CS 175: Project in AI (in Minecraft): Spring 2017

Assignment 2: It's a Dog's Life!

Sameer Singh (with help from Moshe Lichman)
http://sameersingh.org/courses/aiproj/sp17/

In this assignment, we have simulated a reinforcement learning based dog, whose life's purpose, as it tends to be, is to please its owner. You will define the details of the Markov Decision Process, and use the provided implementation of Q-Learning to learn a policy. Then, you will extend the MDP with more states and actions, and see if how well you can get the agent to do.

1 Task Description

Let me introduce you to Chester the dog, the naive and earnest dog that belongs to Moshe. Every day, while Moshe is discussing all your projects with you in his office hours, Chester spends the whole day trying to find and make something to gift Moshe. At the end of the day, when Moshe returns home, Chester presents what it did that day to Moshe, expecting some reward. Since Moshe is usually quite tired, and in no mood to for useless gifts, usually just punishes Chester, unless Chester *really* brings him something cool.

We will be modeling this scenario in the Malmo environment. Each *run* (or *episode*) consists of a single day of Chester putting something together for Moshe, and ends when Chester is satisfied with what it has accomplished that day. The day starts with a number of items, which Chester can pick up in any order, but since he's a dog, he can hold a maximum of three items at any time, and more unfortunately, doesn't know how to drop any items. However, Chester is a little magical, it can combine some items to create new ones (some that might be quite desirable to Moshe)!

1.1 Provided Source Code

We have provided two Python files: the primary one is `assignment2.py` which contains the complete code to setup the Malmo environment and default RL learning code. This file calls some of the methods in the second file, `assignment2_submission.py`, but this file is incomplete. You will be mostly changes to the second file.

1.2 Overview of the Code

The basic Minecraft environment consists a number of predefined items (called `items`) strewn in a circle around the player agent. The playet agent has an inventory, limited to a maximum size of *three*, but can be changed using `inventory_limit`. In terms of actions, based on the current inventory, the agent (Chester) can either *pick up* one of the existing items (implemented via *teleport* command, which is why you won't see the agent running around), combine multiple items in the inventory to *craft* a new item (using the recipes available as `food_recipes`), or decides to present everything it has in its inventory to Moshe (`present_gift`). How Moshe reacts to receiving a gift is stored in `rewards_map`, with reward for multiple items simply being the sum of rewards of each of the items. We have provided an implementation of Q-Learning, as presented in the discussion[1], which follows an ϵ-greedy policy while following off-policy updates.

The source code should be fairly self-explanatory, since it follows the existing Malmo tutorials quite closely (a related tutorial is `tabular_q_learning.py`, which is the solution to `tutorial_6.py`). Go through the implementation for details, and discuss on Piazza if you have any doubts.

1.3 Setup and Running the Code

Assuming you have installed Malmo, all you need to do to run this assignment is to copy the two files above to the `Python_Examples` folder, and after launching Minecraft, run `python assignment2.py`. If everything run successfully, the agent should be doing random things, since the implementation is incomplete (more on this later). The output in the terminal should look like the following:

[1] https://github.com/MosheLichman/CS175-Discussions/tree/master/discussion_3

```
n= 1
1 Learning Q-Table:  pumpkin, c_pumpkin_seeds, egg, present_gift,
Reward:  -75
2 Learning Q-Table:  egg, egg, pumpkin, present_gift, Reward:  -55
3 Learning Q-Table:  pumpkin, sugar, egg, c_pumpkin_pie,
present_gift, Reward:  100
4 Learning Q-Table:  egg, pumpkin, sugar, c_pumpkin_pie,
present_gift, Reward:  100
5 Showing best policy:  egg, egg, sugar, present_gift, with reward
-60.0
6 Learning Q-Table:  pumpkin, egg, sugar, c_pumpkin_seeds,
present_gift, Reward:  -85
7 Learning Q-Table:  pumpkin, egg, egg, c_pumpkin_seeds,
present_gift, Reward:  -100
8 Learning Q-Table:  egg, sugar, present_gift, Reward:  -35
9 Learning Q-Table:  sugar, egg, present_gift, Reward:  -35
10 Showing best policy:  pumpkin, sugar, egg, c_pumpkin_seeds,
present_gift, with reward -85.0
...
```

2 What Do I Submit?

Here we'll describe what exactly you need to submit to the assignment on Canvas.

1. **Code: Define the State (10 points):** As the first simple exercise, implement the `get_curr_state` function. Given the list of items in the inventory, this function returns an indexable Python object (*hint:* for example, a tuple) that represents the state of the inventory. Keep in mind that any possible combination of items will be a different state, and further, the order of the items does not matter (however, the quantity of item of each type does). Solution will be quite small (ours was 5 lines, could have been smaller).
2. **Number: Number of States (5 points):** Assuming the environment contains one copy of each item (including the crafted items), how many states do you think are possible? As when defining the state space above, keep in mind that ordering between items in the inventory does not matter. Just provide a number.
3. **Code: Implement ϵ-Greedy Policy (15 points):** The `choose_action` method currently ignores the `eps` and `q_table` values and just returns a random action. Instead, implement the ϵ-greedy policy that does exactly the above with probability `eps`, but with `1-eps` it picks the action with the highest Q-value (you can get a list of actions and q-values using `q_table[curr_state].items()`). Note that in case of multiple actions having the same maximum q-value, you should *randomly* pick any one of them. Solution can easily be achieved in $10-15$ lines, but probably in less.
4. **Number: Reward From Best Policy (10 points):** Given the list of items, the recipes, and the rewards, what is the maximum possible reward that Chester will get at the end of any episode? Keep in mind that the inventory can only hold a maximum of three items, and Chester cannot drop any item (they get removed only when they are part of a recipe). Use this in the optimality test (`is_solution`), and submit the number.
5. **Output: Smartest Chester Can Be (15 points):** Now your implementation should be complete. Run the agent till it ends with the output indicating it has found the best policy. Submit the last three lines (starts with `XXX Showing best policy ...`, then `Found solution`, and then `Done`).
6. **Code: Expanding the World (10 points):** We will now add more items and recipes to the world. To begin with, add the following items (one `red_mushroom` and two `planks`) and recipes (a `bowl` made from two `planks`, and a `mushroom_stew` made from a `bowl` and a `red_mushroom`). Also include a reward of 5 for the `red_mushroom`, -5 for the `planks`, -1 for the `bowl`, and 100 for the `mushroom_stew`.
7. **Number: Number of States 2 (5 points):** With this expanded list of items and recipes (and assuming the environment has one of each, including crafted items), how many states do you think are possible now? Just provide a number.
8. **Number: Reward From Best Policy 2 (10 points):** Given this expanded list of items, recipes, and rewards, now what is the maximum possible reward that Chester will get? Use this in the optimality test (`is_solution`), and submit the number.
9. **Output: Running on the Expanded World (10 points):** Given this implementation of the the expanded world, run the same code as before. Note that the world is much bigger now (in terms of number of

states), and thus do not be disappointed if your agent does not converge to the optimal policy any time soon (or at all). Run your agent for at least 200 episodes, and submit at least the last 20 lines starting with `XXX Showing best policy ...` (if on a Unix system, *grep* and *tail* will be your friends).

10. **Extra Credit: Improving the Agent (20 points)** If you are feeling adventurous, you can try to run with $n > 1$, or change α, γ, and ϵ, to see if you get better results. You should basically see if you can find a policy that is significantly better than the previous policy, in the same number of episodes (200). Describe in a line or two what you changed, and submit the last 20 lines that start with `XXX Showing best policy ...`, as in the previous part. You will be graded on a combination of what improvements you achieved, and what you tried.
11. **Comments:** Any comments about your submission that you want to bring to our attention as we are grading it. This is completely optional, I expect most of you to leave this empty.
12. **Statement of Collaboration (10 points):** It is **mandatory** to include a *Statement of Collaboration* with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed. You should also include the links to all online resources you used for the assignment in this section.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, I encourage the students to organize (perhaps using Piazza) to discuss the task description, assignment requirements, bugs in our/Malmo code, and the relevant technical content *before* they start working on it. However, you should *not* discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no photographs of the blackboard, written notes, etc.). The same holds for online resources: you are allowed to read the description of algorithms, but your code should be your own. Especially *after* you have started working on the assignment, try to restrict the discussion to Piazza as much as possible, so that there is no doubt as to the extent of your collaboration.