COS 445 : Networks, Economics and Computing - Fall 2012
Programming Assignment #01: Prisoner's Dilemma

Out Wednesday, Sept 19th 2012
Due **23:59** : **Monday, Oct 1, 2012**

Total points: 20. Worth: 10% of the final grade. This is a single-person assignment. Points will be awarded for clarity, correctness and completeness of the answers. You are free to discuss the problem set with other students. However, you should not share answers.

Introduction

The Prisoner's Dilemma is a canonical example in game theory, which is often used to explore behaviors of people in a situation of strategic interdependence. In its basic version, the Prisoner's Dilemma is a one-shot game. Here, we consider the iterated (i.e., repeated) Prisoner's Dilemma (IPD), where the game is repeated over several rounds, and the players can make their decisions based on knowledge of the outcomes in previous rounds. Cooperation can occur in this version of the game because a player is able to punish a defecting opponent.

In his 1984 book *The Evolution of Cooperation*, Robert Axelrod described the results of a tournament he organized for academics to submit IPD strategies for a fixed number of steps. The winner of Axelrod's competition was the strategy "tit-for-tat" – simply choose the same move as was chosen by the opponent in the previous round. In general, Axelrod found that altruistic strategies triumphed over greedy ones.

For this problem set, you will create a submission to an IPD tournament like the one Axelrod set up in 1984 but with a twist. The information your player receives about its opponent's previous action is noisy, i.e. *with probability 0.05 your player receives incorrect information about the previous action of the other player*. The tit-for-tat strategy is not reliable in noisy environments. To understand why this is so, imagine that Alice is told that her opponent Bob defected, when in fact Bob collaborated. Alice would then defect, falsely punishing Bob, and the mistaken defections will echo until someone gets the wrong information once again. What do you think would be the best strategy to overcome a noisy environment for IPD? In discussing the complications that noise adds to his tournament, Axelrod said that "Noise calls for forgiveness, but too much forgiveness invites exploitation." How will you design a winning agent for such an environment?

Setup

We designed a website http://www.cos445.com to test and submit your agents. You need to login to this website with the userid and the password we have sent you. Once you login you will notice a default agent written in Java. This default agent uses a random strategy i.e., cooperates or defects with probability 0.5 each. Its actions are independent of the opponent's actions in the previous rounds. You need to modify this Java file to implement your own strategy. **Your Java class name should be same as your username.** You can submit multiple agents and test

them with each other. Your latest agent is always shown on top. You can also choose one of your agents to play with all the latest agents of the rest of the students, but no more than once every 30 minutes. It is your choice whether to run set of noiseless or noisy games. **For grading, we choose your latest agent that is submitted before the deadline and run it in *both* noiseless and noisy regimes.**

Tasks

1. **[6 Points]** Designing your Agent

   Your task is to create an agent for the (finitely) iterated noisy Prisoner's Dilemma described above. The agent takes no command line parameters. To begin, your agent must read three comma-separated values from `stdin`: the number of rounds (type: `int`), the amount of noise (type: `double`), and the total amount of time allowed (type: `double`). The noise is a value which will be either 0.00 (no noise) or 0.075, and the amount of time allocated is always at least 5.0 seconds. The scores for each round are as follows:

	C	D
C	3,3	0,5
D	5,0	1,1

   In every round, your program reads the opponent's strategy from previous round (a character 'C' or 'D') and the amount of time it has left (a `double`), separated by a comma. It must then respond with either 'C' or 'D' followed by a newline on `stdout` (we recommend using `System.out.println()` for this, as it automatically flushes the output buffer). If your agent fails to produce a valid output, crashes, or its allocated time has expired, we proceed as if it had returned 'C' in all remaining rounds[1] (in the output displayed to the user, timeout-induced 'C's are denoted by 'C(T)').

   Testing : For your convenience, www.cos445.com provides a mechanism to test your agents. After your agent is submitted, the server compiles your code and attempts to pit it against itself. If everything goes smoothly, this new agent appears at the top of your submission list and is ready to be pitted against other successfully-run agents. Otherwise, the server will give you a potentially helpful error message. Each simulation result reports the round number, what action was taken by the two agents, what actions were observed (because the data is noisy this may be different than the actual actions taken), and the two cumulative scores for each round.

2. **[12 Points]** Analysis

   Explain in a few paragraphs how your agents (with noise and without noise) work, why you designed it that way, and how you arrived at this design. How do you think it will do against the agents of the other students? Explain how your strategy works with varying number of rounds and different amounts of noise. **Please limit your explanation to at most two pages**.

3. **[2 Points]** Competition

[1] In the context of Prisoner's Dilemma this is consistent with the principle that "Dead men tell no tales".

We will run each submitted agent against each of the others with and without noise between 1 and 10 times, with each game lasting 100 rounds. We will rank agents by their cumulative score. You will get points proportional to your cumulative score attained in the competition with other agents.
In addition, winners will be invited to present their agents to the class.

More rules

- Each agent can spend at most 5 seconds (cumulative time) for 100 rounds before it times out.
- Memory usage of each agent is limited to at most 12MB.
- Please do not submit agents that hack. For example, writing out of memory, time-outing on purpose, side channel attacks, etc. are not allowed.
- Do not collude with other students to design a joint colluding strategy.
- Public discussion on Piazza (including that of potential strategies) is allowed. If you have any questions related to this assignment, please post your question on Piazza.
- Each student must submit a separate agent and explanation.