Single Agent Planning

Duy Chuan Ha, Said Al Faraby, Christos Louizos, Oana Munteanu

September 20, 2013

Abstract

The aim of this assignment is to make a research according to the planning scenario in which the main goal of the predator is to capture the prey.
The report comprises the methods that have been implemented by using the Dynamic Programming paradigm, namely policy evaluation, policy iteration and value iteration. Our contribution on Single Agent Planning reveals the comparison between the implemented methods in terms of convergence time and results.
+Conclusion

CONTENTS

1 INTRODUCTION

We begin the following report by first explaining the predator-prey environment and defining it's goal in section **??**. Afterwards we present in section **??** the various methods that were implemented in the afforementioned environment. The results that derived from each of these methods are displayed in detail in section **??** and finally there is a discussion as well as a comparison between them in section **??**

1.1 ENVIRONMENT

The environment is encoded as a state of two positions, the one of the predator and the prey. The simulator is defined through a while-loop of a hundred runs where the initial position of the predator is (0,0) and that of the prey (5,5). In each transition, the predator will make a move first and then the prey, according to their respective policies. Each run can be "seen" as an episode where the goal of the predator is to capture the prey, which can be interpreted as the end state of this particular episode.

Both the predator and the prey are initialized with a random policy in the beginning where for the predator it is equiprobable to choose any of the possible actions in any state. The prey has a steady policy and hence it can be modeled as part of the environment. It has 0.8 probability of waiting at any given state and 0.2 probability of moving to any of the adjacent squares.

2 METHODS AND PROCEDURES

2.1 IMPLEMENTATION OVERVIEW

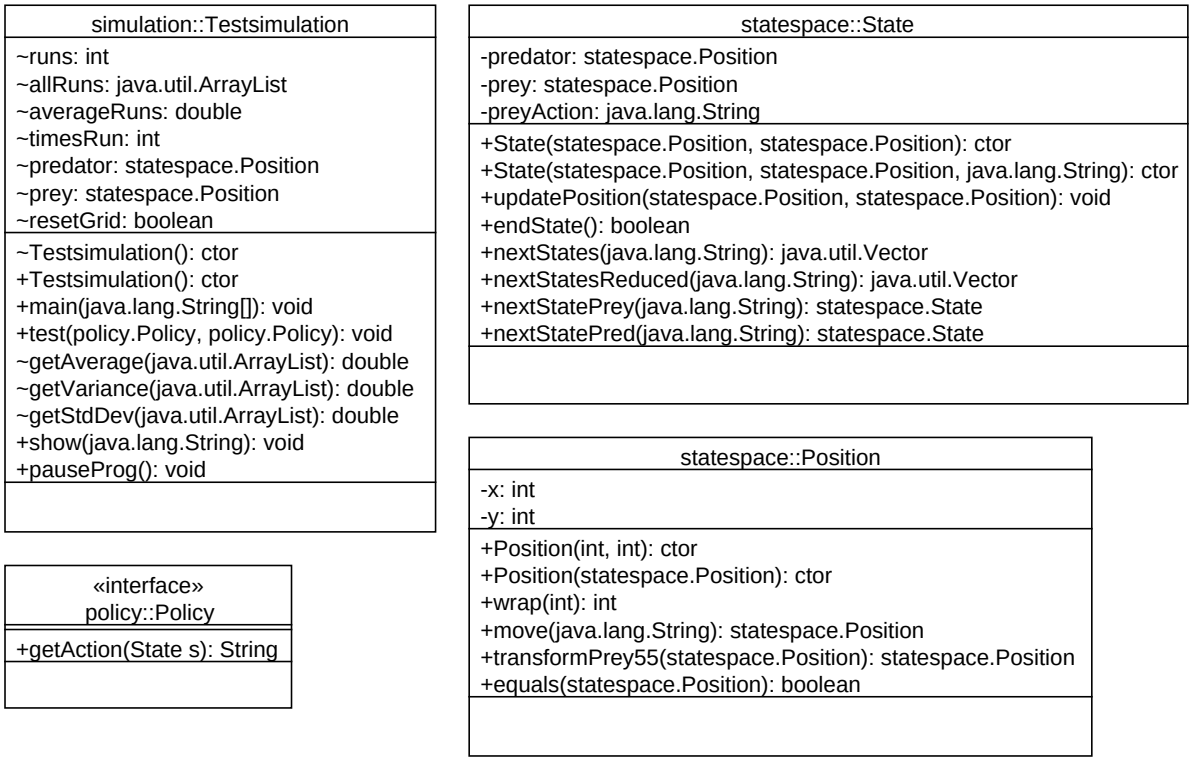

Figure 2.1: Predator-prey domain framework

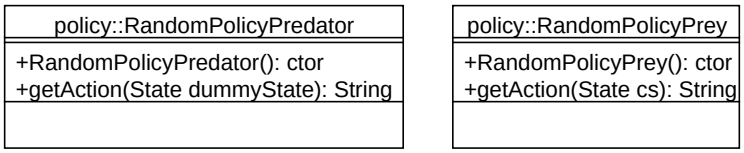

Figure 2.2: Random policies implemented

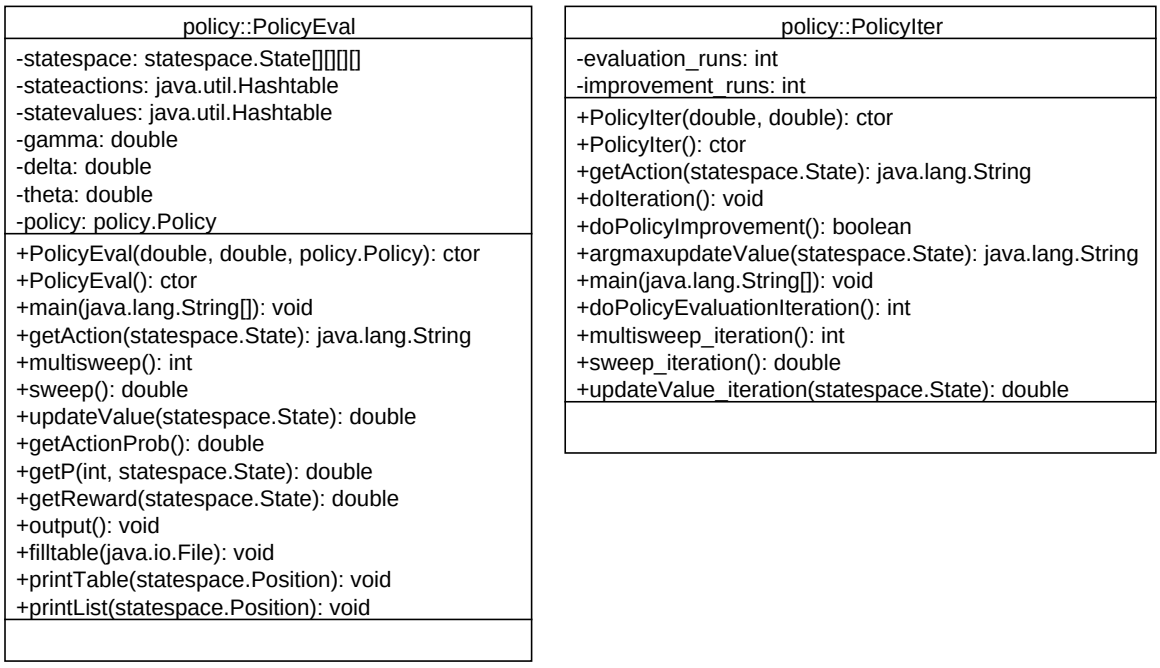

Figure 2.3: Policy evaluation and iteration implemented

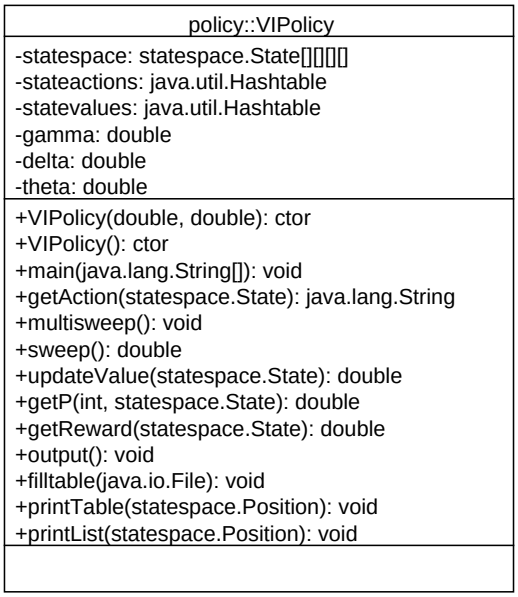

Figure 2.4: Policy value iteration implemented

2.2 Iterative Policy Evaluation

In Marcov Decision Processes, one of the important steps is evaluation step, which is to find the state-value functions V^π for an arbitrary policy, π. The state-value functions estimate value of a state in terms of expected return that can be obtained from that state. Below is the equation for calculating V^π.

$$\begin{aligned} V^\pi(s) &= E_\pi \left\{ r_{t+1} + \gamma V^\pi(s_{s+1}) \mid s_t = s \right\} \\ &= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^\pi(s')] \end{aligned} \tag{2.1}$$

where $\pi(\mathbf{s},\mathbf{a})$ is probability of taking action **a** in state **s** under policy π, $P^a_{ss'}$ is probability of end up at state $\mathbf{s}'$ after taking action **a** in state **s**, $R^a_{ss'}$ is expected immediate reward on transition from **s** to $\mathbf{s}'$ under action **a**, and γ is a discount factor.

Since in our test environment the dinamics are completely known, then equation (**??**) can be solved using iterative solution methods. By iterative solution methods, we can consider there is a sequence of V, $V_0, V_1, V_2, \ldots$, each mapping all states in S^+ to $\Re$. The initial values of V^π are chosen arbitrarily, and the update rule for each successive state-values are gained by following the Bellman equation :

$$\begin{aligned} V_{k+1}(s) &= E_\pi \left\{ r_{t+1} + \gamma V_k(s_{t+1} \mid s_t = s) \right\} \\ &= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V_k(s')] \end{aligned} \tag{2.2}$$

For the technical implementation, we used one array to save state-values and updated the values in place. This implementation is slightly different from (**??**), because there is a possibility that $V_k(s')$ uses new values instead of old ones, but this implementation still guarantees to converge to V^π. Another technical implementation to be considered is the stopping criteria. In formal, iterative policy iteration will converge only in the limit, but for practical purposes stopping condition is necessary. One of common criteria to stop the sweep is when there is no any single state managed to make a significant changes to their values. A complete pseudocode of the whole processes of the iterative policy evaluation is shown in figure **??**.

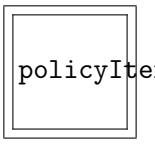

Figure 2.5: Iterative Policy Evaluation

2.3 Policy Iteration

Policy iteration is composed by two steps, the policy evaluation, which was explained above, and the policy improvement. The main reason for calculating the value function for a policy is to use it in order to find better policies. Consequently if we have determined the value function V^π for a deterministic policy π we can then select a different action α for one state s and

afterwards keep on using that specific action for that state, and then continue following the existing policy. Therefore we can determine if the change improved the policy by calculating the following formula:

$$\begin{aligned} Q^{\pi}(s,\alpha) &= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s, \alpha_t = \alpha\} \\ &= \sum_{s'} P^{\alpha}_{ss'}[R^{\alpha}_{ss'} + \gamma V^{\pi}(s')] \end{aligned} \tag{2.3}$$

if this is greater than the value of the existing policy in that state $V^{\pi}(s)$ then it would be better to continue taking action α everytime we encounter state s, which results in a new policy π'. This is true because it derives from the policy improvement theorem where if we have two deterministic policies π and π' such that for all states $s \in \mathbf{S}$ the $Q^{\pi}(s,\pi'(s)) >= V^{\pi}(s)$ then the policy π' must be equal to or better than policy π meaning that it will yield equal or higher values for all state s. Thus if $Q^{\pi}(s,\alpha) > V^{\pi}(s)$ then the changed policy π' is indeed better than π. There is one more step needed in order to finish the policy improvement. Since we can acquire a better reward by doing a different action than the one the policy provide at a given state, then it makes sense to choose the action that maximizes that reward and incorporate it to the new policy π'. Then the formula resulting from this step is:

$$\begin{aligned} \pi'(s) &= arg\max_{\alpha} Q^{\pi}(s,\alpha) \\ &= arg\max_{\alpha} Er_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s, \alpha_t = \alpha \\ &= arg\max_{\alpha} \sum_{s'} P_{ss'}\alpha[R^{\alpha}_{ss'} + \gamma V^{\pi}(s')] \end{aligned} \tag{2.4}$$

where $arg\max_{\alpha}$ denotes the action that maximizes the term that is following. Taking all the above into consideration we can say that the process of making a new policy that improves the existing one by being greedy on the value function of the original policy, is called policy improvement.

The whole pseudocode loop of iterative evaluation of the current policy and improvement of the current policy in order to produce a better policy π' untill we converge to the optimal policy is illustrated in figure **??**.

Figure 2.6: Policy Iteration

We begin by initializing the values and actions for the policy arbitrarily and afterwards we perform policy evaluation untill the difference in the values are smaller than a certain small threshold θ. After this step we begin the policy improvement step where we find the best possible action for each possible state, with respect to maximizing the value. If the resulting actions are the same as the current policy then we have reached the optimal policy, otherwise we perform policy evaluation again on the new policy and keep on the same procedure untill we converge to the optimal one.

2.4 VALUE ITERATION

Value iteration extends on Policy iteration in which the policy evaluation step of policy evaluation is stopped after just one sweep, ie there's one backup of each state. It consists of a backup operation that combines the policy improvement and truncated policy evaluation steps:

$$\begin{aligned} V_{k+1}(s) &= \max_a Er_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a \\ &= \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')] \end{aligned} \tag{2.5}$$

From this we can see it's identical to the Policy evaluation except that it requires the maximum to be taken over all actions. The algorithm, as previous algorithms converge to an optimal policy for discounted finite MPDs. Formally, convergent to V^* is only reached when the number of iterations is taken to infinity. In practice we consider θ as the termination condition of the loop, which keeps track of the change in value funtion. The pseudocode for value iteration is shown in the following figure:

sutton4_5valueiteration.png

Figure 2.7: Value Iteration

As in our implementation of policy evaluation and policy iteration, the backup operation is done "in-place", thus faster convergent is reached. The rate of convergent hereby is very much dependant on the order in which states are backed up during the sweep. In our implementation the order is straightforward; it's aligned with the array[i][j], starting with the x-coordinate x_i. From the second row forward the benefit hereby is valued at most $3V_k$, leaving at minimum $2V_{k-1}$ still containing the old values, where k denotes the current sweep.

2.5 REDUCING THE STATE SPACE

3 RESULTS

3.1 RANDOM POLICY

Table 3.1: Random policy results

For 100 runs	Summary
Average time to capture the prey	336.78
Standard deviation	285.66324859876534

3.2 Iterative Policy Evaluation

The goal of the experiment in this algorithm was to evaluate the arbitrary policy, and to measure the number of iteration it took to converge. There were two parameters should be defined, discount factor γ and a threshold value for stopping condition θ. The values of the parameters along with some results are shown in the following table. Runtime is time that

Table 3.2: Policy evaluation results

	Summary
γ	0.8
θ	1.0E-20
runtime	12248712457 ns
Number of iterations	106

the algorithm took to converge, and it was measured in nanosecond. While the number of iterations is the number of sweep (sweep is backup operation for all states) until it converged.

Table 3.3: State values for the following states

Predator	Prey	$V(s)$
(0,0)	(5,5)	0.005724141401102873
(2,3)	(5,4)	0.18195076385152237
(2,10)	(10,10)	0.18195076385152237
(10,10)	(10,10)	1.1945854778368172

reduced nr iteration = 19 Time : 179385026

3.3 Policy Iteration

For all the experiments with policy Iterations theta was defined heuristically as $\theta = \exp(-20)$.

Table 3.4: Convergence in iterations for different γ

γ	Evaluation runs	Improvement runs
0.1	100	8
0.5	227	7
0.7	323	8
0.9	763	10

The values for for every possible state when the prey is at [5][5] are illustrated in table **??**

3.4 Value Iteration

The values for for every possible state when the prey is at [5][5] are illustrated in table **??**

Table 3.5: Values from policy iteration when the prey is at [5][5]

3.883	4.291	4.742	5.237	5.792	6.251	5.792	5.237	4.742	4.291	3.883
4.291	4.712	5.228	5.802	6.436	6.997	6.436	5.802	5.228	4.712	4.291
4.742	5.228	5.802	6.440	7.148	7.839	7.148	6.440	5.802	5.228	4.742
5.237	5.802	6.440	7.148	7.936	8.780	7.936	7.148	6.440	5.802	5.237
5.792	6.436	7.148	7.936	8.780	10.000	8.780	7.936	7.148	6.436	5.792
6.251	6.997	7.839	8.780	10.000	0.000	10.000	8.780	7.839	6.997	6.251
5.792	6.436	7.148	7.936	8.780	10.000	8.780	7.936	7.148	6.436	5.792
5.237	5.802	6.440	7.148	7.936	8.780	7.936	7.148	6.440	5.802	5.237
4.742	5.228	5.802	6.440	7.148	7.839	7.148	6.440	5.802	5.228	4.742
4.291	4.712	5.228	5.802	6.436	6.997	6.436	5.802	5.228	4.712	4.291
3.883	4.291	4.742	5.237	5.792	6.251	5.792	5.237	4.742	4.291	3.883

Table 3.6: Convergence in iterations for different γ

γ	Nr. of iterations
0.1	18
0.5	25
0.7	27
0.9	29

4 Discussion

References

[1] Richard S. Sutton, Andrew G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, A Bradford Book, 1998.

Table 3.7: Values from value iteration when the prey is at [5][5]

3.883	4.291	4.742	5.237	5.792	6.251	5.792	5.237	4.742	4.291	3.883
4.291	4.712	5.228	5.802	6.436	6.997	6.436	5.802	5.228	4.712	4.291
4.742	5.228	5.802	6.440	7.148	7.839	7.148	6.440	5.802	5.228	4.742
5.237	5.802	6.440	7.148	7.936	8.780	7.936	7.148	6.440	5.802	5.237
5.792	6.436	7.148	7.936	8.780	10.000	8.780	7.936	7.148	6.436	5.792
6.251	6.997	7.839	8.780	10.000	0.000	10.000	8.780	7.839	6.997	6.251
5.792	6.436	7.148	7.936	8.780	10.000	8.780	7.936	7.148	6.436	5.792
5.237	5.802	6.440	7.148	7.936	8.780	7.936	7.148	6.440	5.802	5.237
4.742	5.228	5.802	6.440	7.148	7.839	7.148	6.440	5.802	5.228	4.742
4.291	4.712	5.228	5.802	6.436	6.997	6.436	5.802	5.228	4.712	4.291
3.883	4.291	4.742	5.237	5.792	6.251	5.792	5.237	4.742	4.291	3.883