Autonomous Agents
Report Assignment 2: Single Agent Learning

Authors:

Agnes van Belle *(10363130)*,
Maaike Fleuren *(10350470)*,
Norbert Heijne *(10357769)*,
Lydia Mennes *(10333843)*

October 4, 2012

1 Introduction

This report has been written for the Master Artificial Intelligence course Autonomous Agents. This report will contain the answers, motivation and explanation for our implementations of the tasks we had to accomplish in our second assignment for this course. These tasks were centered around the topic of 'Single Agent Learning'.

1.1 The environment

In all tasks there is assumed to be a grid world (of 11×11) with a predator and a prey in it. The agents can both move one tile forward each iteration. The direction they take (or if they move at all) is affected by probabilities (their policies). If they move over the edge of the grid they end up at the opposing side of the grid. The prey will never step onto the predator. We are focused on improving the decisions of one agent, the predator.

1.2 The state space representation

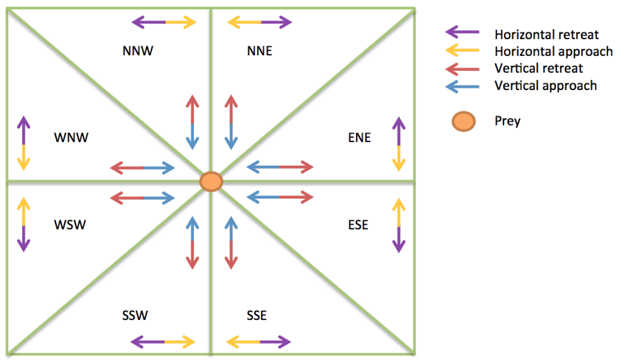
In the experiments described the first report, we initially used a state space that was an intuitive, yet cumbersome representation. We referred to that state space representation as the 'default' state space. The amount of states that was used in the default state space was $(11 \times 11) \times (11 \times 11) = 121 \times 121 = 14641$. We then changed the state space representation to a more efficient one, referred to as the 'efficient' state space, which led to a reduction of 697 times less states, resulting in just 21 different states.

In this assignment, we used this efficient state space representation for the learning algorithms. To give a good understanding of our learning algorithms, which were built on the efficient state space representation, we will once again explain how this representation works.

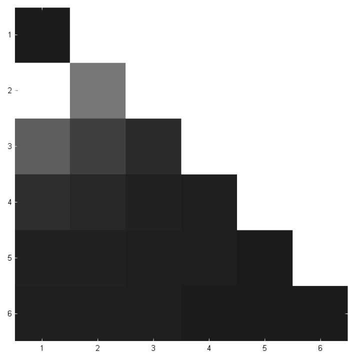
Figure **??** illustrates that there is a symmetry in the default state space, and thus that there were relatively much values redundantly computed. By using this symmetry in the default state space a much smaller state space was achieved.

Each state represents a distance between the prey and predator. These are represented in the lower left diagonal of a matrix, in which the x-axis is the relative horizontal distance in the MDP and the y-axis the relative vertical distance in the MDP. This matrix is shown in Figure **??**. Combinations of positions of prey and predator for which the horizontal and vertical distances are equal are now treated equivalent.

Also two combinations for which the horizontal distance in one equals the vertical distance in the other and vice versa are considered equal. In order to navigate through this state space different actions are required. These are: *horizontal retreat, horizontal approach, vertical retreat, vertical approach*, as illustrated in Figure **??**, and of course the action *wait*. When interacting with the environment these actions are converted into corresponding actions in the real world. This only requires the relative direction of the prey (which is always located at the centre, regardless of its coordinates) with respect to the predator. This is computed by using the difference in location of the prey and predator on the x- and y-axis.



(a) The 11×11 grid divided into eight symmetric pieces, with the corresponding possible moves which are also symmetric.



(b) Colormap of V-values, the brighter the color the higher the corresponding V-value. The prey is always located on the (1, 1) coordinate in this state representation.

Figure 1: Illustration of the symmetry and corresponding values of the new state space representation

1.3 Implementation details

This report will not be about our exact code and implementation details. However, a class diagram of our code is provided in Appendix **??**.

2 Learning algorithms

As mentioned before, we will use the same environment as in the previous assignment. But in this assignment, we will assume the learning scenerio: the agent does not know the transition probabilities, nor the reward structure. On a very high level there are two ways to come to a good solution in this setting: learning the model, and do planning again (model based learning), or not learn the model, and directly try to learn a high-reward policy (model-free learning). In this assignment we will focus on the latter.[1]

2.1 (M) Q-Learning

Q-Learning is an off-policy Temporal-Difference control algorithm. While the distinguishing feature of on-policy methods is that they estimate the value of a policy while using it for control, these two functions are separated in off-policy methods. The behavior policy, used to generate behavior, and the estimation policy, that is evaluated and improved, may in fact be unrelated. This separation is an advantage because the estimation policy may be deterministic (e.g., greedy), while the behavior policy can continue to sample

[1]Roijers (2012) *Assignments Autonomous Agents* p. 6

all possible actions.[2] Its simplest form, *one-step Q-learning*, is defined by[3]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

For this assignment, we implemented Q-learning and used it for our predator agent. We used ϵ-greedy action selection, which behaves greedily most of the time, but with probability ϵ, instead select an action at random, uniformly, independently of the action-value estimates.[4] In this case, we used $\epsilon = 0.1$. We initiated the values of our Q-learning table optimistically with a value of 15 for all cells in the table.

Figure **??** shows plots on the performance of the agent over time for different α and for different γ (discount factor).

2.2 (M) Experiment with ϵ and optimistic initialization

2.3 (SC) Softmax action selection instead of ϵ-greedy

Softmax: the greedy action is still given the highest selection probability, but all the others are ranked and weighted according to their value estimates.[5]

2.4 (SC) On-policy Monte Carlo Control

On-policy methods try to evaluate or improve the policy that is used to make decisions.[6] The on-policy method we implemented uses softmax policies, which was previously explained in section **??**.

2.5 (SC) Off-Policy Monte Carlo Control

2.6 (SC) Sarsa

3 Conclusion

[2]Suttion, Barto (1998) *Reinforcement Learning: An Introduction* Cambridge, Massachusetts: The MIT press. p. 126
[3]Suttion, Barto (1998) *Reinforcement Learning: An Introduction* Cambridge, Massachusetts: The MIT press. p. 148
[4]Suttion, Barto (1998) *Reinforcement Learning: An Introduction* Cambridge, Massachusetts: The MIT press. p. 28
[5]Suttion, Barto (1998) *Reinforcement Learning: An Introduction* Cambridge, Massachusetts: The MIT press. p. 30
[6]Suttion, Barto (1998) *Reinforcement Learning: An Introduction* Cambridge, Massachusetts: The MIT press. p. 122

Appendices

A Class Diagram