Trust or Treachery Tournament

Fusion Academy - The Woodlands

Overview

Welcome to the end-of-year Trust or Treachery Tournament! In this project, you will write a Python function that implements a strategy for a special variant of the prisoner's dilemma game. Your function will compete against those of your classmates over a series of rounds. The goal is to accumulate the highest total score based on the payoff matrix. **Critical Thought:** Does this mean you should also aim to minimize your opponent's payout?

Game Description

The prisoner's dilemma is a classic game theory problem. Two players simultaneously choose to either **cooperate (C)** or **defect (D)**. The payoffs for each player are determined as follows:

- Both cooperate: Each player receives 3 points.
- Both defect: Each player receives 1 point.
- One cooperates, one defects: The cooperator receives 0 points, and the defector receives 5 points.

The dilemma arises because mutual cooperation yields the best collective outcome, but individual incentives can lead to defection, resulting in a worse outcome for both players. The game illustrates the conflict between individual rationality and collective rationality.

Enhanced Game Description

To incorporate a real-world phenomenon, we will introduce the possibility of misinterpreting the actions of others. In each round, there will be a 2% chance that a player's intended action will be flipped. This simulates the occasional misunderstanding or miscommunication that can occur in real-life situations.

Noise/Static in Decisions

- There is a 2% chance that the chosen action (cooperate or defect) will be flipped due to external factors.

Number of Rounds

- Each match will consist of 195 to 205 rounds, randomly selected at the start of each match.

Dynamic Payoff Matrix

- Initially, the payoff matrix is as follows:
  - Both cooperate: Each player receives 3 points.
  - Both defect: Each player receives 1 point.
  - One cooperates, one defects: The cooperator receives 0 points, and the defector receives 5 points.
- After 100 rounds of mutual cooperation, the payoff matrix changes to:
  - Both cooperate: Each player receives 4 points.
  - Both defect: Each player receives 1 point.
  - One cooperates, one defects: The cooperator receives 0 points, and the defector receives 10 points.

Nuclear Option

To add another layer of strategy, we introduce the nuclear option. A player can choose to stop the match with the current score and walk away. This option can be useful when dealing with an untrustworthy opponent.

- If one player invokes the nuclear option (returns 'N'), the match ends immediately.
- The player who invokes the nuclear option receives a +5 reputation bonus, while the opponent gets a -5 reputation penalty.
- If both players invoke the nuclear option simultaneously, the match ends immediately, and no reputation changes are made.

Function Signature

You will need to implement a function with the following signature:

```python
def my_prisoner_strategy(my_history: list[str],
                         opponent_history: list[str],
                         round_number: int,
                         relative_score: int,
                         opponent_reputation: int) -> str:
    """
    Determines whether to cooperate, defect, or use the nuclear option.

    Args:
        my_history (list[str]): A list of 'C' (cooperate) or 'D' (defect) representing my past moves
            .
        opponent_history (list[str]): A list of 'C' or 'D' representing the opponent's past moves.
        round_number (int): The current round number.
        relative_score (int): The difference between the player's score and the opponent's score.
        opponent_reputation (int): The opponent's reputation score.

    Returns:
        str: 'C' for cooperate, 'D' for defect, or 'N' for nuclear option.
    """
    pass
```

Example Strategies

Here are some example strategies to get you thinking:

Always Cooperate

```
def always_cooperate(my_history, opponent_history, round_number, relative_score,
     opponent_reputation):
    return 'C'
```

Always Defect

```
def always_defect(my_history, opponent_history, round_number, relative_score, opponent_reputation):
    return 'D'
```

Tit for Tat

```
def tit_for_tat(my_history, opponent_history, round_number, relative_score, opponent_reputation):
    if round_number == 1:
        return 'C'
    else:
        return opponent_history[-1]
```

Score-Based Strategy

```
def score_based_strategy(my_history, opponent_history, round_number, relative_score,
     opponent_reputation):
    if round_number == 1:
        return 'C'
    if relative_score > 0:
        return 'C'
    return 'D'
```

Reputation-Based Strategy

```
def reputation_based_strategy(my_history, opponent_history, round_number, relative_score,
     opponent_reputation):
    if round_number == 1:
        return 'C'
    if opponent_reputation < -10:
        return 'N'
    if relative_score > 0:
        return 'C'
    return 'D'
```

Nuclear Option Strategy

```python
def nuclear_option_strategy(my_history, opponent_history, round_number, relative_score,
     opponent_reputation):
    if round_number == 1:
        return 'C'
    if opponent_history[-3:] == ['D', 'D', 'D']:
        return 'N'
    return 'C' if relative_score > 0 else 'D'
```

Scoring and Leaderboard

Each match will be scored based on the payoff matrix. Your total score from all matches will determine your ranking on the leaderboard. The leaderboard will be updated in real-time, so you can track your progress throughout the tournament.

Guidelines

- Your function must only use the provided arguments (`my_history`, `opponent_history`, `round_number`, `relative_score`, `opponent_reputation`) to make its decision.
- You must document your strategy, explaining your thought process and expected outcomes.
- Be creative and think critically about how to outsmart other strategies.

Implementation Details

The following code snippets outline the main components of the tournament platform:

Dynamic Payoff Matrix

```python
initial_payoff_matrix = {'CC': (3, 3), 'CD': (0, 5), 'DC': (5, 0), 'DD': (1, 1)}
advanced_payoff_matrix = {'CC': (4, 4), 'CD': (0, 10), 'DC': (10, 0), 'DD': (1, 1)}

def get_payoff_matrix(mutual_cooperations):
    """
    Determine the appropriate payoff matrix based on the number of mutual cooperations.

    Args:
        mutual_cooperations (int): The number of rounds both players cooperated.

    Returns:
        dict: The current payoff matrix.
    """
    if mutual_cooperations >= 100:
        return advanced_payoff_matrix
    return initial_payoff_matrix
```

Noise/Static in Decisions

```python
import random

def add_noise(action: str, noise_level: float = 0.02) -> str:
    """
    Introduce noise to the chosen action with a given probability.

    Args:
        action (str): The intended action ('C' or 'D').
        noise_level (float): Probability of action being toggled.

    Returns:
        str: The possibly toggled action.
    """
    if random.random() < noise_level:
        return 'D' if action == 'C' else 'C'
    return action
```

Play Match Function with Static Noise

```python
def play_match(strategy1, strategy2, rounds):
    history1, history2 = [], []
    score1, score2 = 0, 0
    reputation1, reputation2 = 0, 0
    mutual_cooperations = 0
    noise_level = 0.02

    for round_number in range(1, rounds + 1):
        payoff_matrix = get_payoff_matrix(mutual_cooperations)
        relative_score1 = score1 - score2
        relative_score2 = score2 - score1

        move1 = strategy1(history1, history2, round_number, relative_score1, reputation2)
        move2 = strategy2(history2, history1, round_number, relative_score2, reputation1)

        if move1 == 'N' and move2 == 'N':
            # Both choose nuclear option, end the match without reputation change
            break
        elif move1 == 'N':
            # Only player 1 chooses nuclear option, update reputation
            reputation1 += 5
            reputation2 -= 5
            break
        elif move2 == 'N':
            # Only player 2 chooses nuclear option, update reputation
            reputation2 += 5
            reputation1 -= 5
            break

        move1 = add_noise(move1, noise_level)
        move2 = add_noise(move2, noise_level)

        outcome = move1 + move2
        if outcome == 'CC':
            mutual_cooperations += 1
        score1 += payoff_matrix[outcome][0]
        score2 += payoff_matrix[outcome][1]

        history1.append(move1)
        history2.append(move2)

        # Update reputations
        reputation1 = calculate_reputation(history1, history2)
        reputation2 = calculate_reputation(history2, history1)

    return score1, score2, reputation1, reputation2
```

Tournament Execution

```python
def run_tournament(strategies):
    results = {strategy.__name__: {'score': 0, 'reputation': 0} for strategy in strategies}

    for i, strategy1 in enumerate(strategies):
        for j, strategy2 in enumerate(strategies):
            if i != j:
                rounds = random.randint(195, 205)
                score1, score2, reputation1, reputation2 = play_match(strategy1, strategy2, rounds)
                results[strategy1.__name__]['score'] += score1
                results[strategy1.__name__]['reputation'] = reputation1
                results[strategy2.__name__]['score'] += score2
                results[strategy2.__name__]['reputation'] = reputation2

    return results
```

We hope you enjoy this project and learn a lot about game theory and strategic thinking. Good luck!