Prisoner's Dilemma Tournament Regulations

1 Wait!!

Before you continue, read about Prisoner's Dilemma and Iterated Prisoner's Dilemma here.

2 TL;DR

You are supposed to write a bot (in python) to compete against other bots in an iterated prisoner's dilemma. The number of rounds in each match is 1000. Stick to the interface attached in the sent email. Read on for details about the judge system and the scoring.

3 Judge System

3.1 Source Code

The judge system implementation will be made public on `Github` before the tournament. Any concern about the implementation should be reported as a `Github` issue. Contributions are welcome.

3.2 Security

Although we trust you, the judge system will run in a sandboxed environment and algorithms will be checked (by humans) before running.

4 Rules

4.1 Bot

- All submissions are in Python.
- The bot should stick to the interface provided in the email. If you don't know object-oriented programming, this might help.
- You are **not** allowed to use any third-party libraries.

4.2 Participation

- Any person affiliated with E-JUST can participate, including, but not limited to, undergraduate students, graduate students, and faculty members.
- Fill in the attached form If you wish to participate.
- The form **does not** collect anything other than the uploaded python source code. This means that you participation is completely anonymous. Each submission will be assigned an ID that will be made available to you once submitted. You can use this ID to check your result when available.
- If you don't know Python at all but wish to participate, you can send your algorithm (written in full details) to ahmed.salman@ejust.edu.eg. Feel free to send this email anonymously. There is no guarantee that your algorithm will be implemented though (whether sent anonymously or not).
- A Github issue will be created in this repository for each sent algorithm. If you know python, feel free to implement any of the algorithms. The repository includes contribution details.

4.3 Scoring

Rounds will be scored according to the following payoff matrix:

-	Cooperate	Defect
Cooperate	(3, 3)	(0, 5)
Defect	(5, 0)	(1, 1)

The match score will be the sum of scores of individual rounds. The standings will be based on the sum of scores of individual matches. Ties will be broken arbitrarily.

4.4 Matches

4.4.1 General Rules

- Before the start of each match, the judge system will make a new instance of the two competing classes.
- Before each round, the judge will call `instance_name.play()` to get the move of each player. Your move should either be 1 (indicating cooperation) or 0 (indicating defection).
- After each round, the judge will call `instance_name.set_result(result, opponent_move)` to report the result to each player. `result` will either be 0, 1, 3, or 5, depending on the outcome of the round, and `opponent_move` will either be 0 or 1, depending on your opponent's move in that round.

4.4.2 Time Limit

4.4.2.1 Class Instantiation An instance of the class will be created before every match. The time limit for instantiation is 0.5 milliseconds. Failure to instantiate within the time limit will forfeit the match (receive 0 out of 5000).

4.4.2.2 Generating Move The time used for generating a move should not exceed 0.1 milliseconds. Failure to generate a move in the specified time limit will be considered as a `Cooperation`.

4.4.2.3 Receiving Result The time limit for processing the result received from the judge system is 0.1 milliseconds. After the time has passed the function will automatically terminate (if it hadn't already). Failing to terminate within the time limit will not affect the algorithm score from our side, although it might result in future problems.

4.4.3 Errors

Any error raised by your bot will be treated as if it exceeded the time limit.