Welcome to the Prisoner's Dilemma Programming Challenge!

Challenge Description:

In this challenge, you will write code to simulate a prisoner agent that plays the famous Prisoner's Dilemma game. The Prisoner's Dilemma is a classic example of game theory where two prisoners have to decide whether to cooperate or betray each other.

Rules:

1. You will implement a strategy for your prisoner agent, which determines its decision to either cooperate or betray.
2. Your strategy will be called for each round of the game, providing you with information about your previous moves and your opponent's moves.
3. Based on this information, you must decide whether to cooperate or betray.
4. Your goal is to maximize your own score over multiple rounds of the game.
5. The game will be played against other prisoner agents developed by different participants.
6. The winner will be determined based on the cumulative scores obtained throughout the game.

Additional Rule:

7. An optional parameter called "cake" can be provided in each round. If "cake" is present, the round evaluation will be multiplied by five. Otherwise, the evaluation remains standard.

Example: If your score for a round is 5 and "cake" is present as a parameter, your final round score will be 25.

Agent Requirements:

- Your agent should implement the necessary functions or methods as specified in the challenge guidelines.
- It should provide a strategy for making decisions based on the given information.
- The code for your agent should be well-documented and maintainable.
- You are encouraged to be creative and develop effective strategies to outperform your opponents.

Execution:

The call for your bot is made by the run.sh script in your bot's folder. Please ensure that your bot is properly compiled and executed when the ./run.sh command is called. When called, your bot is required to answer with "Cooperate" or "Defect" by printing to the console.

Parameters:

The following parameters are provided for each round:

- Your Bot Name: The name of your prisoner agent.
- Opponent's Name: The name of the opponent prisoner agent.
- Your move history to your opponent. Nonexistent for the first round.
- Your opponent’s move history towards you. Nonexistent for the first round.
- Cake (optional): An optional parameter indicating whether "cake" is present for the current round. Can appear at any time, including the first round.

Move history is provided as a string of ‘**c**’s and ‘**d**’s to represent cooperation and defection respectively.

Example: The parameters “my_name opponents_name ccd cdc” would represent a history in which you answered towards cooperation two times followed by defection in the third, while your opponent cooperated at first, defected second and returned to cooperation in the third round.

Example Console Call:

To run the game and test your prisoner agent, use the following command in the console:

shell

```shell
$ ./run.sh me_in_jail evil_prisoner cake
```

This command will execute the run.sh script with your bot named "me_in_jail" against the opponent bot named "evil_prisoner", with the optional "cake" parameter present for the current round. The absence of history indicates the first round.

shell

```shell
$ ./run.sh me_in_jail evil_prisoner

$ ./run.sh me_in_jail evil_prisoner ccddd cdccd cake
```

Are both valid calls.

Submission Guidelines:

- Submit your code for the prisoner agent as per the challenge instructions.
- Include any necessary instructions for running or testing your code.

Evaluation Criteria:

- Correctness and functionality of your agent's strategy implementation.
- Performance and efficiency of your code.
- Clarity, readability, and maintainability of your code.
- Creativity and effectiveness of your strategies.

- Overall score and ranking achieved in the game.

Best of luck! May the best prisoner agent prevail in this challenging competition. Start coding and have fun!