Assignments Autonomous Agents

D.M. Roijers

September 3, 2012

1 Rules and schedule

In this course, 50% of your grade will be determined by the practical assignments. There are three practical assignments, each of which weighs 33% for the lab part of the course. The assignments are *Single Agent Planning*, *Single Agent Learning*, and *Multi-Agent Planning and Learning.* All of these assignments are provided in this document.

The assignements should be done in pairs. We will allow maximally one group of three, if the number of participants is odd. Each assignment consists of programming and writing a report. The code and the report should be handed in together (e.g. in a zip file), at the given deadlines.

1.1 Quality demands

The report for each assignment must be written in correct academic English. Points can and will be deducted for if this requirement is not met. All reports must have an introduction and a conclusion. References must be provided whenever used. We assume all students are familiar with proper citation. If this is not familiar to you, please contact the TA; plagiarism will be punished.

The code should be readable and commented. It should be clear from the comments, which function provides which functionality. If you are using an object oriented language, be sure to comment on the class structure as well. The names of variables and classes should be meaningful; do not use 1, 2 or 3 character variable names except for in counters. When in doubt about the amount of comments you should add to your code, use as a guideline that the comments should lead the reader through your code.

Any programming language may be used for programming the assignments. However, the usage of esotheric and low-level languages (like assembly) is strongly discouraged, as this hardly ever leads to readable code (which is required). Recommended languages are C++, JAVA, Mathematica, R-script, PHP, JavaScript and Haskell.[1]

1.2 Schedule

- Assignment 1 is due September 21, 2012, at 11:59PM.
- Assignment 2 is due October 5, 2012, at 11:59PM.
- Assignment 3 is due October 19, 2012, at 11:59PM.

Turning in assignments late is not allowed. Assignments that are turned in late will not be graded. The time registered in blackboard counts. If you are experiencing any trouble with uploading to blackboard, you should e-mail all of your code and your report to the TA, before the deadline, along with screenshot of the experienced problem. We recommend submitting a couple of hours before the deadline, to avoid the system rush hour.

1.3 Grading

Every assignment will consist of different functionalities you have to implement and test. There are two types of functionalities. The *must-have* (M) functionalities *must* be correctly implemented - they form the basic functionality of your program, and the baseline for comparisons against other methods. It is impossible to pass an assignment without correctly having implemented the *must-haves*. Then there are *should/could-haves* (SC), which are functionalities that are less essential to the assignments. Implementing SCs will get you extra points for your grade. Implementing only the Ms will get you a 5.5 out of 10, iff done correctly. Implementing enough SCs, after implementing all the Ms, can increase your grade to a 10 out of 10. We strongly recommend implementing all the Ms first. Having all SCs, but not all the Ms will automatically give you a bad mark (failing the assignment), because you would miss the essential functionality, and the base-lines to compare against.

[1]These are the languages most familiar to the TA, and will therefore make it easier to discuss your code with the TA.

2 Assignment 1: Single Agent Planning

2.1 Intro

In this assignment we are going to use a predator vs prey grid world MDP. In this assignment there will be one predator, and one prey. In later assignments, we will add more predators.

The basic environment is a 11 by 11 grid (see Figure 1). The grid is toroidal, which means that the north side of the grid is attached to the south side, and the east side is attached to the west side. So the adjacent squares to the square with coordinates (0,0), are (1,0), (0,1), (10,0) and (0,10). The predator and the prey can be anywhere on this grid, though they cannot be on the same square. The starting position of the prey is (5,5) and that of the predator (0,0).

The prey behaves in a known way, and can therefore be modelled as part of the environment. The prey stays at the same square with probability 0.8, and moves randomly, and with equal probability to any of the adjacent squares (North, West, South or East from the current position) that are open, at every time-step. Therefore, if time starts at $t = 0$, at time-step $t = 1$, the prey has a 0.05 probability of being on (6,5), (5,6), (4,5), and (5,4), and a 0.8 probability of being at (5,5). Note that the prey would never move into the predator, so the probabilities are different when the prey is standing next to the predator.

The predator is the agent. After implementing the environment, you will mainly focus on writing code for the predator agent. The predator has five possible actions: *North* which results in the predator being moved upwards, a difference of (-1, 0) in coordinates. *South*, moving the predator downwards: a (+1,0) change in coordinates. *East*, (0,+1), and *West*, (0,-1). The final possible action is *Wait*, for which the predator does not move.[2] The goal for the predator is to catch the prey. This happens when the predator stands next to the prey, and moves in the direction of the prey. (The prey has no chance of escape in this case.) When the prey is captured, the episode ends, and the game is reverted to the starting position.

The immediate reward for catching the prey is 10, the immediate reward for anything else is 0.

[2]Note that this might not seem like a very useful action, but it can be useful when we move to the multi-agent setting.

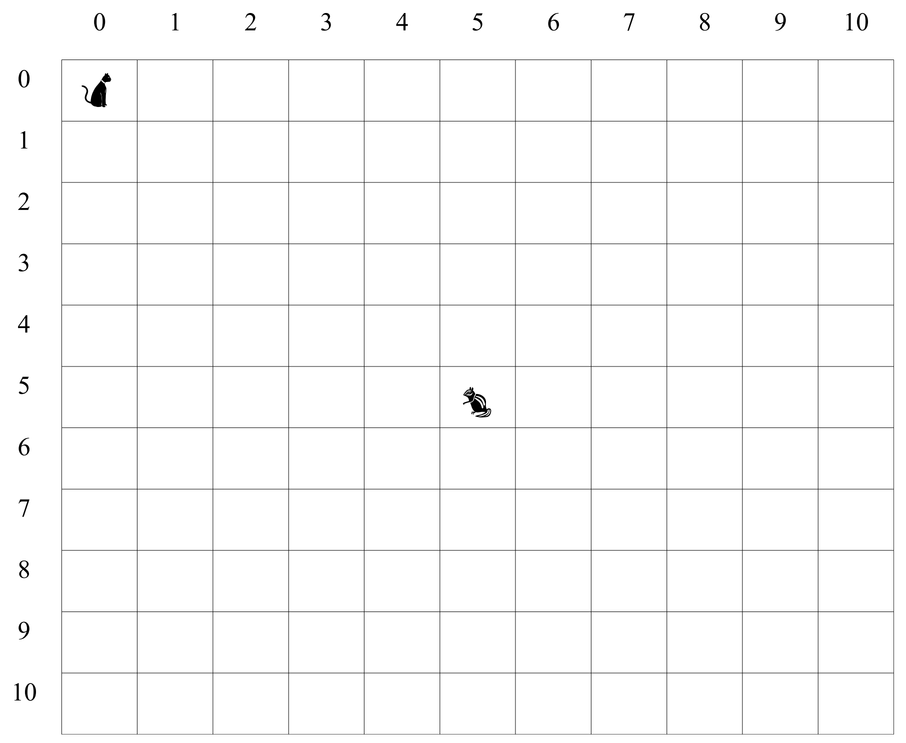

Figure 1: The predator-prey environment: a 11x11 toroidal grid, in the starting position. The predator is depicted by the cat, and the prey by the squirrel.

2.2 Assignments

In Assignment 1, we will consider the planning scenario: the entire MDP is known to your agent. The agent can thus determine the optimal policy even before interating with the environment. If you are asked to run simulations you should first running your planning algorithm, and then execute the policy that results from it.

The following list contains the sub-assignments for assignment 1. A mandatory assignment is denoted with an **M**. Should/Could-haves are indicated by **SC** followed by a number of points. The points you will get for completing all the must-haves is 55, the other 45 points you can get by completing the Should/Could-haves.

We highly recommend that you show the work you did for the Must-haves (code and results) to your TA, before proceeding to the other assignments.

1. **M** Write a simulator for the environment. Be sure to keep the policies of the predator and the prey seperate; both will change in the upcoming assignments. For now, use a random policy for the predator (picking one of the five actions randomly, with equal probability). Test your environment by running a few simulations. Print the states and the actions of the predator, so that you can check whether your simulator is functioning properly. Encode the state as `Predator(X,Y), Prey(X,Y)` (where X and Y are the coordinates). Measure the time it takes on average (for 100 runs) for the predator to catch the prey with this random policy, mention the average and the standard deviation in your report.

2. **SC (10)** Code and use iterative policy evaluation[3] to determine the value of the random policy for all possible states, using a discount factor of 0.8. Put the values of the following states in your report:

   - `Predator(0,0), Prey(5,5)`
   - `Predator(2,3), Prey(5,4)`
   - `Predator(2,10), Prey(10,0)`
   - `Predator(10,10), Prey(0,0)`

   Also report how many iterations it takes to converge.

3. **SC (10)** Think of a smarter way to encode the state-space such that the size of the state-space is reduced as much as possible. How much

[3]see `http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node41.html`

does your solution change the state-space? How does this effect the runtime of the algorithms?

4. **M** Implement Value Iteration[4], and output the values of all states in which the prey is located at $(5, 5)$. Test the convergence speed (in number of iterations), for different discount factors: at least 0.1, 0.5, 0.7 and 0.9, and report on the results.

5. **SC (25)** Implement Policy Iteration[5], and output the values of all states in which the prey is located at $(5, 5)$. Test the convergence speed (in number of iterations), for different discount factors: at least 0.1, 0.5, 0.7 and 0.9, and report on the results. Compare the results to those of Value Iteration.

.

3 Assignment 2: Single Agent Learning

In Assignment 2, we will use the same environment as in Assigment 1, but now we will assume the learning scenario: 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. **M** Implement Q-learning[6] and use this for your predator agent. Use ϵ-greedy action selection with $\epsilon = 0.1$. Initiate the values for your Q-learning table optimistically with a value 15 for all cells in the table. Show plots on the performance of the agent over time for different α (at least $0.1, ..., 0.5$), for different discount factors (at least 0.1, 0.5, 0.7 and 0.9).

2. **M** Experiment with different values of ϵ and the optimistic initialization of the Q-table. Make up good values to test, and explain why you chose these values.

[4]See `http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node44.html`
[5]See `http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node43.html`
[6]See `http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node65.html`

3. **SC (10)** Use Softmax action selection[7], instead of ϵ-greedy. And illustrate the difference, using graphs from your empirical results.

4. **SC (15 points per method)** Implement other ways to do learning, instead of Q-learning, from the following list, explain the difference theoretically, and compare the different methods using informative graphs. Note that your grade cannot become higher than 10 out of 10.

   - On-policy Monte-Carlo Control[8]
   - Off-Policy Monte Carlo Control[9]
   - Sarsa[10]
   - If you would like to test a different learning method, please discuss this with your TA. The TA will have to approve your proposed method. (We highly recommend not to start implementing this until you have recieved this approval.)

4 Assignment 3: Multi-Agent Planning and learning

In this assignment we will make several changes. We will make the prey more intelligent, making it harder to catch.

4.1 Assignments

To be filled in later, please watch blackboard for any changes.

5 Contact info

Please contact your TA for any additional questions. His contact info can be found at `http://staff.science.uva.nl/~roijers`.

[7]See `http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node17.html`
[8]`http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node54.html`
[9]`http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node56.html`
[10]`http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node64.html`