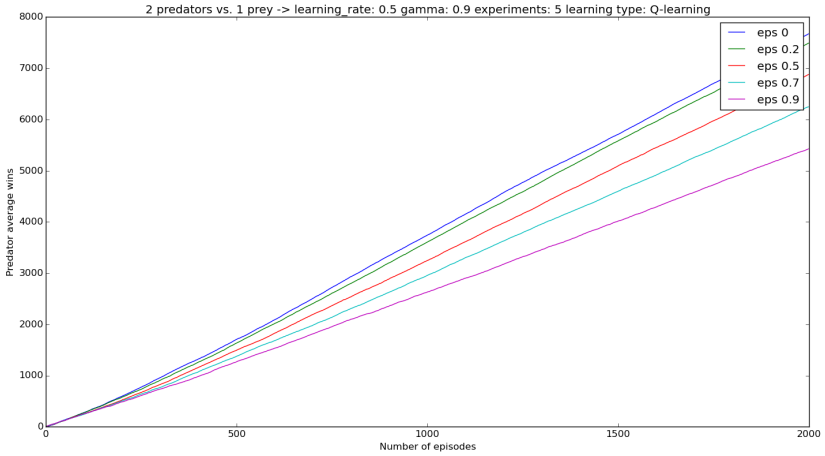

Figure 10: Independent Q-learning: 2 predators vs. 1 prey, ϵ-greedy action selection

Figure shows that greedy action selection yields best results. This is possible, as all predators are initialized at the corners of the grid, starting out with equal distance to the prey. As the prey moves, it will be closer to one predator. Therefore, after learning and without exploration, the prey will be caught by one predator (as it tries to minimize the distance between the prey and itself). As a greedy action, in this case, leads to moving in the direction of the highest Q-value, it is still possible for the predators to bump into each other. However, it seems as if the prey is most often caught before this happens leading to wins for the predators.

ϵ-rate	**Avg wins (first 100)**	**Avg losses (first 100)**	**Avg wins (last 100)**	**Avg losses (last 100)**
0	55	44	76	22
0.2	54	45	77	21
0.5	49	50	72	27
0.7	48	51	66	32
0.9	50	59	55	44

Table 5: Average # wins and losses by the predators with varying ϵ-rates

Though, at first, an absolute greedy policy appears to yield the most promising results, a slightly exploratory policy eventually yields the best results. This is not displayed in the graph as the wins are counted cumulatively. Eventually, the results will be best when still exploring sightly.

4.2 Minimax Q-learning

4.3 Independent SARSA

This section discusses the effects of independent SARSA learning.

4.3.1 1 predator vs. 1 prey

Again, it is interesting to see if independent SARSA works well with the new environment. Te test this, the new environment executed a game of 1 predator vs. 1 prey. The results can be found in figure #.

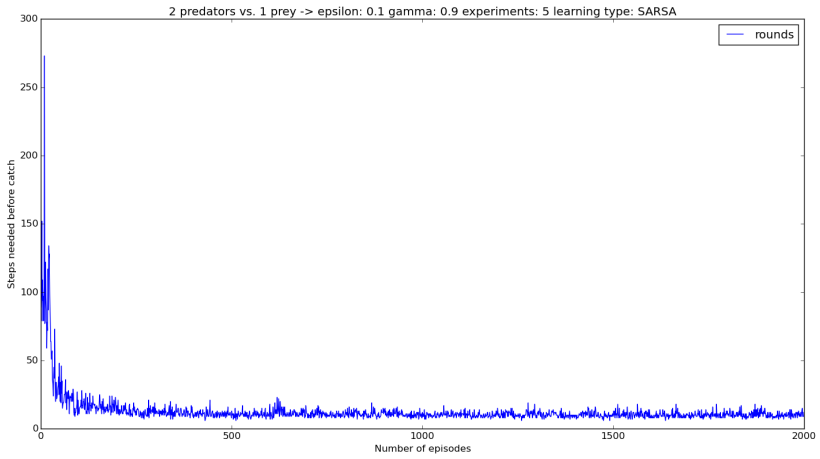

Figure 11: Independent SARSA: 1 predator vs. 1 prey

As seen with independent Q-learning, the predator learns how to catch the prey quicker than before. This confirms the theory that the smaller state space leads to quicker conversion of the algorithms.

Compared to independent Q-learning, SARSA takes much longer to catch the prey in the beginning. Also, after learning the amount of rounds it takes the predators to catch the prey still vary much. However, where there is a spike rounds in Q-learning, there is not in SARSA. This happens due to the fact that SARSA is more careful than Q-learning. As SARSA is an on-policy learning algorithm, it is more careful than Q-learning. This leads to less extreme exploratory actions, choosing the "safe path" more often than Q-learning.

4.3.2 2 predators vs. 1 prey

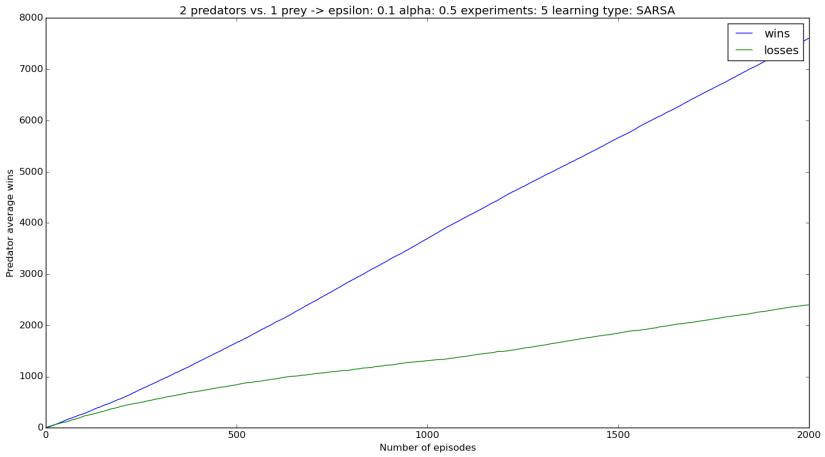

Figure 12: Independent SARSA: 2 predators vs. 1 prey

The graph is very similar to the graph in Q-learning. This is possible, as both are temporal difference learning methods and are based on the same principles. The main difference between these two is that SARSA is an on-policy learning method and Q-learning is an off-policy learning method. It is therefore expected for these two methods to behave similarly.

	Avg wins (first 100)	**Avg losses (first 100)**	**Avg wins (last 100)**	**Avg losses (last 100)**
Predators	55	44	78	21

Table 6: Average # wins and losses by the predators

Compared to independent Q-learning, independent SARSA performs slightly better. This can be attributed to the fact that SARSA is more careful than Q-learning. This behaviour leads to more wins for the predators, on average, compared to Q-learning.

4.3.3 3 predators vs. 1 prey

It is interesting to see what happens when three predators take on one prey. As seen with independent Q-learning, this does not work well. As SARSA is a more careful algorithm, the predators may learn to cooperate better than with Q-learning.

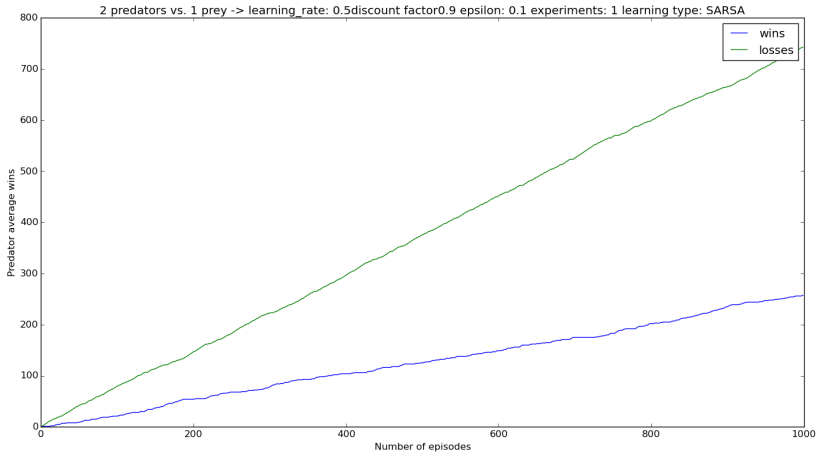

Figure 13: Independent SARSA: 3 predators vs. 1 prey

The graph shows that the prey manages to escape most of the times. This hardly differs from independent Q-learning.

	Avg wins (first 100)	**Avg losses (first 100)**	**Avg wins (last 100)**	**Avg losses (last 100)**
Predators	21	79	22	77

Table 7: Average # wins and losses by the predators with varying learning rates

This shows that SARSA learns very slowly, in this case. Combined with the fact that there are many predators on a small grid, these can bump into one another. Any learning will be much delayed. It is important to note that this is the only test which was run once, with 1000 episodes.

4.3.4 4 predators vs. 1 prey

Again, this is intractable.

4.3.5 Parameter settings

Again, parameter settings were explored for this algorithm. This was tested with two predators and one prey.

4.3.6 Learning rate

First, the effect of the learning rate is researched. As the learning rate determines to what extent the newly acquired information will override the old information, it is interesting to see what happens.

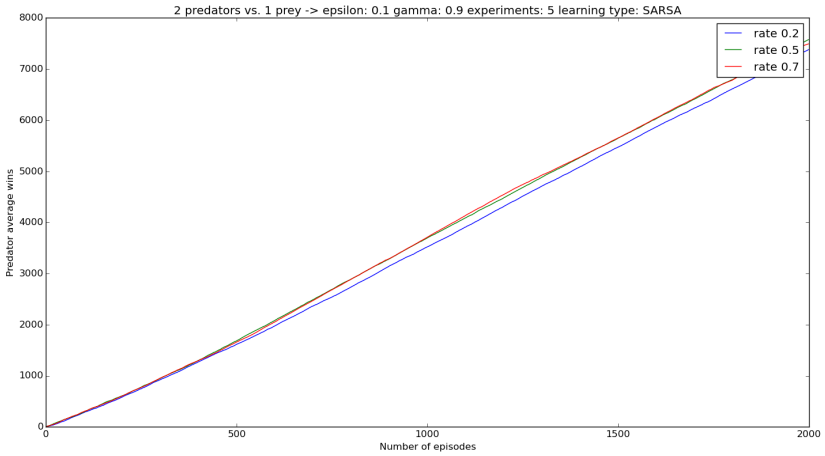

Figure 14: Independent SARSA: 2 predators vs. 1 prey, learning rate

From the graph it is difficult to see if a high or average learning rate affect the implementation in the best possible way. In the end, however it appears as though a learning rate of 0.5 yields best results.

ϵ-rate	Avg wins (first 100)	Avg losses (first 100)	Avg wins (last 100)	Avg losses (last 100)
0.2	56	44	75	23
0.5	58	41	79	19
0.7	59	40	68	30

Table 8: Average # wins and losses by the predators with varying learning rates

The table confirms the theory derived off the graph. A learning rate does yield best results. When the predators are learning while hunting the prey, an equal balance of new and old information yields optimal results.

4.3.7 Discount factor

The discount factor determines the importance of future rewards. In the previous assignment, the a high discount factor yielded best results. This means that the future reward was most important. Only the goal state yielded a reward, making reaching the goal state very important. Currently, there are two terminal states: the win state and the lose state. It is interesting to see what effect the negative rewards have on the importance of the immediate reward.

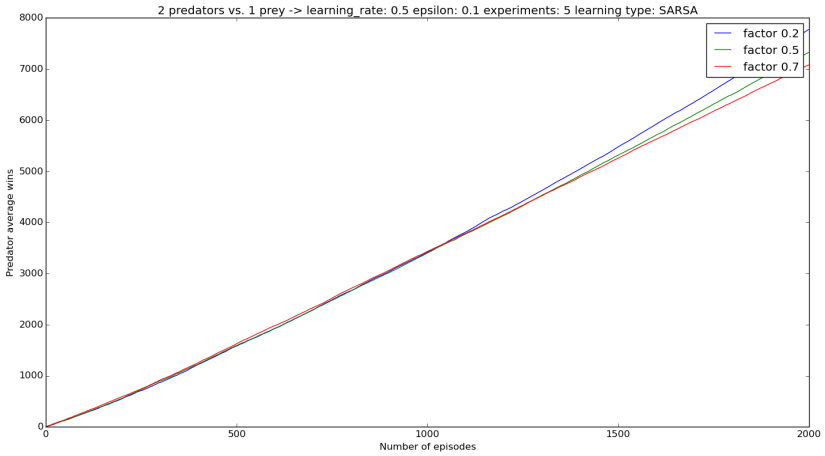

Figure 15: Independent SARSA: 2 predators vs. 1 prey, discount factor

Again, the immediate reward shows to be important. It has been mentioned several times that SARSA is a careful algorithm, as it collects rewards during learning. Independent Q-learning yields similar results, confirming the theory that introducing a negative reward into the game makes the immediate reward more important.

ϵ-rate	Avg wins (first 100)	Avg losses (first 100)	Avg wins (last 100)	Avg losses (last 100)
0.2	52	47	95	3
0.5	53	46	80	18
0.7	57	42	71	27

Table 9: Average # wins and losses by the predators with varying discount factors

The previous conclusions are supported by the table. However, it is interesting to note that keeping into account the immediate reward yields significantly better results than seen before. Winning games 95% of the time, on average, is highly successful. This means that even when the predators are exploring new paths, the predators hardly ever bump into one another. These are excellent results for any algorithm, let alone an on-policy algorithm.

4.3.8 ϵ-greedy action selection

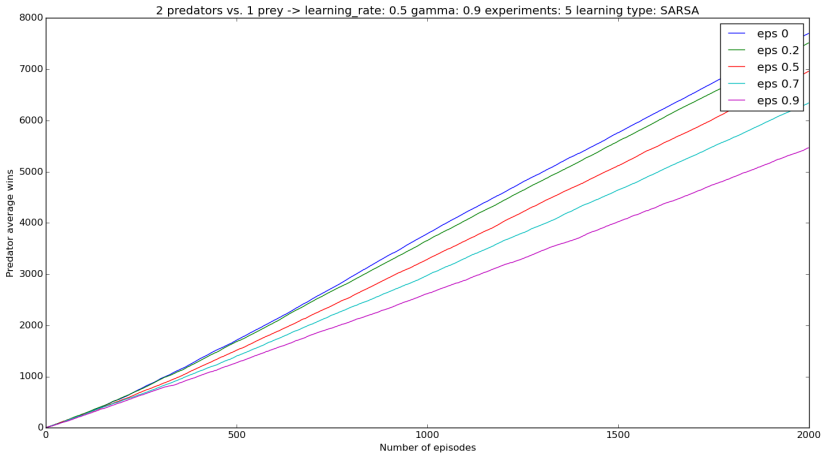

Figure 16: Independent SARSA: 2 predators vs. 1 prey, ϵ-greedy action selection

The graph shows that low exploration leads to the best results. Again, this is to be expected as exploration most likely will lead to predators bumping into one another.

ϵ-rate	Avg wins (first 100)	Avg losses (first 100)	Avg wins (last 100)	Avg losses (last 100)
0	54	45	74	24
0.2	55	45	74	24
0.5	53	47	73	25
0.7	51	48	66	32
0.9	48	51	57	41

Table 10: Average # wins and losses by the predators with varying ϵ-rates

The table confirms almost what the graph shows. Where the graph shows better results for an absolutely greedy action selection, the table shows that the results over the last 100 runs are exactly the same. This will lead to trying to catch the prey in the beginning, but exploring other paths to the prey while avoiding other predators is also important. This can help the predators significantly in the future. However, exploration is needed. Therefore, lowering the learning rate over time will help the predators best. This confirms the theory that the learning rate must be lowered in order to yield best results.

5 Conclusion

This section discusses the conclusions drawn from the analysis.

5.1 Independent learning

Independent learning was performed in two different ways: Q-learning and SARSA. When more than 2 predators enter the grid, the predators lose the game more often than the prey. This is to be expected as the agents learn independently from one another. Though all agents act on what is best for them, they do not work together. Therefore, all agents must learn each others behaviour before being able to catch the prey without bumping into another predator. This takes very long and even then it is not certain that the predators will catch the prey. As all agents learn each others policies, the prey will also learn what the predators will do and may trick them into bumping into one another.

5.1.1 Parameter settings

Parameters settings for both algorithms were tested.

5.2 Minimax Q-learning

Draw thine conclusions and place them here. Oh Romeo, Romeo. Wherefore art thou Romeo? :'(

6 Future work

This section contains information about improvements in the future

6.1 State space encoding

It has shown that the state space encoding has improved the performance of the program. However, in order to run tests with four predators on one grid, a smarter encoding is necessary. With a smarter state space encoding, it will be possible to run the program with up to four predators and this research can be completed. The smarter state space encoding will also allow multiple experiments to be run on grids with three or more predators.

6.2 Grid size

All the tests were performed on an 11x11 grid. As it is believed that the predators bump into each other so much is because the grid is small, it is interesting to see how all agents learn on a larger grid. As stated before, this will significantly increase the state space of the implementation, leading to slow computation. In order to perform these tests, it is imperative to come up with smarter state space encoding. Then these tests can be executed and the effects of these learning algorithms can be evaluated.

7 Files attached

- newstate.py
- agents_new.py
- other_objects.py
- helpers.py ...

8 Sources

References

[1] Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[2] Michael L. Littman. "Markov games as a framework for multi-agent reinforcement learning". In: *IN PROCEEDINGS OF THE ELEVENTH INTERNATIONAL CONFERENCE ON MACHINE LEARNING*. Morgan Kaufmann, 1994, pp. 157–163.

[3] Cheng Wu and Waleed Meleis. "Function Approximation Using Tile and Kanerva Coding For Multi-Agent Systems". In: *Proc. Of Adaptive Learning Agents Workshop (ALA) in AAMAS*. 2009.