CE/CZ4046: INTELLIGENT AGENTS

ASSIGNMENT 2: THREE PLAYER ITERATED PRISONER'S DILEMMA PROBLEM

AGARWAL ANUSHA(U2023105H)

School of Computer Science & Engineering

Nanyang Technological University

Table of Contents

1. INTRODUCTION

In the game of Prisoner's dilemma, the dominant strategy is for agents to defect even though they would have a higher payoff if they cooperate. If however, we play a repeated game, there is a possibility that this lack of cooperation can disappear.

A repeated game is one where the game is played many times such that the players remember their own choices as well as their opponents choices in the past and modify their future choice accordingly. As k (number of rounds played) approaches infinity, the average reward of each player is determined as the limit of the player's total payoff in each round divided by the number of rounds played.

This assignment deals with a three player iterated prisoner's dilemma. We aim to create an agent strategy that gets a high payoff when playing against other agents with possibly different strategies.

2. PROPOSED SOLUTION FRAMEWORK

The agent created in this assignment to play the game of Iterated Prisoner's dilemma can be found in AgarwalAnushaPlayer.java with the player name as AgarwalAnushaPlayer.

Other strategies used as motivation to create the final agent can be found in the ThreePrisonersDilemma.java file. Outputs of conducted trial tournaments between different agents can be found in the submitted text file.

3. OVERVIEW

1. Payoff Chart (Symmetric) & Analysis

C = Cooperate, D = Defect

(A,B,C)	PAYOFF (the higher the better)
(C,D,D)	0
(D,D,D)	2
(C,D,C) (C,C,D)	3
(D,D,C) (D,C,D)	5
(C,C,C)	6
(D,C,C)	8

Fig1 - Payoff Chart

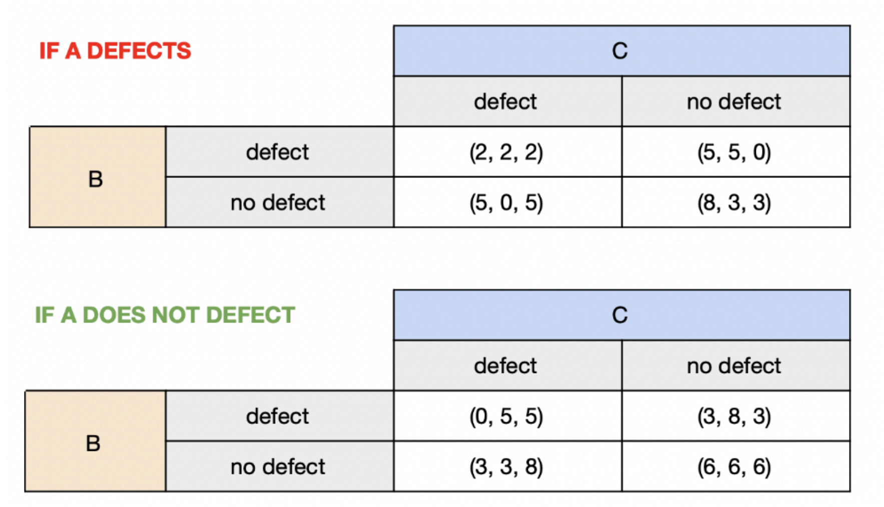
Analysis -

As seen from the chart above, the action combination that results in the highest payoff is the one where our agent defects while both other agents cooperate. However, since we assume the agents to be rational, this would disincentivize the opponents to cooperate in the future. This would result in temporary gain but overall loss in payoff.

Therefore, we must aim to achieve the highest possible payoff by maximizing cooperation but also prevent being exploited.

2. Three Player Prisoner's Dilemma Matrix

The above payoff matrix can be expressed in matrix form as follows.



IF A DEFECTS		C	
		defect	no defect
B	defect	(2, 2, 2)	(5, 5, 0)
	no defect	(5, 0, 5)	(8, 3, 3)

IF A DOES NOT DEFECT		C	
		defect	no defect
B	defect	(0, 5, 5)	(3, 8, 3)
	no defect	(3, 3, 8)	(6, 6, 6)

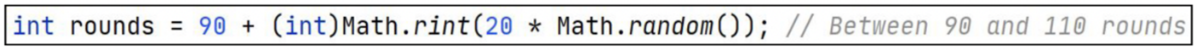
Fig 3. - Prisoner's Dilemma Matrix

Where (SCORE1, SCORE2, SCORE3) == (A,B,C)

As seen from the matrix above the highest average achievable score [C,C,C] as 6 per agent.

3. Iterated Prisoner's dilemma

For our assignment, the game is an iterated problem where we play the same game for 90 to 110 rounds. Each agent remembers its opponents as well as its own past choices and tweaks future strategy based on that.



```
int rounds = 90 + (int)Math.rint(20 * Math.random()); // Between 90 and 110 rounds
```

Fig 3. - Number of rounds

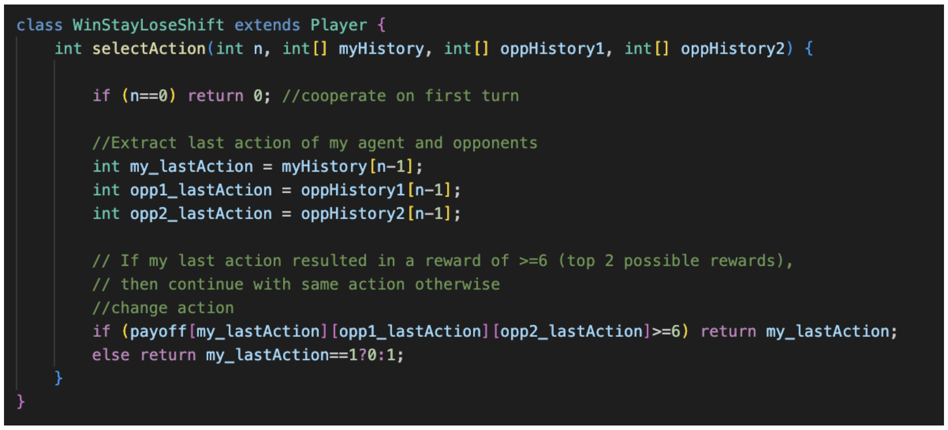
4. ADDITIONAL AGENTS

1. Agent Design

a. Win Stay Lose Shift :

This strategy is an adaptation of Pavlov Player. This agent -

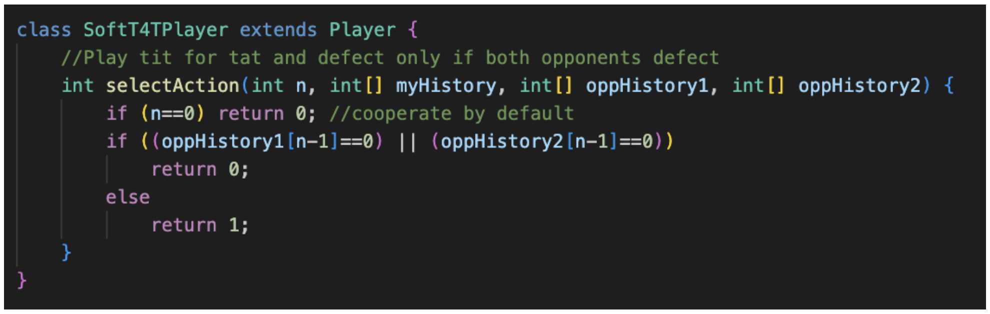
1. Cooperates on first turn
2. If the agent earns 6 or 8 as payoff, then repeat the previous round action otherwise return the opposite action



```java
class WinStayLoseShift extends Player {
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {

        if (n==0) return 0; //cooperate on first turn

        //Extract last action of my agent and opponents
        int my_lastAction = myHistory[n-1];
        int opp1_lastAction = oppHistory1[n-1];
        int opp2_lastAction = oppHistory2[n-1];

        // If my last action resulted in a reward of >=6 (top 2 possible rewards),
        // then continue with same action otherwise
        //change action
        if (payoff[my_lastAction][opp1_lastAction][opp2_lastAction]>=6) return my_lastAction;
        else return my_lastAction==1?0:1;
    }
}
```

b. Soft Tit For Tat Player:

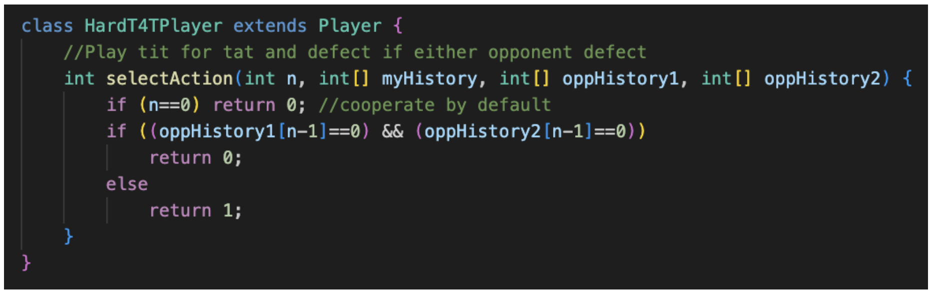
1. Cooperate on the first turn
2. If either opponent cooperates then cooperate otherwise if both defect then punish by defecting



```java
class SoftT4TPlayer extends Player {
    //Play tit for tat and defect only if both opponents defect
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n==0) return 0; //cooperate by default
        if ((oppHistory1[n-1]==0) || (oppHistory2[n-1]==0))
            return 0;
        else
            return 1;
    }
}
```

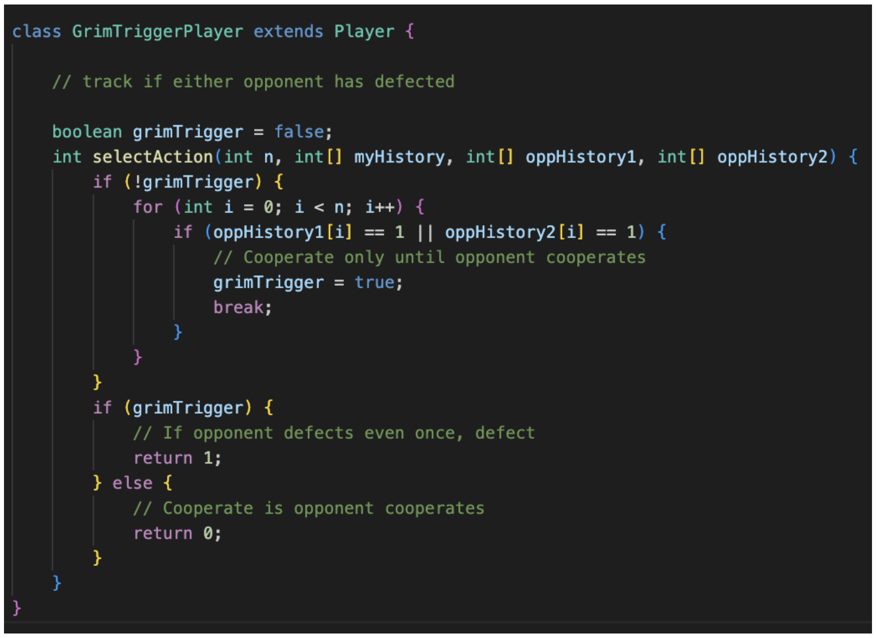
c. Hard Tit For Tat Player:

1. Cooperate on first turn
2. Cooperate only if both opponents cooperate otherwise if either opponent defects then retaliate by defecting



```java
class HardT4TPlayer extends Player {
    //Play tit for tat and defect if either opponent defect
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n==0) return 0; //cooperate by default
        if ((oppHistory1[n-1]==0) && (oppHistory2[n-1]==0))
            return 0;
        else
            return 1;
    }
}
```

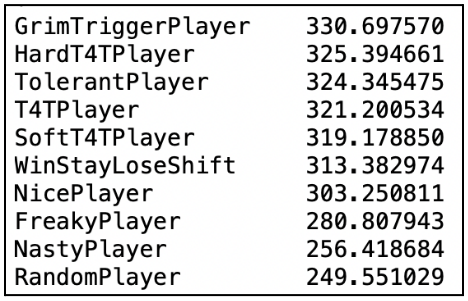
d. Grim Trigger Player:
   1. If either opponent has ever defected in the past, then switch grimTrigger mode to True and defect always
   2. If not, then cooperate until both opponents are cooperating.



```
class GrimTriggerPlayer extends Player {

    // track if either opponent has defected

    boolean grimTrigger = false;
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (!grimTrigger) {
            for (int i = 0; i < n; i++) {
                if (oppHistory1[i] == 1 || oppHistory2[i] == 1) {
                    // Cooperate only until opponent cooperates
                    grimTrigger = true;
                    break;
                }
            }
        }
        if (grimTrigger) {
            // If opponent defects even once, defect
            return 1;
        } else {
            // Cooperate is opponent cooperates
            return 0;
        }
    }
}
```

2. Agents Performance Analysis

After repeating this tournament for given agents as well as additional agents for 50 times, and leveraging the value it was found that GrimTrigger Player usually wins.



```
GrimTriggerPlayer    330.697570
HardT4TPlayer        325.394661
TolerantPlayer       324.345475
T4TPlayer            321.200534
SoftT4TPlayer        319.178850
WinStayLoseShift     313.382974
NicePlayer           303.250811
FreakyPlayer         280.807943
NastyPlayer          256.418684
RandomPlayer         249.551029
```

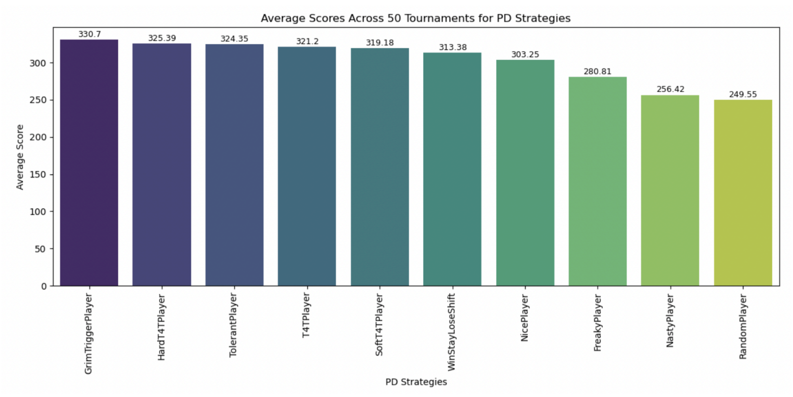


Fig 4. Agents Performance Analysis

We observe that both the winning strategies - GrimTrigger and HardT4T Player do not let any agent take advantage of it, we use the same principle while designing our own agent. Therefore one of the rules should be to always protect oneself and not let anyone take advantage of your cooperation.

5. PROPOSED AGENT - Agarwal_Anusha_Player

1. Assumptions

To create our agent strategy, we assume that the competing agents are rational and want to maximize their overall payoff.

Even though an action combination of (D,C,C) where we defect and others cooperate will get the highest payoff, it is unlikely that the opponents will not retaliate resulting in temporary gain but future loss.

As seen for (C,C,C) we get payoff as 6 (second highest). Therefore we design our agent to cooperate as long as opponents are cooperating as this results in mutual gain. However, our agent must also retaliate for opponent defects and not be taken advantage of.

2. Agarwal_Anusha_Player Design

As seen from above discussions, there needs to be a balance between cooperation (C) as well as protecting itself against opponent defections. Since we have seen that the agent always needs to protect itself and retaliate against defections, and at the same time for social benefit, one should try to be cooperative.

To design an efficient strategies, we abide by Axelrod's guidelines -

1. Don't Be Envious
2. Be Nice
3. Retaliate Properly
4. Don't Hold Grudges

Following these guidelines we create our own strategy for the 3 player Iterated Prisoner's dilemma problem.

1. Rule 1: Balance trusting others and being unpredictable at the same time

Ideally we aim our agent to establish trust but at the same time also be unpredictable. If the other agent understands our strategy and tries to exploit our vulnerability, then unpredictability will throw them off

2. **Rule 2: Prioritize self-protection**

   Throughout the game, we keep track of the opponent's defection probability. If seen that the opponent defects above a threshold, then retaliate firmly with defection

3. **Rule 3: Reciprocate Cooperation**

   As long as the opponents cooperate and have a relatively high probability of cooperation, our agent must also be cooperative for economic welfare.

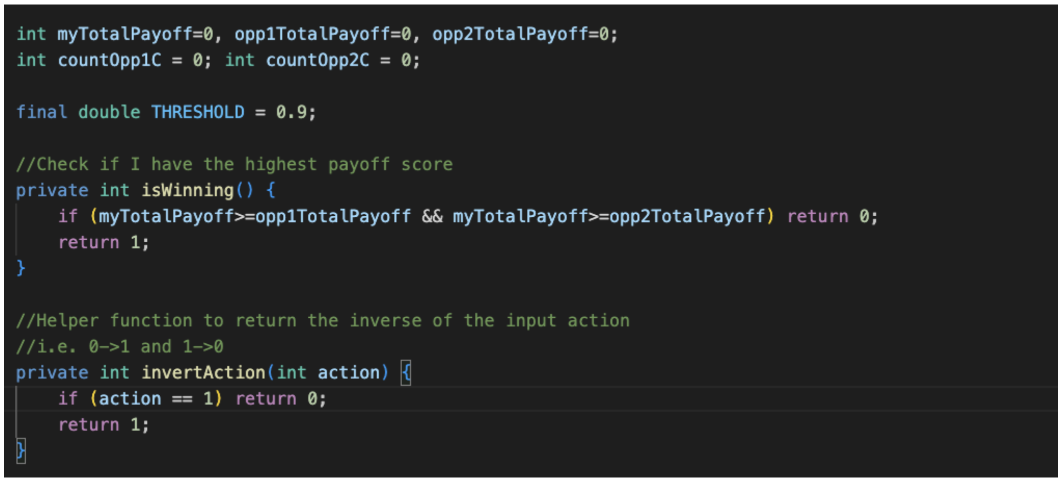
4. **Rule 4: Transform losing scenario to mutual loss**

   Since we also keep track of our current total payoff after each game, we compare it with that of the opponent. Our agent chooses to cooperate as long as we are winning otherwise defects. This is done when in initial rounds we are unable to be sure about opponent's defection probability

3. Agarwal_Anusha_Player Implementation (Code)

To implement our player - 'Agarwal_Anusha_Player',

1. We first start with defining the necessary variables for required information such as storing the total payoff till now for my Agent, and the opponent agents.
   This is done by using the action in the previous round and using the payoff array to find the agent's current payoff and adding it to the payoff till now.
2. We define two other helper functions -
   '`isWinning()`' checks if our agent has the highest payoff score or now.
   '`invertAction()`' inverses the input action i.e. 0->1 and 1->0



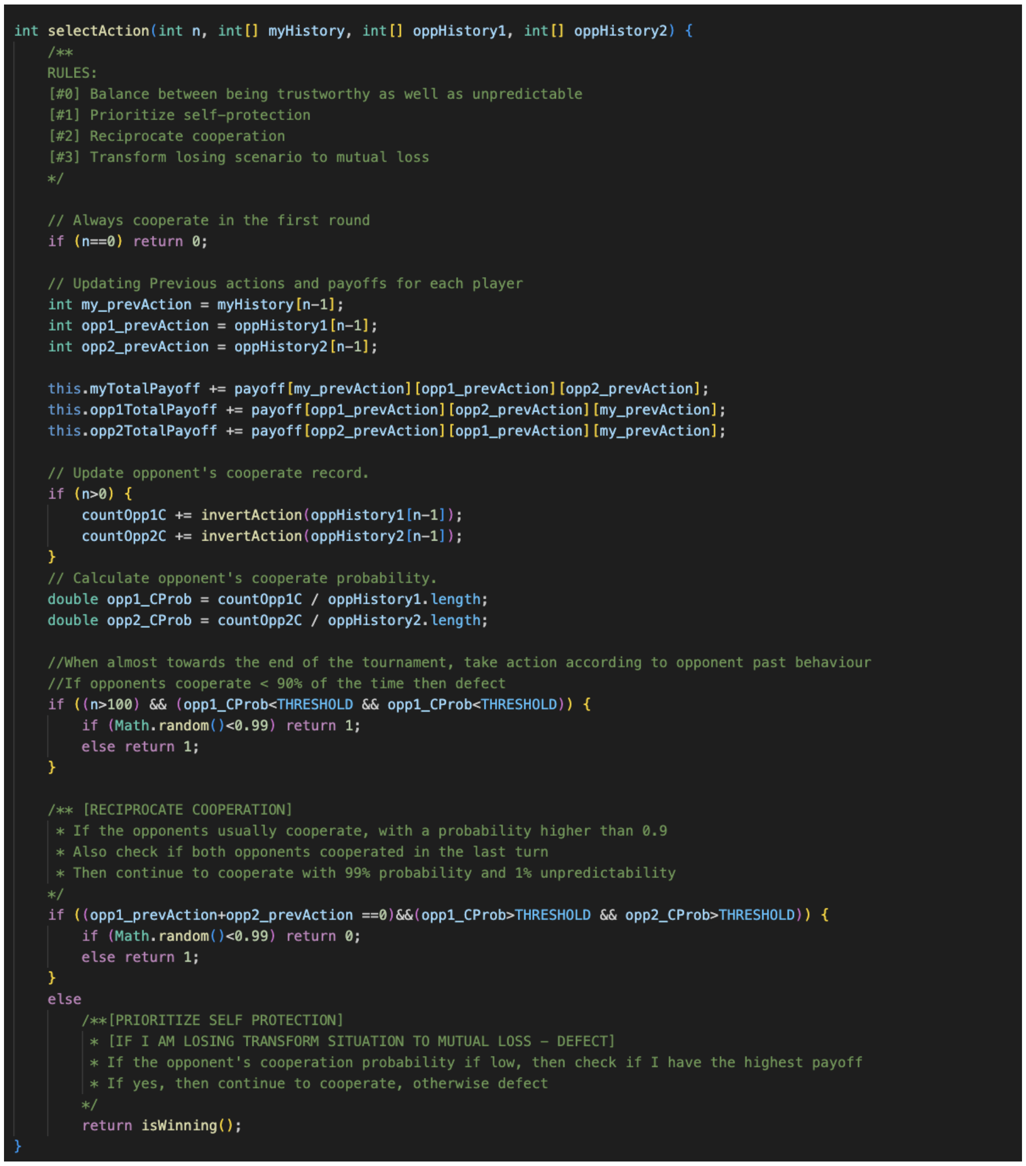
```java
int myTotalPayoff=0, opp1TotalPayoff=0, opp2TotalPayoff=0;
int countOpp1C = 0; int countOpp2C = 0;

final double THRESHOLD = 0.9;

//Check if I have the highest payoff score
private int isWinning() {
    if (myTotalPayoff>=opp1TotalPayoff && myTotalPayoff>=opp2TotalPayoff) return 0;
    return 1;
}

//Helper function to return the inverse of the input action
//i.e. 0->1 and 1->0
private int invertAction(int action) {
    if (action == 1) return 0;
    return 1;
}
```

For the `selectAction()` function the strategy used is as follows -

1. Always cooperate in the first round
2. If we are nearing the end of the tournament, take firm action. If the opponents have a high probability of cooperation (>0.9) then cooperate otherwise defect.

This is done in accordance to rule 2 - [Prioritize Self-Protection]
We take this action with 0.99 probability and, 0.01 randomness to maintain unpredictability of agent as well

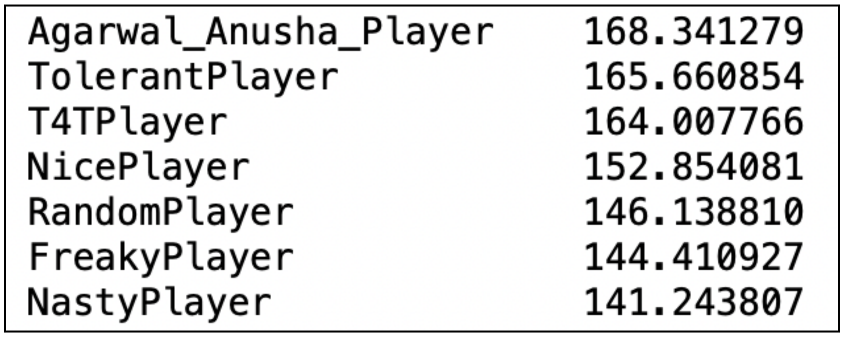
3. If both opponents cooperated in the previous round, AND their cooperation probability is >0.9 then with 0.01 randomness, our agent must cooperate
4. If not, i.e. either one of the agents defected in the last round or their cooperation probability is low, then we decide strategy based on if our agent is winning. If winning, then cooperate for economic welfare and long term mutual benefit, otherwise defect.



```java
int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
    /**
    RULES:
    [#0] Balance between being trustworthy as well as unpredictable
    [#1] Prioritize self-protection
    [#2] Reciprocate cooperation
    [#3] Transform losing scenario to mutual loss
    */

    // Always cooperate in the first round
    if (n==0) return 0;

    // Updating Previous actions and payoffs for each player
    int my_prevAction = myHistory[n-1];
    int opp1_prevAction = oppHistory1[n-1];
    int opp2_prevAction = oppHistory2[n-1];

    this.myTotalPayoff += payoff[my_prevAction][opp1_prevAction][opp2_prevAction];
    this.opp1TotalPayoff += payoff[opp1_prevAction][opp2_prevAction][my_prevAction];
    this.opp2TotalPayoff += payoff[opp2_prevAction][opp1_prevAction][my_prevAction];

    // Update opponent's cooperate record.
    if (n>0) {
        countOpp1C += invertAction(oppHistory1[n-1]);
        countOpp2C += invertAction(oppHistory2[n-1]);
    }
    // Calculate opponent's cooperate probability.
    double opp1_CProb = countOpp1C / oppHistory1.length;
    double opp2_CProb = countOpp2C / oppHistory2.length;

    //When almost towards the end of the tournament, take action according to opponent past behaviour
    //If opponents cooperate < 90% of the time then defect
    if ((n>100) && (opp1_CProb<THRESHOLD && opp1_CProb<THRESHOLD)) {
        if (Math.random()<0.99) return 1;
        else return 1;
    }

    /** [RECIPROCATE COOPERATION]
     * If the opponents usually cooperate, with a probability higher than 0.9
     * Also check if both opponents cooperated in the last turn
     * Then continue to cooperate with 99% probability and 1% unpredictability
    */
    if ((opp1_prevAction+opp2_prevAction ==0)&&(opp1_CProb>THRESHOLD && opp2_CProb>THRESHOLD)) {
        if (Math.random()<0.99) return 0;
        else return 1;
    }
    else
        /**[PRIORITIZE SELF PROTECTION]
         * [IF I AM LOSING TRANSFORM SITUATION TO MUTUAL LOSS - DEFECT]
         * If the opponent's cooperation probability if low, then check if I have the highest payoff
         * If yes, then continue to cooperate, otherwise defect
        */
        return isWinning();
}
```

4. Agarwal_Anusha_Player Performance

When competing the newly created player - Agarwal_Anusha_Player, against the initially given strategies, we find that average over 50 tournaments suggest Agarwal_Anusha_Player as the winner as can be seen below -



```
Agarwal_Anusha_Player    168.341279
TolerantPlayer           165.660854
T4TPlayer                164.007766
NicePlayer               152.854081
RandomPlayer             146.138810
FreakyPlayer             144.410927
NastyPlayer              141.243807
```

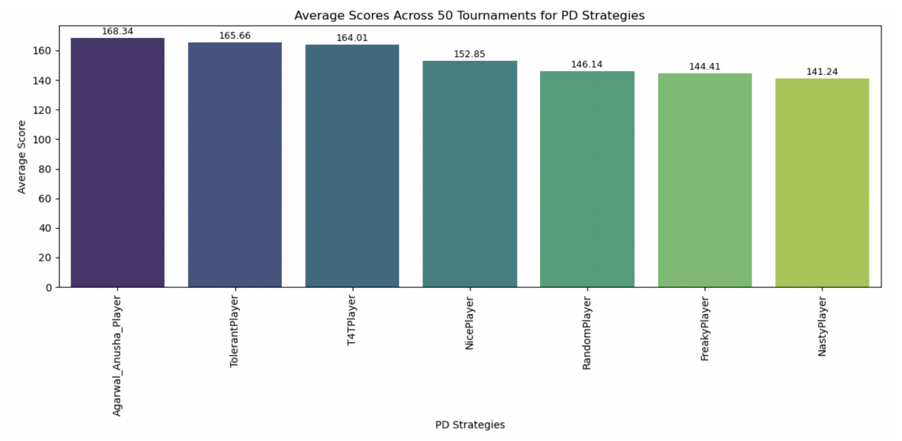
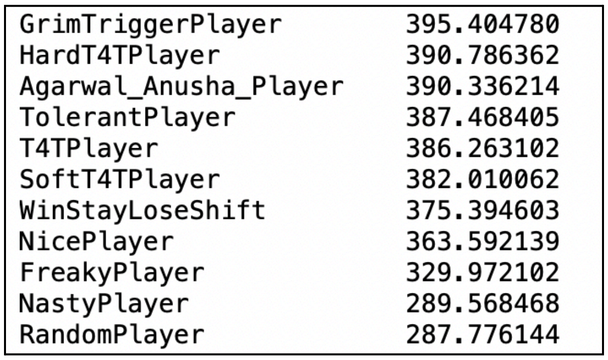


Fig 5. Agarwal_Anusha_Player vs Initial Agents

However, while competing the agents against all other agents, including the additional ones created in this assignment we find the result different. We see that even though still averaging over 50 tournaments, our agent still comes in top 3, it might not be the top performing agent.



```
GrimTriggerPlayer        395.404780
HardT4TPlayer            390.786362
Agarwal_Anusha_Player    390.336214
TolerantPlayer           387.468405
T4TPlayer                386.263102
SoftT4TPlayer            382.010062
WinStayLoseShift         375.394603
NicePlayer               363.592139
FreakyPlayer             329.972102
NastyPlayer              289.568468
RandomPlayer             287.776144
```

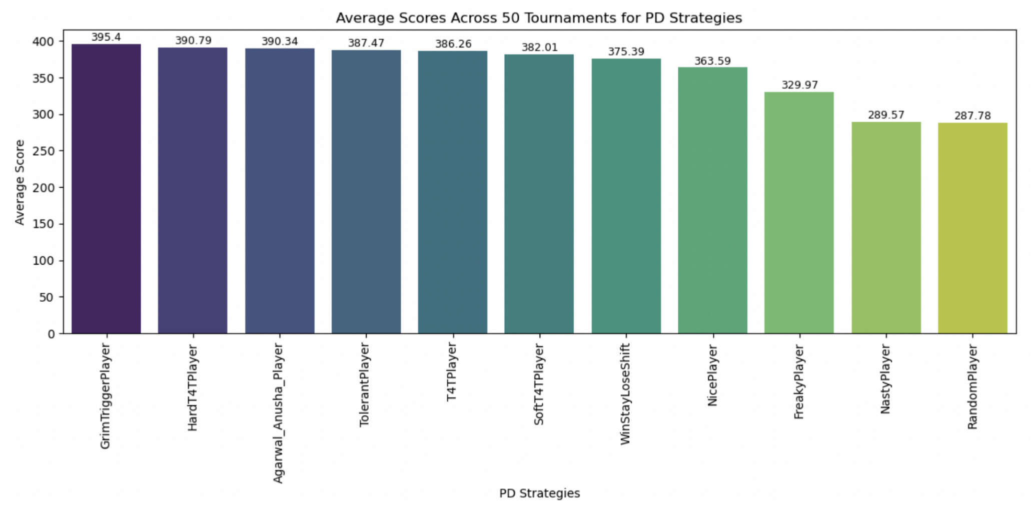


Fig 6. Agarwal_Anusha_Player vs All Agents

4.1. Agarwal_Anusha_Player vs Random Player

Random Player is purely driven by randomness, which may also work towards its disadvantage. While it may be hard to decipher such an agent's next action, our agent does reasonably well by purely reacting to the opponent's previous action.

4.2. Agarwal_Anusha_Player vs Nasty Player

As seen in the rules that define our agent, our agent does not let anyone take advantage of it by using unpredictability as well as self-preservation. It retaliates defection of opponent

4.3. Agarwal_Anusha_Player vs Nice Player

Our agent follows rule 3, to always cooperate in a cooperative environment. Since Nice Player is always cooperative, our agent rewards it by reciprocating cooperation which results in mutual gain.

4.4. Agarwal_Anusha_Player vs Freaky Player

Freaky Player decides to either be Nice or Nasty, and our agent is able to win against both strategies

4.5. Agarwal_Anusha_Player vs T4T Player

Tit for Tat strategy is too predictable as it just rewards or punishes based on the previous action of the opponent, other opponents who learn its strategy may take advantage of this.
Therefore, learning from this strategy, our agent introduces unpredictability so as to not be easily exploited.

4.6. Agarwal_Anusha_Player vs Tolerant Player

While Tolerant Player only considers probabilities of cooperation of opponents, our agent also considers the last action of opponents to help make a decision about the next action. Because of this extra layer of reactivity, our agent does reasonably well against Tolerant Player.

4.7. Agarwal_Anusha_Player vs WinStayLoseShift Player

As seen, WinStayLoseShift only focuses on the previous action of the agents and overlooks the entire history. However, learning from this, our agent is able to calculate cooperation probability of its opponents using their entire history and is hence more fair and reactive.

6. CONCLUSION

To conclude, our agent does reasonably well against all other strategic agents and outperforms them while also being cooperative wherever possible for the greater economic welfare and social benefit.

Our agent abides by Axelrod's guidelines, as well as follows the predefined rules to be trustworthy yet not predictable, to prioritize self preservation, to be cooperative where cooperation is reciprocated as well as not let other agents take advantage of it.