

**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**

---

**SINGAPORE**

# **CZ4046 INTELLIGENT AGENTS**

## **Assignment 2**

**Submitted By: Soh Qian Yi**

**Matriculation Number: U1922306C**

## Table of Contents

<b>1. Introduction.....</b>	<b>3</b>
<b>1.1. Rewards.....</b>	<b>3</b>
<b>2. Approach .....</b>	<b>4</b>
<b>2.1. Understanding the Example Strategies .....</b>	<b>4</b>
2.1.1. NicePlayer .....	4
2.1.2. NastyPlayer .....	5
2.1.3. RandomPlayer .....	5
2.1.4. TolerantPlayer .....	5
2.1.5. FreakyPlayer .....	6
2.1.6. T4TPlayer (Tit-for-Tat Player).....	6
<b>2.2. Additional Strategies .....</b>	<b>7</b>
2.2.1. SuspiciousT4TPlayer (Suspicious Tit-for-Tat Player).....	7
2.2.2. T42TPlayer (Tit-for-Two-Tat Player).....	7
2.2.3. ForgivingT4TPlayer (Forgiving Tit-for-Tat Player) .....	8
2.2.4. GrimTriggerPlayer .....	9
2.2.5. SpitefulPlayer .....	9
2.2.6. PavlovPlayer .....	10
2.2.7. GradualPlayer .....	10
2.2.8. AlternateCooperateDefectPlayer.....	11
<b>2.3. Tournament .....</b>	<b>12</b>
2.3.1. Results of Strategies .....	12
<b>3. Implementation .....</b>	<b>13</b>
3.1. Combination of GrimTriggerPlayer and GradualPlayer.....	13
3.2. Combination of GrimTriggerPlayer and ForgivingT4TPlayer .....	15
3.3. Combination of GradualPlayer and ForgivingT4TPlayer .....	16
3.4. Combination of GrimTriggerPlayer, GradualPlayer and ForgivingT4TPlayer .....	17
<b>4. Results for Agent Performances .....</b>	<b>19</b>
4.1. Evaluation of Selected Agent.....	20
<b>5. Conclusion .....</b>	<b>20</b>
<b>6. Reference .....</b>	<b>20</b>

# 1. Introduction

This project aims to explore the topic of the repeated Prisoners' Dilemma game and the development of strategies for agents to cooperate. In the standard Prisoners' Dilemma, the dominant strategy for agents is to defect, even though it is mutually beneficial for both to cooperate. However, in a repeated version of the game, this lack of cooperation can potentially disappear. In this project, three players will play with each other repeatedly in a match, and a player's score from that match is the average of the payoffs from each round.

The objective of this assignment is to develop a strategy for an agent in a three-player repeated Prisoners' Dilemma that would maximize the agent's payoff over a series of 100-round matches. The strategy will be a code fragment that looks at the previous plays and computes the next action. This report will discuss the different strategies developed for this game, their implementations, and their effectiveness in achieving the goal of mutual cooperation.

## 1.1. Rewards

The table below shows the possible combinations of actions taken by three players in a repeated Prisoner's Dilemma game, along with the corresponding rewards to Player 1. The rewards are listed in the last column. The actions are denoted using the integers 0 and 1, where 0 represents cooperation and 1 represents defection.

*Table 1: Table of Combination of Payoffs*

Player 1	Player 2	Player 3	Rewards to Player 1
0	0	0	6
0	0	1	3
0	1	0	3
0	1	1	0
1	0	0	8
1	0	1	5
1	1	0	5
1	1	1	2

From the table, it is observed that the dominant action for Player 1 is defection, regardless of the actions taken by the other players, as the highest rewards of 6, 8, 5, and 5 (highlighted in green) are mostly obtained when Player 1 defects.

For instance, if Player 1 defects and Players 2 and 3 cooperate, Player 1 gets a payoff of 8, which is the highest possible payoff. Similarly, if Player 1 cooperates while Players 2 and 3 both defect, Player 1 gets the lowest possible payoff of 0.

This presents a dilemma for each agent, as they must balance their individual payoff against the collective payoff of all agents. The challenge for each agent is to develop a strategy that navigates this tension between individual and collective interests. They can either choose to cooperate and rely on the other agents to cooperate or defect and aim for the highest payoff while avoiding the worst-case scenario of all agents defecting.

## 2. Approach

### 2.1. Understanding the Example Strategies

In the `ThreePrisonersDilemma.java` file, six different strategies were provided to be used by the agents in the assignment: *NicePlayer*, *NastyPlayer*, *RandomPlayer*, *TolerantPlayer*, *FreakyPlayer*, and *T4TPlayer*. These strategies were based on different algorithms and approaches to the game, and studying them were beneficial to get a sense of how the different players behave in the tournament.

#### 2.1.1. NicePlayer

This strategy involves always cooperating with the other player regardless of their move. It is a cooperative strategy that assumes the other player will reciprocate with cooperation.

```
class NicePlayer extends Player {  
    // NicePlayer always cooperates  
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {  
        return 0;  
    }  
}
```

Figure 1: Code Snippet of NicePlayer

### 2.1.2. NastyPlayer

This strategy involves always defecting regardless of the other player's move. It is a competitive strategy that assumes the other player will also defect.

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

*Figure 2: Code Snippet of NastyPlayer*

### 2.1.3. RandomPlayer

This strategy involves choosing a move at random (cooperate or defect) with a 50/50 chance for each move. It is an unpredictable strategy that does not rely on the other player's move.

```
class RandomPlayer extends Player {
    // RandomPlayer randomly picks his action each time
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (Math.random() < 0.5)
            return 0; // cooperates half the time
        else
            return 1; // defects half the time
    }
}
```

*Figure 3: Code Snippet of RandomPlayer*

### 2.1.4. TolerantPlayer

This strategy looks at both opponents' histories and only defects if at least half of their moves have been defects. It is a forgiving strategy that allows the opponents to make a few mistakes before retaliating with a defection.

```
class TolerantPlayer extends Player {
    // TolerantPlayer looks at his opponents' histories, and only defects
    // if at least half of the other players' actions have been defects
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        int opponentCoop = 0;
        int opponentDefect = 0;
        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;
        else
            return 0;
    }
}
```

*Figure 4: Code Snippet of TolerantPlayer*

### 2.1.5. FreakyPlayer

This strategy randomly chooses at the start of the game whether to always cooperate or always defect and then follows that action for the rest of the game. It is an unpredictable strategy that aims to confuse the other player by not revealing its intentions until the game has started.

```
class FreakyPlayer extends Player {
    // FreakyPlayer determines, at the start of the match,
    // either to always be nice or always be nasty.
    // Note that this class has a non-trivial constructor.
    int action;

    FreakyPlayer() {
        if (Math.random() < 0.5)
            action = 0; // cooperates half the time
        else
            action = 1; // defects half the time
    }

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        return action;
    }
}
```

*Figure 5: Code Snippet of FreakyPlayer*

### 2.1.6. T4TPlayer (Tit-for-Tat Player)

This strategy starts with cooperation and then copies the other player's previous move. It is a cooperative strategy that rewards the other player's cooperation and punishes their defection.

```
class T4TPlayer extends Player {
    // 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 0; // cooperate by default
        if (Math.random() < 0.5)
            return oppHistory1[n - 1];
        else
            return oppHistory2[n - 1];
    }
}
```

*Figure 6: Code Snippet of T4TPlayer*

## 2.2. Additional Strategies

In addition to the provided strategies, 8 additional strategies (Kuhn, 2019) were analyzed and added to `ThreePrisonersDilemma.java` file in order to have more options for strategy combinations. These 8 strategies include: *SuspiciousT4TPlayer*, *T42TPlayer*, *ForgivingT4TPlayer*, *GrimTriggerPlayer*, *SpitefulPlayer*, *PavlovPlayer*, *GradualPlayer* and *AlternateCooperateDefectPlayer*.

### 2.2.1. SuspiciousT4TPlayer (Suspicious Tit-for-Tat Player)

This strategy involves the player initially defecting instead of cooperating and then using a variation of the T4T strategy. The player cooperates in the next round if both opponents cooperated in the previous round. If either opponent defects in the previous round, the player will defect as well, but also defect in the next round regardless of the opponents' actions. This approach is intended to be more cautious, as it will not continue to cooperate with an opponent that defects even once.

```
class SuspiciousT4TPlayer extends Player {
    // instead of T4T which starts with cooperate, this strategy starts with defect
    // by default
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 1; // defect by default
        if (oppHistory1[n - 1] == 0 && oppHistory2[n - 1] == 0)
            return 0; // cooperate if both opponents cooperated last round
        else
            return Math.min(oppHistory1[n - 1], oppHistory2[n - 1]); // tit-for-tat
    }
}
```

Figure 7: Code Snippet of *SuspiciousT4TPlayer*

### 2.2.2. T42TPlayer (Tit-for-Two-Tat Player)

This strategy involves the player initially cooperating for the first two rounds, and then plays T4T until the opponent defects twice in a row. Once the opponent defects twice, the player defects in response, and then goes back to playing T4T. This is a variation of the T4T strategy with a forgiving approach in the first two rounds, allowing the opponent to show their intentions before reacting accordingly.

```

class T42TPlayer extends Player {
    // Cooperates then plays T4T until opponent defects twice in a row
    // defects, then goes back to playing T4T
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n < 2) {
            return 0; // cooperate by default for first two rounds
        } else {
            // If the opponent defected twice in a row, defect in response
            if (oppHistory1[n - 1] == 1 && oppHistory1[n - 2] == 1) {
                return 1;
            } else {
                return oppHistory1[n - 1]; // copy opponent's previous move
            }
        }
    }
}

```

*Figure 8: Code Snippet of T42TPlayer*

### 2.2.3. ForgivingT4TPlayer (Forgiving Tit-for-Tat Player)

This strategy involves the player initially cooperating, and forgiving a number of instances of the opponent's defection equal to the forgiveness threshold, before switching to defect. This is a variation of the T4T strategy, but with a more forgiving approach that allows for occasional mistakes by the opponent.

```

class ForgivingT4TPlayer extends Player {
    // Cooperates then plays T4T
    // if opponent defects, cooperates for a threshold before switching to defect
    int forgivenessThreshold = 2;

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0) {
            return 0; // cooperate by default for first round
        } else {
            // If the opponent defected last round and we have not reached the forgiveness
            // threshold, cooperate this round
            if (oppHistory1[n - 1] == 1 && n <= forgivenessThreshold) {
                return 0;
            } else {
                return oppHistory1[n - 1]; // copy opponent's previous move
            }
        }
    }
}

```

*Figure 9: Code Snippet of ForgivingT4TPlayer*



### 2.2.4. GrimTriggerPlayer

This strategy involves the player initially cooperating but will defect for the remainder of the game if the opponent defects at any point. This strategy is quite harsh and unforgiving, but can be effective in deterring defection.

```
class GrimTriggerPlayer extends Player {
    // Cooperates as long as both players cooperates
    // but defects if other player defects
    boolean defect = false;

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (defect) {
            return 1; // always defect after initial defection
        } else if (n > 0 && oppHistory1[n - 1] == 1) {
            defect = true; // trigger defection if opponent ever defects
            return 1;
        } else {
            return 0; // cooperate otherwise
        }
    }
}
```

Figure 10: Code Snippet of GrimTriggerPlayer

### 2.2.5. SpitefulPlayer

This strategy involves the player initially cooperating but will defect for the remainder of the game unless both opponents cooperate for at least half of the game. The strategy is motivated by spite and seeks to punish opponents who do not cooperate consistently.

```
class SpitefulPlayer extends Player {
    // Always defects unless both opponents cooperates for at least half the game
    int rounds = 0; // number of rounds played
    int cooperations = 0; // number of times both opponents have cooperated

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 0; // cooperate by default
        if (oppHistory1[n - 1] == 1 || oppHistory2[n - 1] == 1) {
            cooperations++;
        }
        rounds++;
        if (cooperations >= rounds / 2)
            return 0; // cooperate if both opponents have cooperated for at least half of the game
        else
            return 1; // always defect
    }
}
```

Figure 11: Code Snippet of GrimTriggerPlayer

### 2.2.6. PavlovPlayer

This strategy involves starting with cooperation and then switching actions based on the payoff of the previous round. Specifically, if the player received a high payoff in the previous round (both players cooperated or both players defected), then the player will continue with the same action. If the player received a low payoff in the previous round (the player defected while the opponent cooperated or vice versa), then the player will switch to the opposite action. This strategy aims to maintain cooperation by adjusting the player's action based on the success or failure of the previous round.

```
class PavlovPlayer extends Player {
    // Starts with cooperation
    // Switch to opposite action if low payoff
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0) {
            return 0; // start with cooperation
        } else if ((myHistory[n - 1] == oppHistory1[n - 1] && myHistory[n - 1] == 0)
            || (myHistory[n - 1] != oppHistory1[n - 1] && myHistory[n - 1] == 1)) {
            return 0; // continue with cooperation if last round was successful or I defected
        } else {
            return 1; // defect otherwise
        }
    }
}
```

Figure 12: Code Snippet of PavlovPlayer

### 2.2.7. GradualPlayer

This strategy involves the player initially cooperating and gradually becoming more likely to defect if the opponent defects. This strategy aims to avoid immediate retaliation while still maintaining a degree of deterrence and can be effective against players who defect occasionally.

```
class GradualPlayer extends Player {
    // Starts with cooperation
    // gradually becomes more likely to defect as game progresses
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0) {
            return 0; // cooperate by default for first round
        } else {
            int numDefects = 0;

            // Count the number of times the opponent has defected in the last four rounds
            for (int i = n - 1; i >= n - 4 && i >= 0; i--) {
                if (oppHistory1[i] == 1) {
                    numDefects++;
                }
            }

            // If the opponent has defected more than once in the last four rounds, defect
            // this round
            if (numDefects > 1) {
                return 1;
            } else {
                return 0; // otherwise, cooperate
            }
        }
    }
}
```

Figure 13: Code Snippet of GradualPlayer

### 2.2.8. AlternateCooperateDefectPlayer

This strategy involves the player alternating between cooperating and defecting regardless of the opponent's actions. This strategy is simple and does not take the opponent's actions into account, and may be effective against other simple strategies, but may be less effective against more complex and adaptive strategies.

```
class AlternateCooperateDefectPlayer extends Player {
    // Alternates between cooperation and defection
    // Starts with cooperation
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0) {
            return 0; // cooperate by default for first round
        } else {
            // Alternate between cooperating and defecting each round
            if (n % 2 == 0) {
                return 1; // defect on even rounds
            } else {
                return 0; // cooperate on odd rounds
            }
        }
    }
}
```

*Figure 14: Code Snippet of AlternateCooperateDefectPlayer*

## 2.3. Tournament

In the tournament, two strategies from the example strategies, *RandomPlayer* and *FreakyPlayer*, were excluded because of their unpredictable nature. *RandomPlayer* chooses its move at random with a 50/50 chance for each move, while *FreakyPlayer* randomly determines at the start of the match to either always cooperate or always defect. These unpredictable strategies could confuse and make it difficult for other players to predict their next move, making it harder to form a coherent strategy in response. Therefore, they were excluded from the tournament to ensure that the competition remained fair and the strategies used were more calculable.

### 2.3.1. Results of Strategies

Table 2: Table of Results for all Strategies

Player (Ranked by Average Points)	Points (1 <sup>st</sup> Run)	Points (2 <sup>nd</sup> Run)	Points (3 <sup>rd</sup> Run)	Average Points
<b>GrimTriggerPlayer</b>	468.8816	465.2712	464.95682	<b>466.369873</b>
<b>GradualPlayer</b>	462.0946	462.4742	461.20438	<b>461.924393</b>
<b>ForgivingT4TPlayer</b>	459.31348	459.0363	458.87607	<b>459.075283</b>
<b>T42TPlayer</b>	459.21878	459.00952	458.87607	<b>459.03479</b>
<b>PavlovPlayer</b>	455.34335	456.10834	455.7367	<b>455.729463</b>
<b>T4TPlayer</b>	452.03394	455.5164	454.7063	<b>454.085547</b>
<b>TolerantPlayer</b>	438.94092	439.13287	439.8664	<b>439.313397</b>
<b>NicePlayer</b>	435.70502	435.49347	435.50314	<b>435.56721</b>
<b>SuspiciousT4TPlayer</b>	404.90005	405.1801	404.741	<b>404.940383</b>
<b>SpitefulPlayer</b>	397.42404	396.8107	397.56903	<b>397.267923</b>
<b>AlternateCooperateDefectPlayer</b>	362.29626	361.5872	361.43845	<b>361.77397</b>
<b>NastyPlayer</b>	344.69263	344.22498	345.29584	<b>344.737817</b>

By analyzing the results of the tournament for three runs, the top three players were determined to be *GrimTriggerPlayer*, *GradualPlayer*, and *ForgivingT4TPlayer*, with average points of 466.369873, 461.924393, and 459.075283, respectively. These players have consistently performed well in the tournament across multiple runs. Exploring a combination of these three

strategies can potentially lead to a dominant strategy that performs better than any of the individual strategies alone.

### 3. Implementation

To determine the most effective combination of players, the top 3 performing players from the previous simulations were chosen, namely (a) *GrimTriggerPlayer*; (b) *GradualPlayer*; and (c) *ForgivingT4TPlayer*. A total of four combinations of these players were created, with three combinations including two players each, as well as one combination with all three players.

#### 3.1. Combination of *GrimTriggerPlayer* and *GradualPlayer*

The combination of *GrimTriggerPlayer* and *GradualPlayer* (Figure 15) creates a strategy that is forgiving towards the other player's cooperation, but punishes them for defection. The *GrimTriggerWithGradualPlayer* starts with cooperation, like *GradualPlayer*, and only defects if the other player defects. However, once the other player defects, the *GrimTriggerWithGradualPlayer* switches to defect mode and only cooperates again after a set number of rounds of mutual cooperation, similar to the *GrimTriggerPlayer*. This combination allows the player to maintain cooperation as long as possible, while still being able to retaliate against defection, creating a balance between cooperation and defection. The threshold value of 3 was chosen for balancing the need to punish defection and maintain cooperation in the long run. It allows enough rounds to pass before switching back to cooperation mode to prevent immediate retaliation. A higher threshold would be more forgiving, while a lower threshold would be overly aggressive, damaging long-term cooperation. Therefore, 3 was considered a reasonable compromise.

```

class GrimTriggerWithGradualPlayer extends Player {
    // This strategy combines elements of GrimTriggerPlayer and GradualPlayer.
    // It cooperates until the opponent defects, at which point it enters a
    // "grim trigger" mode and defects for a certain number of rounds before
    // reverting back to cooperation.

    private boolean defectMode = false; // Tracks whether the strategy is in defect mode
    private int roundsSinceLastDefect = -1; // Counts the number of rounds since the last defect
    private int threshold = 3; // The number of rounds to defect before reverting back to cooperation

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

        // Check if opponent defected last move
        if (oppHistory1[oppHistory1.length - 1] == 1) {
            // If the opponent defected, enter defect mode and reset the rounds counter
            defectMode = true;
            roundsSinceLastDefect = 0;
        }

        // Play according to strategy
        if (defectMode) {
            if (roundsSinceLastDefect >= threshold) {
                // If enough rounds have passed since the last defect, exit defect mode
                defectMode = false;
            } else {
                // Otherwise, increment the rounds counter and continue to defect
                roundsSinceLastDefect++;
            }
            return 1; // Defect if in defect mode
        } else {
            return 0; // Cooperate if not in defect mode
        }
    }
}

```

*Figure 15: Code Snippet of GrimTriggerWithGradualPlayer*

### 3.2. Combination of *GrimTriggerPlayer* and *ForgivingT4TPlayer*

The *GrimTriggerWithForgivenessPlayer* (Figure 16) combines elements of both the *GrimTriggerPlayer* and the *ForgivingT4TPlayer* strategies. Similar to *GrimTriggerPlayer*, the player will defect if either opponent has defected, and will continue to defect until both opponents cooperate for two consecutive rounds. However, the *ForgivingT4TPlayer* element comes into play when the opponent cooperates again after triggering the GrimTrigger strategy. Instead of continuing to defect, the player will switch back to a T4T strategy of cooperating as long as both opponents continue to cooperate. This combination of strategies allows the player to maintain a cooperative relationship with both opponents as long as they continue to cooperate, but also includes a punishing element to deter defection. This combination was eventually chosen as the selected player for Soh\_QianYi\_Player as it produced the best results.

```
class GrimTriggerWithForgivenessPlayer extends Player {
    // Cooperates first, plays T4T as long as opponent cooperates
    // but defects if other player defects
    // until opponent cooperates twice in a row

    boolean triggered = false; // whether the player has been triggered (GrimTrigger)
    int roundsSinceDefection = 0; // number of rounds since the opponent last defected (ForgivingT4T)

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 0; // cooperate by default
        if (triggered)
            return 1; // always defect if triggered (GrimTrigger)
        if (oppHistory1[n - 1] == 0 && oppHistory2[n - 1] == 0)
            return 0; // cooperate if both opponents cooperated last round (ForgivingT4T)
        else if (oppHistory1[n - 1] == 1 || oppHistory2[n - 1] == 1) {
            if (roundsSinceDefection >= 2) {
                triggered = true;
                return 1; // always defect if triggered (GrimTrigger)
            } else {
                roundsSinceDefection++;
                return Math.min(oppHistory1[n - 1], oppHistory2[n - 1]); // tit-for-tat (ForgivingT4T)
            }
        } else {
            roundsSinceDefection = 0;
            return 0; // cooperate if both opponents defected last round (ForgivingT4T)
        }
    }
}
```

Figure 16: Code Snippet of *GrimTriggerWithForgivenessPlayer*

### 3.3. Combination of *GradualPlayer* and *ForgivingT4TPlayer*

The *GradualPlayer* and *ForgivingT4TPlayer* strategies are combined in the *GradualWithForgivenessPlayer* (Figure 17) to create a player that gradually increases its cooperation level while forgiving occasional defections. The *GradualPlayer* strategy ensures that the player starts by cooperating, then gradually increases its cooperation level until it reaches a threshold of two consecutive defections from the opponent, at which point it defects. The *ForgivingT4TPlayer* strategy ensures that the player uses a T4T strategy, but forgives occasional defections from the opponent if they have cooperated for a certain number of rounds in a row. By combining these strategies, the *GradualWithForgivenessPlayer* aims to strike a balance between cooperation and defection, gradually increasing its cooperation level while avoiding being too forgiving or too retaliatory.

```
class GradualWithForgivenessPlayer extends Player {
    // Cooperates first, plays T4T as long as opponent cooperates
    // but defects if other player defects
    // until opponent cooperates twice in a row
    // Also implements Forgiving T4T, by playing cooperatively if the
    // opponent has defected fewer than 'forgivenessThreshold' times in the past
    int forgivenessThreshold = 2; // Threshold for forgiveness
    int cooperateCount = 0; // Count of times opponent cooperated in a row
    int lastAction = 0; // Last action played
    int n = 0; // Round number

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        int oppLastAction = (oppHistory1.length == 0) ? 0 : oppHistory1[oppHistory1.length - 1];

        // Forgiving T4T: plays cooperatively if the opponent has defected fewer than
        // 'forgivenessThreshold' times in the past
        if (oppHistory1.length < forgivenessThreshold
            || oppHistory1[oppHistory1.length - forgivenessThreshold] == 0) {
            lastAction = 0; // Cooperate
        } else {
            lastAction = oppLastAction; // Play the same as the opponent's last action
        }

        // Gradual: cooperates first, plays T4T as long as opponent cooperates but
        // defects if other player defects
        // until opponent cooperates twice in a row
        if (oppLastAction == 1) { // Opponent cooperated
            cooperateCount = 0;
            return 1; // Cooperate
        } else if (cooperateCount >= 2) { // Opponent defected twice in a row
            cooperateCount = 0;
            return 0; // Defect
        } else { // Opponent defected but less than twice in a row
            cooperateCount++;
            return 1; // Cooperate
        }
    }
}
```

Figure 17: Code Snippet of *GradualWithForgivenessPlayer*



### 3.4. Combination of *GrimTriggerPlayer*, *GradualPlayer* and *ForgivingT4TPlayer*

The *GrimTriggerPlayer*, *GradualPlayer* and *ForgivingT4TPlayer* strategies are combined in the *GrimTriggerGradualWithForgivenessPlayer* (Figure 18) to create a player that is initially cooperative and forgiving, but will retaliate if either opponent defects, and will gradually become more aggressive over time if neither opponent defects. The player starts by implementing all three strategies simultaneously. In the beginning, the player follows the Grim Trigger strategy, which means it will cooperate until either opponent defects, at which point it will always defect for the rest of the game. However, the player also implements the *GradualPlayer* and *ForgivingT4TPlayer* strategies at the same time.

If both opponents continue to cooperate, the player follows the *GradualPlayer* strategy, which means it will cooperate until one of the opponents defects, at which point it will immediately defect for the next turn. This strategy gradually becomes more aggressive over time, as it requires two consecutive rounds of cooperation to reset.

If one of the opponents defects, the player follows the *ForgivingT4TPlayer* strategy, which means it will cooperate if the opponent has defected fewer than two times in the past. This strategy forgives the opponent for a certain number of defections before retaliating.

```

class GrimTriggerGradualWithForgivenessPlayer extends Player {
    // Cooperates first, then checks opponent histories to determine which strategy to implement.
    // If an opponent defected, it stops using ForgivingT4T and GrimTrigger, and switches to always defecting.
    // If either opponent defects, it switches from Gradual Player to always defecting.
    // If none of the above conditions are met, it cooperates.
    int opponent1LastMove = -1;
    int opponent2LastMove = -1;
    boolean isOpponent1Defected = false;
    boolean isOpponent2Defected = false;
    boolean isGrimTrigger = false;
    boolean isForgiving = false;
    boolean isGradual = false;

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0) {
            isGrimTrigger = true;
            isForgiving = true;
            isGradual = true;
            return 0; // First move: Cooperate
        } else {
            // Check opponent 1 history
            if (opponent1LastMove == 1 && oppHistory1[n - 1] == 0) {
                isOpponent1Defected = true;
            }
            opponent1LastMove = oppHistory1[n - 1];

            // Check opponent 2 history
            if (opponent2LastMove == 1 && oppHistory2[n - 1] == 0) {
                isOpponent2Defected = true;
            }
            opponent2LastMove = oppHistory2[n - 1];

            // Implement Grim Trigger strategy
            if (isGrimTrigger) {
                if (isOpponent1Defected || isOpponent2Defected) {
                    isGrimTrigger = false;
                    return 1;
                } else {
                    return 0;
                }
            }

            // Implement Forgiving T4T strategy
            if (isForgiving) {
                if (isOpponent1Defected || isOpponent2Defected) {
                    isForgiving = false;
                    return 1;
                } else {
                    return opponent1LastMove == 0 && opponent2LastMove == 0 ? 0 : 1;
                }
            }

            // Implement Gradual Player strategy
            if (isGradual) {
                if (opponent1LastMove == 1 || opponent2LastMove == 1) {
                    isGradual = false;
                    return 1;
                } else {
                    return 0;
                }
            }

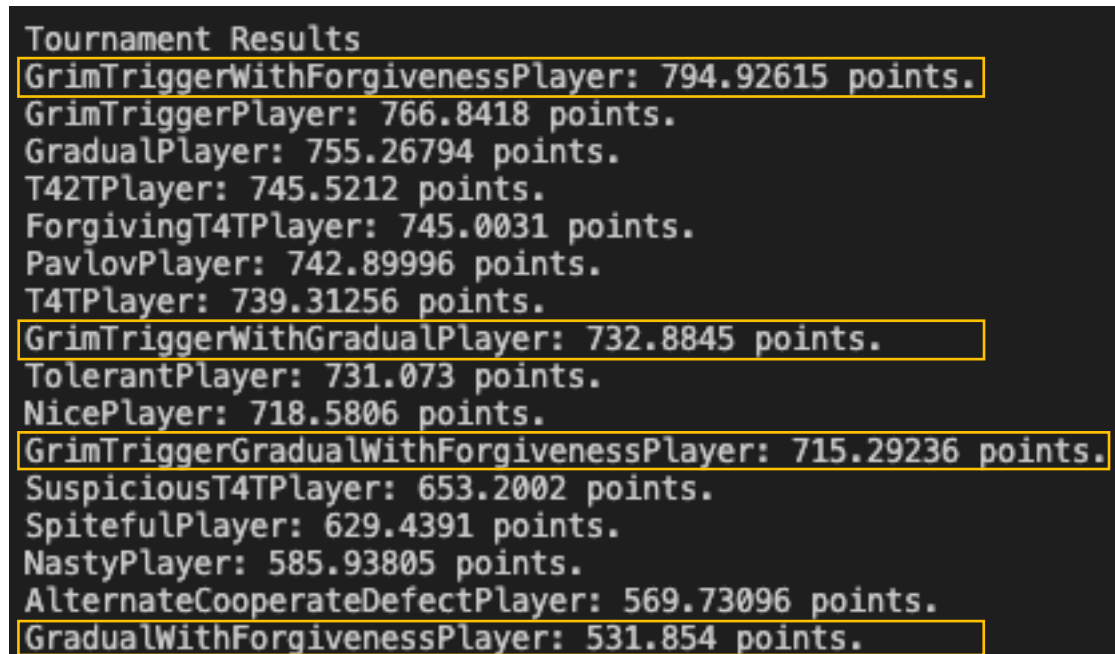
            // If none of the above strategies are active, cooperate
            return 0;
        }
    }
}

```

Figure 18: Code Snippet of GrimTriggerGradualWithForgivenessPlayer

## 4. Results for Agent Performances

The performance results of the agents were evaluated after incorporating a combination of players, and the following outcomes were observed.



```
Tournament Results
GrimTriggerWithForgivenessPlayer: 794.92615 points.
GrimTriggerPlayer: 766.8418 points.
GradualPlayer: 755.26794 points.
T42TPlayer: 745.5212 points.
ForgivingT4TPlayer: 745.0031 points.
PavlovPlayer: 742.89996 points.
T4TPlayer: 739.31256 points.
GrimTriggerWithGradualPlayer: 732.8845 points.
TolerantPlayer: 731.073 points.
NicePlayer: 718.5806 points.
GrimTriggerGradualWithForgivenessPlayer: 715.29236 points.
SuspiciousT4TPlayer: 653.2002 points.
SpitefulPlayer: 629.4391 points.
NastyPlayer: 585.93805 points.
AlternateCooperateDefectPlayer: 569.73096 points.
GradualWithForgivenessPlayer: 531.854 points.
```

*Figure 19: Performance Results*

These results show the points earned by each of the four combined strategies in a tournament against other player strategies.

- **GrimTriggerWithForgivenessPlayer** had the highest score of 794.92615 points, beating all other players. This suggests that combining the Grim Trigger and ForgivingT4T strategies was effective in winning points against other player strategies.
- **GrimTriggerWithGradualPlayer** had a lower score of 732.8845 points compared to GrimTriggerWithForgivenessPlayer. This suggests that combining the Grim Trigger and Gradual Player strategies was less effective in winning points against other player strategies compared to the combination with ForgivingT4T.
- **GrimTriggerGradualWithForgivenessPlayer** had an even lower score of 715.29236 points. This suggests that combining all three strategies was not as effective as the Grim Trigger and ForgivingT4T combination alone.
- **GradualWithForgivenessPlayer** had the lowest score of 531.854 points. This suggests that combining Forgiving Tit for Tat and Gradual Player strategies alone was not very effective in winning points against other player strategies.

## 4.1. Evaluation of Selected Agent

Based on the results in Figure 19, it is clear that the *GrimTriggerWithForgivenessPlayer* strategy, which is also my chosen agent, was the most effective in winning points against other player strategies, including the provided example strategies like *NicePlayer*, *NastyPlayer*, etc. This is likely due to the combination of the Grim Trigger and ForgivingT4T strategies, which allowed the player to both punish defection, while also forgiving and cooperating if the opponent returned to cooperation.

While the combination of Grim Trigger and Gradual Player strategies were not as effective, it is worth noting that the Gradual Player strategy may still have some value in certain scenarios where the opponent is slow to defect. On the other hand, combining all three strategies into *GrimTriggerGradualWithForgivenessPlayer* did not perform as well as *GrimTriggerWithForgivenessPlayer* alone. This may be due to the complexity of the strategy, which may have led to more mixed outcomes and less consistent behaviour.

Overall, the evaluation showed that *GrimTriggerWithForgivenessPlayer* is a strong choice for playing against a variety of example strategies.

## 5. Conclusion

In conclusion, the results of the tournament show that the combination of player strategies can greatly affect their overall performance. The *GrimTriggerWithForgivenessPlayer* strategy proved to be the most effective combination in terms of earning points against other player strategies. This suggests that the Grim Trigger and ForgivingT4T strategies complement each other well, allowing the player to take advantage of the opponent's cooperation while forgiving their occasional defection. Hence, based on these results, *GrimTriggerWithForgivenessPlayer* was chosen as my player strategy.

## 6. Reference

Kuhn, S. (2019). *Strategies for the Iterated Prisoner's Dilemma*. Retrieved from Stanford Encyclopedia of Philosophy: <https://plato.stanford.edu/entries/prisoner-dilemma/strategy-table.html>