Assignment 5: Prisoner's Dilemma Agent

CS486/686 – Winter 2015

Out: November 18, 2015
Due: December 4, 2015 at 12pm

Submit your assignment on LEARN and by uploading your code as described below.
******* Late assignments will be accepted up until December 14th at 12pm ******

The iterated prisoner's dilemma is a classic two-person game which consists of a number of rounds. In each round, each person can either *defect* by taking \$1 from a (common) pile, or *cooperate* by giving \$2 from the same pile to the other person [4]. In the one-round game, there is one Nash equilibrium in which both players defect[1]. However, when humans play the iterated (multi-round) game they often are able to achieve cooperation over many rounds. A purely rational agent will optimise over his expected long-term payoffs, possibly by averaging over his expectations of his opponent's type (or strategy). The prisoner's dilemma has long been studied, starting with the work of [2, 3]. Recent work has shown that strategies exist for iterated PD in which one player can dominate another [5]. These so-called zero-determinant (ZD) strategies have been shown, however, to be evolutionarily unstable [1].

In this assignment you will write a computer program to play the Prisoner's dilemma. Your program *must* be written in Python 2.X, and must implement the following abstract base class with three functions:

```
class PDBot:
    @abstractmethod
    def init(self):
        pass
    @abstractmethod
    def get_play(self):
        pass
    @abstractmethod
    def make_play(self, opponent_play):
        pass
```

You will be provided with a file `PDbot.py` that has this abstract base class in it, as well as the file `AwesomeBot.py` with an implementation of an *"almost-tit-for-tat"* agent that plays *tit-for-tat* (start with give \$2 and always repeat what the opponent did in the last turn afterwards) 90% of the time, and does a random action 10% of the time. The file `AwesomeBot.py` also includes a simple interactive test program (main function) so you can try it out.

Your functions will be called in the same order that they are in the test program. At the start of all games, your constructor will be called (if you provided one). Then, at the start of each game, `init` is called. Then, at each round, first `get_play` followed by `make_play` is called with whatever the opponent played in the last round. This means that you can't see what your opponent is going to play before deciding what to play (and the same holds for your opponent). Your function `get_play` **must** return either of two strings `''give 2''` or `''take 1''`. If your function returns anything else, it will be replaced with `''give 2''`. Your agent **must** be able to play a round in under 5 seconds.

[1] The exact payoffs are not necessarily \$2 and \$1, but it must be the case that a game with actions of give \$$R$ and take \$$P$ has $(R+P) > R > P$ for mutual defection to be the Nash equilibrium of the game. If $2R > (R + P)$, then mutual cooperation is the best global outcome.

What to hand in (on LEARN):

- Your code in a single file called `<YOURBOTNAME>_<YOURSTUDENTID>.py`. For example, my student number is 12345678, so I submit AwesomeBot in a file named `AwesomeBot_12345678.py`. Your agent name (and first part of the file name) **must be identical (case sensitive)** to the name of your subclass of `PDBot`.
- A name for your agent (be creative!)
- A short (max 500 words) description of how your agent works, including why you believe it to be the best agent for this game. Be concise. If you've implemented a strategy described elsewhere (e.g. in a paper), give a clear reference. If your bot is designed to collude with other bots (see Notes below), indicate this clearly.

Your agent will be played in a tournament against each other submitted agent, as well as against 5 agents that will be submitted by the instructors. Your agent will play 5 consecutive games against each other submitted agent, each game consisting of 15 ± 3 rounds (plays of *give 2* or *take 1* by both agents).

Your grade will be assessed as follows for a total of 7 marks (3 marks for CS686) with an available 3 marks bonus. Each mark is 1% of your final grade.

- 3 marks (CS486 only) for submitting an agent correctly according to the instructions above.
- 1 mark for your agent's name (be creative!)
- 3 marks (2 marks for CS686) for how innovative, creative, or strategically powerful your agent is (even if just in theory). To assign these marks, I will primarily look at your write-up. If you submit one of the examples described above or *tit-for-tat* (or something equivalently simple), you can still get full marks by giving a convincing reason why this agent is best. You can even argue for an agent other than the one you submitted, but only if you explicitly say so in your write-up.
- 3 bonus marks (same for CS686) for performance in the tournament. All agents will be ranked after the tournament by average score per round.
  - Agents ranked between the 25% and 50% of ranks will get 1 bonus mark.
  - Agents ranked the top 25% of ranks will get 2 bonus marks.
  - Agents ranked in the top 10 ranks will and additional bonus mark for a total of 3 bonus marks

  The ranks will include ties, so it is possible for more than $1/2$ of the class to score bonus marks.

So, you can get full marks in this assignment by writing either a basic agent that does really well, or a highly innovative agent that does really poorly. You can score up to 10 marks by implementing something really innovative that beats all but 9 (or less) of the other agents.

In addition, the creator of the two top scoring agents (including those submitted by the instructors!) will each receive a $\$50$ Amazon gift card.

Notes:

- Your code must run with no special libraries or files, be written in Python 2.X, and must be submitted according to the instructions above. If your code does not run, or uses too much memory, your agent may be disqualified. If your agent is disqualified, you may still get full marks for the assignment as described above.
- Your agent **must** be able to play a round in under 5 seconds. If we find some agent that is consistently taking longer than this, that agent may be disqualified.

- Your agent will play the other agents in a random order. For each opponent, your agent will be instantiated **once** at the start. Then, at the start of each game, your agent's `init` method will be called. Your agent therefore can have a memory of all games it has played against an opponent, but it will have no memory across opponents (i.e. its constructor will be called once at the start of all games against an opponent, but its `init` method will be called at the start of each game, so you can re-initialize if you want). Each set of 5 games against each other agent will be played consecutively.
- It may be possible to form coalitions in the game. Although each student must submit his/her own agent, students are not prevented from discussing strategies beforehand, or even from designing agents that collude with each other. However, during the tournament, your agent will not be told which other agent it is playing against, so collusion will require some form of *handshaking*. If you decide to do this, be sure to indicate who you are colluding with in your write-up. Further, the instructor will be submitting 5 agents and these may also take advantage of any coalitions that are formed.
- Your code is not allowed to read/write to the file system or the network. Any code found to be doing this will be disqualified.

References

[1] Christoph Adami and Arend Hintze. Evolutionary instability of zero-determinant strategies demonstrates that winning is not everything. *Nature Communications*, 4, 2013.

[2] R Axelrod and WD Hamilton. The evolution of cooperation. *Science*, 211(4489):1390–1396, 1981.

[3] David M. Kreps, Paul Milgrom, John Roberts, and Robert Wilson. Rational cooperation in the finitely repeated prisoners dilemma. *Journal of Economic Theory*, 27:245–252, 1982.

[4] Roger B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge, Massachussetts, 1991.

[5] William H. Press and Freeman J. Dyson. Iterated prisoners dilemma contains strategies that dominate any evolutionary opponent. *Proceedings of the National Academy of Sciences*, 109(26):10409–10413, 2012.