Game of Trust

This two player game is inspired by the prisoner's dilemma where each player has two options - either to offer 1 coin to a common pool (`trust`) or to not offer (`cheat`). The exact reward function is irrelevant to the problem for now, but let's say that when both players `trust` they both make a profit on their investment, whereas if both players `cheat` they get nothing, and finally if one player `cheats` and other player `trusts`, person who `cheats` makes a larger profit at the expense of the other player who makes a loss.
The solution to an isolated game is usually to `cheat` irrespective of the opponent's strategy, however things get a lot more interesting if players play multiple rounds.

Background

Imagine you walk into a bar and discover a group of people playing the *Game of Trust*. Intrigued, you ask the bartender who introduces you to the game. She also tells you that each player always follows some fixed (unknown) strategy from a pool of strategies (also unknown).
Every game consists of two players playing a variable number of rounds (the game stopping decision after a round is some unknown distribution). Each player plays according to the output of their fixed strategy and communicates their play to the bartender who tallies the rewards and posts the rounds on a board. But being in a noisy environment, sometimes a player's play gets flipped (we call this particular operation a `miscommunication`). You can assume that the `miscommunication` rate is $< 10\%$. Also, when calculating the play in the subsequent round, the players utilize the previous rounds' information from the board that the bartender maintains i.e. `miscommunications` get factored in.
All strategies being used are algorithmic and depend on the previous rounds' information played in the corresponding game only.
One example strategy is:

> Start with move: TRUST, TRUST, CHEAT, CHEAT.
>
> Then if the other player has cheated more than 2 times in the last 4 moves, play CHEAT else play TRUST.

Problem 1: Small world

Being immediately drawn to the game, you set out to discover the number of distinct strategies being followed. You were able to procure a dataset of the past games from the bartender, with the following format:

game_id	p1_id	p2_id	p1_action	p2_action	turn
1	2	0	TRUST	CHEAT	1
1	2	0	TRUST	CHEAT	2
1	2	0	CHEAT	CHEAT	3
1	2	0	CHEAT	TRUST	4
1	2	0	TRUST	TRUST	5
1	2	0	TRUST	TRUST	6
1	2	0	TRUST	TRUST	7
1	2	0	TRUST	CHEAT	8
1	2	0	TRUST	CHEAT	9

where:

- `game_id` is the unique id of a game
- `p1_id` is the id of the player 1
- `p2_id` is the id of the player 2
- `turn` is the turn number within the given game
- `p1_action` is the action of player 1 in this turn
- `p2_action` is the action of player 2 in this turn

Your task is to find out the total number of distinct strategies N being used in the above dataset (ref file `input_game.csv`). To accomplish this, you would need to find ways to characterize how moves are made based on mathematical representation of prior rounds' in this game. We encourage you to think out of the box here about what possible state variables (or features) could influence the next move. When finished, enter the value of N on the problem page.

Problem 2: Birds of a feather

You now set out to find the groups of people who follow the same strategies. You must form a partition of all players P into sets of those playing the same strategy $\{S_1, S_2, \ldots, S_N\}$. Here the strategies need to be ordered from $1 \ldots N$ based on the smallest player id in the set of players playing that strategy. When finished enter the list of lists of $[S_1, S_2, \ldots, S_N]$ on the problem page. Each S_i should also be sorted by player ids. As an example:
[[1, 5, 12, 17, 20], [2, 13], [3, 4, 8], [6, 9, 10, 16, 18, 19], [7, 11, 14, 15]]

Problem 3: Now I see you

Nearing the limit on your tab, you plan to figure out the strategies being followed by players and then play against them. For this section, you are not only incentivized to be able to learn the strategy algorithms but also to interpret them. We encourage you to construct training and test sets for strategy moves based on your features and to make sure your models accurately model each strategy. For bonus marks, for each strategy S_i, enter a brief description of how you think it works on the problem page. Note, that because of the `miscommunication` probability described above you will not be able to get perfect predictions.

Problem 4: Pecking Order

Now that you have figured out (and hopefully reverse engineered) the set of strategies in play, you want to determine which strategy is optimal. For this, design a Monte-Carlo simulation environment where each strategy plays every other strategy (including itself) a fixed number of games (with a variable number of rounds as above). The reward matrix for each round in the game depending on player actions is:

	Trust	Cheat
Trust	$(2, 2)$	$(-1, 3)$
Cheat	$(3, -1)$	$(0, 0)$

When done, enter the relative ranking of strategies (starting from the highest reward accruing to the least) on the problem page.

Problem 5: Dark Forest

Let us now introduce `miscommunication` to the above simulation. Examine the change in the relative ordering as you introduce a `miscommunication` probability increases.
As specified above, you can also imagine `miscommunication` occurs because the two parties are communicating their decision over the noisy bar counter. This process is described in detail in the Background section.
You can try out various settings for `miscommunication` such as $5\%, 10\%, 25\%$. You should see that as `miscommunication` increases certain types of strategies start outperforming. Enter a brief summary of your observations on the problem page.

Problem 6: Master of All

Finally, to tie it all together you try to make a strategy that outperforms all existing strategies around you. This problem is open ended and you may end up proving that its hard to beat some of the existing strategies. Your strategy should be in the form of the following function implementation:

Sample Strategy

```
from enum import Enum
from typing import List

class Action(Enum):
    TRUST = 1
    CHEAT = 2

def strategy(past_self: List[Action],
             past_opponent: List[Action]) -> Action:
    if len(past_self) % 2 == 0:
        return Action.TRUST
    else:
        return Action.CHEAT
```

Enter your python code on the problem page. You can use all standard python libraries.