

Autonomous Agents 1

Assignment 1

By Group 4: Gieske, Gornishka, Loor, Radscum

November 14, 2014

Introduction

This report discusses a predator versus prey Markov Decision Process (MDP) implementation, focused on single agent planning. The planning is focused on the predator. This MDP consists of an 11×11 toroidal grid. The predator and the prey are placed on the grid, after which the predator must catch the prey. Both can move vertically and horizontally across the grid as well as stay wait at their current location until the next time step. Describing the movements about the grid as North, East, South, West and Wait, the policies of the predator and the prey are as follows:

	North	East	South	West	Wait
Predator	0.2	0.2	0.2	0.2	0.2
Prey	0.05	0.05	0.05	0.05	0.8

The predator and prey move about on the grid as specified by the policy. However, the prey does not move towards the predator. After the prey is caught, the episode ends and the game is reverted to starting positions. Catching the prey gives an immediate reward of 10, 0 otherwise.

This implementation contains an execution of the game, policy evaluation, policy iteration and value iteration. The performance of these functions are analyzed in order to research the behaviour of the agents. The results of these functions are also compared with one another as part of analyzation.

Theory

Iterative policy evaluation

Iterative policy iteration is used to compute the state-value function v_π for an arbitrary policy π . It is a stationary algorithm where the goal state and the arbitrary policy are static. In this case, it means that the goal state, the prey, remains on the same location. By analyzing different cases for policy evaluation, the policy of the agent can be analyzed for improvement. It is expected that the policy evaluation values increase around the location of the prey. Therefore, if the agent moves in the direction of the increasing numbers on the grid, it will catch the prey. It uses the following algorithm as described in Barto and Sutton [source]:

```
Input  $\pi$ , the policy to be evaluated
Initialize an array  $v(s) = 0$ , for all  $s \in S^+$ 
Repeat
     $\Delta \leftarrow 0$ 
    For each  $s \in S$ :
        temp  $\leftarrow v(s)$ 
         $v(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma v(s')]$ 
         $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$ 
until  $\Delta < \theta$  (a small positive number)
Output  $v \approx v(s)$ 
```

Where:

$\pi(a|s)$ is an action chosen, given the state.

$p(s'|s, a)$ is a transition function.

$r(s, a, s')$ is a reward function.

$\gamma v(s')$ is the discounted state.

This algorithm gives the agent insight in the consequences taking each action has. This can help the agent in deciding which action to take according to its policy.

Policy improvement

Policy improvement is used to find an optimal, deterministic policy. This is, again a stationary function. This algorithm exists of two steps: policy evaluation and policy iteration. Policy evaluation is demonstrated in the previous section. Policy iteration finds the optimal policy. As this algorithm first performs policy evaluation until convergence and then performs policy improvement, this algorithm is relatively slow and computationally expensive. Again, from Barto and Sutton [source]:

1. Initialization
 $v(s) \leftarrow \text{arbitrary}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$
2. Policy evaluation
Repeat
 $\Delta \leftarrow 0$
For each $s \in \mathcal{S}$:
 $\text{temp} \leftarrow v(s)$
 $v(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) [r(s, a, s') + \gamma v(s')]$
 $\Delta \leftarrow \max(\Delta, |\text{temp} - v(s)|)$
until $\Delta < \theta$ (a small positive number)
3. Policy improvement
Policy stable \leftarrow true
For each $s \in \mathcal{S}$:
 $\text{temp} \leftarrow \pi(s)$
 $v(s) \leftarrow \arg \max_a \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma v(s')]$
if $\text{temp} \neq \pi(s)$, policy stable \leftarrow false

This algorithm is also used to help the agent decide which action to take according to its policy. The difference with iterative policy evaluation, however, is that the policy is updated after the full grid is evaluated. This leads to a finite, static policy, as this is a stationary function.

Value iteration

This algorithm is a quicker version of policy improvement. Where policy improvement first performs policy evaluation and then computes the optimal policy, value iteration performs these steps together. This makes value iteration quicker and computationally less expensive than policy improvement.

From Barto and Sutton [source]

```
Repeat
 $\Delta \leftarrow 0$ 
For each  $s \in \mathcal{S}$ :
     $\text{temp} \leftarrow v(s)$ 
     $v(s) \leftarrow \max_a \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma v(s')]$ 
     $\Delta \leftarrow \max(\Delta, |v(s) - \text{temp}|)$ 
until  $\Delta < \theta$  (a small positive number)
Output a deterministic policy  $\pi$ , such that
 $\pi(s) = \arg \max_a \sum_{s'} p(s'|s, a) [r(s, a, s') + \gamma v(s')]$ 
```

After performing this algorithm, the results can also be used for agent planning.

Implementation

The current implementation consists of the following files:

Agents

This file implements the agents, the prey class and the predator class. Both agents have a policy and actions as described in the introduction as well as other functions to get and set data. As the predator is the agent this implementation focuses on, this class contains more functions than the prey.

Helpers

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

Game

This file contains the game and the environment classes. The environment of the game as well as the rules are implemented in this class. The game class contains the implementation of running the game as well as the agent planning algorithms.

Analysis

Simulator for the environment

Avarage run time	Standard deviation
296 rounds	286.580390118

Iterative policy evaluation

As described in the assignment, policy evaluation is implemented. The results will be analyzed in terms of agent (predator) and goal state (prey) locations. The following cases have been analyzed:

Case	Discount Factor	Predator	Value	Prey	Value
1	(0,0)	0.8	0.003357	(5,5)	2.850622
2	(2,3)	0.8	0.198928	(5,4)	2.850622
3	(2,10)	0.8	0.198928	(10,0)	2.850622
4	(10,10)	0.8	1.650667	(0,0)	2.850622

Case 1 is the maximum distance between the predator and the prey. In this case, it can be seen that moving in a diagonal direction gets the agent to the prey fastest. Since this is not possible, the predator can move in a horizontal - vertical fashion to reach the goal state. This takes at least ten timesteps. As this is the maximum distance between the predator and the prey, it will take relatively long for the predator to catch the prey.

Case two starts with the agent at (2,3) with the prey located at (5,4). The distance between the agent and the goal state is (3, 1). This means that the agent is fairly close to the predator. It will take the predator at least four timesteps to reach the goal state. After moving South for two consecutive timesteps, the agent can choose to move East or South in order to optimize the immediate reward. However, that does assume an optimal policy, while the predator has a random policy.

In case 3, the agent starts at (2,10) and the prey at (10,0). Using the properties of the toroidal grid, the shortest distance between the agent and the goal state is (3, 1), again. This shows that the distance between the agent and the goal state is exactly the same as in case 2.

Case 4 has the agent starting at (10,10) and the prey located at (0,0). The distance between the agent and the goal state is (1, 1). The predator is very close to the goal state. The difference between the values of the predator and the prey is quite small, especially compared with the previous cases.

Predator	Prey	Value	Discount Factor	Iterations to converge
(0, 0)	(5, 5)	0.00335	0.8	33
(2, 3)	(5, 4)	0.19892	0.8	33
(2, 10)	(10, 0)	0.19892	0.8	33
(10, 10)	(0, 0)	1.65066	0.8	33

The table above proves that policy evaluation to converge always takes equally long for the same size of the grid. This is to be expected, as the size of the grid has not changed.

Discount Factor	Iterations to converge
0.1	5
0.5	13
0.7	22
0.9	64

The discount factor appears to affect the number of iterations necessary to converge. This makes sense as the discount factor discounts the value of a state. A small discount value discounts the value of the state quite radically, leading to quick conversion. However, this quick conversion leaves many states with a

random policy. This is most likely undesired. Using a higher discount value leads to more iterations before convergence. With a less radical discount, policy evaluation can be optimized in such a way that every state has a value. The discount factor should, however, not be too large. This will lead to faster convergence. Do note that in order to reach convergence, the discount factor must lie between 0-1.

Policy iteration

Policy iteration is a stationary algorithm to find the optimal policy. After performing policy evaluation, the policy is updated. This is repeated until the policy is optimized. The table below shows the results for policy iteration with the prey located at (5,5).

Policy Iteration Grid in loop 3, discount 0.8											
Indices y\x	0	1	2	3	4	5	6	7	8	9	10
0	3.7281 ES	4.6602 ES	5.8252 ES	7.2816 ES	9.1020 ES	11.3776 S	9.1020 WS	7.2816 WS	5.8252 WS	4.6602 WS	3.7281 WS
1	4.6602 ES	5.8252 ES	7.2816 ES	9.1020 ES	11.3776 ES	14.2220 S	11.3776 WS	9.1020 WS	7.2816 WS	5.8252 WS	4.6602 WS
2	5.8252 ES	7.2816 ES	9.1020 ES	11.3776 ES	14.2220 ES	17.7776 S	14.2220 WS	11.3776 WS	9.1020 WS	7.2816 WS	5.8252 WS
3	7.2816 ES	9.1020 ES	11.3776 ES	14.2220 ES	17.7776 ES	22.2220 S	17.7776 WS	14.2220 WS	11.3776 WS	9.1020 WS	7.2816 WS
4	9.1020 ES	11.3776 ES	14.2220 ES	17.7776 ES	22.2220 ES	27.7776 S	22.2220 WS	17.7776 WS	14.2220 WS	11.3776 WS	9.1020 WS
5	11.3776 E	14.2220 E	17.7776 E	22.2220 E	27.7776 E	22.2220 WENS	27.7776 W	22.2220 W	17.7776 W	14.2220 W	11.3776 W
6	9.1020 EN	11.3776 EN	14.2220 EN	17.7776 EN	22.2220 EN	27.7776 N	22.2220 WN	17.7776 WN	14.2220 WN	11.3776 WN	9.1020 WN
7	7.2816 EN	9.1020 EN	11.3776 EN	14.2220 EN	17.7776 EN	22.2220 N	17.7776 WN	14.2220 WN	11.3776 WN	9.1020 WN	7.2816 WN
8	5.8252 EN	7.2816 EN	9.1020 EN	11.3776 EN	14.2220 EN	17.7776 N	14.2220 WN	11.3776 WN	9.1020 WN	7.2816 WN	5.8252 WN
9	4.6602 EN	5.8252 EN	7.2816 EN	9.1020 EN	11.3776 EN	14.2220 N	11.3776 WN	9.1020 WN	7.2816 WN	5.8252 WN	4.6602 WN
10	3.7281 EN	4.6602 EN	5.8252 EN	7.2816 EN	9.1020 EN	11.3776 N	9.1020 WN	7.2816 WN	5.8252 WN	4.6602 WN	3.7281 WN

As shown in the table above, the states surrounding the goal state have higher values than the ones around it. Also, the policy shows which actions are optimal. Policy iteration shows that a state can make multiple optimal transitions. The optimal transitions all have the same probability of being chosen, while all other transition probabilities are set to zero. In the cases where there are multiple optimal actions, one of the actions was selected. This creates an optimal, deterministic policy.

Predator	Prey	Value	Discount Factor	Iterations to converge
(0, 0)	(5, 5)	0.00335	0.8	2
(2, 3)	(5, 4)	0.19892	0.8	2
(2, 10)	(10, 0)	0.19892	0.8	2
(10, 10)	(0, 0)	1.65066	0.8	2

It takes a few iterations to converge. These are always the same. This can be expected as policy evaluation always takes the same amount of time and policy improvement is applied to the entirety of the grid. This means that each computation is essentially the same and should take the same amount of time

Discount Factor	Iterations to converge
0.1	6
0.5	7
0.7	7
0.9	7

I have no idea what to say here, because I cannot test this, right now.

Value iteration

Prey is located at (5, 5)

Value Iteration Grid in loop 8											
Indices y\x	0	1	2	3	4	5	6	7	8	9	10
0	0.000000	0.000000	0.000027	0.000168	0.001049	0.006554	0.001049	0.000168	0.000027	0.000000	0.000000
1	0.000000	0.000027	0.000168	0.001049	0.006554	0.040960	0.006554	0.001049	0.000168	0.000027	0.000000
2	0.000027	0.000168	0.001049	0.006554	0.040960	0.256000	0.040960	0.006554	0.001049	0.000168	0.000027
3	0.000168	0.001049	0.006554	0.040960	0.256000	1.600000	0.256000	0.040960	0.006554	0.001049	0.000168
4	0.001049	0.006554	0.040960	0.256000	1.600000	10.000000	1.600000	0.256000	0.040960	0.006554	0.001049
5	0.006554	0.040960	0.256000	1.600000	10.000000	0.000000	10.000000	1.600000	0.256000	0.040960	0.006554
6	0.001049	0.006554	0.040960	0.256000	1.600000	10.000000	1.600000	0.256000	0.040960	0.006554	0.001049
7	0.000168	0.001049	0.006554	0.040960	0.256000	1.600000	0.256000	0.040960	0.006554	0.001049	0.000168
8	0.000027	0.000168	0.001049	0.006554	0.040960	0.256000	0.040960	0.006554	0.001049	0.000168	0.000027
9	0.000000	0.000027	0.000168	0.001049	0.006554	0.040960	0.006554	0.001049	0.000168	0.000027	0.000000
10	0.000000	0.000000	0.000027	0.000168	0.001049	0.006554	0.001049	0.000168	0.000027	0.000000	0.000000

Discount Factor	Iterations to converge
0.1	1
0.5	7
0.7	7
0.9	8

Smarter state-space encoding

Conclusion

Performing policy evaluation gives detailed insight in the which actions the agent can take and how this affects reaching the goal state. In order to reach the goal state as quickly as possible, the policy of the agent needs to be adjusted. In order to update the policy of the agent to an optimal policy, one of two algorithms can be used: policy improvement or value iteration. The name policy improvement sounds very straightforward and effective. It is both. However, value iteration is faster, due to performing policy evaluation and policy improvement at the same time rather than sequentially. In the end, both yield the same results.

The discount factors of these algorithms have also been evaluated. It has shown that a low discount factor (0.1) leads to quick convergence, but is quite radical. Many states in the state space will contain random policies. As this is not the goal of planning algorithms, this is undesired. Too high a discount rate (0.9) will lead to slow convergence, but each state will be evaluated and have optimal actions to take. This does take longer than often is necessary. This leads to the conclusion that, in this case, a learning rate of about 0.7 or 0.8 is optimal.

Files attached

Sources