Autonomous Agents 1: Assignment 1

Tran Cong Nguyen, Lautaro Quiroz, Joost van Doorn, Roger Wechsler

November 14, 2014

1 Introduction

In this assignment we implemented the predator-prey environment and three algorithms to determine state values and an optimal policy for the predator.

Our code consists of one file that defines the agent, its environment including the prey and the three algorithms. In addition, we wrote a simulator that visualizes the steps of the predator and the prey on the grid (figure 1).

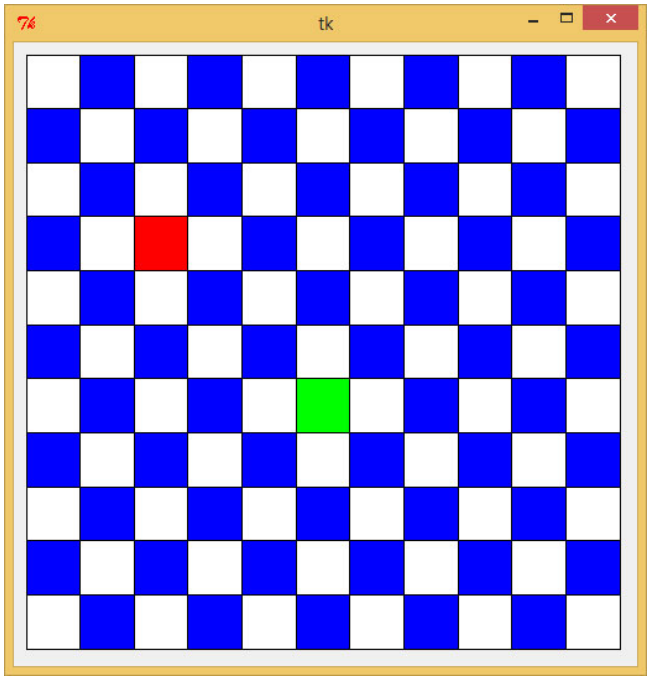

Figure 1: Screenshot of the simulation. The red square is the predator, and the green one is the prey.

In the following sections we evaluate a random policy of the predator and report some properties of the iterative policy evaluation, policy iteration and value iteration. Finally, we show how we can reduce the state-space, which results in faster computations for all algorithms.

2 Task 1: Random Policy

For the first task we let the predator behave according to a random policy. For each time step the predator randomly chooses one of five possible actions (up, right, down, left, stay) independet of its state. A run finishes if the predator randomly runs into the prey.

Table 1 shows the number of steps that were necessary for the predator to catch the prey over 100 runs. The number of steps that led to success varies highly from a very lucky run of only 14 steps to a very unlucky one of almost 2,000 steps. This explains the high standard deviation. Later in this report, table 5 shows how an optimal policy significantly reduces those numbers.

Min	Max	Mean	Standard deviation
14	1956	375.76	367.26

Table 1: Number of iterations needed to catch the prey in 100 simulations using a random policy

3 Task 2: Iterative Policy Evaluation

Table 2 shows the values for the states (predator, prey) after implementing iterative policy evaluation. The algorithm converged after 16 iterations using a discount factor $\gamma = 0.8$.

Coordinates		Values
Predator	Prey	
0, 0	5, 5	0.0013
2, 3	5, 4	0.1240
2, 10	10, 0	0.1240
10, 10	0, 0	1.2415

Table 2: Values for four selected states after iterative policy evaluation using a random policy ($\gamma = 0.8$).

4 Task 3: Policy Iteration

Table 3 illustrates the number of iterations needed until convergence for different discount factors using policy iteration. Since we initialize the policy randomly at the beginning of the algorithm, we report averaged numbers over 10 runs.

γ	Iterations
0.1	13.8
0.5	13.1
0.7	23.2
0.9	102.3

Table 3: Random sample of number of iterations needed until convergence using policy iteration for different discount factors γ, averaged over 10 runs

The state values for all states where the prey is located at (5,5) are illustrated in figure 2.

	0	1	2	3	4	5	6	7	8	9	10
0	0.0	0.0	0.0	0.0001	0.0006	0.0023	0.0006	0.0001	0.0	0.0	0.0
1	0.0	0.0	0.0001	0.0006	0.004	0.0187	0.004	0.0006	0.0001	0.0	0.0
2	0.0	0.0001	0.0006	0.0042	0.0285	0.1491	0.0285	0.0042	0.0006	0.0001	0.0
3	0.0001	0.0006	0.0042	0.0285	0.1928	1.2747	0.1928	0.0285	0.0042	0.0006	0.0001
4	0.0006	0.004	0.0285	0.1928	1.2747	8.8889	1.2747	0.1928	0.0285	0.004	0.0006
5	0.0023	0.0187	0.1491	1.2747	8.8889	8.8889	8.8889	1.2747	0.1491	0.0187	0.0023
6	0.0006	0.004	0.0285	0.1928	1.2747	8.8889	1.2747	0.1928	0.0285	0.004	0.0006
7	0.0001	0.0006	0.0042	0.0285	0.1928	1.2747	0.1928	0.0285	0.0042	0.0006	0.0001
8	0.0	0.0001	0.0006	0.0042	0.0285	0.1491	0.0285	0.0042	0.0006	0.0001	0.0
9	0.0	0.0	0.0001	0.0006	0.004	0.0187	0.004	0.0006	0.0001	0.0	0.0
10	0.0	0.0	0.0	0.0001	0.0006	0.0023	0.0006	0.0001	0.0	0.0	0.0

(a) $\gamma = 0.1$

	0	1	2	3	4	5	6	7	8	9	10
0	0.0465	0.0875	0.1644	0.3082	0.5809	0.9353	0.5809	0.3082	0.1644	0.0875	0.0465
1	0.0875	0.1611	0.3084	0.591	1.1247	1.8978	1.1247	0.591	0.3084	0.1611	0.0875
2	0.1644	0.3084	0.591	1.1325	2.169	3.8615	2.169	1.1325	0.591	0.3084	0.1644
3	0.3082	0.591	1.1325	2.169	4.165	7.8919	4.165	2.169	1.1325	0.591	0.3082
4	0.5809	1.1247	2.169	4.165	7.8919	16.0	7.8919	4.165	2.169	1.1247	0.5809
5	0.9353	1.8978	3.8615	7.8919	16.0	16.0	16.0	7.8919	3.8615	1.8978	0.9353
6	0.5809	1.1247	2.169	4.165	7.8919	16.0	7.8919	4.165	2.169	1.1247	0.5809
7	0.3082	0.591	1.1325	2.169	4.165	7.8919	4.165	2.169	1.1325	0.591	0.3082
8	0.1644	0.3084	0.591	1.1325	2.169	3.8615	2.169	1.1325	0.591	0.3084	0.1644
9	0.0875	0.1611	0.3084	0.591	1.1247	1.8978	1.1247	0.591	0.3084	0.1611	0.0875
10	0.0465	0.0875	0.1644	0.3082	0.5809	0.9353	0.5809	0.3082	0.1644	0.0875	0.0465

(b) $\gamma = 0.5$

	0	1	2	3	4	5	6	7	8	9	10
0	1.2129	1.6918	2.3581	3.2823	4.5923	5.9168	4.5923	3.2823	2.3581	1.6918	1.2129
1	1.6918	2.3224	3.2764	4.6251	6.5111	8.5993	6.5111	4.6251	3.2764	2.3224	1.6918
2	2.3581	3.2764	4.6251	6.529	9.2156	12.5236	9.2156	6.529	4.6251	3.2764	2.3581
3	3.2823	4.6251	6.529	9.2156	13.0186	18.2868	13.0186	9.2156	6.529	4.6251	3.2823
4	4.5923	6.5111	9.2156	13.0186	18.2868	26.6667	18.2868	13.0186	9.2156	6.5111	4.5923
5	5.9168	8.5993	12.5236	18.2868	26.6667	26.6667	26.6667	18.2868	12.5236	8.5993	5.9168
6	4.5923	6.5111	9.2156	13.0186	18.2868	26.6667	18.2868	13.0186	9.2156	6.5111	4.5923
7	3.2823	4.6251	6.529	9.2156	13.0186	18.2868	13.0186	9.2156	6.529	4.6251	3.2823
8	2.3581	3.2764	4.6251	6.529	9.2156	12.5236	9.2156	6.529	4.6251	3.2764	2.3581
9	1.6918	2.3224	3.2764	4.6251	6.5111	8.5993	6.5111	4.6251	3.2764	2.3224	1.6918
10	1.2129	1.6918	2.3581	3.2823	4.5923	5.9168	4.5923	3.2823	2.3581	1.6918	1.2129

(c) $\gamma = 0.7$

	0	1	2	3	4	5	6	7	8	9	10
0	31.4943	34.8064	38.4582	42.4787	46.9748	50.7257	46.9748	42.4787	38.4582	34.8064	31.4943
1	34.8064	38.213	42.3995	47.0569	52.194	56.7821	52.194	47.0569	42.3995	38.213	34.8064
2	38.4582	42.3995	47.0569	52.2287	57.9677	63.6111	57.9677	52.2287	47.0569	42.3995	38.4582
3	42.4787	47.0569	52.2287	57.9677	64.3486	71.3295	64.3486	57.9677	52.2287	47.0569	42.4787
4	46.9748	52.194	57.9677	64.3486	71.3295	80.0	71.3295	64.3486	57.9677	52.194	46.9748
5	50.7257	56.7821	63.6111	71.3295	80.0	80.0	80.0	71.3295	63.6111	56.7821	50.7257
6	46.9748	52.194	57.9677	64.3486	71.3295	80.0	71.3295	64.3486	57.9677	52.194	46.9748
7	42.4787	47.0569	52.2287	57.9677	64.3486	71.3295	64.3486	57.9677	52.2287	47.0569	42.4787
8	38.4582	42.3995	47.0569	52.2287	57.9677	63.6111	57.9677	52.2287	47.0569	42.3995	38.4582
9	34.8064	38.213	42.3995	47.0569	52.194	56.7821	52.194	47.0569	42.3995	38.213	34.8064
10	31.4943	34.8064	38.4582	42.4787	46.9748	50.7257	46.9748	42.4787	38.4582	34.8064	31.4943

(d) $\gamma = 0.9$

Figure 2: State values for different positions of predator with prey fixed at (5,5) for different discount factors γ.

5 Task 4: Value Iteration

Table 4 shows the number of iterations needed to converge under the value iteration algorithm.

γ	Iterations
0.1	7
0.5	20
0.7	36
0.9	112

Table 4: Random sample of number of iterations needed until convergence using value iteration for different discount factors γ. All state values were initialized with 0.

The state values for all states where the prey is located at (5,5) are the same as in figure 2.

The value iteration algorithm also returns a deterministic optimal policy. Table 5 shows the number of steps the predator needed to catch the prey using the optimal policy calculated by value iteration. In comparison to a random policy, the optimal policy significantly reduces the number of steps the predator needs to catch the prey.

Min	Max	Mean	Standard deviation
7	14	10.29	1.39

Table 5: Number of iterations needed to catch the prey in 100 simulations using an optimal policy

6 Task 5: State-space Reduction

The naive way of implementing the state space would be to encode both coordinates of the predator and the prey into a four-dimensional vector. However, for a board with size 11×11 used in the assignment, this would mean there will be $11^4 = 14641$ different states.

To define a vector in a 2-dimensional space, only 2 independent coordinates are needed. Therefore, we improved the implementation of the state space by using only the distances in X and Y directions between the predator and the prey. The size of the state space for the same board is significantly reduced to $11^2 = 121$. Figure 3 illustrates an example of how the new state vector is computed.

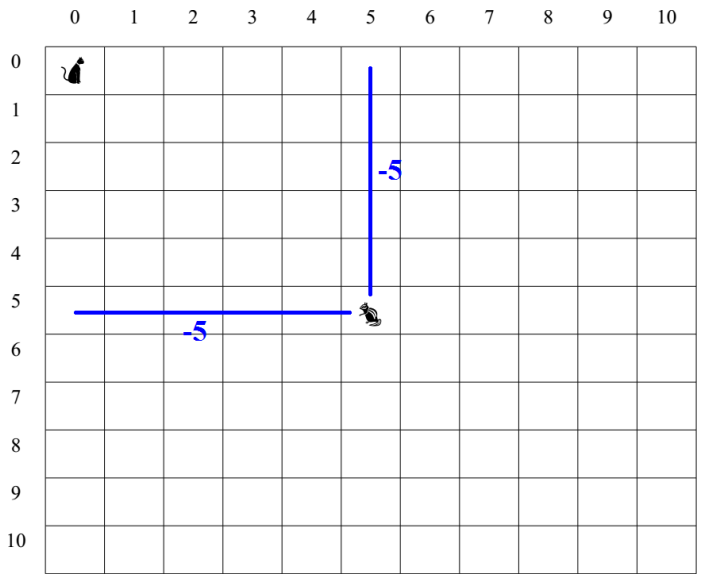

Figure 3: The reduced state only needs to encode two values for the distances in X and Y directions between the predator and the prey. In this case the state vector is $(-5, -5)$.

The following tables compare the performance achieved by using the previous three algorithms (iterative policy evaluation, policy iteration, and value iteration) under the original 11^4 state space and the new 11^2 one:

Discount	Original space	Reduced space
0.8	16	6

Table 6: Comparison between state spaces: Number of iterations needed to converge using iterative policy evaluation for a discount factor $\gamma = 0.8$.

Discount	Original space	Reduced space
0.1	13.8	8.3
0.5	13.2	6.6
0.7	23.2	6.1
0.9	102.3	6.3

Table 7: Comparison between state spaces: Number of iterations needed to converge using policy iteration for different discount factors γ, averaged over 10 runs.

Discount	Original space	Reduced space
0.1	7	7
0.5	20	17
0.7	36	30
0.9	112	87

Table 8: Comparison between state spaces: Number of iterations needed to converge using value iteration for different discount factors γ. All state values were initialized with 0.

The above tables show that the algorithms converge faster in the reduce state space.

7 Conclusion

In this assignment we have shown how we can calculate the state values of the predator-prey game using three different algorithms. We experimented with different discount factors, which varies the number of iterations required to converge. We observed that high discount factors generally result in a larger number of iterations.

Moreover, the value iteration and the policy iteration algorithms calculate an optimal policy, which allows our predator to catch its prey in only a few steps. Compared to the baseline in which the predator follows a random policy, the optimal policy reduces the number of steps of a game dramatically.

Furthermore, we have shown how the state-space can be reduced significantly to only 121 states, which speeds up the runtime of the algorithms, but does not affect the final optimal policy.

Finally, to visualize a predator-prey game, we implemented a simulator which can be used to watch the prey's nerve-racking, thrilling but hopeless escape from the predator.