Nanyang Technological University

College of Computing and Data Science

SC4003 CE/CZ4046 INTELLIGENT AGENTS

Assignment 2

Name	Matric No.
Lian Hong Shen Jordan	U2122011E

Contents

Repeated Prisoners Dilemma

This report will serve to cover the problem in Repeated Prisoner's Dilemma. In Prisoner's Dilemma, the dominant strategy equilibrium is for agents to defect even though both agents would be best off cooperating. However, when the game is played repeatedly, cooperation becomes more viable. In a repeated setting, players have the opportunity to establish trust and develop mutual beneficial strategies over time. When agents know they will meet again in future interactions, the incentive to defect diminishes significantly. This is because short-term gains from defection can be outweighed by the long-term costs of retaliation in subsequent rounds.

The setting up of the environment along with the solution to the problem was implemented in java.

1. Ensure java is installed, either from OpenJDK, https://openjdk.org/, or OracleJDK, https://www.oracle.com/java/. The java version I used is OpenJDK 23.0.2 2025-01-2.
2. To run any java file (in this case **ThreePrisonersDilemma.java)** , in unix terminal, type **javac ThreePrisonersDilemma.java** and then **java ThreePrisonersDilemma**.

I will briefly describe what each the files' function and also the files they output if any:

- **ThreePrisonersDilemma.java**: This file was as defined as given to us in this assignment, with extra custom strategies I have implemented.
- **ThreePrisonersDilemmaExperiment.java**: This file runs 50 matches with the extra strategies I have implemented. It saves outputs to four files under a new "experiment_results" directory.
    - **experiment_results/tournament_log.txt**: A log file that records the outcome of every match in all 50 tournaments, including individual matchup scores and final rankings.
    - **experiment_results/average_scores_bar_chart.png**: A bar chart that shows the average score for each strategy across all 50 matches, sorted in descending order of performance.
    - **experiment_results/consolidated_scores.csv**: A spreadsheet file containing raw performance data for each strategy across all 50 matches, plus their average scores.
    - **experiment_results/strategy_scores.png**: A line graph showing how each strategy's performance varied across the 50 matches.
- **ThreePrisonersDilemmaExperimentDefect.java**: This is similar to that of the experiment file but replaces most strategies with defectors like (NastyPlayers) to analyse how the competitive environment changes when defection becomes the dominant strategy. It creates similar output files to **ThreePrisonersDilemmaExperiment.java**, but saves them to a separate "experimentdefect_results" directory.
    - **experimentdefect_results/tournament_log.txt**: Similar to that **experiment_results/tournament_log.txt** but with majority defectors.
    - **experimentdefect_results /consolidated_scores.csv**: Similar to that **experiment_results/consolidated_scores.csv** but with majority defectors.
    - **experimentdefect_results /average_scores_bar_chart.png**: Similar to that **experiment_results/average_scores_bar_chart.png** but with majority defectors.
    - **experimentdefect_results /strategy_scores.png**: Similar to that **experiment_results/strategy_scores.png** but with majority defectors.
- **Lian_HongShenJordan_Player.java**: This file is the single class best strategy that I have found based on the findings below.

Strategies in **ThreePrisonersDilemma.java**

I have implemented 12 strategies in **ThreePrisonersDilemma.java** to compare which approaches perform best.

1. SuspiciousT4TPlayer

```
class SuspiciousT4TPlayer extends Player {
    // Impelments the Suspicious Tit-for-tat strategy
    // Picks a random opponent at each play,
    // and uses the 'tit-for-tat' strategy against them
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 1; // Defect on first round
        if (Math.random() < 0.5)
            return oppHistory1[n - 1];
        else
            return oppHistory2[n - 1];
    }
}
```

SuspiciousT4TPlayer is exactly the same as the demo strategy that was introduced in the original **ThreePrisonersDilemma.java** file which picks a random opponent at each play and uses Tit-for-Tat against them, but it defects on the first round instead of cooperating.

2. StandardT4TPlayer

```
class StandardT4TPlayer extends Player {
    // Implements the Standard Tit-for-tat strategy
    // Does not pick a random opponent at each play, considers both opponents
    // defects if either of them defected in the previous round
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 0; // Cooperate on first round

        // Defect if any opponent defected in the previous round
        if (oppHistory1[n - 1] == 1 || oppHistory2[n - 1] == 1) {
            return 1;
        }

        return 0; // Otherwise cooperate
    }
}
```

StandardT4TPlayer implements the Standard Tit-for-Tat strategy. Unlike the original T4TPlayer which picks a random opponent at each play, this strategy considers both opponents simultaneously. It cooperates on the first round, then defects if either opponent defected in the previous round.

3. SuspiciousStandardT4TPlayer

```
class SuspiciousStandardT4TPlayer extends Player {
    // Implements the Suspicious Standard Tit-for-tat strategy
    // Does not pick a random opponent at each play, considers both opponents
    // defects if either of them defected in the previous round
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 1; // Defect on first round

        // Defect if any opponent defected in the previous round
        if (oppHistory1[n - 1] == 1 || oppHistory2[n - 1] == 1) {
            return 1;
        }

        return 0; // Otherwise cooperate
    }
}
```

SuspiciousStandardT4TPlayer is similar to StandardT4TPlayer but starts with defection instead of cooperation. This strategy considers both opponents simultaneously. It defects on the first round, then defects if either opponent defected in the previous round.

4. GenerousT4TPlayer

```
class GenerousT4TPlayer extends Player {
    // Implements the Generous Tit-for-tat strategy
    // Similar to Standard T4T but with min{1- (T-R)/(R-S) , (R-P)/(T-P)} probability of forgiving defections
    // R (Reward) = 6 (payoff[0][0][0]), P (Punishment) = 2 (payoff[1][1][1]), T (Temptation) = 8 (payoff[1][0][0]), S (Sucker) = 0 (payoff[0][1][1])
    // Math.min(1 - (payoff[1][0][0] - payoff[0][0][0]) / (payoff[0][0][0] - payoff[0][1][1]),(payoff[0][0][0] - payoff[1][1][1]) / (payoff[1][0][0] - payoff[1][1][1]))
    // Math.min(1 - (8-6)/(6-0), (6-2)/(8-2)) = Math.min(1 - 2/6, 4/6) = Math.min(2/3, 2/3) = 2/3
    double gCooperate = 2 / 3;

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 0; // Cooperate on first round

        // If any opponent defected in the previous round
        if (oppHistory1[n - 1] == 1 || oppHistory2[n - 1] == 1) {
            // 2/3 chance to forgive and cooperate anyway
            if (Math.random() < gCooperate) {
                return 0;
            } else {
                return 1; // Defect to retaliate
            }
        }

        return 0; // Both opponents cooperated, so cooperate
    }
}
```

GenerousT4TPlayer implements the Generous Tit-for-Tat strategy. It extends StandardT4TPlayer but adds forgiveness - even when opponents defect, it has a 2/3 probability of cooperating anyway.

5. JossPlayer

```
class JossPlayer extends Player {
    // Implements the Joss strategy
    // Similar to Standard T4T but once in a while, defects randomly
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 0; // Cooperate on first round

        // If any opponent defected, defect
        if (oppHistory1[n - 1] == 1 || oppHistory2[n - 1] == 1) {
            return 1;
        }

        // 10% chance to defect even when opponents cooperated
        if (Math.random() < 0.1) {
            return 1;
        }

        return 0; // Otherwise cooperate
    }
}
```

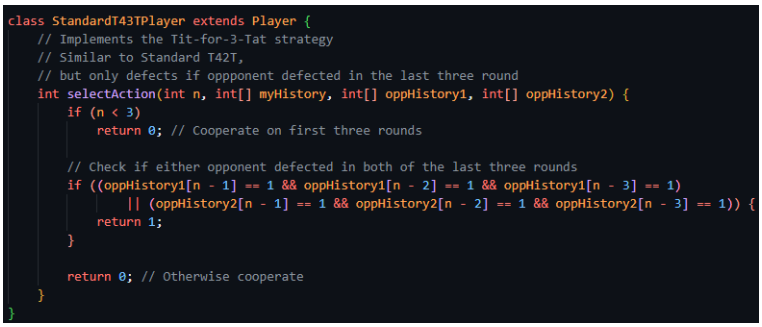
JossPlayer implements the Joss strategy, a variant of StandardT4TPlayer with occasional unpredictable defection. It cooperates on the first round and defects if any opponent defected in the previous round, like StandardT4TPlayer. However, it introduces a 10% random chance to defect even when both opponents cooperated.

6. StandardT42TPlayer

```
class StandardT42TPlayer extends Player {
    // Implements the Tit-for-2-Tat strategy
    // Similar to Standard T4T,
    // but only defects if oppponent defected in the last two round
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n < 2)
            return 0; // Cooperate on first two rounds

        // Check if either opponent defected in both of the last two rounds
        if ((oppHistory1[n - 1] == 1 && oppHistory1[n - 2] == 1)
                || (oppHistory2[n - 1] == 1 && oppHistory2[n - 2] == 1)) {
            return 1;
        }

        return 0; // Otherwise cooperate
    }
}
```

StandardT42TPlayer implements the Tit-for-2-Tat strategy. Similar to StandardT4T, but with increased forgiveness - it only defects if the same opponent defected in both of the last two rounds.

7. StandardT43TPlayer



```
class StandardT43TPlayer extends Player {
    // Implements the Tit-for-3-Tat strategy
    // Similar to Standard T42T,
    // but only defects if oppponent defected in the last three round
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n < 3)
            return 0; // Cooperate on first three rounds

        // Check if either opponent defected in both of the last three rounds
        if ((oppHistory1[n - 1] == 1 && oppHistory1[n - 2] == 1 && oppHistory1[n - 3] == 1)
                || (oppHistory2[n - 1] == 1 && oppHistory2[n - 2] == 1 && oppHistory2[n - 3] == 1)) {
            return 1;
        }

        return 0; // Otherwise cooperate
    }
}
```

StandardT43TPlayer implements the Tit-for-3-Tat strategy. Similar to StandardT42T, but with even greater forgiveness - it only defects if the same opponent defected in all three of the last rounds.

8. StandardT44TPlayer

```
class StandardT44TPlayer extends Player {
    // Implements the Tit-for-4-Tat strategy
    // Similar to Standard T42T,
    // but only defects if oppponent defected in the last four round
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n < 4)
            return 0; // Cooperate on first four rounds

        // Check if either opponent defected in both of the last four rounds
        if ((oppHistory1[n - 1] == 1 && oppHistory1[n - 2] == 1 && oppHistory1[n - 3] == 1
                && oppHistory1[n - 4] == 1)
                || (oppHistory2[n - 1] == 1 && oppHistory2[n - 2] == 1 && oppHistory2[n - 3] == 1
                        && oppHistory2[n - 4] == 1)) {
            return 1;
        }

        return 0; // Otherwise cooperate
    }
}
```

StandardT44TPlayer implements the Tit-for-4-Tat strategy. This is the most forgiving variant in the T4T family, only defecting if the same opponent defected in all four of the last rounds.

9. TesterPlayer

```
class TesterPlayer extends Player {
    // Implements the Tester strategy
    // Tests opponents by defecting on the first round if opponent defects, switch to Standard T4T
    // if opponent cooperates, exploit by mixing cooperation and defection exploit every 5 rounds
    boolean retaliationDetected = false;

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 1; // Defect on first round

        // Check if any opponent retaliated to our initial defection
        if (n == 1 && (oppHistory1[0] == 1 || oppHistory2[0] == 1)) {
            retaliationDetected = true;
        }

        if (retaliationDetected) {
            // Use standard Tit-for-Tat
            if (oppHistory1[n - 1] == 1 || oppHistory2[n - 1] == 1) {
                return 1;
            } else {
                return 0;
            }
        } else {
            // No retaliation detected, periodically defect
            if (n % 5 == 0) { // Defect every 5 rounds
                return 1;
            } else {
                return 0;
            }
        }
    }
}
```

TesterPlayer implements the Tester strategy, which probes opponents' reactions and adjusts accordingly. It defects on the first round to test opponents, then observes their response. If any opponent retaliates with defection, it switches to StandardT4T behaviour. However, if opponents continue to cooperate despite the initial defection (no retaliation detected), it exploits this weakness by periodically defecting every 5 rounds while cooperating in between.

10.PavlovPlayer

```
class PavlovPlayer extends Player {
    // Implements the Pavlov Strategy
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 0; // Cooperate on first round

        // Get previous payoff
        int lastPayoff = ThreePrisonersDilemma.payoff[myHistory[n - 1]][oppHistory1[n - 1]][oppHistory2[n - 1]];

        // If payoff was good (≥ 5), stick with previous move
        if (lastPayoff >= 5) {
            return myHistory[n - 1];
        } else {
            // Otherwise change move
            return 1 - myHistory[n - 1];
        }
    }
}
```

PavlovPlayer implements the Pavlov Strategy. It cooperates on the first round, then bases future moves on the payoff received. If the previous round's payoff was good (≥ 5), it repeats its last move. If the payoff was poor (< 5), it switches to the opposite move. Setting the threshold at 5 creates a natural division as the

- Good outcomes (≥ 5) - should repeat the same action:
    - DCC (8): Successfully exploiting cooperators
    - CCC (6): Mutual cooperation
    - DCD/DDC (5): Defecting with one other defector
- Bad outcomes (< 5) - should change action:
    - CCD/CDC (3): Being partially exploited
    - DDD (2): Everyone defecting
    - CDD (0): Being fully exploited

This creates a "win-stay, lose-shift" pattern that rewards successful moves and changes unsuccessful ones.

11.TriggerPlayer

```
class TriggerPlayer extends Player {
    // Implements the Trigger/Grim/Friedman strategy
    // This strategy cooperates until any opponent defects, then defects forever
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 0; // Cooperate on first round

        // Check if any opponent has ever defected
        for (int i = 0; i < n; i++) {
            if (oppHistory1[i] == 1 || oppHistory2[i] == 1) {
                return 1; // Defect forever if betrayed
            }
        }

        return 0; // Otherwise cooperate
    }
}
```

TriggerPlayer implements the Trigger/Grim/Friedman strategy, characterized by extreme punishment for defection. It cooperates until any opponent defects even once, then defects permanently for all remaining rounds.

12.EndGameDefactorPlayer

```
class EndGameDefactorPlayer extends Player {
    // Implements the End Game Defector strategy
    // This strategy cooperates with Standard TFT
    // until the last 10 rounds, then defects
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 0; // Cooperate on first round

        // Defect in the last ~10 rounds (we know there are ~90-110 rounds)
        if (n >= 100) {
            return 1;
        }

        // Use Tit-for-Tat strategy before the end
        if (oppHistory1[n - 1] == 1 || oppHistory2[n - 1] == 1) {
            return 1;
        }

        return 0; // Otherwise cooperate
    }
}
```

EndGameDefactorPlayer implements the End Game Defector strategy, which exploits game length knowledge. It cooperates using Standard TFT strategy for most of the game but defects in approximately the last 10 rounds (after round 100), since we know that there are approximately ~90-110 rounds.

I have done 3 runs with all 18 agents to get the performance of these strategies against each other.

```
Tournament Results
StandardT42TPlayer: 922.17145 points.
StandardT43TPlayer: 918.26904 points.
TolerantPlayer: 906.70654 points.
StandardT44TPlayer: 900.61035 points.
GenerousT4TPlayer: 870.1021 points.
T4TPlayer: 865.839 points.
StandardT4TPlayer: 863.926 points.
EndGameDefactorPlayer: 862.13654 points.
TriggerPlayer: 847.9173 points.
NicePlayer: 834.4522 points.
PavlovPlayer: 799.11426 points.
SuspiciousT4TPlayer: 697.51904 points.
FreakyPlayer: 697.297 points.
JossPlayer: 677.3991 points.
SuspiciousStandardT4TPlayer: 649.8887 points.
TesterPlayer: 633.59344 points.
RandomPlayer: 564.7026 points.
NastyPlayer: 561.4298 points.
```

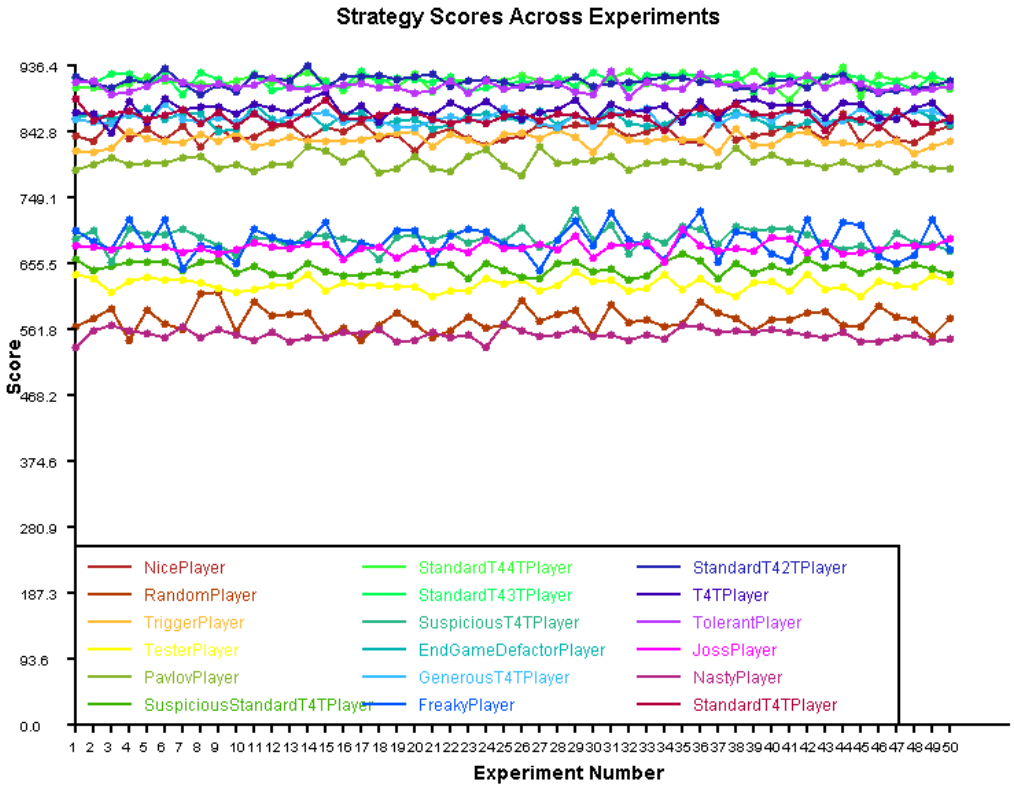
```
Tournament Results
StandardT42TPlayer: 931.3047 points.
StandardT43TPlayer: 916.5067 points.
StandardT44TPlayer: 910.42505 points.
TolerantPlayer: 892.7815 points.
T4TPlayer: 880.39185 points.
StandardT4TPlayer: 877.3808 points.
GenerousT4TPlayer: 865.7948 points.
EndGameDefactorPlayer: 854.79926 points.
TriggerPlayer: 839.69196 points.
NicePlayer: 834.07605 points.
PavlovPlayer: 799.2872 points.
SuspiciousT4TPlayer: 700.6246 points.
JossPlayer: 685.98553 points.
FreakyPlayer: 662.5422 points.
SuspiciousStandardT4TPlayer: 643.60614 points.
TesterPlayer: 628.70044 points.
RandomPlayer: 594.6605 points.
NastyPlayer: 565.5234 points.
```

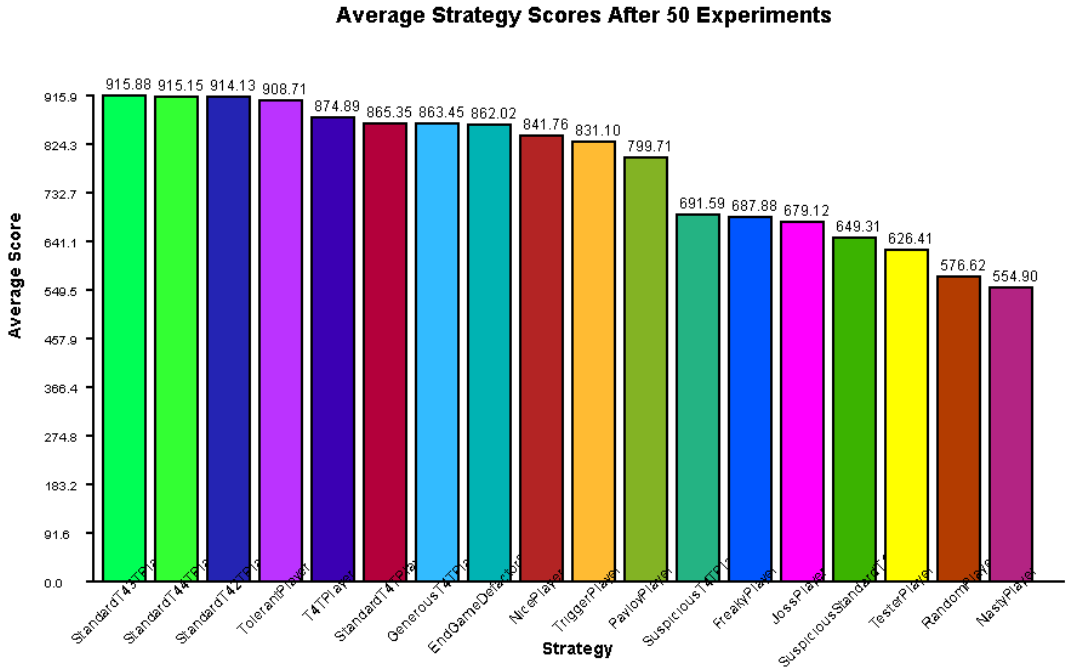
```
Tournament Results
StandardT42TPlayer: 921.4474 points.
StandardT44TPlayer: 914.2729 points.
StandardT43TPlayer: 906.6582 points.
TolerantPlayer: 897.76074 points.
EndGameDefactorPlayer: 884.02545 points.
T4TPlayer: 866.1069 points.
StandardT4TPlayer: 862.0401 points.
NicePlayer: 859.2833 points.
GenerousT4TPlayer: 858.0758 points.
TriggerPlayer: 831.49744 points.
PavlovPlayer: 800.7663 points.
SuspiciousT4TPlayer: 702.2917 points.
FreakyPlayer: 699.57654 points.
JossPlayer: 668.6528 points.
SuspiciousStandardT4TPlayer: 637.1175 points.
TesterPlayer: 630.38477 points.
NastyPlayer: 554.2854 points.
RandomPlayer: 547.8592 points.
```

In the 3 separate runs, we can see that StandardT42TPlayer has gotten the highest score across all matches. This consistent dominance of StandardT42TPlayer suggests that a moderate level of forgiveness provides an optimal balance between cooperation and protection against exploitation in this three-player environment.

Results from in **ThreePrisonersDilemmaExperiment.java**

To better understand the performance of the agents in longer matches, I have created **ThreePrisonersDilemmaExperiment.java** which runs 50 matches with all implemented strategies and records comprehensive statistics. The program generates detailed analytics including tournament logs, average scores across all matches, and visualizations showing how each strategy performs over time.





We notice that StandardT34TPlayer scored the highest when averaged across 50 runs, closely followed by StandardT44TPlayer and StandardT42TPlayer. The consistent top performance of these forgiving Tit-for-Tat variants demonstrates the importance of extended forgiveness in multi-player repeated games. StandardT43TPlayer's success illustrates that in environments with multiple interacting agents, strategies that can maintain cooperative relationships despite occasional defections ultimately accumulate greater benefits. In the three-player repeated Prisoner's Dilemma, the uncertainty about when the game will end encourages sustained cooperation. Strategies that can both establish and maintain cooperative relationships while protecting against systematic exploitation perform best. The T4nT family's superior performance over standard T4T shows that allowing for some "mistakes" or noise in opponent behaviour without immediate retaliation creates more stable and profitable long-term interactions.

Results from in **ThreePrisonersDilemmaExperimentDefect.java**

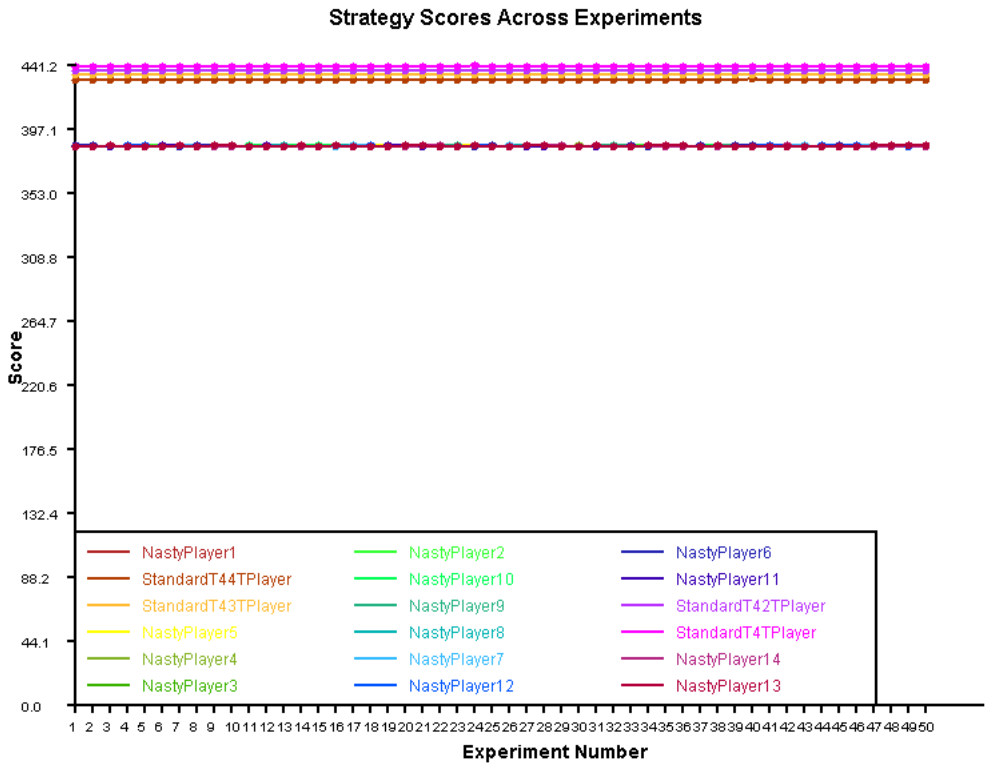
To test how environmental conditions affect strategy performance, I created **ThreePrisonersDilemmaExperimentDefect.java** to simulate a defection-dominant environment. I replaced most strategies with NastyPlayers (always defect), while retaining the T4nT family strategies to see how they would perform when surrounded by defectors.

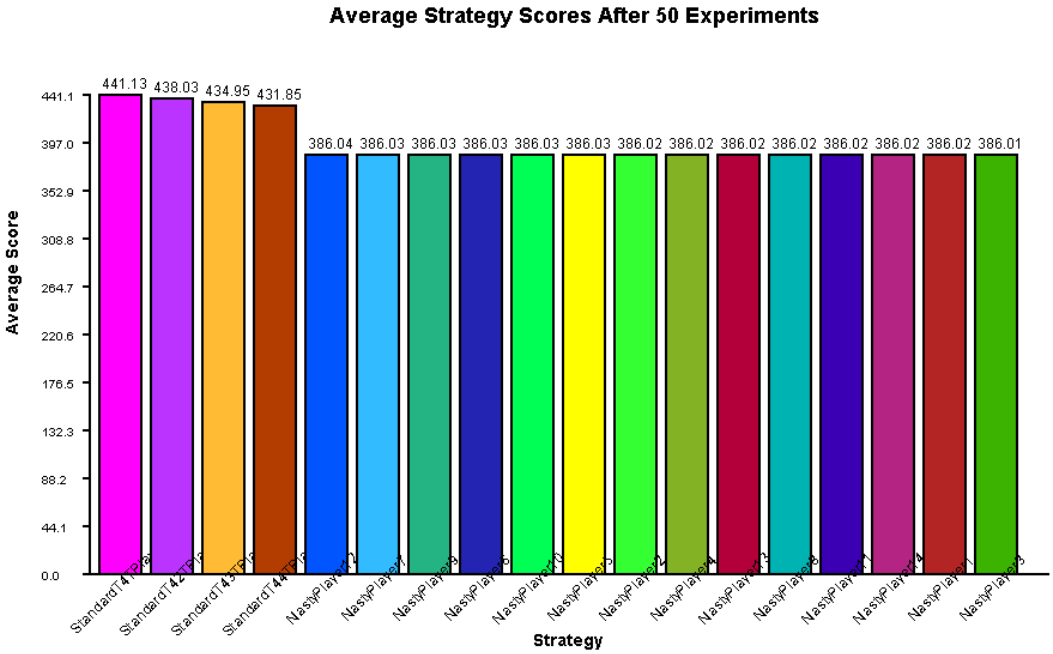
```
Player makePlayer(int which) {
    switch (which) {
    case 0:
        return new NamedNastyPlayer(name:"NastyPlayer1");
    case 1:
        return new NamedNastyPlayer(name:"NastyPlayer2");
    case 2:
        return new NamedNastyPlayer(name:"NastyPlayer3");
    case 3:
        return new NamedNastyPlayer(name:"NastyPlayer4");
    case 4:
        return new NamedNastyPlayer(name:"NastyPlayer5");
    case 5:
        return new NamedNastyPlayer(name:"NastyPlayer6");
    case 6:
        return new NamedNastyPlayer(name:"NastyPlayer7");
    case 7:
        return new NamedNastyPlayer(name:"NastyPlayer8");
    case 8:
        return new NamedNastyPlayer(name:"NastyPlayer9");
    case 9:
        return new NamedNastyPlayer(name:"NastyPlayer10");
    case 10:
        return new StandardT4TPlayer();
    case 11:
        return new StandardT42TPlayer();
    case 12:
        return new StandardT43TPlayer();
    case 13:
        return new StandardT44TPlayer();
    case 14:
        return new NamedNastyPlayer(name:"NastyPlayer11");
    case 15:
        return new NamedNastyPlayer(name:"NastyPlayer12");
    case 16:
        return new NamedNastyPlayer(name:"NastyPlayer13");
    case 17:
        return new NamedNastyPlayer(name:"NastyPlayer14");
    }

    throw new RuntimeException(message:"Bad argument passed to makePlayer");
}
```

```
abstract class Player {
    // This procedure takes in the number of rounds elapsed so far (n), and
    // the previous plays in the match, and returns the appropriate action.
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        throw new RuntimeException(message:"You need to override the selectAction method.");
    }

    // Used to extract the name of this player class.
    String name() {
        String result = getClass().getName();
        return result.substring(result.indexOf(ch:'$') + 1);
    }
}
```

```
class NamedNastyPlayer extends Player {
    private final String customName;

    // Constructor that takes a custom name
    public NamedNastyPlayer(String name) {
        this.customName = name;
    }

    // Override the name() method to return the custom name
    @Override
    String name() {
        return customName;
    }

    // NastyPlayer always defects
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        return 1;
    }
}
```

Keeping the same number of 18 agents, I maintained StandardT4TPlayer, StandardT42TPlayer, StandardT43TPlayer, and StandardT44TPlayer while replacing the other 14 strategies with NastyPlayers (named NastyPlayer1 through NastyPlayer14). This allowed me to observe how varying levels of forgiveness would perform in a highly adversarial environment.



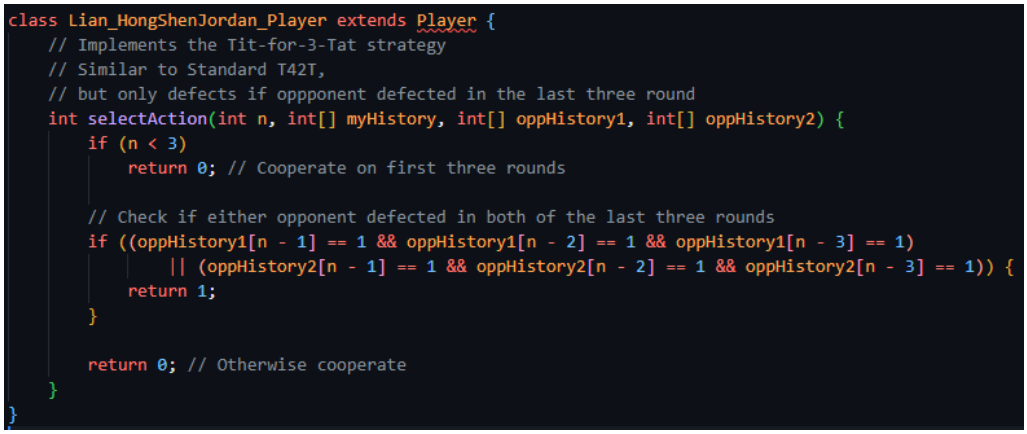


We notice that the T4T variants performed better than the larger T4nT family because in a defection-dominant environment, excessive forgiveness becomes a liability rather than an advantage. As shown in both figures, StandardT4TPlayer achieved the highest average score followed by StandardT42TPlayer, while StandardT43TPlayer and StandardT44TPlayer performed worse as forgiveness increased. This clear pattern demonstrates that when surrounded by consistently hostile agents, quickly retaliating against defection while still maintaining the ability to return to cooperation is the optimal strategy. The performance gap between these four strategies and all NastyPlayers highlights that even in highly adversarial scenarios, having some capacity for conditional cooperation remains advantageous, though with much less forgiveness than in mixed environments.

Conclusion for **Lian_HongShenJordan_Player.java**

After the results from the testing across the different scenarios, I chose to use the StandardT43TPlayer (Tit-for-3-Tat) strategy as my final solution. This was based on several key observations:

1. In cooperative environments with mixed strategies (as shown in **ThreePrisonersDilemmaExperiment.java** results), StandardT43TPlayer consistently performed at the top tier, often outperforming other strategies including its T4nT variants.
2. While StandardT4TPlayer performed best in highly defective environments (as demonstrated in **ThreePrisonersDilemmaExperimentDefect.java**), this scenario is less likely in an academic setting where students understand game theory principles.
3. For this Intelligent Agents course, most students will recognize that mutual cooperation yields higher collective outcomes. In such an environment, a more forgiving strategy like T43T provides the optimal balance - it maintains cooperative relationships despite occasional defections while still protecting against systematic exploitation.
4. The choice of forgiving exactly 3 defections present a "sweet spot" - more forgiving than StandardT42T to maximize cooperation in mostly cooperative settings, but not as vulnerable as StandardT44T when faced with occasional defectors.



```java
class Lian_HongShenJordan_Player extends Player {
    // Implements the Tit-for-3-Tat strategy
    // Similar to Standard T42T,
    // but only defects if oppponent defected in the last three round
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n < 3)
            return 0; // Cooperate on first three rounds

        // Check if either opponent defected in both of the last three rounds
        if ((oppHistory1[n - 1] == 1 && oppHistory1[n - 2] == 1 && oppHistory1[n - 3] == 1)
                || (oppHistory2[n - 1] == 1 && oppHistory2[n - 2] == 1 && oppHistory2[n - 3] == 1)) {
            return 1;
        }

        return 0; // Otherwise cooperate
    }
}
```