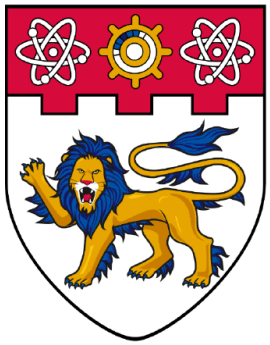


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

AY2023/24 Semester 2
CC4046 - Intelligent Agents

Assignment 2: Repeated Prisoners Dilemma

Name	Matriculation Number
Dann, Wee Zi Jun	U2122790K

Setup Guide

Environment Context

Language: Java - High level, class-based, object-oriented programming language.
Build Tool: Apache Maven - Build automation tool to compile, build and test Java Programming Language projects.
IDE: Visual Studio Code - Streamlined source code editor that supports development operations such as debugging, task running, and version control.

Setup Environment

Download IDE

Download Visual Studio Code via https://code.visualstudio.com/download

Installation Steps

1. Install Java Development Kit (JDK 17+)

```
java --version
```

2. Install Apache Maven (Version 3.9+)

```
mvn -version
```

3. Clone the repository

```
git clone https://github.com/dannweeeee/CZ4046-Intelligent-Agents.git
```

4. Change directory to assignment-two

```
cd assignment-two
```

5. Compile assignment-two

```
mvn compile
```

Debugging Source Code

There are 3 executable files that will be runned for assignment 2

1. **ThreePrisonersDilemma.java**
   Executable file - ThreePrisonersDilemma (Source code given by Assignment Question).

```
mvn exec:java -Dexec.mainClass="cz4046.main.ThreePrisonersDilemma"
```

2. **ThreePrisonersDilemmaWithStrategies.java**
   Executable file - ThreePrisonersDilemma (Source code with Agent Strategies).

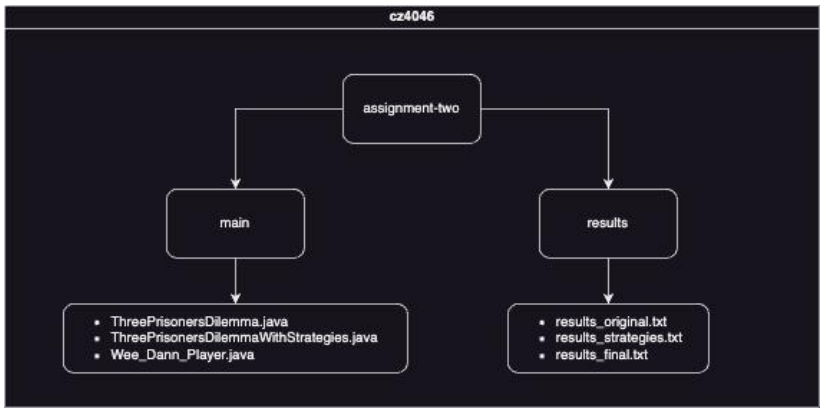
```
mvn exec:java -Dexec.mainClass="cz4046.main.ThreePrisonersDilemmaWithStrategies"
```

3. **Wee_Dann_Player.java**
   Executable file - Wee_Dann_Player (Implemented Source Code Fragment).

```
mvn exec:java -Dexec.mainClass="cz4046.main.Wee_Dann_Player”
```

Source Code

The source code is decomposed into only 2 major folder structures, Main and Results.



Source Code Folder Structure

The tree structure of the source code is shown as follows:

```
dannwee@Dann-Macbook-Pro assignment-two % tree
.
|____target
| |____generated-sources
| | |____annotations
| |____classes
| | |____cz4046
| | | |____main
| | | | |____Wee_Dann_Player$FreakyPlayer.class
| | | | |____ThreePrisonersDilemmaWithStrategies$RandomPlayer.class
| | | | |____Wee_Dann_Player$BestPlayer.class
| | | | |____ThreePrisonersDilemmaWithStrategies$FirmButFairPlayer.class
| | | | |____ThreePrisonersDilemmaWithStrategies$BestPlayer.class
| | | | |____Wee_Dann_Player$T4TPlayer.class
| | | | |____ThreePrisonersDilemma$Player.class
| | | | |____ThreePrisonersDilemmaWithStrategies$Player.class
| | | | |____ThreePrisonersDilemma.class
| | | | |____Wee_Dann_Player$NastyPlayer.class
| | | | |____ThreePrisonersDilemmaWithStrategies$HardMajorityPlayer.class
| | | | |____Wee_Dann_Player.class
| | | | |____ThreePrisonersDilemma$NicePlayer.class
| | | | |____ThreePrisonersDilemmaWithStrategies$GrimTriggerPlayer.class
| | | | |____ThreePrisonersDilemma$FreakyPlayer.class
| | | | |____ThreePrisonersDilemma$T4TPlayer.class
| | | | |____ThreePrisonersDilemmaWithStrategies$ReverseT4TPlayer.class
| | | | |____ThreePrisonersDilemma$TolerantPlayer.class
| | | | |____ThreePrisonersDilemma$RandomPlayer.class
| | | | |____Wee_Dann_Player$NicePlayer.class
| | | | |____ThreePrisonersDilemmaWithStrategies$GradualPlayer.class
| | | | |____ThreePrisonersDilemmaWithStrategies$FreakyPlayer.class
| | | | |____ThreePrisonersDilemmaWithStrategies$NastyPlayer.class
| | | | |____Wee_Dann_Player$TolerantPlayer.class
| | | | |____ThreePrisonersDilemmaWithStrategies.class
| | | | |____ThreePrisonersDilemmaWithStrategies$TolerantPlayer.class
| | | | |____ThreePrisonersDilemmaWithStrategies$NicePlayer.class
| | | | |____ThreePrisonersDilemma$NastyPlayer.class
| | | | |____ThreePrisonersDilemmaWithStrategies$T4TPlayer.class
| | | | |____Wee_Dann_Player$Player.class
| | | | |____Wee_Dann_Player$RandomPlayer.class
| | |____maven-status
| | | |____maven-compiler-plugin
| | | | |____compile
| | | | | |____default-compile
| | | | | | |____inputFiles.lst
| | | | | | |____createdFiles.lst
|____pom.xml
|____README.md
|____results
| |____results_strategies.txt
| |____results_final.txt
| |____results_original.txt
|____src
| |____main
| | |____java
| | | |____cz4046
| | | | |____main
| | | | | |____ThreePrisonersDilemma.java
| | | | | |____Wee_Dann_Player.java
| | | | | |____ThreePrisonersDilemmaWithStrategies.java
```

Source Code Tree

Introduction

In the game Prisoners' Dilemma, the dominant strategy equilibrium entails both agents defecting, despite the fact that cooperation would yield the best outcome for both. However, transitioning to a repeated iteration of the Prisoners' Dilemma may potentially mitigate this tendency toward non-cooperation.

In a repeating game, participants engage in the same game repeatedly, often indefinitely. As the number of repetitions (k) approaches infinity, the average reward of a player in the repeated game is calculated as the limit of the total payoffs accumulated by the player across all rounds, divided by the total number of rounds played. This can be computed in the following formula:

$$\lim_{k\to\infty} \sum_{j=1}^{k} r_j/k$$

The objective of this assignment is to devise a strategy for an agent participating in a three-player repeated Prisoners' Dilemma. The report outlines the development, implementation, and evaluation of various strategies. The performance analysis that we get from the agent strategies will lead to the creation of the proposed Agent, referred to as ***BestPlayer***. Additionally, the report elaborates on the rules and outcomes associated with ***BestPlayer***, presenting the findings of the conducted trials in accompanying text files. The Agent class is located in the ***Wee_Dann_Player.java*** file, while the alternative strategies discussed in the report can be found in the ***ThreePrisonersDilemmaWithStrategies.java*** file.

Strategies

Upon initially executing the provided Agents in ***ThreePrisonersDilemma.java***, it became evident that the Tit-for-Tat Agent and Tolerant Agent consistently outperformed others in most tournaments by clinching the highest number of top 2 finishes across 20 tournaments.

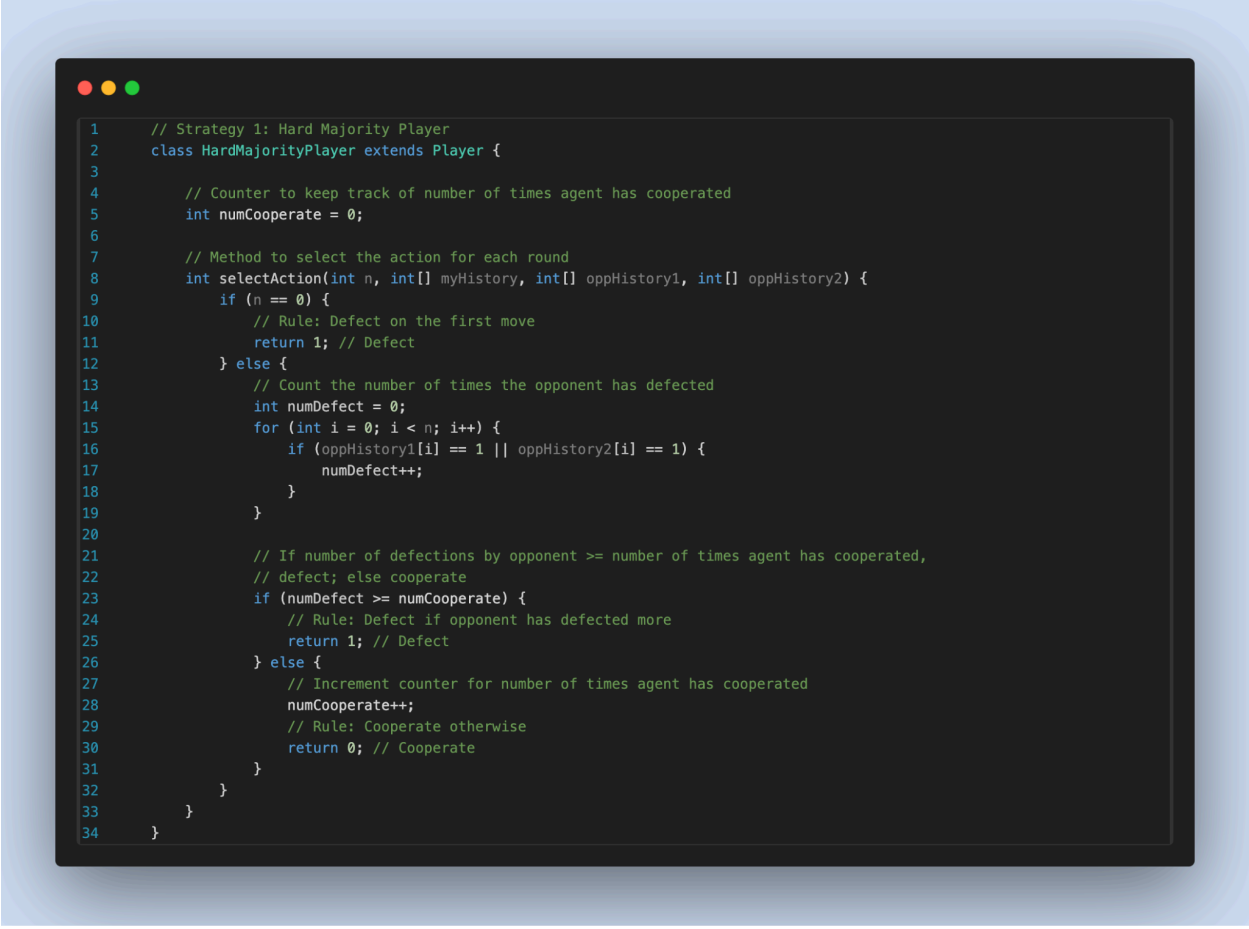
Players	Average Points
TolerantPlayer	124.5059519
T4TPlayer	124.204044
RandomPlayer	119.5700528
NastyPlayer	118.7366135
FreakyPlayer	116.8146272
NicePlayer	116.5254906

Consequently, the primary motivation for agent creation was to assess the varying degrees of cooperation and punishment across different strategies. The following sections detail the initial agent strategies that were investigated.

Strategy 1: Hard Majority Player

The Hard Majority strategy initiates by defecting on the first move, establishing an initial stance that leans towards cooperation. Subsequently, it keeps track of the ratio of its own cooperative and defective actions compared to those of the opponent. If the number of opponent defections surpasses or equals the agent's cooperative actions, the strategy defects; otherwise, it cooperates.

This method operates on a "majority" principle, whereby the agent evaluates the historical interactions to determine if the opponent has exhibited a higher frequency of defections compared to the agent's cooperative actions.



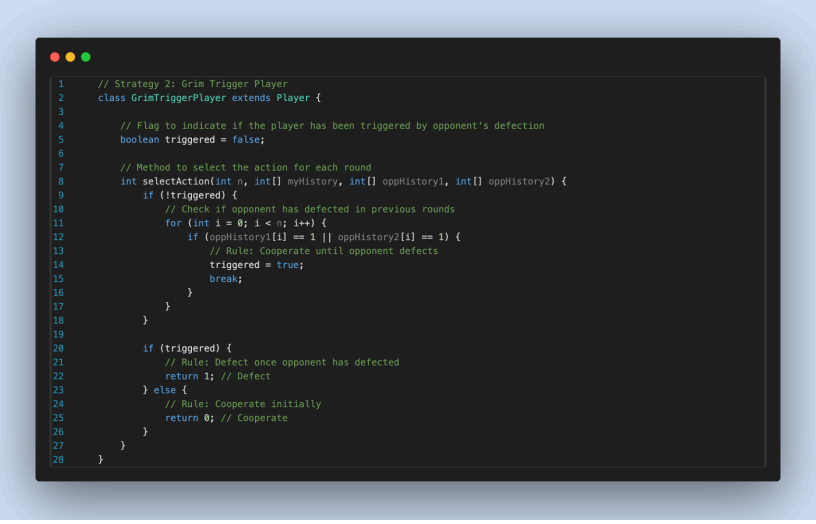
```
// Strategy 1: Hard Majority Player
class HardMajorityPlayer extends Player {

    // Counter to keep track of number of times agent has cooperated
    int numCooperate = 0;

    // Method to select the action for each round
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0) {
            // Rule: Defect on the first move
            return 1; // Defect
        } else {
            // Count the number of times the opponent has defected
            int numDefect = 0;
            for (int i = 0; i < n; i++) {
                if (oppHistory1[i] == 1 || oppHistory2[i] == 1) {
                    numDefect++;
                }
            }

            // If number of defections by opponent >= number of times agent has cooperated,
            // defect; else cooperate
            if (numDefect >= numCooperate) {
                // Rule: Defect if opponent has defected more
                return 1; // Defect
            } else {
                // Increment counter for number of times agent has cooperated
                numCooperate++;
                // Rule: Cooperate otherwise
                return 0; // Cooperate
            }
        }
    }
}
```

Source Code for Hard Majority Player Class

Strategy 2: Grim Trigger Player

The Grim Trigger strategy starts with cooperation with the adversary. Nevertheless, once the opponent defects, the strategy switches to a consistent pattern of defection for the remainder of the game.

This strategy showcases a stringent retaliatory approach, where even a single instance of defection by the opponent triggers a permanent shift to continuous defection in retaliation.



```java
// Strategy 2: Grim Trigger Player
class GrimTriggerPlayer extends Player {

    // Flag to indicate if the player has been triggered by opponent's defection
    boolean triggered = false;

    // Method to select the action for each round
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (!triggered) {
            // Check if opponent has defected in previous rounds
            for (int i = 0; i < n; i++) {
                if (oppHistory1[i] == 1 || oppHistory2[i] == 1) {
                    // Rule: Cooperate until opponent defects
                    triggered = true;
                    break;
                }
            }
        }

        if (triggered) {
            // Rule: Defect once opponent has defected
            return 1; // Defect
        } else {
            // Rule: Cooperate initially
            return 0; // Cooperate
        }
    }
}
```

Source Code for Grim Trigger Player Class

Strategy 3: Reverse Tit for Tat Player

The Reverse Tit-for-Tat technique is the reverse from the traditional Tit-for-Tat approach. Instead of mirroring the opponent's last move, Reverse Tit-for-Tat executes the inverse action. It commences with a defection on the first move, potentially setting a confrontational tone.

Subsequently, it observes the opponent's preceding move and performs the opposite action. If the opponent defected in the previous move, Reverse Tit-for-Tat cooperates. Otherwise, it defects.

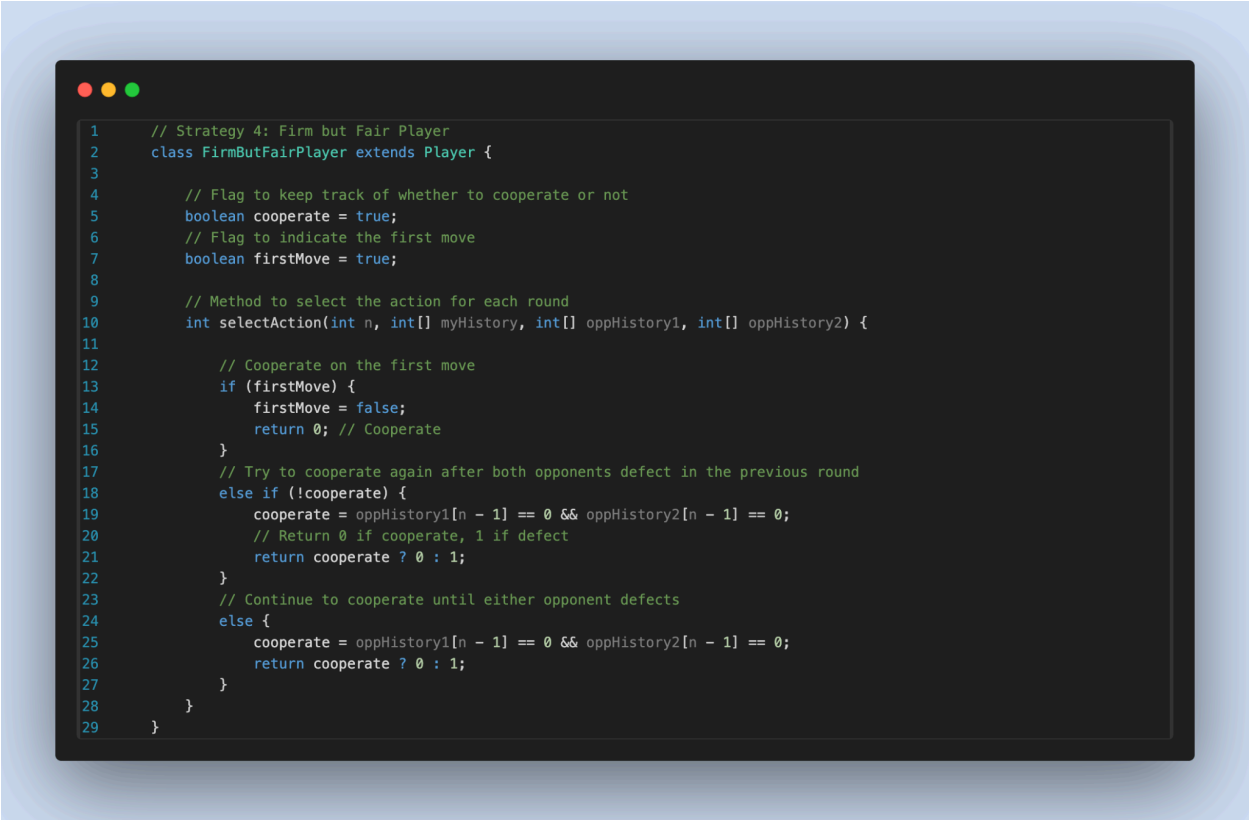
```java
// Strategy 3: Reverse Tit for Tat Player
class ReverseT4TPlayer extends Player {

    // Method to select the action for each round
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0) {
            return 1; // Defect on the first move
        } else {
            int lastOpponentMove = oppHistory1[n - 1]; // Get opponent's last move
            return lastOpponentMove == 0 ? 1 : 0; // Reverse opponent's last move
        }
    }
}
```

Source Code for Reverse Tit for Tat Player Class

Strategy 4: Firm but Fair Player

The Firm But Fair approach epitomizes a cooperative tactic where an agent begins by cooperating and persists in cooperation unless the opponent defects. Upon detecting defection from the opponent, the strategy reciprocates with defection. However, in cases of both sides defecting (D|D), the strategy strives to reestablish cooperation by returning to cooperation in the subsequent move.

This strategy is notable for its initial inclination towards cooperation and its commitment to cooperative behavior as long as the opponent reciprocates. Conversely, it inherits a "tit-for-tat" strategy in response to defection, aiming to strike a balance between assertiveness and fairness. While it retaliates against defection, it also actively seeks to renew cooperation after instances of mutual defection.



```java
// Strategy 4: Firm but Fair Player
class FirmButFairPlayer extends Player {

    // Flag to keep track of whether to cooperate or not
    boolean cooperate = true;
    // Flag to indicate the first move
    boolean firstMove = true;

    // Method to select the action for each round
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {

        // Cooperate on the first move
        if (firstMove) {
            firstMove = false;
            return 0; // Cooperate
        }
        // Try to cooperate again after both opponents defect in the previous round
        else if (!cooperate) {
            cooperate = oppHistory1[n - 1] == 0 && oppHistory2[n - 1] == 0;
            // Return 0 if cooperate, 1 if defect
            return cooperate ? 0 : 1;
        }
        // Continue to cooperate until either opponent defects
        else {
            cooperate = oppHistory1[n - 1] == 0 && oppHistory2[n - 1] == 0;
            return cooperate ? 0 : 1;
        }
    }
}
```

Source Code for Firm but Fair Player Class

Strategy 5: Gradual Player

The Gradual strategy initially embraces cooperation but adopts a more retaliatory stance following the opponent's defection. It commences with one defection and two cooperative moves after the first instance of opponent defection.

Subsequently, it retaliates with N consecutive defections following the opponent's Nth defection. In cases where the opponent exhibits forgiveness, the strategy employs two cooperative moves to reset the interaction pattern.

```java
// Strategy 5: Gradual Player
class GradualPlayer extends Player {

    // Variables to track defections and forgiveness status
    private int numDefections = 0;
    private int consecutiveDefections = 0;
    private boolean opponentIsForgiving = true;

    // Method to select the action for each round
    @Override
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0) {
            // Cooperate on the first move
            return 0; // Cooperate
        }

        // Check if the opponent has ever defected
        boolean opponentHasDefected = false;
        for (int i = 0; i < n; i++) {
            if (oppHistory1[i] == 1 || oppHistory2[i] == 1) {
                opponentHasDefected = true;
                break;
            }
        }

        if (!opponentHasDefected) {
            // Keep cooperating if the opponent has never defected
            return 0; // Cooperate
        }

        // If the opponent has defected at least once, start applying the gradual
        // strategy
        if (oppHistory1[n - 1] == 1 || oppHistory2[n - 1] == 1) {
            // Opponent defected on the previous move
            numDefections++;
            consecutiveDefections++;
            return 1; // Defect
        } else {
            // Opponent cooperated on the previous move
            if (numDefections > 0 && consecutiveDefections == numDefections) {
                // Calm down the opponent after N consecutive defections
                consecutiveDefections = 0;
                numDefections = 0;
                opponentIsForgiving = true;
                return 0; // Cooperate
            } else if (opponentIsForgiving) {
                // Cooperate if the opponent is forgiving
                return 0; // Cooperate
            } else {
                // Defect if the opponent is not forgiving
                consecutiveDefections++;
                return 1; // Defect
            }
        }
    }
}
```

Source Code for Gradual Player Class

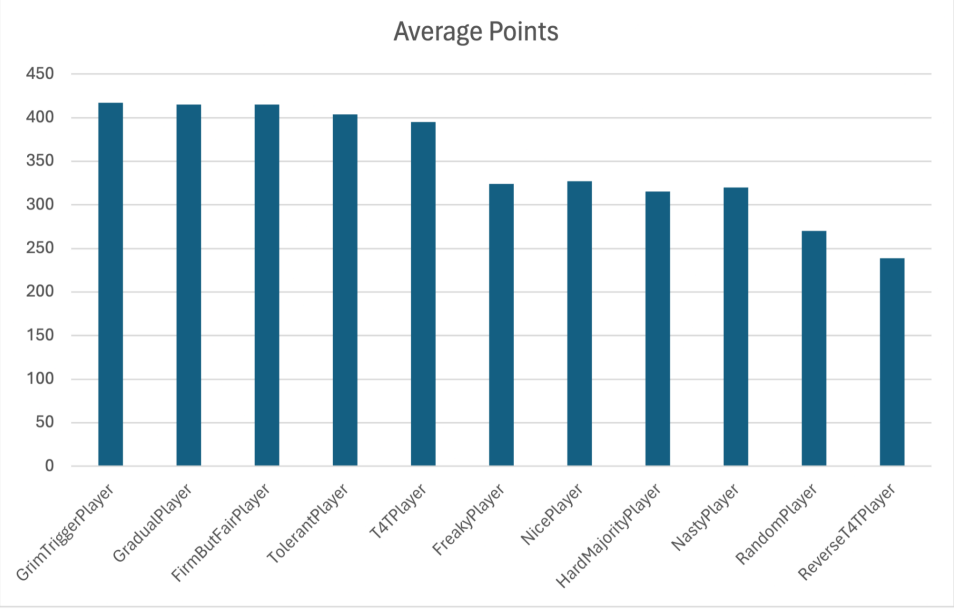
Results of Strategies

A series of 20 tournaments were carried out to evaluate the efficacy of the strategies. The table below presents the average scores attained by each agent.

Additionally, detailed raw results are also available in the *results_strategies.txt* file.

Players	Average Points
GrimTriggerPlayer	417.0256835
GradualPlayer	415.124651
FirmButFairPlayer	415.007275
TolerantPlayer	403.599055
T4TPlayer	395.2070605
FreakyPlayer	324.000722
NicePlayer	327.227053
HardMajorityPlayer	315.215952
NastyPlayer	319.8613745
RandomPlayer	269.96069
ReverseT4TPlayer	238.8716605

Table Results for results_strategies.txt



Bar Plot for results_strategies.txt

Analysis

The outcomes indicate that the ***Grim Trigger Player, Gradual Player, Firm but Fair Player***, and ***Tolerant Player*** emerged as the top-performing strategies across the 20 tournaments. As a result, the strategy to build the ***Best Player*** is grounded in the fundamental rules and principles underlying these 4 well-performing agents.

Best Player Agent

The ***BestPlayer*** class design implementation is motivated by the top 4 agents based on the ***Results of Strategies*** section in this report.

Explanation

At the core of the `BestPlayer` decision-making is the `selectAction` method, which determines the player's action for the current round. This method takes into account several factors, including whether the opponents have defected in previous rounds and the player's own history of actions.

When the opponents have defected previously, the player employs a strategy based on whether it has been `triggered` by these defections. If triggered, the player always chooses to defect, indicating a retaliatory response to its opponents' actions. Otherwise, it evaluates the cooperation (`opponentCoop`) and defection (`opponentDefect`) rates of its opponents and decides whether to cooperate or defect based on this analysis.

If the opponents have not defected, the player's strategy varies. On the first move, it cooperates by default. For subsequent moves, the player evaluates whether its last move was cooperation or defection and adjusts its action accordingly. It also considers the recent actions of its opponents, aiming to cooperate if both opponents cooperated in the previous round, or defect otherwise.

Throughout its decision-making process, the BestPlayer class also maintains internal variables to track its state, such as whether it has been triggered by previous defections and whether it should continue to cooperate or defect based on recent history. These variables allow the `BestPlayer` to dynamically adapt its strategy to maximize its payoff in the repeated game scenario.

The `BestPlayer` agent is heavily inspired by the following agent strategies:

1. **Grim Trigger** - The agent employs a Grim Trigger approach, where it always defects (`return 1`) if triggered (`triggered == true`), determined by the opponent's past defection (`opponentHasDefected`). It maintains its triggered status by scanning the opponent's history for any past defections (using a for loop with `oppHistory1` and `oppHistory2`).

2. **Gradual** - Like Firm but Fair, the agent cooperates (`return 0`) if both opponents cooperated in the previous round (`cooperate == true`), otherwise it defects (`return 1`). However, it also checks if it cooperated in the previous round (`if (cooperate)`), and if so, it sets cooperate using `oppHistory1[n - 1] == 0` and `oppHistory2[n - 1] == 0`. This strategy embodies Gradual behavior, where the agent forgives and cooperates as long as the opponents do, but retaliates with a defection if it was the one who cooperated.

3. **Firm but Fair** - Similar to Tolerant, the agent cooperates (`return 0`) if both opponents cooperated in the previous round (`cooperate == true`), otherwise it defects (`return 1`). Additionally, it checks if it defected in the previous round (`if (!cooperate)`), and if so, it sets cooperate using `oppHistory1[n - 1] == 0` and `oppHistory2[n - 1] == 0`. This reflects the Firm but Fair strategy, where the agent forgives and cooperates as long as the opponents do, but retaliates with a defection if it was the one who defected.

4. **Tolerant** - Initially, the agent cooperates on the first move (`if (firstMove)`) if the opponent has not defected (`opponentHasDefected == false`). Afterwards, it cooperates (`return 0`) if both opponents cooperated in the previous round (`cooperate == true`), otherwise it defects (`return 1`). This strategy aligns with Tolerant behavior, where the agent cooperates as long as the opponents do, but retaliates with a defection in response to their defections.

Implementation

```java
/*
 * Solution: Best Player
 * Based on Grim Trigger Player, Gradual Player, Firm but Fair Player, and
 * Tolerant Player
 */
class BestPlayer extends Player {

    // Private instance variables
    private boolean triggered = false;
    private int opponentCoop = 0;
    private int opponentDefect = 0;
    private boolean cooperate = true;
    private boolean firstMove = true;

    // Overriding the selectAction method in the superclass
    @Override
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        // Determine if the opponent has defected in previous rounds
        boolean opponentHasDefected = false;
        for (int i = 0; i < n; i++) {
            if (oppHistory1[i] == 1 || oppHistory2[i] == 1) {
                opponentHasDefected = true;
                break;
            }
        }

        // Implementing strategy when the opponent has defected
        if (opponentHasDefected) {
            // Check if the player is triggered by a previous defection
            if (!triggered) {
                for (int i = 0; i < n; i++) {
                    if (oppHistory1[i] == 1 || oppHistory2[i] == 1) {
                        triggered = true;
                        break;
                    }
                }
            }

            // If triggered, always choose to defect
            if (triggered) {
                return 1; // Defect
            } else {
                // If not triggered, assess opponent's cooperation and defection rates
                for (int i = 0; i < n; i++) {
                    if (oppHistory1[i] == 0)
                        opponentCoop++;
                    else
                        opponentDefect++;
                }
                for (int i = 0; i < n; i++) {
                    if (oppHistory2[i] == 0)
                        opponentCoop++;
                    else
                        opponentDefect++;
                }
                if (opponentDefect > opponentCoop)
                    return 1; // Defect
                else
                    return 0; // Cooperate
            }
        } else {
            // Implement strategy when the opponent has not defected
            // Cooperate on the first move
            if (firstMove) {
                firstMove = false;
                return 0; // Cooperate
            } else if (!cooperate) {
                // If the last move was a defection
                cooperate = oppHistory1[n - 1] == 0 && oppHistory2[n - 1] == 0;
                // Cooperate if both opponents cooperated, otherwise defect
                return cooperate ? 0 : 1;
            } else {
                // If the last move was cooperation
                cooperate = oppHistory1[n - 1] == 0 && oppHistory2[n - 1] == 0;
                // Cooperate if both opponents cooperated, otherwise defect
                return cooperate ? 0 : 1;
            }
        }
    }
}
```

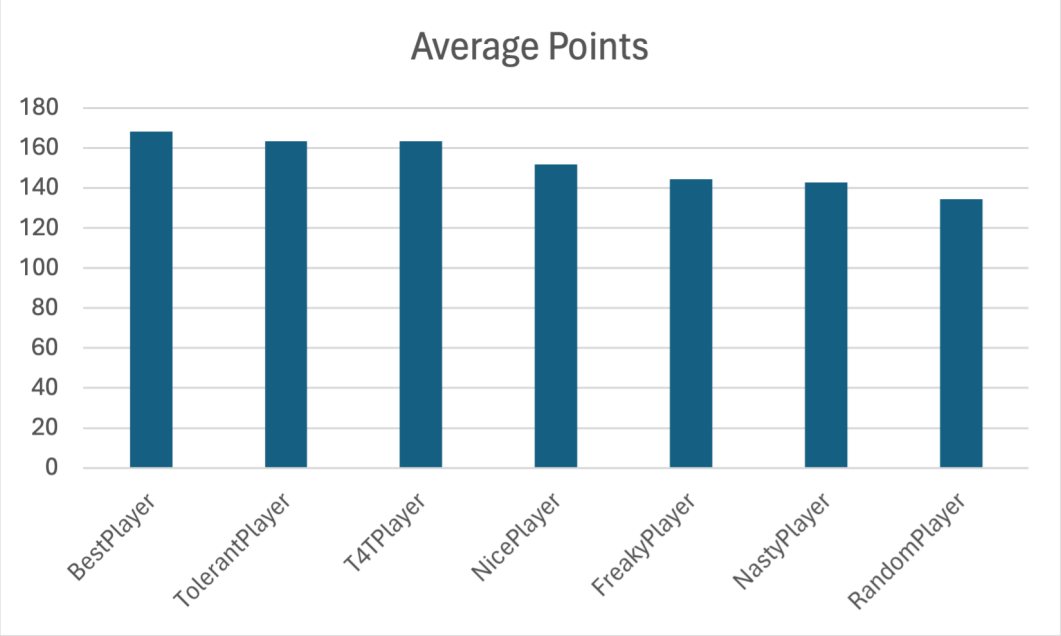
Source Code for Best Player Class

Final Results

A total of 20 tournaments was conducted to test ***BestPlayer***. The table below displays the average score of each of the agents in ***Wee_Dann_Player.java***. Raw results can also be found in *results_final.txt* file.

Player	Average Point
BestPlayer	168.084194
TolerantPlayer	163.4325325
T4TPlayer	163.3266305
NicePlayer	151.7648915
FreakyPlayer	144.391815
NastyPlayer	142.7680385
RandomPlayer	134.2722438

Table Results for results_final.txt



Bar Plot for results_final.txt

In summary, ***BestPlayer*** showcases an impressive performance, boasting the highest average points among 20 tournaments.

Although ***BestPlayer*** consistently outshines other competitors, there were occasional occurrences where it placed second or at most third, trailing behind Tolerant and T4T agents. This highlights the variability in performance under specific circumstances.