

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

CZ4046 - Intelligent Agents

Assignment 2 - Three Prisoner's Dilemma

Done By: Mukizhitha Rajkumar (U2020288H)

Table of Contents

Agent's Design.....	3
Forgiveness (Default Cooperation) and Retaliation (Defection Probability) Mechanisms.....	3
Adaptive Decision-Making	3
Exploration-Exploitation Trade-off	3
Mimicry of Successful Strategies.....	3
Probabilistic Decision-Making	4
Agent's Implementation.....	4
Full Pseudocode	4
selectAction ()	6
thresholdUpdate().....	7
calculateForgiveness and calculateRetaliation.....	8
calculateForgiveness:	8
calculateRetaliation:.....	9
Evaluation.....	10
1. NicePlayer Comparison	10
2. NastyPlayer Comparison	10
3. RandomPlayer Comparison.....	10
4. TolerantPlayer Comparison	10
5. FreakyPlayer Comparison.....	10
6. T4TPlayer Comparison	10
Tournament Results	11

Agent's Design

The Rajkumar_Mukizhitha_Player strategy is designed to excel in the Iterated Prisoner's Dilemma (IPD) game by dynamically adapting to opponent behaviour. I developed this agent by incorporating the advantages of the 6 default agents and counter the limitations through enforcing threshold probabilities and more as will be detailed in this report.

Forgiveness (Default Cooperation) and Retaliation (Defection Probability) Mechanisms

Forgiveness and retaliation are important aspects of the Rajkumar_Mukizhitha_Player strategy.

1. Forgiveness enables the player to maintain cooperation with cooperative opponents by forgiving occasional defections.
2. Retaliation serves as a deterrent against repeated defections by opponents.

By dynamically adjusting forgiveness and retaliation probabilities based on observed behaviour, the strategy optimises its response to different opponent strategies.

Adaptive Decision-Making

In the Repeated Prisoner's Dilemma, players engage in repeated interactions, allowing for ongoing strategic adjustments based on feedback from previous rounds. Therefore, by employing an adaptive approach, players can adapt to shifting dynamics, anticipate opponents' tactics, and exploit emerging opportunities over time. This continuous learning and adaptation process not only enhances players' long-term performance but also equips them to counteract opponents' evolving strategies effectively, thereby maintaining competitiveness and adaptability throughout the game.

Exploration-Exploitation Trade-off

Through experimentation and analysis, it became apparent that a certain amount of randomness, implemented through an initial period of exploration in the first 10 rounds of the game.

This phase is essential because it allows the player to gain insights into the behaviour of its adversaries without committing to a fixed course of action. By randomly selecting actions during this exploratory period, the player can observe how opponents respond to different stimuli and identify potential patterns or vulnerabilities in their strategies. This exploration phase balances between exploring new strategies and exploiting successful ones, laying the foundation for effective decision-making in subsequent rounds.

Mimicry of Successful Strategies

Extending from the exploration phase where the agent explores new strategies, the agent also has the propensity to mimic successful opponent strategies. By default, the player mimics the opponent who cooperated in the previous round, thereby adapting to prevailing game conditions. This mimicking behaviour allows the strategy to capitalise on successful opponent strategies, enhancing its own performance over time.

Probabilistic Decision-Making

The strategy introduces an element of randomness through probabilistic decision-making. By incorporating probabilities into forgiveness, retaliation, and exploration, the player adds an unpredictable dimension to its actions. This probabilistic approach prevents opponents from accurately predicting the player's moves, enhancing its strategic versatility.

Agent's Implementation

Full Pseudocode

The figure below shows the pseudocode for the implemented strategy:

```
class Rajkumar_Mukizhitha_Player extends Player:
    forgivenessThreshold = 0.99
    retaliationThreshold = 0.01

    function selectAction(n, myAgentPrevMoves, otherAgent1PrevMoves,
otherAgent2PrevMoves):
        if n < 10 and random() < 0.01:
            return random() < 0.5 ? 0 : 1

        thresholdUpdate(myAgentPrevMoves, otherAgent1PrevMoves,
otherAgent2PrevMoves)

        if n <= 0:
            return 0

        otherAgent1LastMove = otherAgent1PrevMoves[n - 1]
        otherAgent2LastMove = otherAgent2PrevMoves[n - 1]

        forgivenessProbability = calculateForgiveness(myAgentPrevMoves,
otherAgent1PrevMoves, otherAgent2PrevMoves)
        retaliationProbability = calculateRetaliation(myAgentPrevMoves,
otherAgent1PrevMoves, otherAgent2PrevMoves)

        if random() < forgivenessProbability and random() < forgivenessThreshold:
            return 0
        else if random() < retaliationProbability and random() <
retaliationThreshold:
            return 1
        else:
            return otherAgent1LastMove == 0 ? otherAgent1LastMove :
otherAgent2LastMove

    function thresholdUpdate(myAgentPrevMoves, otherAgent1PrevMoves,
otherAgent2PrevMoves):
```

```

        otherAgent1CooperationRate =
calculateCooperationRate(otherAgent1PrevMoves)
        otherAgent2CooperationRate =
calculateCooperationRate(otherAgent2PrevMoves)

        if otherAgent1CooperationRate > 0.7 or otherAgent2CooperationRate > 0.7:
            forgivenessThreshold += 0.05
        else if otherAgent1CooperationRate < 0.3 and otherAgent2CooperationRate <
0.3:
            forgivenessThreshold -= 0.05

        forgivenessThreshold = max(0, min(1, forgivenessThreshold))

        if otherAgent1CooperationRate < 0.3 or otherAgent2CooperationRate < 0.3:
            retaliationThreshold += 0.05
        else if otherAgent1CooperationRate > 0.7 and otherAgent2CooperationRate >
0.7:
            retaliationThreshold -= 0.05

        retaliationThreshold = max(0, min(1, retaliationThreshold))

    function calculateForgiveness(myAgentPrevMoves, oppHistory1, oppHistory2):
        recentDefections = 0
        for i from max(0, length(myAgentPrevMoves) - 10) to
length(myAgentPrevMoves):
            if oppHistory1[i] == 1 or oppHistory2[i] == 1:
                recentDefections++
        return 1.0 - (recentDefections / 10.0)

    function calculateRetaliation(myAgentPrevMoves, otherAgent1PrevMoves,
otherAgent2PrevMoves):
        recentMutualDefections = 0
        for i from max(0, length(myAgentPrevMoves) - 10) to
length(myAgentPrevMoves):
            if myAgentPrevMoves[i] == 1 and (otherAgent1PrevMoves[i] == 1 or
otherAgent2PrevMoves[i] == 1):
                recentMutualDefections++
        return recentMutualDefections / 10.0

    function calculateCooperationRate(history):
        totalCooperation = 0
        for move in history:
            if move == 0:
                totalCooperation++
        return totalCooperation / length(history)

```

selectAction ()

```
class Rajkumar_Mukizhitha_Player extends Player {
    // initial forgiveness threshold: probability that even if other players
    // defect, agent still forgives and cooperates
    double forgivenessThreshold = 0.99;
    double retaliationThreshold = 0.01; // Initial retaliation threshold

    int selectAction(int n, int[] myAgentPrevMoves, int[]
otherAgent1PrevMoves, int[] otherAgent2PrevMoves) {
        // exploration probability refers to the probability that the agent
explores by
        // randomly choosing
        final double EXPLORATION_PROBABILITY = 0.01; // Probability of
exploration

        // check if we're still in the early rounds for exploration
        if (n < 10 && Math.random() < EXPLORATION_PROBABILITY) {
            // if we are in the early rounds and the exploration probability
threshold is
            // met, the agent explores by randomly selecting cooperation or
defection

            return Math.random() < 0.5 ? 0 : 1;
        }
    }
}
```

The main flow of the selectAction method in the Rajkumar_Mukizhitha_Player class is as follows:

1. Exploration Phase: If the current round n is less than 10 and a random number generated falls below the exploration probability threshold, the agent explores by randomly selecting either cooperation or defection.
2. Threshold Update: The method then proceeds to update the forgiveness and retaliation thresholds based on observed behaviour by calling the thresholdUpdate function.
3. Default Action: If the current round number n is invalid (less than or equal to 0), the agent returns a default action of cooperation (0).
4. Analysis of Opponents' Moves: The agent retrieves the last move of each opponent (otherAgent1LastMove and otherAgent2LastMove) from their respective history arrays.
5. Calculation of Probabilities: The agent calculates the forgiveness and retaliation probabilities based on its own history and the history of the other agents.
6. Decision Making: Using the calculated probabilities and the predefined forgiveness and retaliation thresholds, the agent decides its action. If the randomly generated number falls within the forgiveness probability and threshold, the agent cooperates despite opponents' defections. If it falls within the retaliation probability and threshold, the agent defects in retaliation. Otherwise, the agent mimics the opponent who cooperated in the previous round by returning the last move of that opponent.

thresholdUpdate()

```
void thresholdUpdate(int[] myAgentPrevMoves, int[]
otherAgent1PrevMoves, int[] otherAgent2PrevMoves) {
    // first, calculate the cooperation rates of other agents to
    decide whether to
    // increase or decrease the thresholds
    double otherAgent1CooperationRate =
calculateCooperationRate(otherAgent1PrevMoves);
    double otherAgent2CooperationRate =
calculateCooperationRate(otherAgent2PrevMoves);

    // if other agent cooperated more than it defected then increase
    forgiveness
    if (otherAgent1CooperationRate > 0.7 || otherAgent2CooperationRate
    > 0.7) {
        forgivenessThreshold += 0.05; // Increase forgiveness
        threshold
        // if not decrease the threshold
    } else if (otherAgent1CooperationRate < 0.3 &&
otherAgent2CooperationRate < 0.3) {
        forgivenessThreshold -= 0.05; // Decrease forgiveness
        threshold
    }

    // need to make sure threshold stays within bounds of 0 to 1
    forgivenessThreshold = Math.max(0, Math.min(1,
    forgivenessThreshold));

    // likewise, update the retaliation threshold but the logic is
    opposite
    if (otherAgent1CooperationRate < 0.3 || otherAgent2CooperationRate
    < 0.3) {
        // increasing the retaliation threshold by a little
        retaliationThreshold += 0.05;
    } else if (otherAgent1CooperationRate > 0.7 &&
otherAgent2CooperationRate > 0.7) {
        // decreasing the retaliation threshold by a little
        retaliationThreshold -= 0.05;
    }
    // need to make sure threshold stays within bounds of 0 to 1
    retaliationThreshold = Math.max(0, Math.min(1,
    retaliationThreshold));
}
```

The thresholdUpdate method is responsible for dynamically updating the forgiveness and retaliation thresholds based on the observed behaviour of the other agents.

1. Calculation of Cooperation Rates: The method first calculates the cooperation rates of the two other agents (`otherAgent1CooperationRate` and `otherAgent2CooperationRate`). This rate represents the proportion of times these agents cooperated in their previous moves.
2. Adjustment of Forgiveness Threshold: Depending on the cooperation rates calculated, the forgiveness threshold is adjusted. If either of the other agents cooperated more than they defected (cooperation rate > 0.7), the forgiveness threshold is increased by 0.05, allowing the player to forgive more often. Conversely, if both agents cooperated less frequently (cooperation rate < 0.3), the forgiveness threshold is decreased by 0.05, making the player less forgiving.
3. Bound Checking: After adjusting the forgiveness threshold, it is ensured that the threshold value stays within the bounds of 0 to 1.
4. Adjustment of Retaliation Threshold: Similar to the forgiveness threshold, the retaliation threshold is adjusted based on the opposite logic. If either of the other agents cooperated less frequently, indicating potential aggression, the retaliation threshold is increased slightly. Conversely, if both agents cooperated more often, indicating cooperation, the retaliation threshold is decreased slightly.
5. Bound Checking: Again, after adjusting the retaliation threshold, it is ensured that the threshold value remains within the bounds of 0 to 1.

calculateForgiveness and calculateRetaliation

calculateForgiveness:

```
double calculateForgiveness(int[] myAgentPrevMoves, int[] oppHistory1,
int[] oppHistory2) {
    // Count the number of recent defections by opponents
    int recentDefections = 0;
    for (int i = Math.max(0, myAgentPrevMoves.length - 10); i <
myAgentPrevMoves.length; i++) {
        if (oppHistory1[i] == 1 || oppHistory2[i] == 1) {
            recentDefections++;
        }
    }
    // Calculate forgiveness probability based on recent defections
    return 1.0 - (double) recentDefections / 10.0;
}
```

This function determines the forgiveness probability based on the recent history of the player and the opponents.

1. It counts the number of recent defections by the opponents within the last 10 rounds.
2. For each round within the recent history, it checks whether either opponent defected (i.e., played a move represented by 1).
3. The function then calculates the forgiveness probability as the complement of the fraction of recent rounds in which opponents defected. This value represents the likelihood that the player will forgive the opponents despite their recent defections.
4. The forgiveness probability is returned, with a value ranging from 0 to 1, where 0 indicates no forgiveness and 1 indicates complete forgiveness.

calculateRetaliation:

```
double calculateRetaliation(int[] myAgentPrevMoves, int[]
otherAgent1PrevMoves, int[] otherAgent2PrevMoves) {
    int recentMutualDefections = 0;
    for (int i = Math.max(0, myAgentPrevMoves.length - 10); i <
myAgentPrevMoves.length; i++) {
        if (myAgentPrevMoves[i] == 1 && (otherAgent1PrevMoves[i] == 1
|| otherAgent2PrevMoves[i] == 1)) {
            recentMutualDefections++;
        }
    }
    // calculate retaliation probability based on recent mutual
defections
    return (double) recentMutualDefections / 10.0;
}
```

This function calculates the retaliation probability based on the recent mutual defections between the player and the opponents.

1. It counts the number of recent rounds where both the player and at least one opponent defected.
2. For each round within the recent history, it checks whether both the player and at least one opponent defected.
3. The function then calculates the retaliation probability as the fraction of recent rounds with mutual defections.
4. The retaliation probability is returned, with a value ranging from 0 to 1, indicating the likelihood that the player will retaliate by defecting in response to mutual defections by the opponents.

This function provides insight into the player's inclination to retaliate against aggressive behaviour exhibited by the opponents, contributing to the adaptive nature of the player's strategy.

Evaluation

Let's compare the `Rajkumar_Mukizhitha_Player` to each of the other players:

1. NicePlayer Comparison

Compared to `NicePlayer`, `Rajkumar_Mukizhitha_Player` is more adaptive. While `NicePlayer` always cooperates, `Rajkumar_Mukizhitha_Player` adjusts its strategy based on opponents' behaviour. This adaptability allows `Rajkumar` to respond strategically to different opponents, potentially achieving higher payoffs in scenarios where cooperation is not always the best option.

2. NastyPlayer Comparison

`Rajkumar_Mukizhitha_Player` outperforms `NastyPlayer` by being more cooperative when beneficial. While `NastyPlayer` always defects, `Rajkumar` forgives and cooperates under certain conditions, which can lead to better outcomes in scenarios where cooperation yields higher rewards or fosters cooperation from opponents.

3. RandomPlayer Comparison

Unlike `RandomPlayer`, which selects actions randomly, `Rajkumar`'s decisions are based on observed opponent behavior and adaptive thresholds. This allows `Rajkumar` to exhibit more nuanced behavior, strategically adjusting its actions to promote cooperation or retaliate against defections, potentially achieving higher payoffs than `RandomPlayer`'s unpredictable strategy.

4. TolerantPlayer Comparison

Compared to `TolerantPlayer`, `Rajkumar_Mukizhitha_Player`'s strategy is more dynamic and adaptive. While `TolerantPlayer`'s strategy is based solely on a fixed threshold of opponent defections, `Rajkumar_Mukizhitha_Player` continuously updates its forgiveness and retaliation thresholds based on observed opponent behaviour, allowing for more nuanced and effective decision-making in diverse scenarios.

5. FreakyPlayer Comparison

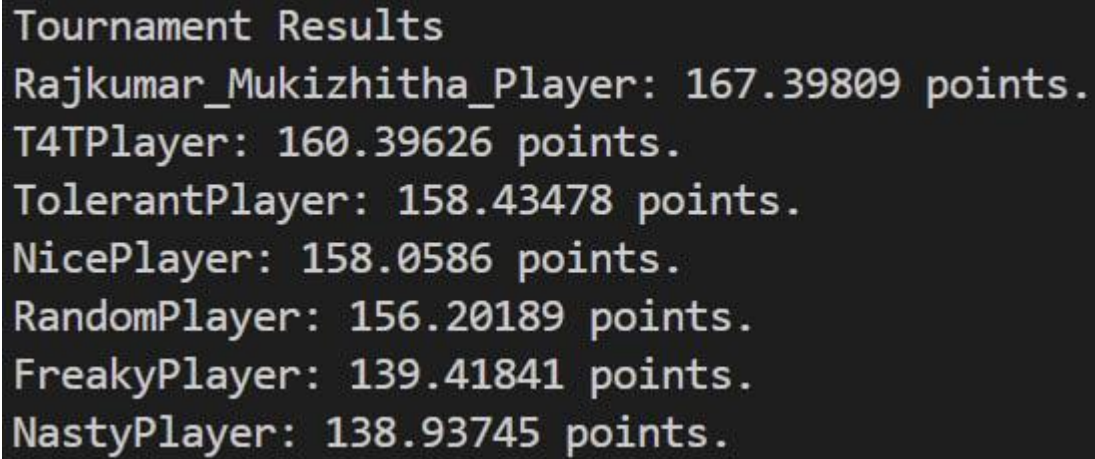
`Rajkumar_Mukizhitha_Player` adaptive strategy outperforms `FreakyPlayer`'s randomized approach. While `FreakyPlayer` randomly chooses between being nice and nasty, `Rajkumar` dynamically adjusts its behaviour based on observed opponent actions, potentially achieving higher payoffs by strategically forgiving or retaliating against opponents.

6. T4TPlayer Comparison

`Rajkumar_Mukizhitha_Player`'s adaptive strategy offers advantages over `T4TPlayer`'s tit-for-tat approach. While `T4TPlayer` mimics opponents' previous moves, `Rajkumar_Mukizhitha_Player` considers forgiveness and retaliation thresholds, allowing for more strategic responses. `Rajkumar_Mukizhitha_Player` can forgive or retaliate based on opponents' behaviour patterns, potentially achieving better outcomes in complex game scenarios.

Tournament Results

I compared the performance of the agent I created against the 6 default agents and got the results displayed in the figure below:



```
Tournament Results
Rajkumar_Mukizhitha_Player: 167.39809 points.
T4TPlayer: 160.39626 points.
TolerantPlayer: 158.43478 points.
NicePlayer: 158.0586 points.
RandomPlayer: 156.20189 points.
FreakyPlayer: 139.41841 points.
NastyPlayer: 138.93745 points.
```

As shown in the figure, the custom agent Rajkumar_Mukizhitha_Player outperformed the 6 default agents.