Autonomous Agents 1 – Assignment 1

R. van der Heijden, F. Xumara, A. Qodari

November 14, 2014

1 Introduction

In this report we describe our implementation of Single Agent Planning implemented in an 11×11 grid world MDP. In this case, we model one predator as a single agent and one prey as a part of the environment.

We developed a web-based application in this project. The core functions are written in Javascript and a single HTML page illustrates the environment. The main reason of using web-based application is so that we can easily show each action taken by our agent in a graphical user interface.

This report is organized as follows. Section 2 describes our implementation of the predator-prey environment environment. Section 3 details the results of our simulations when using a random policy for the predator. In Section 4, we re-design our state-space encoding in order to reduce computation complexity. Section 5, 6, and 5 describe our implementation of Policy Evaluation, Policy Iteration, and Value Iteration respectively. At the end, we explain some remarks that can be concluded based in our experiments.

In addition, we also include code snippets, experiment design, and graphs represent our results.

2 Environment

In this section we describe how the environment behaves. We first describe the dimensions of the grid world, next what actions and policies are and finally we detail the reward structure.

2.1 Simple state-space

In this problem, the predator and prey move through a 2-dimensional toroidal world (Figure 1). It is a discrete world of 11 rows and 11 columns. There are therefore 11×11 possible positions for the predator or prey to be in. The combined positions of the predator and prey defines the state s of the world.

To ensure that the world is toroidal, we implemented a function that makes sure every movement of both the predator and the prey are valid movement. When an agent tries to move outside the grid, this function positions it instead at the opposite side.

```
// toroidal world
if (finalState.x < 0) {
  finalState.x = world.size - 1;
}

if (finalState.x >= world.size) {
  finalState.x = 0;
}

if (finalState.y < 0) {
  finalState.y = world.size - 1;
}

if (finalState.y >= world.size) {
  finalState.y = 0;
}
```

Another constraint for the prey is that it can not move to the predator's position (suicide) when they are adjacent.

```
// check suicide action (if forbiddenState exists)
if (options && options.forbiddenState) {
  for (var i = 0; i < options.forbiddenState.length; i++) {
    if (isSamePositions(finalState, options.forbiddenState[i])) {
      return false;
    }
  }
}
```

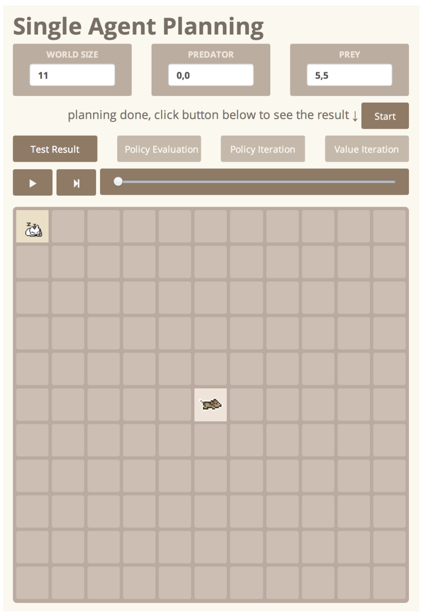

Figure 1: 11×11 Grid World MDP with one predator and one prey

2.2 Policies and Actions

At each time step, both the predator and the prey take an action, according to its policy. A policy π associates for a given state s every action a with a probability $\pi(a, s) \in [0, 1]$. An action is an element of the set $\{\text{wait}, \text{up}, \text{down}, \text{left}, \text{right}\}$. When an agent performs the action wait, it remains in its current position; any other action move the agent in the corresponding direction. Note that not all actions are possible in every state: the prey is not allowed to move onto the spot occupied by the predator. The set of possible moves for the prey in a state s is denoted by $\mathcal{S}$.

For this problem we have used the policy of the prey defined by:

$$\pi_{prey}(a, s) = \begin{cases} 0.8 & \text{if } a = \text{wait} \\ \frac{1-0.8}{|\mathcal{A}(s)|-1} & \text{if } a \in \mathcal{A}(s) \\ 0 & \text{otherwise.} \end{cases}$$

For example when the location of predator is $(5, 4)$ and location of the prey is $(5, 5)$, the possible moves of prey are going up to $(4, 5)$, going down to $(6, 5)$, going right to $(5, 6)$ and stay at $(5, 5)$. Going left is not a possible move. The size of $\mathcal{A}(s)$ is therefore 4, and the probability to move up, down, and right is equal to 0.067.

The policy of the predator will be determined by the algorithms we implemented.

2.3 Reward

Each action returns an immediate reward of either 0 or 10. Only when the predator catches the prey, it is rewarded with 10. For all other actions, the reward is 0. This reward is calculated by the following function:

```
// reward function
var rewardFunction = function (predatorState) {
  for (var i = 0; i < world.preys.length; i++) {
    if (isSamePositions(predatorState, world.preys[i].state)) {
      return world.maxReward;
    }
  }

  return 0;
};
```

3 Random policy

As a reference for the other algorithms, we first examine a policy for the predator where it takes a random action at each time step: $\forall a, s : \pi_{\text{predator}}(a, s) = 0.2$. The initial position of the predator and prey are set at $(0, 0)$ and $(5, 5)$ respectively. We run our simulator 100 times and measure the time needed for the predator to catch the prey.

3.1 Result

Figure 2 illustrates our results after running the program 100 times. Each datapoint represents the number of iterations needed for the predator to catch the prey. Note that the datapoints are centered to the mean. We got an average time needed for the predator to catch the prey is **252.01**, with standard deviation **234.08**. Notice that the standard deviation is very large compared to the average. This is probably due to the random nature of the policy of the predator: there is no intelligence involved whatsoever.

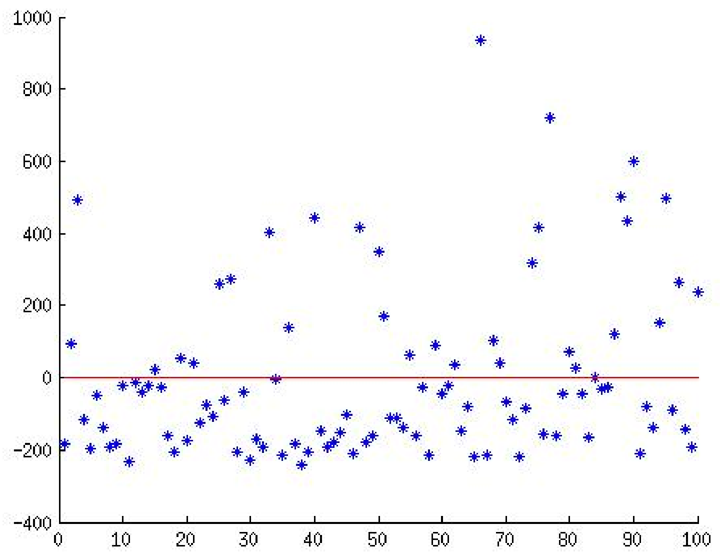

Figure 2: Simulation Result. Each datapoint is centered to the mean.

4 State-space Encoding

In this section we describe a way to decrease the number of states.

In the original problem, a state was characterized by the positions of the predator and the prey. Since the world is an 11×11 grid, both of these agents have $11^2 = 121$ possible positions to be in, which results in a total of $11^2 \cdot 11^2 = 14641$ different states. Because the algorithms we use examine all states in every iteration, a decrease in the number of states could have a major effect on execution performance.

To decrease the number of states, we considered two different models.

4.1 State-space model based on Manhattan-distance

First, we considered encoding a state by the distance between the predator and the prey using the Manhattan-distance. Ideally, this would reduce the number of states to 6, because the predator and prey are always between 0 and 5 (inclusive) steps away from each other. However, this model does not work, since the transition function between states can not be uniquely described using only this information.

To show this, conside the states S_1 where predator and prey are in positions $(0, 0)$ and $(2, 2)$ respectively and S_2 where predator and prey are in positions $(0, 0)$ and $(4, 0)$ respectively. Note that in both S_1 and S_2 the (Manhattan) distance between predator and prey equals 4, and therefore in this model these states would be considered the same. Now suppose the predator waits in both states. In S_1 there are 2 moves (up and left) for the prey to decrease the distance and 2 moves (down and right) to increase the distance, whereas in state S_2 the prey has 3 possible moves to increase the distance (up, down, right) and only 1 move (left) to decrease the distance. This means the transition functions are not equal and the states can therefore not be considered the same.

4.2 State-space model based on relative position

Because the model based on the distance between the predator and prey is unsuitable, we looked at a state-space modelbased on the relative position of the prey to the predator. This means that if the predator is at position (x_1, y_1) and the prey is at position (x_2, y_2), then the encoded state $(\hat{x}, \hat{y})$ becomes

$$\begin{aligned} \hat{x} &= x_1 - x_2 + a, \\ \hat{y} &= y_1 - y_2 + b, \end{aligned} \tag{1}$$

where $a, b \in \{11, 0, -11\}$ such that $\hat{x}, \hat{y} \in \{-5, -4, \ldots, 4, 5\}$ is satisfied. The resulting state-space contains only $11^2 = 121$ states instead of the old 14641.

This model can be interpreted as if the predator is always at position $(0, 0)$ and it's actions move the prey closer to or farther away from it. This interpretation works because the world is toroidal.

5 Policy Evaluation

Policy Evaluation is an iterative algorithm that approximates the score of every state under a given policy. The algorithm is detailed in Sutton & Barto [1], chapter 4.1. In this case we examined the random policy as described Section 3.

5.1 Experiment and Result

We ran Policy Evaluation on this problem to approximate the scores of the random policy. Below are the parameter settings used in this experiment:

- `theta` = 0.001
- `gamma` (discount factor) = 0.8

Our implementation needs 23 iterations to solve all possible states in the reduced state-space. Table 1 shows 4 different state values. It can be seen that the closer predator to the prey, the higher state value we get. This matches our expectations of the algorithm.

State	Value
Predator(0,0), Prey(5,5)	0.093
Predator(2,3), Prey(5,4)	1.384
Predator(2,10), Prey(10,0)	1.384
Predator(10,10), Prey(0,0)	6.742

Table 1: Experiment Result of Policy Evaluation

6 Policy Iteration

Policy Iteration is an algorithm that alternates between evaluating a given policy and improving the policy. It will continue alternating until the policy does not change anymore. We implemented policy iteration according to Sutton & Barto [1] chapter 4.3.

6.1 Experiment and Result

Below are the value of parameters used in our experiment:

- `theta` = 0.001
- `gamma` (discount factor) $\in \{0.1, 0.5, 0.7, 0.9\}$
- Initial position of predator = (0,0)
- Initial position of prey = (5,5)

Table 2 illustrates our experiment result for 4 different values of discount factor (γ).

Discount factor	Number of iterations
0.1	10
0.5	10
0.7	7
0.9	6

Table 2: Experiment Result of Policy Iteration

7 Value Iteration

While Policy Evaluation and Policy Iteration require a policy as part of the input, Value Iteration does not. Instead it takes all actions into account in every state. We followed the algorithm as described by Sutton & Barto [1] in chapter 4.3.

7.1 Experiment

Below are the values of the parameters we used in our experiment:

- `theta` $= 0.001$
- `gamma` (discount factor) $\in \{0.1, 0.5, 0.7, 0.9\}$
- Initial position of predator $= (0, 0)$
- Initial position of prey $= (5, 5)$

7.2 Result

Table 3 shows our experiment result for 4 differect values of discount factor (γ). The higher value of discount factor needs more iterations to cenverge. This is make sense that with high value of discount factor ($\gamma = 1$), the policy evaluation algorithm [1] always consider state value of the all possible next states. This makes the state value at current state will keep changes until the changes is very small.

Discount factor	Number of iterations
0.1	5
0.5	11
0.7	13
0.9	15

Table 3: Experiment Result of Value Iteration

Once we have result from this experiment, we can compare the result with our previous result from Section 4 (Table 4).

	Number of Iterations	
Discount factor	Policy Iteration	Value Iteration
0.1	10	5
0.5	10	11
0.7	7	13
0.9	6	15

Table 4: Comparison Result from Policy Iteration and Value Iteration

8 Conclusion

In this report we demonstrated that Policy Evaluation, Policy Iteration and Value Iteration are well suited for this problem. All of them perform much better than the random policy. Furthermore, we show that Value Iteration is more efficient with regards to computation complexity compared to Policy Iteration and Policy Evaluation. The discount factor also effects the number of iterations needed for convergence, even though the resulting policies are the same.

The reduction of the state-space greatly reduced the computation time needed for all algorithms. The use of this state-space based on the relative position can also be used for single agent learning algorithms.

References

[1] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.