Autonomous Agents 1
Assignment 1

By Group 4: Gieske, Gornishka, Koster, Loor

November 15, 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 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:

Agent	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 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. The implementation the following algorithm as described in Barto and Sutton [1]:

Input π, 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.
- $v(s')$ is the value of the new state.
- γ is a discount factor. ...

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 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 [1]:

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

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 [1]
Repeat
$\Delta \leftarrow 0$
For each s $\in$ S:
   $temp \leftarrow v(s)$
   $v(s) \leftarrow \max_a \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma v(s')]$
   $\Delta \leftarrow max(temp, |v(s)|)$
until $\Delta \leftarrow \theta$ (a small positive number)
Output a deterministic policy π, 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 be used for agent planning.

Implementation

The current implementation consists of the following files:

Agents
: This file contains implementions of the agents, the prey class and the predator class. Both agents have a policy and actions as described in the introduction. Both agents contain functions to retrieve information from the agent as well as functions to change information. 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 part of 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

Average run time	Standard deviation
291	302.93

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	Predator	Prey	Value	Discount Factor	Iterations to converge
1	(0, 0)	(5, 5)	0.0049406	0.8	24
2	(2, 3)	(5, 4)	0.1766713	0.8	24
3	(2, 10)	(10, 0)	0.1766713	0.8	24
4	(10, 10)	(0, 0)	1.1605528	0.8	24

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, nor the discount factor has not changed.

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.

Discount Factor	Iterations to converge
0.1	3
0.5	8
0.7	15
0.8	24
0.9	55

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 an algorithm for finding 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 - Value Grid in loop 7											
Indices y\x	0	1	2	3	4	5	6	7	8	9	10
0	3.710664	4.638330	5.807816	7.259770	9.084616	11.355770	9.084616	7.259770	5.807816	4.638330	3.710664
1	4.638330	5.807816	7.259770	9.084616	11.355770	14.204616	11.355770	9.084616	7.259770	5.807816	4.638330
2	5.807816	7.259770	9.084616	11.355770	14.204616	17.755770	14.204616	11.355770	9.084616	7.259770	5.807816
3	7.259770	9.084616	11.355770	14.204616	17.755770	22.204616	17.755770	14.204616	11.355770	9.084616	7.259770
4	9.084616	11.355770	14.204616	17.755770	22.204616	27.755770	22.204616	17.755770	14.204616	11.355770	9.084616
5	11.355770	14.204616	17.755770	22.204616	27.755770	22.204616	27.755770	22.204616	17.755770	14.204616	11.355770
6	9.084616	11.355770	14.204616	17.755770	22.204616	27.755770	22.204616	17.755770	14.204616	11.355770	9.084616
7	7.259770	9.084616	11.355770	14.204616	17.755770	22.204616	17.755770	14.204616	11.355770	9.084616	7.259770
8	5.807816	7.259770	9.084616	11.355770	14.204616	17.755770	14.204616	11.355770	9.084616	7.259770	5.807816
9	4.638330	5.807816	7.259770	9.084616	11.355770	14.204616	11.355770	9.084616	7.259770	5.807816	4.638330
10	3.710664	4.638330	5.807816	7.259770	9.084616	11.355770	9.084616	7.259770	5.807816	4.638330	3.710664

Policy Iteration Grid in loop 7											
Indices y\x	0	1	2	3	4	5	6	7	8	9	10
0	>	>	>	>	>	v	<	<	<	<	<
1	>	>	>	>	>	v	<	<	<	<	<
2	>	>	>	>	>	v	<	<	<	<	<
3	>	>	>	>	>	v	<	<	<	<	<
4	>	>	>	>	>	v	<	<	<	<	<
5	>	>	>	>	>	X	<	<	<	<	<
6	>	>	>	>	>	^	<	<	<	<	<
7	>	>	>	>	>	^	<	<	<	<	<
8	>	>	>	>	>	^	<	<	<	<	<
9	>	>	>	>	>	^	<	<	<	<	<
10	>	>	>	>	>	^	<	<	<	<	<

The table shown above depicts the optimal policy, where the prey is located at (5,5). Policy iteration can assign multiple optimal transitions to a state. 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.

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
0.8	7
0.9	7

There is a slight difference in number of iterations when performing policy iteration. Though convergence of policy evaluation is relatively quick with a low discount factor, this appears to hardly affect policy improvement.

Value iteration

Prey is located at (5, 5)

v in loop 16											
Indices y\x	0	1	2	3	4	5	6	7	8	9	10
0	1.320840	1.629618	2.009615	2.476236	3.060473	3.584043	3.060473	2.476236	2.009615	1.629618	1.320840
1	1.629618	1.987387	2.471198	3.074265	3.817584	4.538915	3.817584	3.074265	2.471198	1.987387	1.629618
2	2.009615	2.471198	3.074265	3.824652	4.756831	5.759223	4.756831	3.824652	3.074265	2.471198	2.009615
3	2.476236	3.074265	3.824652	4.756831	5.928854	7.272727	5.928854	4.756831	3.824652	3.074265	2.476236
4	3.060473	3.817584	4.756831	5.928854	7.272727	10.000000	7.272727	5.928854	4.756831	3.817584	3.060473
5	3.584043	4.538915	5.759223	7.272727	10.000000	0.000000	10.000000	7.272727	5.759223	4.538915	3.584043
6	3.060473	3.817584	4.756831	5.928854	7.272727	10.000000	7.272727	5.928854	4.756831	3.817584	3.060473
7	2.476236	3.074265	3.824652	4.756831	5.928854	7.272727	5.928854	4.756831	3.824652	3.074265	2.476236
8	2.009615	2.471198	3.074265	3.824652	4.756831	5.759223	4.756831	3.824652	3.074265	2.471198	2.009615
9	1.629618	1.987387	2.471198	3.074265	3.817584	4.538915	3.817584	3.074265	2.471198	1.987387	1.629618
10	1.320840	1.629618	2.009615	2.476236	3.060473	3.584043	3.060473	2.476236	2.009615	1.629618	1.320840

This shows that value iteration has the exact same results as policy iteration. Also, we see that the algorithm converges quicker than policy iteration.

Discount Factor	Iterations to converge
0.1	4
0.5	12
0.7	15
0.8	16
0.9	17

For states with the same Manhattan distance, the diagonal states have a higher value. This happens, due to the fact that the prey is twice as likely to move toward the predator instead of away from it. When observing the distance between the predator and the prey in a straight line, the prey has a 0.05×3 probability of moving away from the predator. However, when taking into account the same distance diagonally, the prey only has a probability of 0.05×2 of moving away.

Smarter state-space encoding

State space encoding was implemented to improve performance. By taking into account the distance between the predator and the prey, the valus of less states need to be computed compared to the stated of the entire grid. The state space is reduced from 11^4 to 11^2, by calculating distances and only updating states whose distance has not been updated yet. This greatly reduces the state space and thus increases the speed of the algorithms

Case	Algorithm	γ	Time to converge	Avg. # steps	Standart deviation
1	Value Iteration	0.8	189 s	20	14.10
2	Encoded Value Iteration	0.8	12 s	19	13.57
3	Policy Iteration	0.8			
4	Encoded Policy Iteration	0.8			
5	Policy Evaluation	0.8			
6	Encoded Policy Evaluation	0.8			

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 algortihms 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.

Smarter state-space encoding does indeed improve performance.

Files attached

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

Sources

1 Barto and Sutton (http://webdocs.cs.ualberta.ca/ sutton/book/the-book.html) ...