Due Date: November 20, 2017 at 11:59pm

Submission Instructions: Assignments are to be submitted through LEARN, in the Dropbox labelled *Assignment 4 Submissions* in the Assignment 4 folder. Late assignments will be accepted up until November 22 at 11:59pm. Please read the course policy on assignments submitted after the official due date. *No assignment will be accepted, for any reason, after 11:59pm on November 22.*

Lead TA: Ben Armstrong (ben.armstrong@uwaterloo.ca). Office hours on Fridays from 4:00pm-5:30pm in DC 2306 (the AI Lab).

Announcements: The following is to be done independently. You need to use python 3.X for your implementation.

Question 1

As you know, in the game the Prisoners' Dilemma, the dominant strategy equilibrium is for agents to defect (i.e. confess) even though both agents would be better off cooperating with each other and keeping quiet. When we move to the repeated version of the game, however, the possibility of cooperation begins to appear.

In a repeated game, a given game (typically in the normal form representation) is played multiple times (possibly infinitely often) by the same set of players. We compute the (average) reward of a player in a repeated game to be

$$\lim_{k\to\infty} \sum_{j=1}^{k} r_j/k.$$

A strategy in a repeated game specifies what action the agent should play in each state of the game, given all actions taken by all players in the previous rounds. For example, one famous strategy for the two-player repeated prisoners' dilemma is Tit-for-Tat (TfT). In TfT agents start by cooperating and thereafter chooses in round $j+1$ the same action that the other agent chose in round j. If both agents play TfT then we have an equilibrium (with some additional conditions), however, this is not the only equilibrium that may arise and, in particular, there are infinitely many strategies which agents may consider.

In this assignment you will develop a strategy for an agent in a **three player repeated prisoners dilemma with noisy channel**. Triples of agents will play each other repeatedly in a "match" with a *randomized* number of rounds (drawn independently for each game from a truncated normal distribution with mean 100). Your score from that match is the *average* of the payoffs from each round of the match. For each round, given a list of the previous plays (so you can remember what your opponents did) your strategy must compute the next action. We represent cooperation by the integer 0 and defection by the integer 1.

To add a minor complication to the game, the communication channel with the game is *noisy*. That is, in each round, there is a 2% chance (drawn independently for each player) that the action chosen by the agent is reversed. For instance, if you specify an agent that always cooperates, it will actually submt the defect action in approximately 2% of the rounds.

Your strategy will be a code fragment that looks at the previous plays (by you and your opponents) for that match and computes your next play. For example, here is a code fragment that implements a Nice Agent who always cooperates

```
from prison import Player

class NicePlayer(Player):
    """
    The nicePlayer always cooperates (plays 0)
    """
    def studentID(self):
        return "42"
    def agentName(self):
        return "Nice Player"
    def play(self, myHistory, oppHistory1, oppHistory2):
        return 0;
```

We provide code in the files prison.py and runTournament.py. The first file provides definitions necessary to create your agent; the latter file contains code for running the game. You need to extend the class **Player(object)** by modifying the **studentID**, **agentName**, and **play** functions. We are also supplying you with sample code for running the prisoner's dilemma scenario to help with testing, but you should not modify this at all.

We will run all agents against each other in a tournament, ranking agents by their performance. Throughout the tournament, we will only initialize your agent once using the default constructor; for example `agent001 = NicePlayer()`. Therefore, you may maintain internal states in your player and expect them to persist between games.

To Submit

- Your code for your agent in a single file called `YOURSTUDENTID_YOURAGENTNAME.py` For example if my ID is 12345678 and my agent name is WonderAgent, then your file should be `12345678_WonderAgent.py`
- A name for your agent (be creative!)
- A short (max 500 word) description of how your agent works, including why you believe it is the best agent for this game. If you have implemented a strategy you found elsewhere,

then cite the reference. If your agent is designed to collude with others (see Notes) then note this.

Marking

- 3 marks for submitting an agent correctly according to the instructions
- 1 mark for a great name
- 4 marks for how innovative, creative or strategically powerful your agent is (even if only in theory). This will come from your writeup. If you submit something simple like TfT, you can still get full marks if you provide a convincing argument as to why the agent is best. For instance, you may wish to elaborate on how you are handling the noisy channel and the randomized number of rounds.
- 2 marks for basic performance of your agent. We will play your agent in several simple scenarios, against simplistic agents, *without* the noisy channel. These scenarios will act as a basic evaluation of your agent, and you will receive full marks if your agent performs reasonably in these tests — importantly, they *do not need to perform optimally* to receive full marks.
- Up to 2 bonus marks for performance in the tournament. All agents will be ranked by average score per round.
  - Agents ranked in the top 50% will get 1 bonus mark
  - Agents ranked in the top 10% will get 2 bonus marks

Notes

- Your agent must be able to play a round in under 1 second. If we find that it takes longer than this, then the agent may be disqualified.
- Your code must run with no special libraries or file, be written in Python 3.X and follow the instructions above. If your code does not run, or uses too much memory your agent may be disqualified.
- Your agents will play the others in random order, and for each match your agent will be instantiated once at the start.
- Your code is not allowed to read/write to the file system or the network. Your code is not allowed to tamper with the code running the tournament or the code of other agents. Any code found doing this will be disqualified.

- It may be possible to form coalitions in the game. Although each student must implement their own agent, you are allowed to discuss strategies beforehand and may even design agents that try to collude with each other. However, during the tournament, your agent will not be told which agents they are playing against, so collusion would require some sort of *hand-shaking*. If you do decide to try colluding, you should include it in the writeup.