The Repeated Three Prisoners' Dilemma

CO 456 Project 2, Fall 2008

Due Date: November 11 2008 in class

In this project you are asked to analyze a repeated strategic game and implement a Java strategy for it.

Format

In this project you can work alone or in a team of 2. You can keep the same team you had last time, or change teams if you want. Each team submits *one writeup* and *one Java submission.* You are not permitted to discuss the project with anybody from another team.

1 The Game

Background: Two sly criminals, who are known to police only by their aliases "Player 1" and "Player 2," have just been caught by the police. However, there is only circumstantial evidence against them, and with this evidence they would only each be convicted for 1 year in jail. The police interrogate the suspects separately, and hope to get each one to confess to the crime. But, why would any criminal in their right mind confess? The police sweeten the deal by telling Player 1 that if she confesses, then she will go free, while her partner will receive three years in jail. They offer Player 2 the same deal. But what they don't disclose is that if both prisoners confess, then they will both get two years in jail. As it turned out, at the trial, the prosecutors talked endlessly about "Nash equilibria" and "dominating strategies" instead of presenting any evidence, so both of the prisoners went free.

Our game: Our story begins with the capture of the evil duo's little-known accomplice, "Player 3." We want the 3-player prisoner's dilemma to resemble the 2-player prisoner's dilemma as much as possible. Write $U(a_1 : a_2a_3)$ for the utility to a player who plays action a_1 when the other players play actions a_2 and a_3. We know that in the 2-player game,

$$U(F : Q) > U(Q : Q) > U(F : F) > U(F : Q).$$

If any one player has a fixed strategy, then we want the game for the other two players to be a 2-player prisoner's dilemma. So for example, we must have

$$U(F : QQ) > U(Q : QQ) > U(F : FQ) > U(Q : FQ).$$

Using this reasoning, the unique possible ordering of outcomes in the 3-player dilemma is

$$U(F : QQ) > U(Q : QQ) > U(F : FQ) > U(Q : FQ) > U(F : FF) > U(Q : FF).$$

Specifically, we will model the game by the payoffs shown in Figure 1.

In the competition, you will be playing an iterated version of this game.

$p1\backslash p2$	Q	F
Q	6,6,6	3,8,3
F	8,3,3	5,5,0

$p3$: Q

$p1\backslash p2$	Q	F
Q	3,3,8	0,5,5
F	5,0,5	2,2,2

$p3$: F

Figure 1: The 3-player prisoner's dilemma.

2 Writeup Questions

2.1 The 2-Player Dilemma

The first few problems are not directly related to the 3-player game used in the tournament.

Osborne exercise 426.1 asks the following: "Show that a finitely repeated *Prisoner's Dilemma* has a unique subgame perfect equilibrium, in which each player chooses fink in every period, regardless of the history." This is a good question although its solution is available at

`http://www.economics.utoronto.ca/osborne/igt/solsp5.pdf`

We want you to solve the following slightly different question.

Question 1. *Assume there are $T \geq 1$ periods in the game and that in each period we use the numerical payoffs given in Figure 2. In this finitely repeated game, define each player's utility be the sum of their*

$p1\backslash p2$	Q	F
Q	2,2	0,3
F	3,0	1,1

Figure 2: The 2-player prisoner's dilemma.

payoffs from the individual rounds. (Each player knows T.)

Suppose we now modify the game as follows: if both players stay quiet for all T rounds they each get an additional bonus of value 1.5 *at the end.*

(a) *(4 points) When $T = 2$, find all (pure) subgame perfect equilibria.*

(b) *(4 points) Fix $T \geq 2$. Find a (pure) SPE s whose outcome is that both players stay quiet for all T rounds. (Be sure to specify the strategies fully, including choices they specified in subgames that are not actually reached when players play according to s — i.e. what they would do in the event of a mistake.) You should use the one-deviation property to prove that s is an SPE.*

Question 2 (3 points)**.** *Consider the infinitely-repeated version of the strategic game shown in Figure 2, where each player's utility is their δ-discounted average, and $0 < \delta < 1$ is fixed. Let s be a pure strategy profile whose outcome is that player 1 always stays quiet and player 2 always finks. (I.e., $O(s) = ((Q, F), (Q, F), \dots)$.) Show that s is not a Nash equilibrium. (Hence, s is not an SPE.)*

2.2 The 3-Player Dilemma

Question 3 (5 points)**.** *Consider the infinitely-repeated version of the strategic game shown in Figure 1, where each player's utility is their δ-discounted average. Find a value $\delta \in (0, 1)$ so that the following strategy profile $s = (s_1, s_2, s_3)$ is an SPE:*

- *At every history h, $s_1(h) = F$ iff either player 1 or 2 has finked in any round of h (i.e. s_1 is like a modified grim trigger strategy that ignores player 3)*
- *s_2 is the same as s_1*
- *At every history h, $s_3(h) = F$*

(You must prove that s is an SPE by using the one-deviation property.)

Question 4 (2 points)**.** *Suppose that the payoff when you fink and the other two players stay quiet is changed from 8 to 13. (So replace every 8 in Figure 1 with a 13.) If you could collaborate with and trust the other players, how could you take advantage of this change? (Your answer can be somewhat informal, and may assume that a large number of rounds take place.)*

3 Code Fragment and Tournament Structure

Your team will be asked to submit a Java code fragment that will be entered in a tournament against the other teams. Specifically, we will run an iterated version of the game shown in Figure 1. Each *match* consists of a number of rounds. In each round, you will be given a list of the actions chosen by each player in all previous rounds, and you will need to choose either to *fink* or stay *quiet*.

Matches

Each match will consist of at least 1 round. After each round, with probability $\Omega = 1/100$ the match ends, and otherwise there is another round. (This implies that the expected number of rounds per match will be 100.) In order to calculate your score for a match, we will take the sum of all utilities obtained by each player in that match and divide it by the total number of rounds in that match. So, your score for a given match will be a number between 0 and 8.

Tournament Format

We will run every possible combination of teams in several (`repCount`) matches, without regard to order, and with repetition allowed. E.g., for each pair of distinct teams T_1 and T_2, there will be exactly `repCount` matches in which 2 copies of T_1 play against a single copy of T_2. Your score for the tournament is the sum of the scores your strategy obtains for each time it is used in a match. *We will remove all sample strategies before running the tournament.* The provided code has `repcount = 1` but the official tournament will use `repCount = 7`.

Code Format

We represent "fink" by the boolean value `true` and "quiet" by the boolean value `false`. Your strategy will consist of a method `fink` whose input consists of the following: `int n`, the number of rounds elapsed so far; `boolean[] iFinked`, an array of n integers representing the actions that you took in rounds $0, 1, \ldots, (n-1)$; `boolean[] o1Finked` and `int[] o2Finked`, n-element arrays representing the actions that your opponents took in the preceding rounds. So informally, `iFinked`, `o1Finked`, `o2Finked` are the histories of the match so far, e.g., `iFinked[0] = true` means that you finked in the first round. Your procedure should return `true` if you want to fink that round, and `false` if you want to stay quiet. Here is one sample strategy.

```
public class Flipper extends Player {
// Flipper always plays quiet, fink, quiet, fink...
  boolean fink(int n, boolean[] iFinked, boolean[] o1Finked,
                   boolean[] o2Finked) {
    if (n == 0 || iFinked[n-1] == true)
      return false;  // stay quiet!
    else
      return true;  // fink!
  }
}
```

You can access the code at

`http://www.math.uwaterloo.ca/~dagpritc/co456/Project2Tournament.java`

You should be able to run the code as written, and after typing in your own strategy, running the code will test it against the sample strategies. You change change the level of output detail by altering the value of `verbosity` and you can output to a text file as indicated at the start of `runTournament`.

3.1 Submission and Grading

In your writeup, include an answer to the following question.

Question 5 (2 points)**.** *Describe your submission to the tournament, and give us an idea of the logic behind its design.*

You need to send your code fragment to the following email address by 11:30 AM on November 11.

`dagpritc@math.uwaterloo.ca`

Your team should bring one hard copy of your writeup to class on that day.

Your team can obtain a maximum of 20 points on the writeup. You will get three additional points for submitting a strategy. The remaining 10 points will be given based on the performance of your strategy in the tournament. So there are 33 points in total.

- The top team will get 11/10 points
- The remaining top 1/6 of all teams will get 10/10

⋮

- The bottom 1/6 of all teams will get 5/10.

We will run the tournament multiple times and give you a score corresponding to your best finish, in order to alleviate random fluctuations.

Technical Requirements

The technical requirements are the same as in the last project; the main restriction is that you cannot use the `static` modifier. Informally, your code should have no memory from one match to the next. However, you can assume that we will not re-instantiate your class between rounds of the same match (this is how the given tournament code is set up).