The Wumpus World

This document gives a detailed instruction on how to run the Wumpus World testbed, how to tune the RL parameters and how to add arguments to improve RL's performance. Note that all instructions below are based on the Linux system.

Step 1: Download and run the testbed

1. Download 'WWRL.tar.gz' from CATE and uncompress it (you can use command 'tar -zxvf WWRL.tar.gz'). A folder named 'WWRL' will be extracted.

2. Open a terminal, change directory to WWRL, and run the testbed with the following command:

java -jar WWRL.jar

Ensure that you run this command in folder WWRL. Then the console GUI will appear, as shown below:

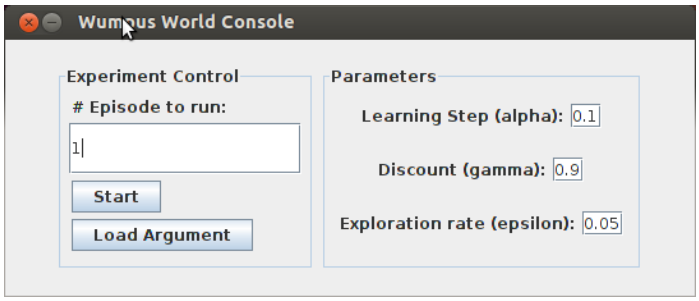

Figure 1. Wumpus World Console

The console has two parts: experiment control (left) and parameter tuning (right). In experiment control, you can input the number of episodes you want to perform, start an experiment and load arguments (details about loading arguments will be given later). Note that the number of episodes should be an integer larger than (excluding) 0. We suggest **100 episodes** when you use the default learning parameters. The RL parameters can be tuned in the right part. Recall that each parameter is a real number in [0,1]. The default parameters will be used if you do not change them.

3. After all parameters are set, you can press the 'Start' button to start an experiment, and two windows will pop up: the Environment window and the Agent window, as shown in Figures 2 and 3 below, respectively.

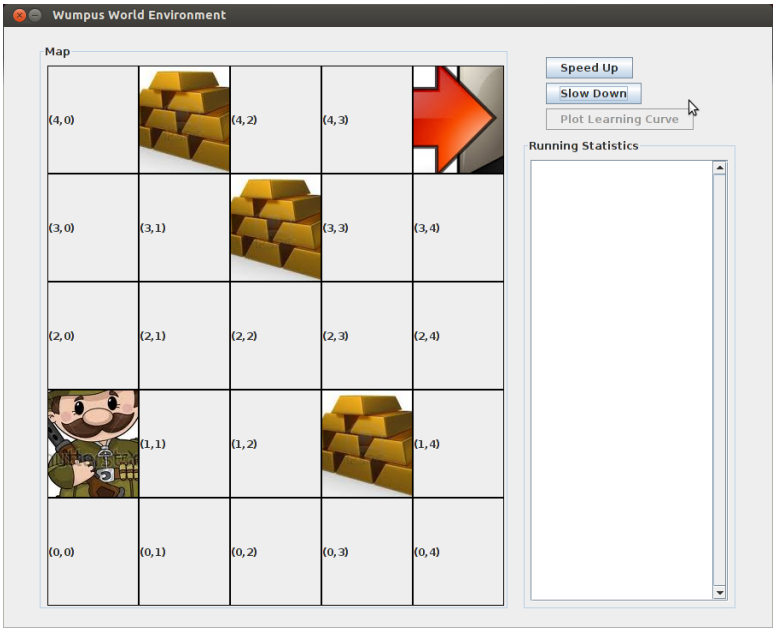

Figure 2. Wumpus World Environment

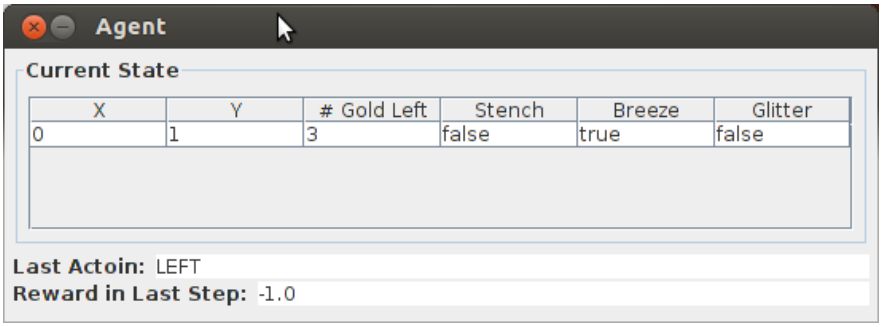

Figure 3. RL Agent

The Environment window shows the current state of the Wumpus World. By default, each step takes 0.5 seconds. You can tune the speed of the experiment by pressing the buttons ‘Speed Up’ and ‘Slow Down’. After each episode ends, a summary of the agent’s performance will be presented in the ‘Running Statistics’ field.

The Agent window shows the agent’s observation of the world and its action in the current state. By tuning the speed of the experiment in the Environment window, the updating frequency of the Agent window will also be changed accordingly.

When the experiment ends, the button ‘Plot Learning Curve’ will be activated and you can press it to see the learning curve of this experiment. Typically it will take around 5 seconds for the learning curve to pop up after you press the button. An example curve is shown in Fig. 4.

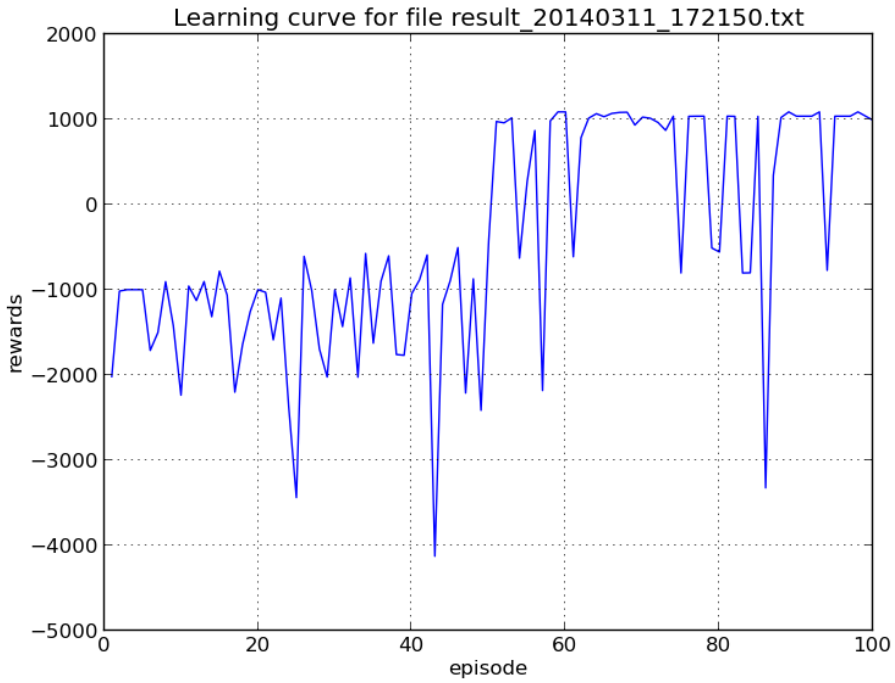

Figure 4. An example learning curve

Note that after each experiment finishes, the learning performance in this experiment is automatically saved in a file named ‘result_DATE_TIME.txt’ in the WWRL folder. The title on top of the curve indicates which text file the curve relates to.

Step 2: Add arguments to improve the learning performance

1. Why add arguments into RL

RL chooses which action to perform at each state by comparing each action's Q-value. However, in the Wumpus World, the goodness of some actions can be deduced given the goodness of other actions. For example, consider the following situation: the agent comes to a new square and it smells stench. The agent has visited the square in up, left and down directions before, and the agents knows that stepping into these three squares is safe (this can be easily found by checking the Q-values of these actions). But the agent has not visited the right square before. According to the rules of the Wumpus World, we can easily deduce that there must be a Wumpus in the right square now, and the best action to perform is to SHOOT_RIGHT. However, classical RL does not have this reasoning ability, and may still choose to perform action RIGHT. Therefore, in order to improve the learning speed of RL, we can add some logic rules describing, in some situations, which actions are good and which actions are bad. In this tutorial, we use arguments to represent these logic rules.

2. What is an argument in the Wumpus World

An argument is a logic sentence describing which actions are recommended and/or discouraged in certain situations. Consider the earlier example: here the agent's knowledge can be described by an argument as follows:

```
IF
left	ok
up	ok
down	ok
right	unknown
stench	true
THEN
+	SHOOT_RIGHT
-	RIGHT
DONE
```

'+ SHOOT_RIGHT' means: SHOOT_RIGHT is deemed a good action in this situation;
'- RIGHT' means: RIGHT is deemed a bad action in this situation.

More generically, an argument is of the following format:

```
IF
condition1
……
conditionN
THEN
+/-	action1
……
+/-	actoinM
DONE
```

This argument means: when condition1, …, conditionN are all true, then action1 is good/bad, …, actionM is good/bad. Also note that IF, THEN, DONE must appear in an argument, and they must occupy a line, i.e. no conditions or actions should be in the same line with them. A condition in an argument can be of the form:

up/down/left/right	ok/bad/unknown

or

stench/glitter/breeze	true/false

where the lefthand side and the righthand side are **tab** separated. Actions can be chosen from LEFT, RIGHT, UP, DOWN, SHOOT_LEFT, SHOOT_RIGHT, SHOOT_UP and SHOOT_DOWN.

Another issue worth noticing is about un-mentioned conditions in an argument: if some conditions do not appear in an argument, then the applicability of this argument does not depend on this condition. Still consider the earlier example: we did not mention glitter or breeze in the conditions of that argument, then no matter whether glitter/breeze is true or false, if all the conditions mentioned in the argument are satisfied, SHOOT_RIGHT will be recommended and RIGHT will be discouraged.

To help you better understand the format requirement of arguments, we give a file in the WWRL folder named 'example_args' with more examples and comments.

3. Create an argument file and load it

Create a text file containing all arguments you want to use in an experiment. Then in the Console window, press the 'Load Argument' button, and a new window will pop up to let you choose which file you want to use. Choose an argument file (e.g. 'example_args'), and the system will check whether the arguments in the file satisfy format requirements. Dialogue windows will pop up to let you know whether the arguments in the file are all right or any problems exist.

4. How arguments improve RL

Here we briefly describe how arguments affect the RL process. The basic idea is that in a state, if some actions are regarded as good/bad in some of your arguments, some extra rewards/punishments will be given to these actions in RL. Still consider our earlier example argument: if in a state all conditions satisfied, then when RL is choosing an action in this state, action SHOOT_RIGHT will be given some extra rewards so as to encourage this action, and action RIGHT will be given some extra punishments (negative rewards) to discourage its execution.

Exercises

1. Describe the following domain knowledge in the form of arguments, and add these arguments in the given file 'example_args':

   If one direction is unknown, the other three directions are ok, and the agent feels breeze, then disencourage going to the unknown direction

   (Hints: you should write four arguments to describe this knowledge)

2. Set epsilon=0, alpha=0.1 and gamma=1, and run two experiments, each with 100 episodes: one without any arguments and the other one with arguments in the extended 'example_args' (after exercise 1). Observe the learning curves of these two experiments and compare them. (Hints: you can compare the learning curves in terms of the initial performance, the best performance, the first time the best performance is reached, the worst performance, the average performance, etc.)

3. Design some more domain knowledge using arguments. Put these arguments in a file called 'extra_args' and test whether these arguments can improve the learning performance (these arguments can be used jointly with the arguments in 'example_args' and in exercise 1). Describe the meaning of each argument as comments (for details of making comments in argument file, see 'example_arg').