CZ4046: Intelligent Agents

Assignment 2: Repeated Prisoners Dilemma

Name: Singh Aishwarya
Matriculation Number: U1923952C
Email: SI0001YA@e.ntu.edu.sg

Table of Contents

Introduction **2**
Initial Strategies **2**
Firm but Fair 2
Gradual 3
Grim Trigger 4
Hard Majority 5
Reverse Tit-for-Tat 6
Intermediate Results (Trials) **7**
Final Agent **8**
Agent Rules 8
Agent Implementation 9
Final Results and Evaluation **11**

Introduction

The dominating strategy equilibrium in the game of Prisoners' Dilemma is for players to defect, even when both agents would be better off collaborating. This lack of collaboration, however, might theoretically diminish in a repeated iteration of the Prisoners' Dilemma. A repeating game is one in which the same group of participants plays the same game many times, sometimes forever. As k approaches infinity, the (average) reward of a player in a repeated game is determined as the limit of the total of the player's payoffs in each round divided by the number of rounds played.

The aim of this assignment is to develop a strategy for an agent in a three-player repeated Prisoners' Dilemma. The report describes various strategies that were designed, implemented, and tested. The performance of the tested strategies is analysed and used to implement the proposed Agent, named *BestPlayer*. The report also discusses the rules and results of *BestPlayer*, and presents the outputs of the conducted trials in the submitted text files. The Agent (class) can be found in the `SinghAishwaryaPlayer.java` file, and the other tested strategies discussed in the report can be found in the `ThreePrisonersDilemma.java` file.

Initial Strategies

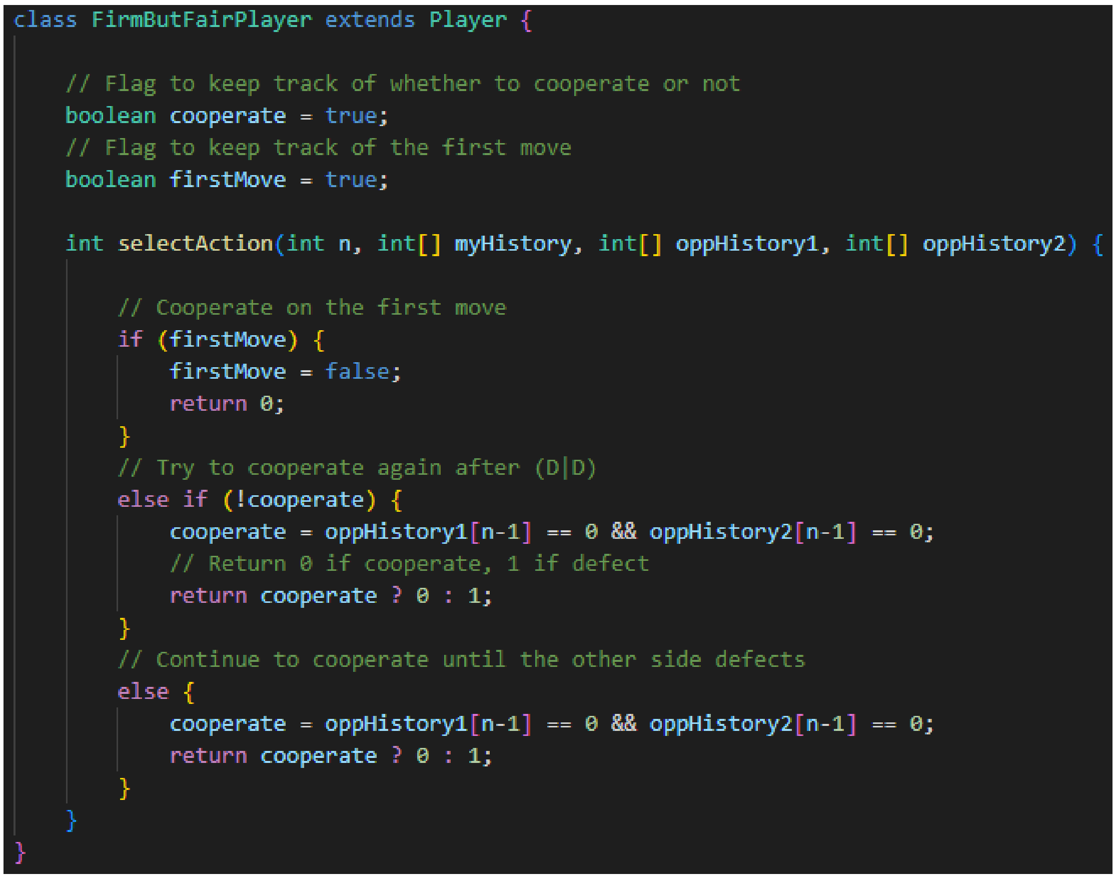
Upon first running the provided Agents in `ThreePrisonersDilemma.java`, it was observed that *Tolerant* Agent and *T4T* Agent were the top players for the majority of tournaments. The initial motivation to create an agent was therefore to test the relative levels of cooperation and punishment of different strategies. The sections below illustrate the initial punishing strategies that were tested.

Firm but Fair

The "Firm But Fair" approach is a cooperative strategy in which an agent first cooperates and continues to cooperate until the opponent defects. When the opponent defected, the strategy defected as well. However, once both sides defect (D|D), the strategy strives to reestablish cooperation by reverting to cooperation in the following move.

The approach is distinguished by its readiness for cooperation at first and to continue cooperating as long as the opponent cooperates. It, on the other hand, responds to defection with defection, using a "tit-for-tat" strategy. The technique strives to strike a balance between assertiveness and fairness by retaliating against defection while simultaneously attempting to reestablish cooperation following mutual defections. The figure below illustrates the implementation of this strategy -

Figure 1: Implementation of Firm but Fair Strategy

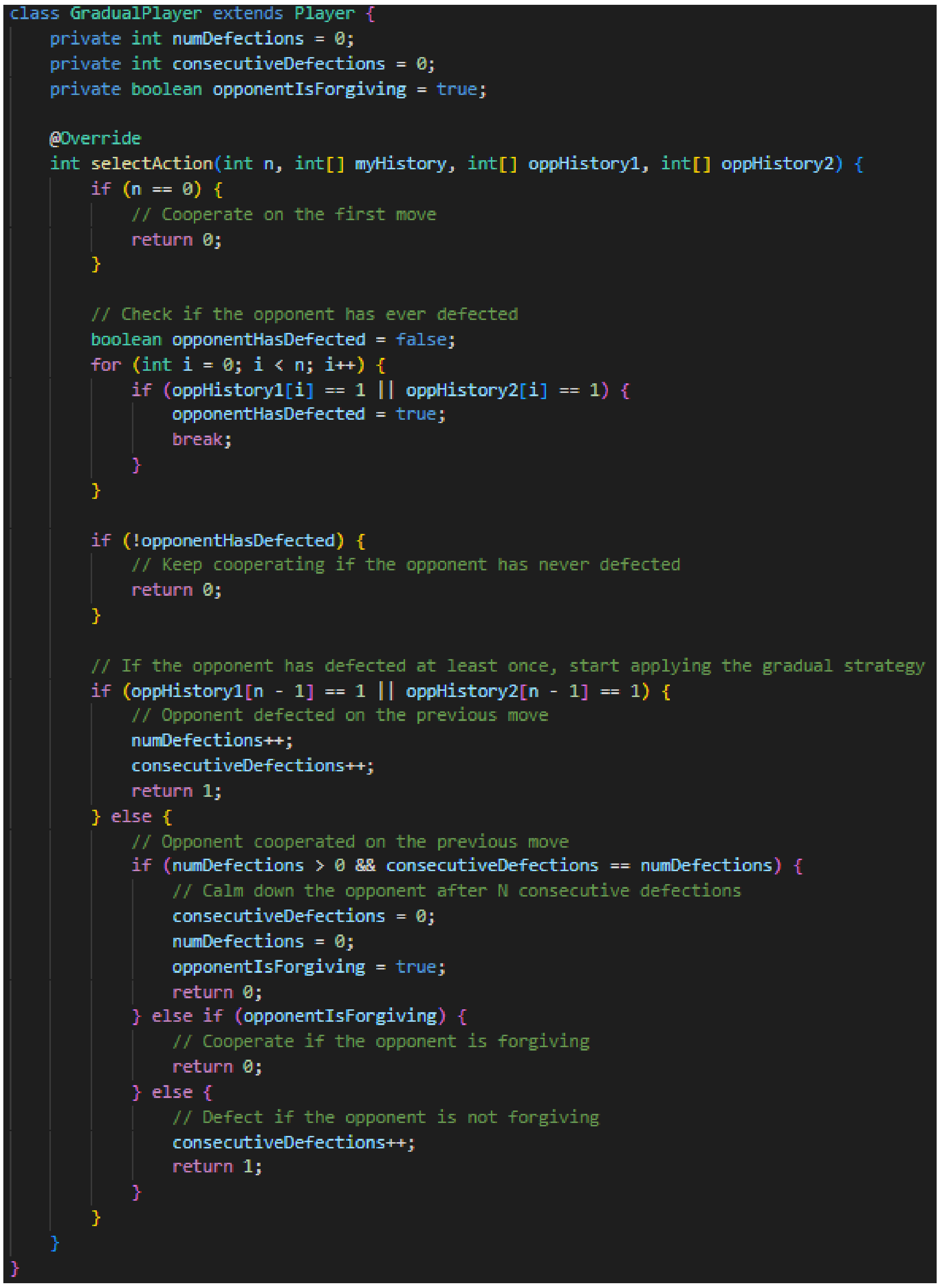


```java
class FirmButFairPlayer extends Player {

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

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {

        // Cooperate on the first move
        if (firstMove) {
            firstMove = false;
            return 0;
        }
        // Try to cooperate again after (D|D)
        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 the other side defects
        else {
            cooperate = oppHistory1[n-1] == 0 && oppHistory2[n-1] == 0;
            return cooperate ? 0 : 1;
        }
    }
}
```

Gradual

The Gradual strategy cooperates at first, but becomes more retaliatory if the opponent defected. It begins with one defection and two co-operations after the first defection, and then retaliates with N consecutive defections after the opponent's Nth defection. If the opponent is forgiving, it soothes them down with two co-operations to reset their conduct. The figure below illustrates the implementation of this strategy -

Figure 2: Implementation of Gradual Strategy

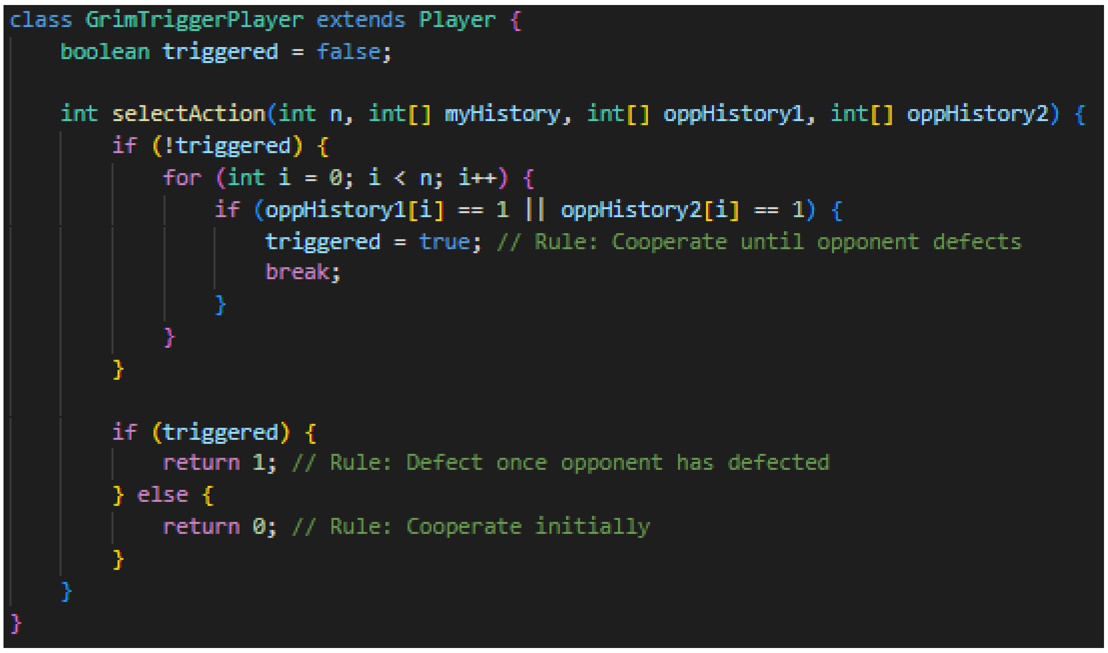


```java
class GradualPlayer extends Player {
    private int numDefections = 0;
    private int consecutiveDefections = 0;
    private boolean opponentIsForgiving = true;

    @Override
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0) {
            // Cooperate on the first move
            return 0;
        }

        // 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;
        }

        // 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;
        } 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;
            } else if (opponentIsForgiving) {
                // Cooperate if the opponent is forgiving
                return 0;
            } else {
                // Defect if the opponent is not forgiving
                consecutiveDefections++;
                return 1;
            }
        }
    }
}
```

Grim Trigger

The Grim Trigger tactic begins by collaborating with the adversary. However, after the opponent defected, the strategy changed to constantly defecting for the rest of the game. This strategy exhibits rigorous retaliatory behaviour, in which a single defection by the opponent causes a permanent change to always defect in retaliation. The figure below illustrates the implementation of this strategy -

Figure 3: Implementation of Grim Trigger Strategy

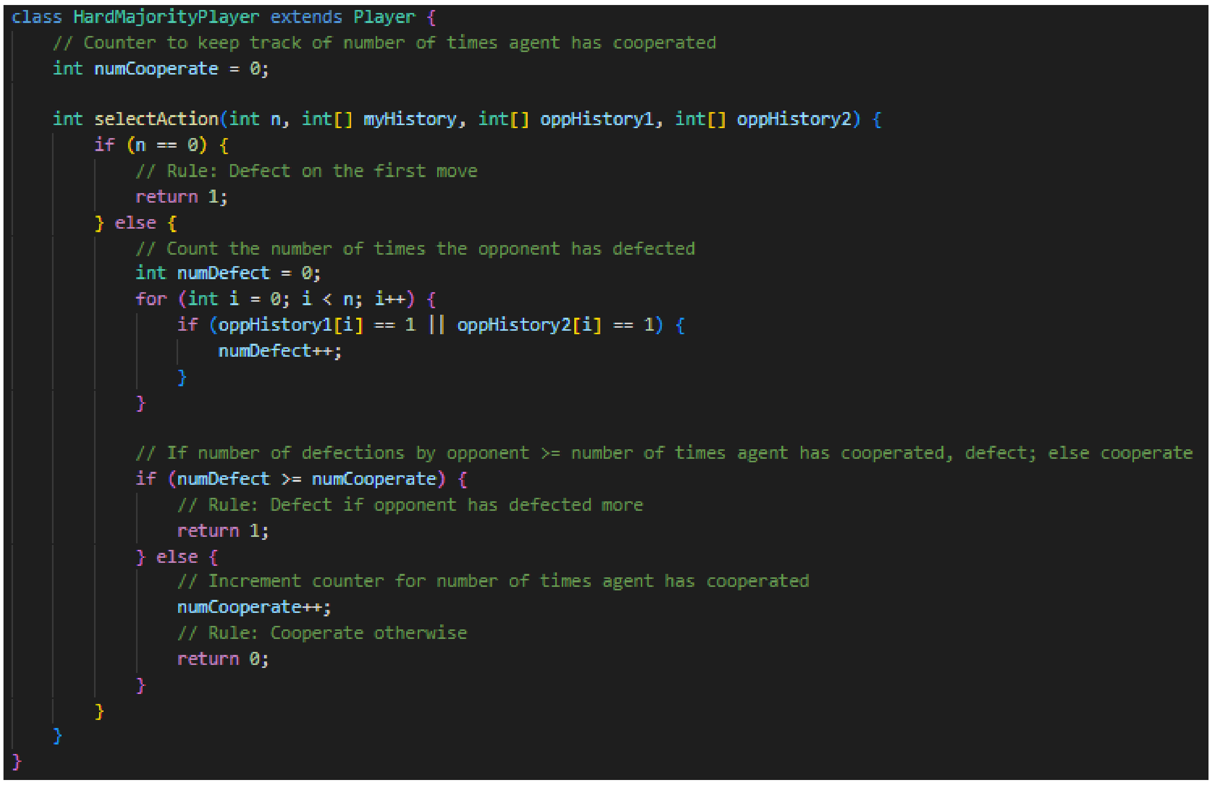


```java
class GrimTriggerPlayer extends Player {
    boolean triggered = false;

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (!triggered) {
            for (int i = 0; i < n; i++) {
                if (oppHistory1[i] == 1 || oppHistory2[i] == 1) {
                    triggered = true; // Rule: Cooperate until opponent defects
                    break;
                }
            }
        }

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

Hard Majority

The Hard Majority strategy begins with a defect on the first move, establishing a precedent for a possibly hostile position. Then, for successive actions, it maintains note of how many times it cooperated and how many times the opponent defected. The agent will defect if the number of defections is higher than or equal to the number of times it has cooperated. Otherwise, it will comply. This technique is based on a "majority" rule, in which the agent examines the history of encounters and decides whether the opponent has defected more frequently than the agent has cooperated. The figure below illustrates the implementation of this strategy -

Figure 4: Implementation of Hard Majority Strategy

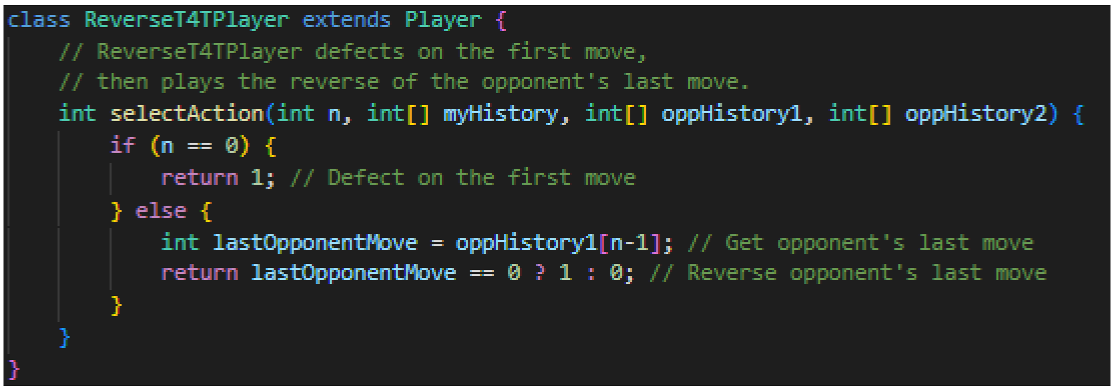


```java
class HardMajorityPlayer extends Player {
    // Counter to keep track of number of times agent has cooperated
    int numCooperate = 0;

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0) {
            // Rule: Defect on the first move
            return 1;
        } 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;
            } else {
                // Increment counter for number of times agent has cooperated
                numCooperate++;
                // Rule: Cooperate otherwise
                return 0;
            }
        }
    }
}
```

Reverse Tit-for-Tat

The Reverse Tit-for-Tat technique, as the name implies, is a variant of the Tit-for-Tat method. Reverse Tit-for-Tat, on the other hand, executes the reverse of the opponent's last move rather than replicating it. It begins by defecting on the first move, perhaps creating an aggressive tone. Then, for successive movements, it looks at the opponent's last move and reverses it. If the opponent defected on the previous move, Reverse Tit-for-Tat will cooperate; otherwise, it will defect. The figure below illustrates the implementation of this strategy -

Figure 5: Implementation of Reverse Tit-for-Tat Strategy



```java
class ReverseT4TPlayer extends Player {
    // ReverseT4TPlayer defects on the first move,
    // then plays the reverse of the opponent's last move.
    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
        }
    }
}
```

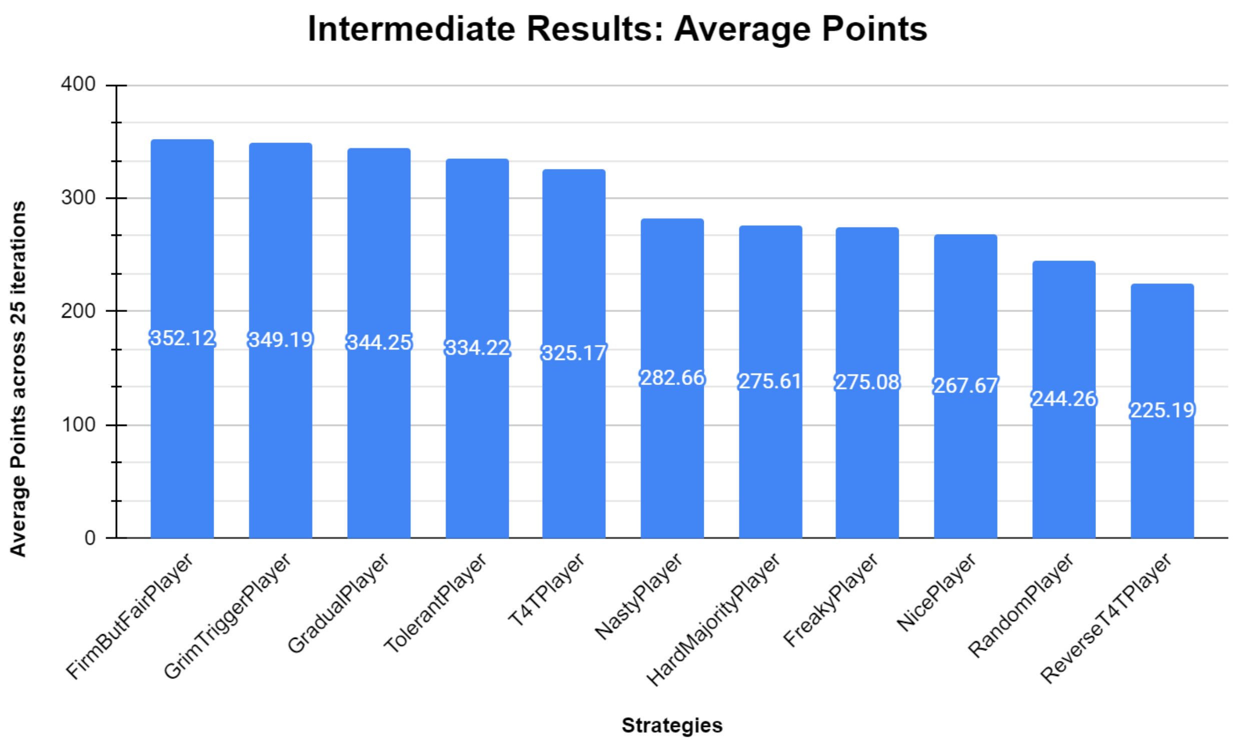
Intermediate Results (Trials)

There were a total of 25 tournaments conducted to test the effectiveness of the strategies. The table below displays the average scores of each of the agents, which is illustrated in **Figure 6** as well. Raw results for this can also be found in the `results_intermediate.txt` file.

Table 1: Intermediate Results - Average Points

Strategy	Average Points
FirmButFairPlayer	352.12418
GrimTriggerPlayer	349.18806
GradualPlayer	344.25418
TolerantPlayer	334.2217
T4TPlayer	325.16515
NastyPlayer	282.66238
HardMajorityPlayer	275.6112
FreakyPlayer	275.0823
NicePlayer	267.6697
RandomPlayer	244.26407
ReverseT4TPlayer	225.19312

Figure 6: Illustration of Intermediate Results



The results illustrate that the top 4 strategies across 25 tournaments were Firm but Fair, Grim Trigger, Gradual and Tolerant Strategy. Therefore the motivation to build an agent stems from the core rules and concepts of these strategies.

Final Agent

The design and implementation of the proposed Agent *BestPlayer* has been motivated by the top 4 agents in the intermediate results from the previous section. The sub-sections below elaborate more on the same.

Agent Rules

Rule 1: If the opponent defected in a previous round, the agent checks to see if it has already been triggered. If not, it investigates the opponent's history for any past defections. If discovered, it is triggered and begins defecting in all following rounds. When the agent is already triggered, it always defects.

Rule 2: If the opponent has never defected before, the agent will cooperate in the first move. Then, in each subsequent move, the agent will check if it defected in the previous move. If it did, it looks to see if the opponent defected in the previous move. If the opponent cooperated in the previous move, the agent will cooperate in the current move. If the opponent defected in the previous move, the agent will also defect in the current move. If the agent cooperated in the previous move, it checks to see if the opponent did the same. If the opponent cooperated in the previous move, the agent will cooperate in the current move. If the agent's opponent defected in the previous move, the agent will defect in the current move.

It can be seen that the rules designed for the agent are heavily motivated from the strategies -

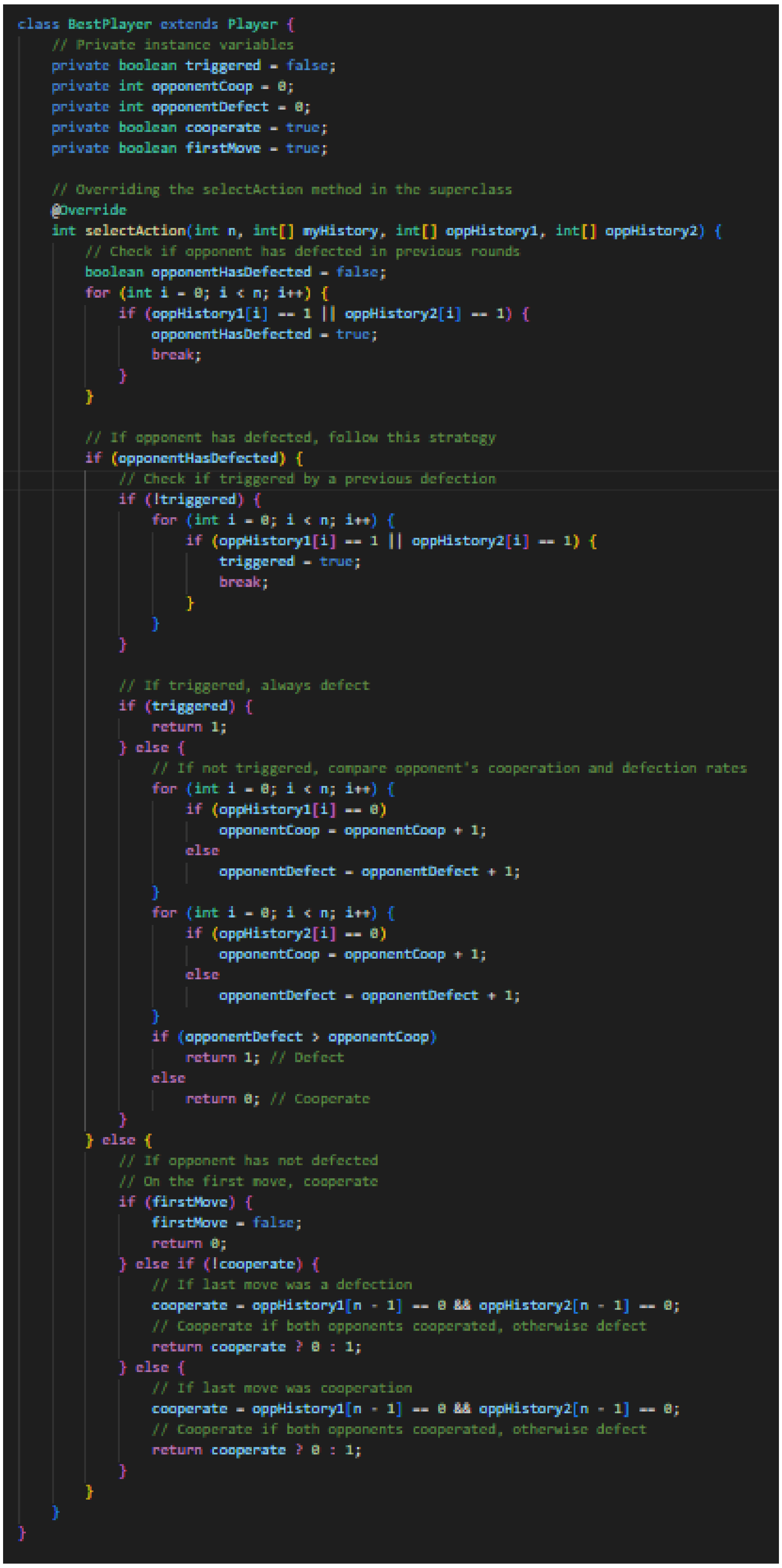
1. **Grim Trigger** - The agent determines if the opponent defected in a previous round (`opponentHasDefected`). If the agent is triggered (`triggered == true`), it always defects (return 1), according to the Grim Trigger approach. The agent also determines if it has been triggered previously by scanning the opponent's history for any past defection (for loop with `oppHistory1` and `oppHistory2`). If discovered, it gets triggered (`triggered = true`), which again corresponds to the Grim Trigger approach.

2. **Tolerant:** If the opponent has not defected (`opponentHasDefected == false`), the agent cooperates on the first move (`if (firstMove)`). After the first move, the agent cooperates (return 0) if both opponents cooperated in the previous round (`cooperate == true`), otherwise defects (return 1). This corresponds to the Tolerant strategy, where the agent cooperates as long as the opponents cooperate, but defects in response to defections.
3. **Firm but Fair**: Similar to the Tolerant strategy, the agent cooperates (return 0) if both opponents cooperated in the previous round (`cooperate == true`), otherwise defects (return 1). However, unlike the Tolerant strategy, the agent also 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 corresponds to the Firm but Fair strategy, where the agent is forgiving and continues to cooperate as long as the opponents cooperate, but retaliates with a defection if it was the one who defected.
4. **Gradual:** Similar to the Firm but Fair strategy, the agent cooperates (return 0) if both opponents cooperated in the previous round (`cooperate == true`), otherwise defects (return 1). However, unlike the Firm but Fair strategy, the agent 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 corresponds to the Gradual strategy, where the agent is forgiving and continues to cooperate as long as the opponents cooperate, but retaliates with a defection if it was the one who cooperated.

Agent Implementation

The figure below depicts how the Agent BestPlayer was implemented following the rules corresponding to the previous section -

Figure 7: BestPlayer Implementation



```java
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) {
        // Check if 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;
            }
        }

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

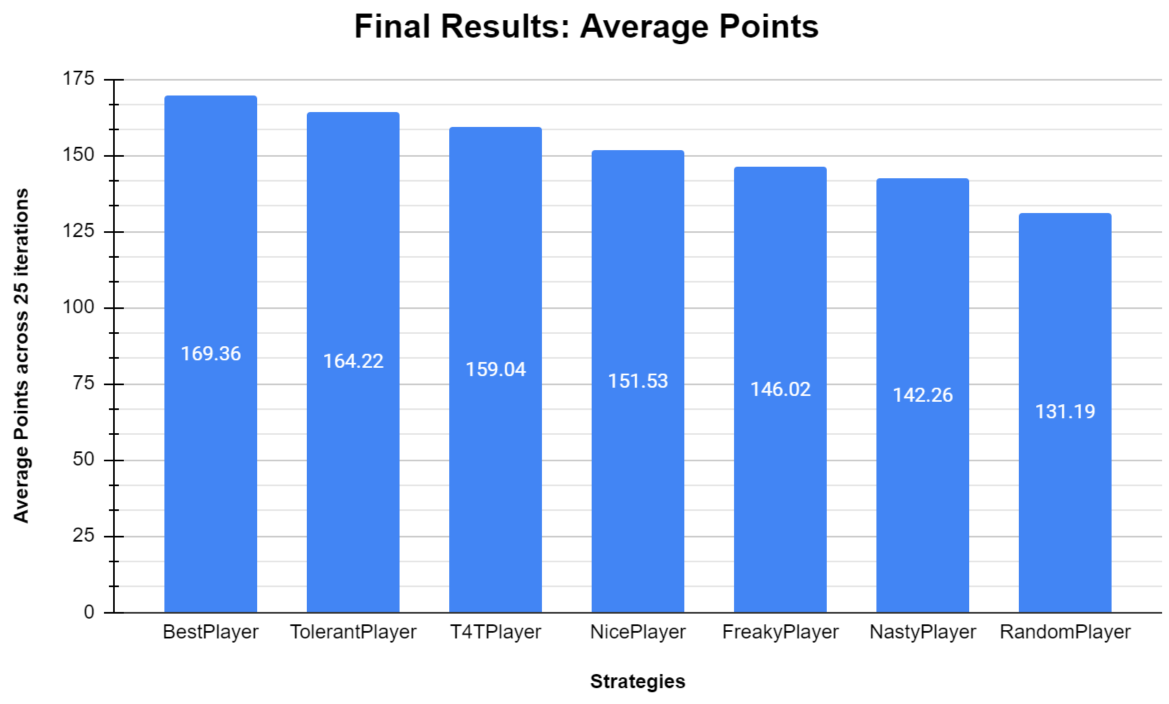
Final Results and Evaluation

There were a total of 25 tournaments conducted to test *BestPlayer*. The table below displays the average scores of each of the agents, which is illustrated in Figure 8 as well. Raw results for this can also be found in the `results_final.txt` file.

Table 2: Final Results - Average Points

Strategy	Average Points
BestPlayer	169.3646
TolerantPlayer	164.2162
T4TPlayer	159.0424
NicePlayer	151.5269
FreakyPlayer	146.0165
NastyPlayer	142.2603
RandomPlayer	131.1873

Figure 8: Illustration of Final Results



In conclusion, *BestPlayer* demonstrates a strong performance with the highest average points across 25 iterations, outperforming even *Tolerant* Player (its inspiration). While *BestPlayer* consistently wins against other players, there were instances where it came in second or at most third to *Tolerant* and *T4T* agents, indicating that performance can vary in specific situations.