NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CE/CZ4046 Intelligent Agents

Assignment 2

11th April 2023

Jolene Tan

U1921255B

Table of Contents

1 Introduction

1.1 Problem Description

Develop a strategy for an agent in a three player repeated prisoners' dilemma.

1.2 Environment

In this simulation,

- Triples of players will play each other repeatedly in a 'match'
- A match consists of about 100 rounds
- Your score from that match is the average of the payoffs from each round of that match

For each round,

- Your strategy is given a list of the previous plays, so you can remember what your opponent did
- Must compute the next action

Agent actions,

- Cooperation represented by integer 0
- Defection represented by integer 1

1.3 Payoff Matrix

The payoffs are as follows,

		Opponent 1 action, Opponent 2 action			
		Cooperate, Cooperate	*Cooperate, Defect*	*Defect, Cooperate*	*Defect, Defect*
My action	*Cooperate*	6	3	8	5
	Defect	3	0	5	2

Table 1. Payoff Matrix

2 Agent Analysis

2.1 Given Agents

2.1.1 NicePlayer

- Player always cooperates

2.1.2 NastyPlayer

- Player always defects

2.1.3 RandomPlayer

- Player chooses his action, cooperate or defect, at random every time

2.1.4 TolerantPlayer

- Player examines the histories of his opponents
- Only defects if at least half of their actions have been defects

2.1.5 FreakyPlayer

- Player determines at the beginning of the match
- Either to consistently cooperate or consistently defect

2.1.6 T4TPlayer

- Tit-For-Tat strategy
- Player randomly chooses an opponent at each round
- Cooperate in the first round
- For subsequent rounds, mirror the opponent's previous move

2.1.7 Evaluation

Metric	Results

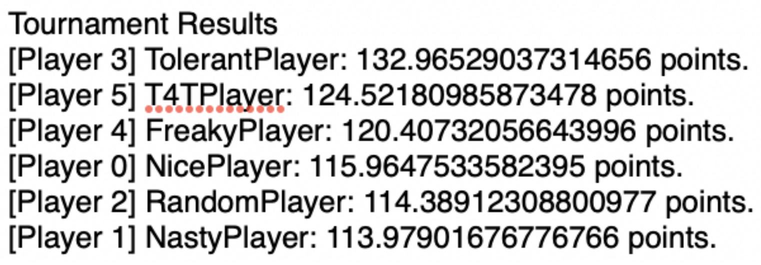
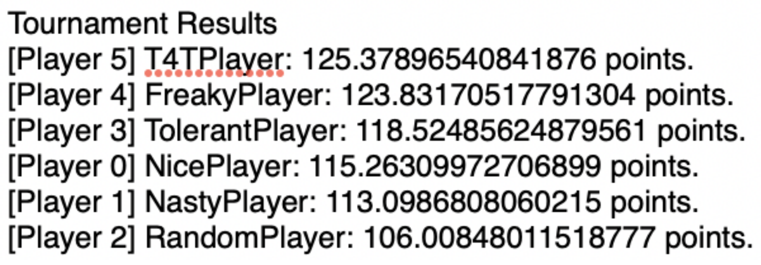
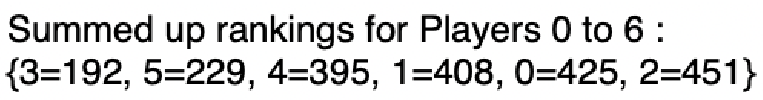
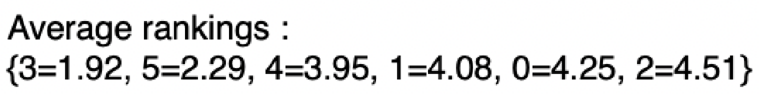
Round 99	 Tournament Results [Player 3] TolerantPlayer: 132.96529037314656 points. [Player 5] T4TPlayer: 124.52180985873478 points. [Player 4] FreakyPlayer: 120.40732056643996 points. [Player 0] NicePlayer: 115.9647533582395 points. [Player 2] RandomPlayer: 114.38912308800977 points. [Player 1] NastyPlayer: 113.97901676776766 points.
Round 100	 Tournament Results [Player 5] T4TPlayer: 125.37896540841876 points. [Player 4] FreakyPlayer: 123.83170517791304 points. [Player 3] TolerantPlayer: 118.52485624879561 points. [Player 0] NicePlayer: 115.26309972706899 points. [Player 1] NastyPlayer: 113.0986808060215 points. [Player 2] RandomPlayer: 106.00848011518777 points.
Summed Up Rankings	 Summed up rankings for Players 0 to 6 : {3=192, 5=229, 4=395, 1=408, 0=425, 2=451}
Average Rankings	 Average rankings : {3=1.92, 5=2.29, 4=3.95, 1=4.08, 0=4.25, 2=4.51}

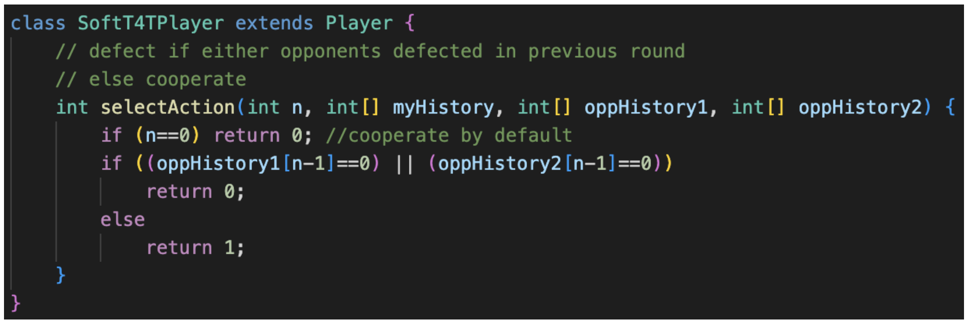
Table 2. Match 1 Results

From the results shown in Table 2 above, TolerantPlayer, T4TPlayer and FreakyPlayer are the top 3 performing players out of the given agents. TheTolerantPlayer performs the best which suggests that adopting a strategy that tolerates the actions of others can be effective as it permits cooperation and forgiveness, even in response to the opponents' defection. In repeated games such as this, players can establish a reputation and develop trust with their opponents. A player who demonstrates tolerance and forgiveness can signal that they are a reliable and trustworthy partner, thereby encouraging their opponents to cooperate in future rounds. Furthermore, such a strategy can prevent the cycle of retaliation and defection, which can lead to harm for all players involved. Ultimately, adopting a tolerant strategy can lead to a cooperative and mutually beneficial outcome in the long term.

2.2 Additional Agents

2.2.1 SoftT4TPlayer

- Soft Tit-For-Tat strategy
- Cooperate in the first round
- For subsequent rounds, as long as one opponent defects, player defects
- If both opponents cooperate, player cooperates

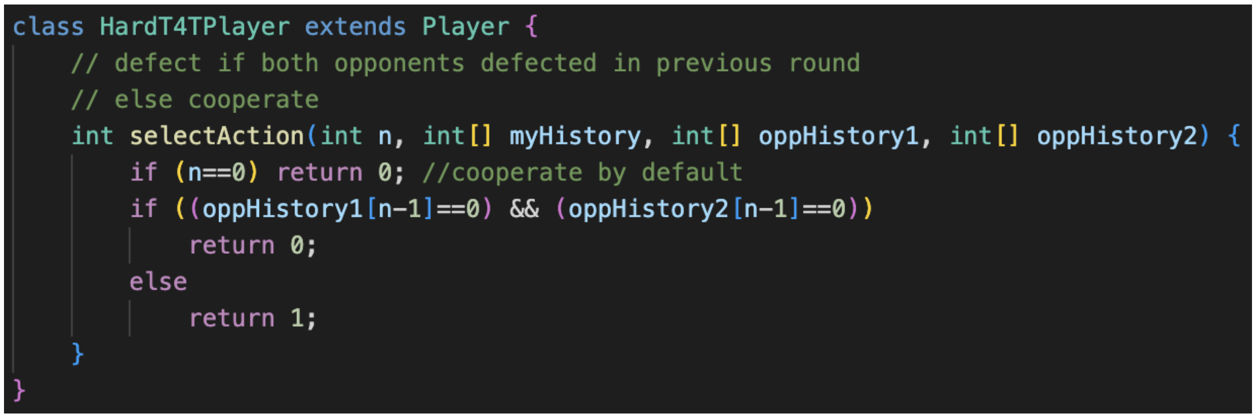


```
class SoftT4TPlayer extends Player {
    // defect if either opponents defected in previous round
    // else cooperate
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n==0) return 0; //cooperate by default
        if ((oppHistory1[n-1]==0) || (oppHistory2[n-1]==0))
            return 0;
        else
            return 1;
    }
}
```

Figure 1. Soft Tit-For-Tat Implementation

2.2.2 HardT4TPlayer

- Hard Tit-For-Tat strategy
- Cooperate in the first round
- For subsequent rounds, as long as one opponent cooperates, player cooperates
- If both opponents defect, player defects



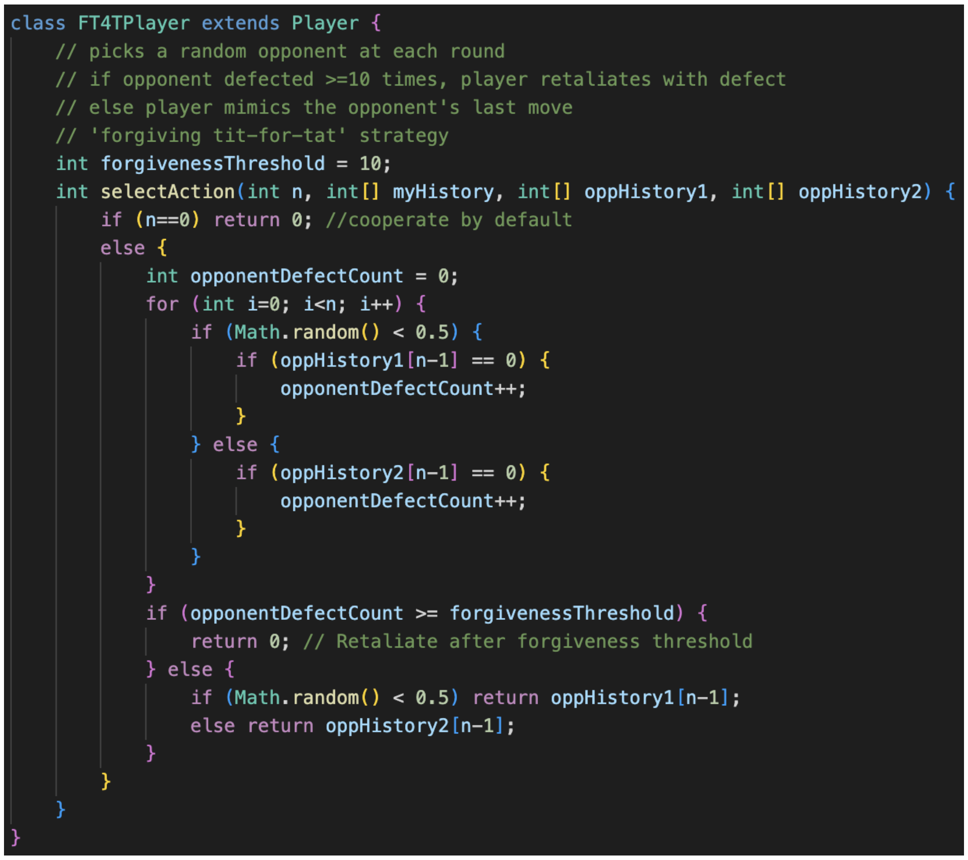
```
class HardT4TPlayer extends Player {
    // defect if both opponents defected in previous round
    // else cooperate
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n==0) return 0; //cooperate by default
        if ((oppHistory1[n-1]==0) && (oppHistory2[n-1]==0))
            return 0;
        else
            return 1;
    }
}
```

Figure 2. Hard Tit-For-Tat Implementation

2.2.3 FT4TPlayer

- Forgiving Tit-For-Tat strategy
- Player randomly chooses an opponent at each round

- Cooperate in the first round
- For subsequent rounds, counts how many times opponent has defected up to that point
- If opponent has defected at least a certain number of times, player retaliates with a defection
- If opponent has not, player mimics opponent's last move



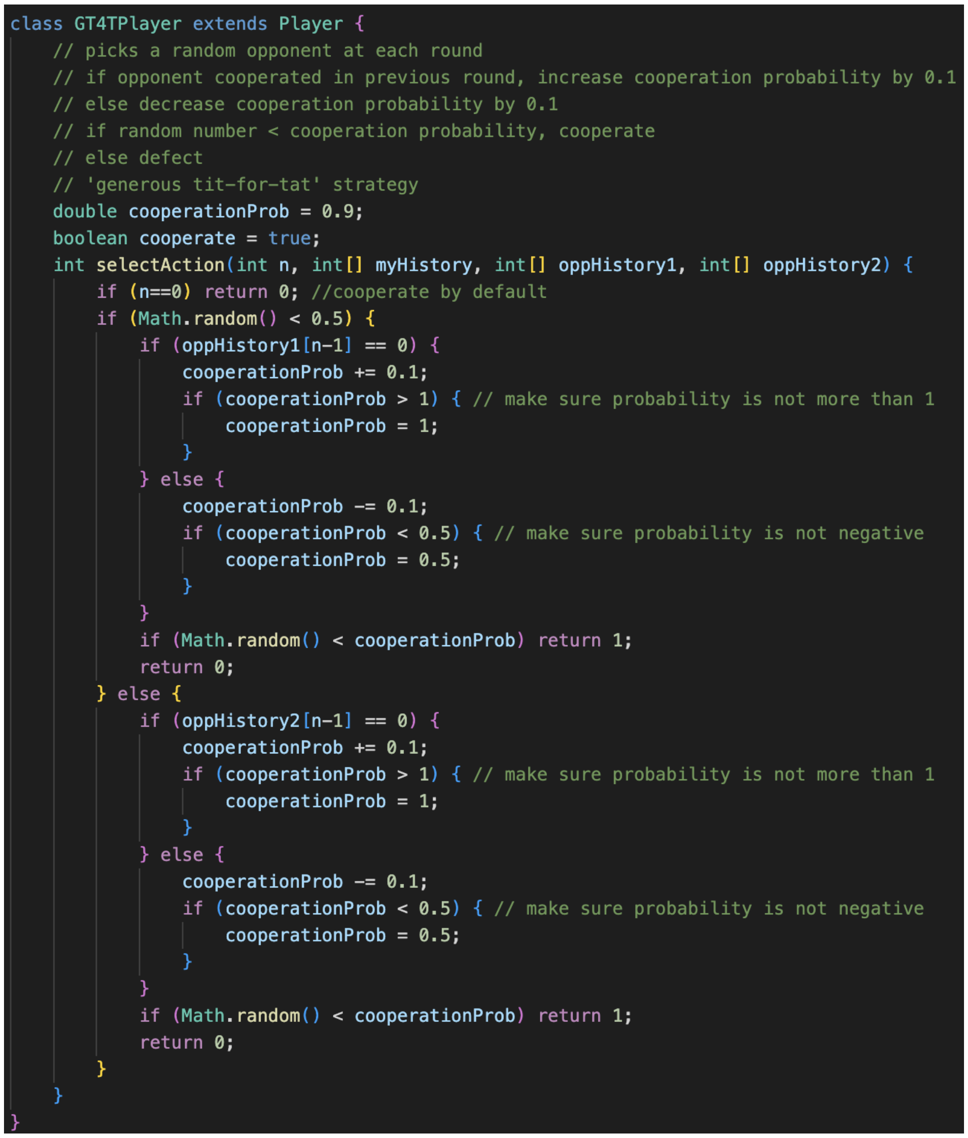
```
class FT4TPlayer extends Player {
    // picks a random opponent at each round
    // if opponent defected >=10 times, player retaliates with defect
    // else player mimics the opponent's last move
    // 'forgiving tit-for-tat' strategy
    int forgivenessThreshold = 10;
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n==0) return 0; //cooperate by default
        else {
            int opponentDefectCount = 0;
            for (int i=0; i<n; i++) {
                if (Math.random() < 0.5) {
                    if (oppHistory1[n-1] == 0) {
                        opponentDefectCount++;
                    }
                } else {
                    if (oppHistory2[n-1] == 0) {
                        opponentDefectCount++;
                    }
                }
            }
            if (opponentDefectCount >= forgivenessThreshold) {
                return 0; // Retaliate after forgiveness threshold
            } else {
                if (Math.random() < 0.5) return oppHistory1[n-1];
                else return oppHistory2[n-1];
            }
        }
    }
}
```

Figure 3. Forgiving Tit-For-Tat Implementation

2.2.4 GT4TPlayer

- Generous Tit-For-Tat strategy
- Player randomly chooses an opponent at each round
- Cooperate in the first round
- For subsequent rounds, player's move is determined by generating a random number between 0 and 1 and comparing it to the cooperation probability
- If random number is less than cooperation probability, player cooperates
- Otherwise, player defects
- If opponent cooperated in previous round, player increases the cooperation probability by 0.1

- If opponent defected in previous round, player decreases the cooperation probability by 0.1

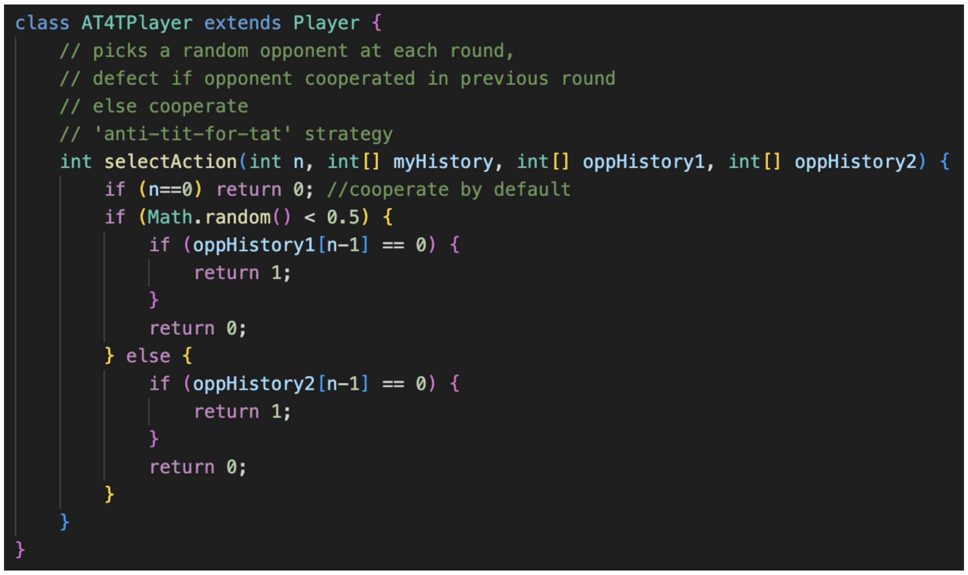


```java
class GT4TPlayer extends Player {
    // picks a random opponent at each round
    // if opponent cooperated in previous round, increase cooperation probability by 0.1
    // else decrease cooperation probability by 0.1
    // if random number < cooperation probability, cooperate
    // else defect
    // 'generous tit-for-tat' strategy
    double cooperationProb = 0.9;
    boolean cooperate = true;
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n==0) return 0; //cooperate by default
        if (Math.random() < 0.5) {
            if (oppHistory1[n-1] == 0) {
                cooperationProb += 0.1;
                if (cooperationProb > 1) { // make sure probability is not more than 1
                    cooperationProb = 1;
                }
            } else {
                cooperationProb -= 0.1;
                if (cooperationProb < 0.5) { // make sure probability is not negative
                    cooperationProb = 0.5;
                }
            }
            if (Math.random() < cooperationProb) return 1;
            return 0;
        } else {
            if (oppHistory2[n-1] == 0) {
                cooperationProb += 0.1;
                if (cooperationProb > 1) { // make sure probability is not more than 1
                    cooperationProb = 1;
                }
            } else {
                cooperationProb -= 0.1;
                if (cooperationProb < 0.5) { // make sure probability is not negative
                    cooperationProb = 0.5;
                }
            }
            if (Math.random() < cooperationProb) return 1;
            return 0;
        }
    }
}
```

Figure 4. Generous Tit-For-Tat Implementation

2.2.5 AT4TPlayer

- Anti-Tit-For-Tat strategy
- Player randomly chooses an opponent at each round
- Cooperate in the first round
- For subsequent rounds, if opponent cooperates in previous round, player defects
- If opponent defects in previous round, player cooperates

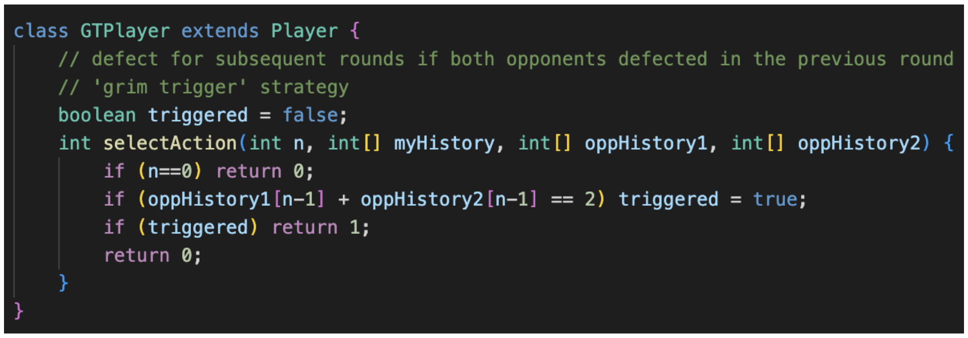


```java
class AT4TPlayer extends Player {
    // picks a random opponent at each round,
    // defect if opponent cooperated in previous round
    // else cooperate
    // 'anti-tit-for-tat' strategy
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n==0) return 0; //cooperate by default
        if (Math.random() < 0.5) {
            if (oppHistory1[n-1] == 0) {
                return 1;
            }
            return 0;
        } else {
            if (oppHistory2[n-1] == 0) {
                return 1;
            }
            return 0;
        }
    }
}
```

Figure 5. Anti-Tit-For-Tat Implementation

2.2.6 GTPlayer

- Grim Trigger strategy
- Cooperate in the first round
- Player cooperates until both opponents defect
- For subsequent rounds, always defect



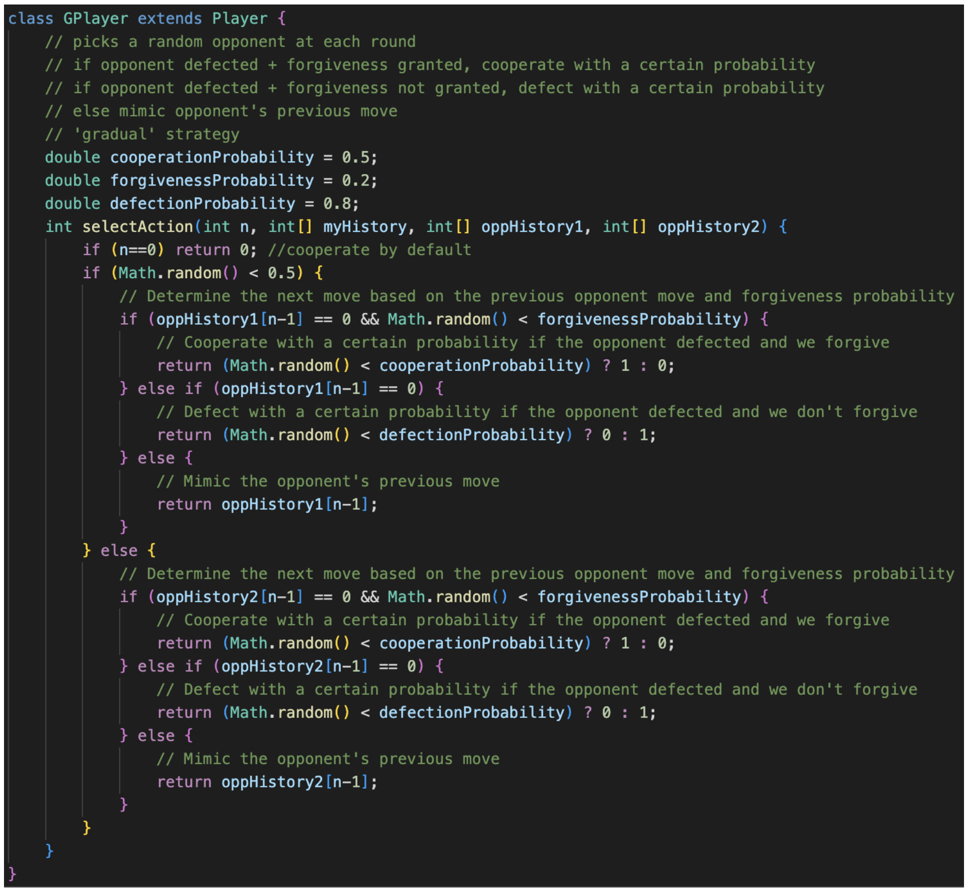
```java
class GTPlayer extends Player {
    // defect for subsequent rounds if both opponents defected in the previous round
    // 'grim trigger' strategy
    boolean triggered = false;
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n==0) return 0;
        if (oppHistory1[n-1] + oppHistory2[n-1] == 2) triggered = true;
        if (triggered) return 1;
        return 0;
    }
}
```

Figure 6. Grim Trigger Implementation

2.2.7 GPlayer

- Gradual strategy
- Cooperate in the first round
- Player randomly chooses an opponent at each round
- For subsequent rounds, get the opponent's previous move and determine next move based on that move and the forgiveness probability
- If opponent defected and forgiveness is granted, cooperate with a certain probability

- If opponent defected and forgiveness is not granted, defect with a certain probability
- Otherwise, mimic opponent's previous move

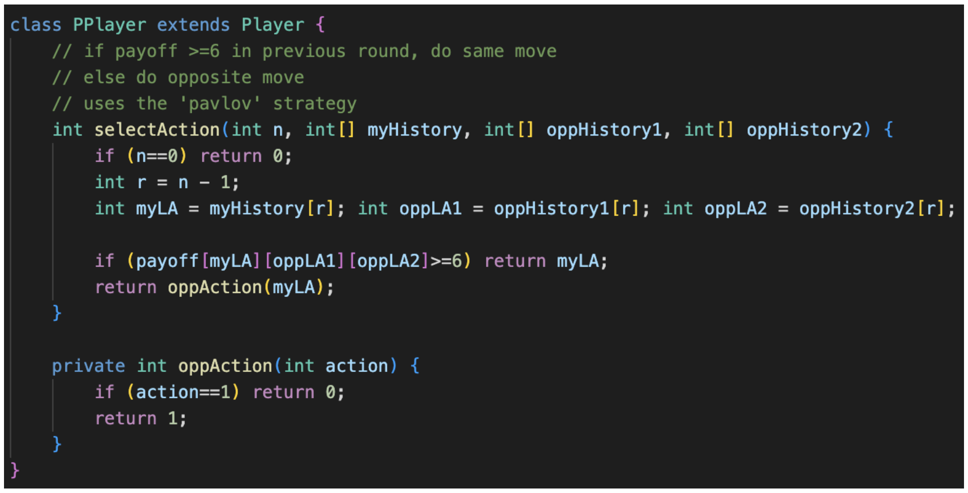


```java
class GPlayer extends Player {
    // picks a random opponent at each round
    // if opponent defected + forgiveness granted, cooperate with a certain probability
    // if opponent defected + forgiveness not granted, defect with a certain probability
    // else mimic opponent's previous move
    // 'gradual' strategy
    double cooperationProbability = 0.5;
    double forgivenessProbability = 0.2;
    double defectionProbability = 0.8;
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n==0) return 0; //cooperate by default
        if (Math.random() < 0.5) {
            // Determine the next move based on the previous opponent move and forgiveness probability
            if (oppHistory1[n-1] == 0 && Math.random() < forgivenessProbability) {
                // Cooperate with a certain probability if the opponent defected and we forgive
                return (Math.random() < cooperationProbability) ? 1 : 0;
            } else if (oppHistory1[n-1] == 0) {
                // Defect with a certain probability if the opponent defected and we don't forgive
                return (Math.random() < defectionProbability) ? 0 : 1;
            } else {
                // Mimic the opponent's previous move
                return oppHistory1[n-1];
            }
        } else {
            // Determine the next move based on the previous opponent move and forgiveness probability
            if (oppHistory2[n-1] == 0 && Math.random() < forgivenessProbability) {
                // Cooperate with a certain probability if the opponent defected and we forgive
                return (Math.random() < cooperationProbability) ? 1 : 0;
            } else if (oppHistory2[n-1] == 0) {
                // Defect with a certain probability if the opponent defected and we don't forgive
                return (Math.random() < defectionProbability) ? 0 : 1;
            } else {
                // Mimic the opponent's previous move
                return oppHistory2[n-1];
            }
        }
    }
}
```

Figure 7. Gradual Implementation

2.2.8 PPlayer

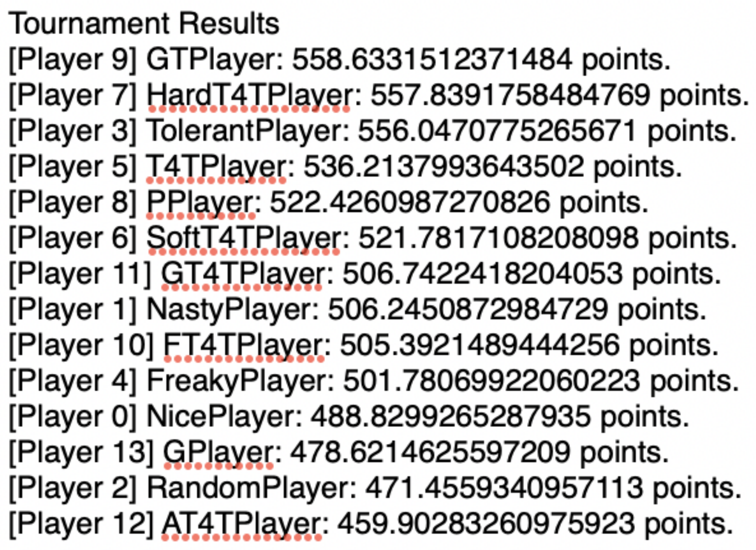
- Pavlov strategy
- Cooperate in the first round
- For subsequent rounds, player changes his move if opponent's move resulted in a low payoff in the previous round



```java
class PPlayer extends Player {
    // if payoff >=6 in previous round, do same move
    // else do opposite move
    // uses the 'pavlov' strategy
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n==0) return 0;
        int r = n - 1;
        int myLA = myHistory[r]; int oppLA1 = oppHistory1[r]; int oppLA2 = oppHistory2[r];

        if (payoff[myLA][oppLA1][oppLA2]>=6) return myLA;
        return oppAction(myLA);
    }

    private int oppAction(int action) {
        if (action==1) return 0;
        return 1;
    }
}
```

Figure 8. Pavlov Implementation

2.2.9 Evaluation

Metric	Results
Round 99	 Tournament Results [Player 9] GTPlayer: 558.6331512371484 points. [Player 7] HardT4TPlayer: 557.8391758484769 points. [Player 3] TolerantPlayer: 556.0470775265671 points. [Player 5] T4TPlayer: 536.2137993643502 points. [Player 8] PPlayer: 522.4260987270826 points. [Player 6] SoftT4TPlayer: 521.7817108208098 points. [Player 11] GT4TPlayer: 506.7422418204053 points. [Player 1] NastyPlayer: 506.2450872984729 points. [Player 10] FT4TPlayer: 505.3921489444256 points. [Player 4] FreakyPlayer: 501.78069922060223 points. [Player 0] NicePlayer: 488.8299265287935 points. [Player 13] GPlayer: 478.6214625597209 points. [Player 2] RandomPlayer: 471.4559340957113 points. [Player 12] AT4TPlayer: 459.90283260975923 points.

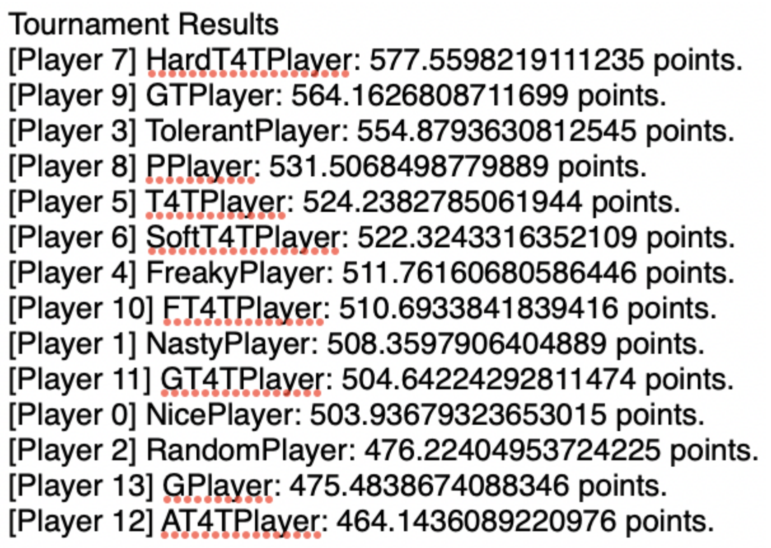
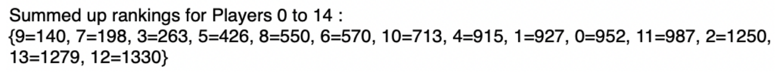
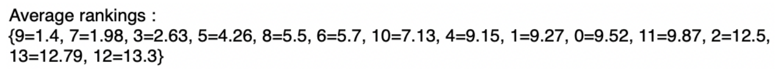
Round 100	 Tournament Results [Player 7] HardT4TPlayer: 577.5598219111235 points. [Player 9] GTPlayer: 564.1626808711699 points. [Player 3] TolerantPlayer: 554.8793630812545 points. [Player 8] PPlayer: 531.5068498779889 points. [Player 5] T4TPlayer: 524.2382785061944 points. [Player 6] SoftT4TPlayer: 522.3243316352109 points. [Player 4] FreakyPlayer: 511.76160680586446 points. [Player 10] FT4TPlayer: 510.6933841839416 points. [Player 1] NastyPlayer: 508.3597906404889 points. [Player 11] GT4TPlayer: 504.64224292811474 points. [Player 0] NicePlayer: 503.93679323653015 points. [Player 2] RandomPlayer: 476.22404953724225 points. [Player 13] GPlayer: 475.4838674088346 points. [Player 12] AT4TPlayer: 464.1436089220976 points.
Summed Up Rankings	 Summed up rankings for Players 0 to 14 : {9=140, 7=198, 3=263, 5=426, 8=550, 6=570, 10=713, 4=915, 1=927, 0=952, 11=987, 2=1250, 13=1279, 12=1330}
Average Rankings	 Average rankings : {9=1.4, 7=1.98, 3=2.63, 5=4.26, 8=5.5, 6=5.7, 10=7.13, 4=9.15, 1=9.27, 0=9.52, 11=9.87, 2=12.5, 13=12.79, 12=13.3}

Table 3. Match 2 Results

From the results shown in Table 3 above, GTPlayer, HardT4TPlayer and TolerantPlayer are the top 3 performing players out of the given agents. GTPlayer performs the best which suggests that adopting a Grim Trigger strategy that punishes defection severely can be effective. This strategy starts with cooperation, but any defection by an opponent triggers a permanent switch to defection by the player throughout the game. Such a response effectively punishes the opponent's initial defection and discourages them from repeating it, leading to a greater overall payoff for the player. Furthermore, due to its deterministic and inflexible nature, this strategy is less vulnerable to being exploited by other players. This means that it is difficult for opponents to manipulate the player into cooperating more frequently than necessary, which is a concern with certain other strategies.

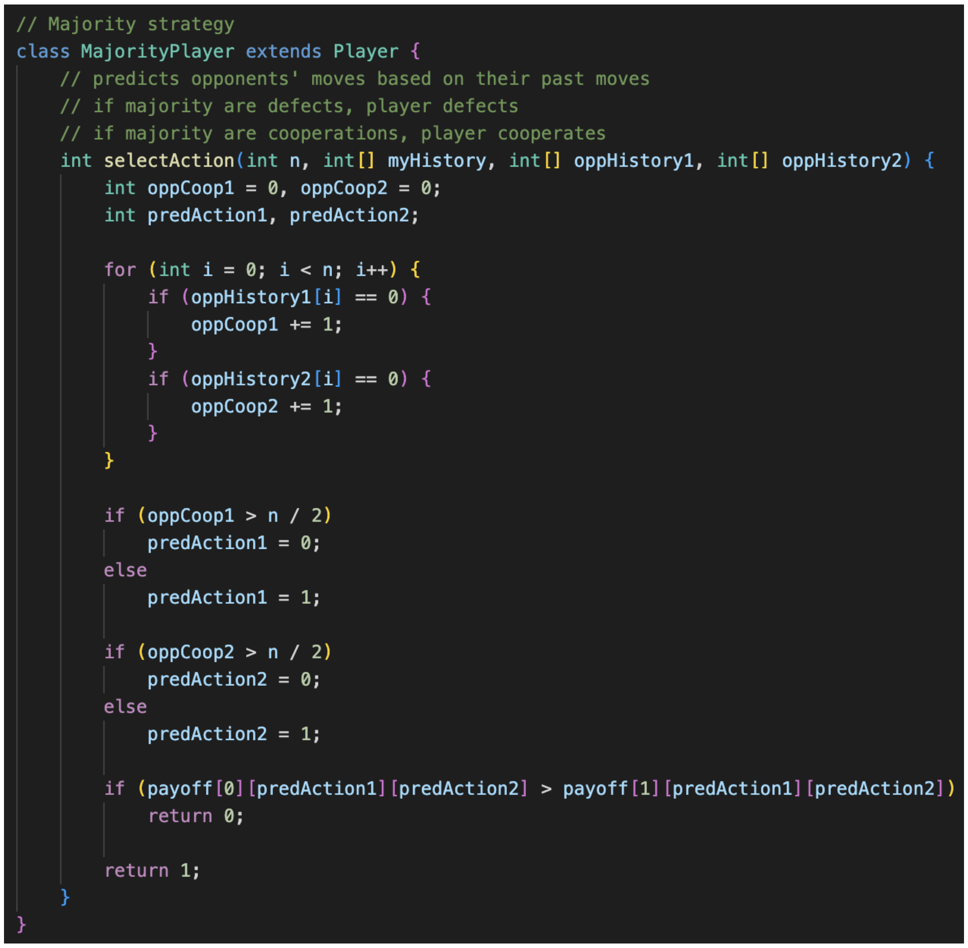
3 Agent Design

3.1 Agent Strategy

My agent utilizes a combination of two strategies, Majority and Expected Utility.

3.1.1 Majority Strategy

The majority strategy involves the agent making decisions based on its predictions of its opponents' actions in the next round. This prediction is done by finding the opponents' past majority moves. When an opponent has performed both actions equally in all previous rounds, the agent predicts that the opponent will defect in the next round. If an opponent has a majority history of performing a particular action, the agent predicts that the opponent will continue to perform that action. The agent then selects the action that maximizes its payoff if its opponents perform the actions predicted by the agent.

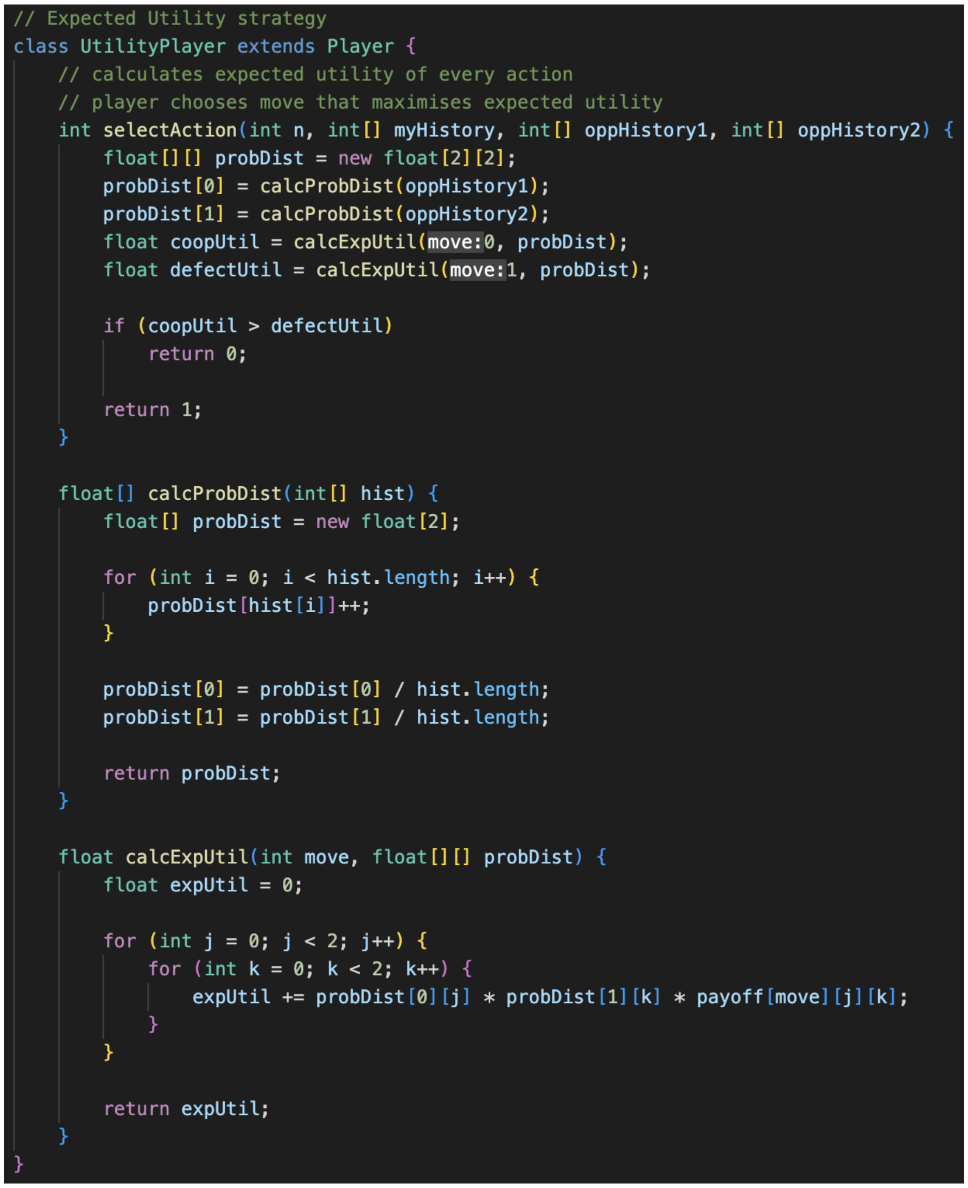


```
// Majority strategy
class MajorityPlayer extends Player {
    // predicts opponents' moves based on their past moves
    // if majority are defects, player defects
    // if majority are cooperations, player cooperates
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        int oppCoop1 = 0, oppCoop2 = 0;
        int predAction1, predAction2;

        for (int i = 0; i < n; i++) {
            if (oppHistory1[i] == 0) {
                oppCoop1 += 1;
            }
            if (oppHistory2[i] == 0) {
                oppCoop2 += 1;
            }
        }

        if (oppCoop1 > n / 2)
            predAction1 = 0;
        else
            predAction1 = 1;

        if (oppCoop2 > n / 2)
            predAction2 = 0;
        else
            predAction2 = 1;

        if (payoff[0][predAction1][predAction2] > payoff[1][predAction1][predAction2])
            return 0;

        return 1;
    }
}
```

Figure 9. Majority Implementation

3.1.2 Expected Utility Strategy

The expected utility strategy involves the agent computing the expected utility for both available actions, cooperate and defect, thereafter choosing the action that maximizes its expected utility. In order to determine the expected utility of its actions, the agent first estimates the probability of each opponent performing each of their actions. This is done based on the opponents' previous actions. Once these probabilities have been determined, the agent uses them to calculate the expected utility that would be gained for each action.

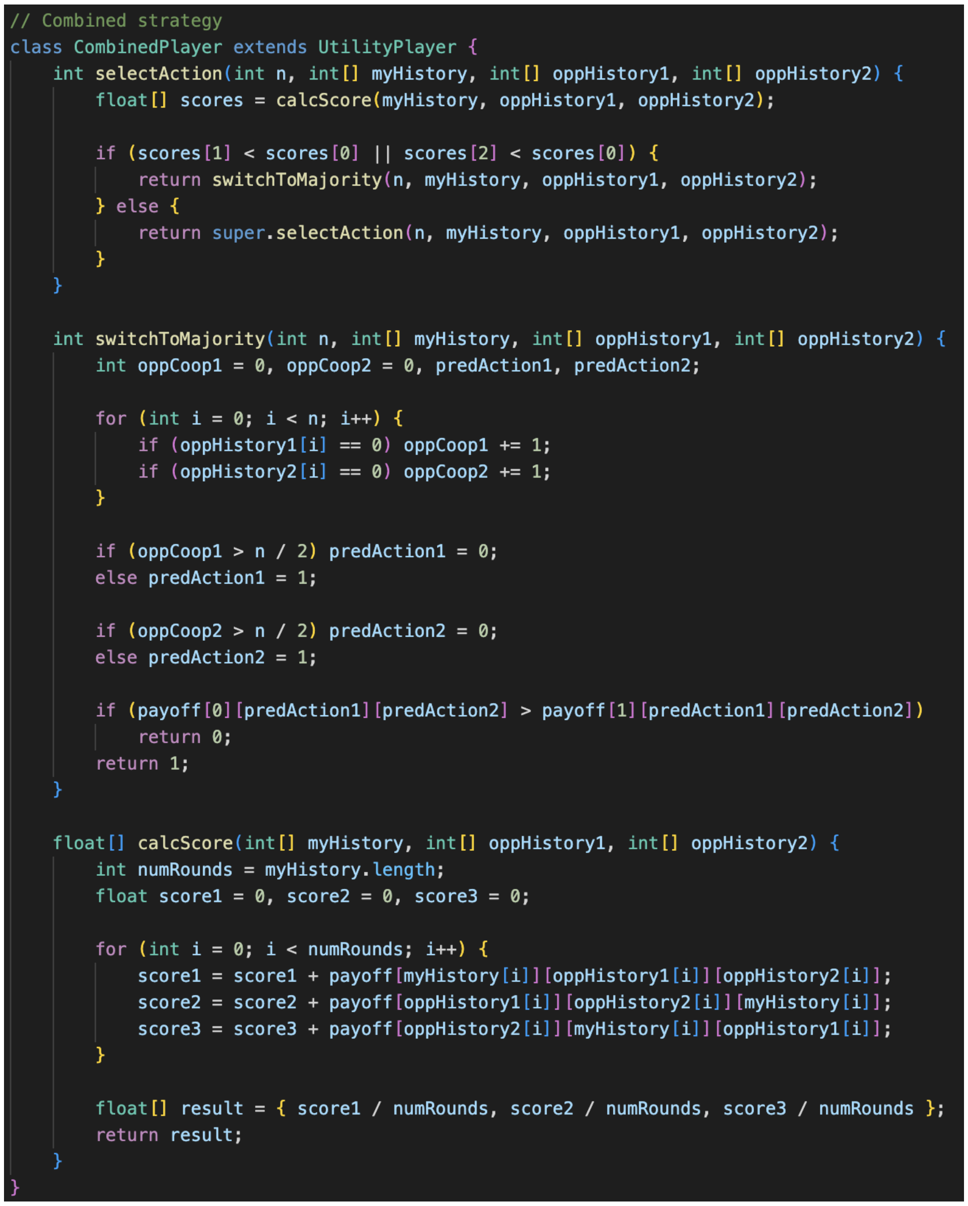


```
// Expected Utility strategy
class UtilityPlayer extends Player {
    // calculates expected utility of every action
    // player chooses move that maximises expected utility
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        float[][] probDist = new float[2][2];
        probDist[0] = calcProbDist(oppHistory1);
        probDist[1] = calcProbDist(oppHistory2);
        float coopUtil = calcExpUtil(move:0, probDist);
        float defectUtil = calcExpUtil(move:1, probDist);

        if (coopUtil > defectUtil)
            return 0;

        return 1;
    }

    float[] calcProbDist(int[] hist) {
        float[] probDist = new float[2];

        for (int i = 0; i < hist.length; i++) {
            probDist[hist[i]]++;
        }

        probDist[0] = probDist[0] / hist.length;
        probDist[1] = probDist[1] / hist.length;

        return probDist;
    }

    float calcExpUtil(int move, float[][] probDist) {
        float expUtil = 0;

        for (int j = 0; j < 2; j++) {
            for (int k = 0; k < 2; k++) {
                expUtil += probDist[0][j] * probDist[1][k] * payoff[move][j][k];
            }
        }

        return expUtil;
    }
}
```

Figure 10. Expected Utility Implementation

3.1.3 Combined Strategy

The combined strategy involves the agent choosing to use either the Majority strategy or the Expected Utility strategy, depending on its current performance in the match. In order to determine its current standing, the agent calculates its own score and the scores of its opponents. This is done by analyzing their actions in previous rounds. If the agent has the lowest score, it employs the Expected Utility strategy to decide its next action. Otherwise, it employs the Majority strategy to decide its next action.

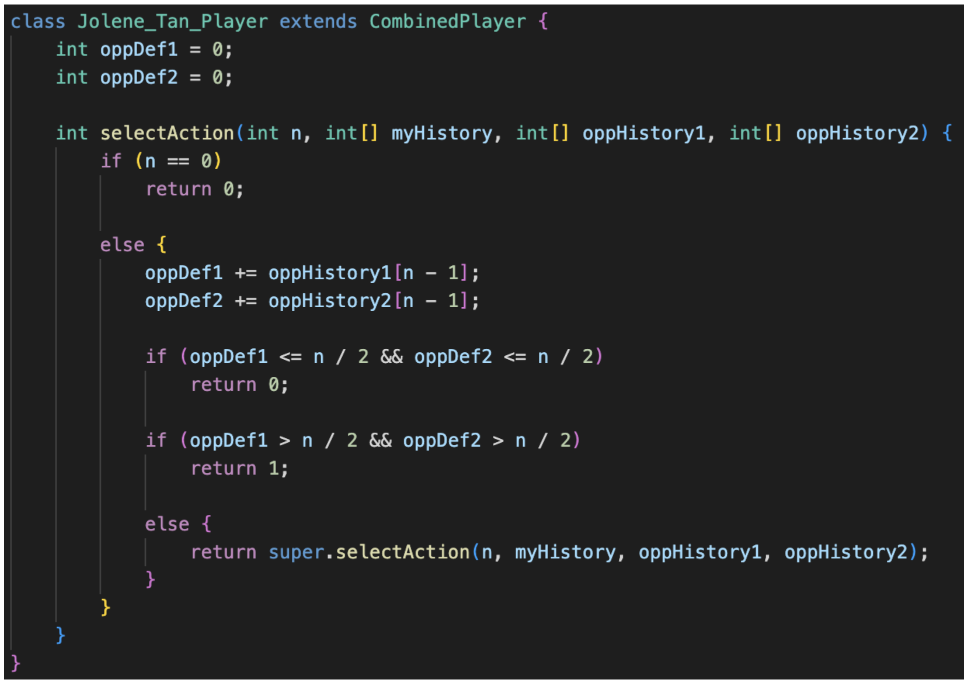


```java
// Combined strategy
class CombinedPlayer extends UtilityPlayer {
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        float[] scores = calcScore(myHistory, oppHistory1, oppHistory2);

        if (scores[1] < scores[0] || scores[2] < scores[0]) {
            return switchToMajority(n, myHistory, oppHistory1, oppHistory2);
        } else {
            return super.selectAction(n, myHistory, oppHistory1, oppHistory2);
        }
    }

    int switchToMajority(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        int oppCoop1 = 0, oppCoop2 = 0, predAction1, predAction2;

        for (int i = 0; i < n; i++) {
            if (oppHistory1[i] == 0) oppCoop1 += 1;
            if (oppHistory2[i] == 0) oppCoop2 += 1;
        }

        if (oppCoop1 > n / 2) predAction1 = 0;
        else predAction1 = 1;

        if (oppCoop2 > n / 2) predAction2 = 0;
        else predAction2 = 1;

        if (payoff[0][predAction1][predAction2] > payoff[1][predAction1][predAction2])
            return 0;
        return 1;
    }

    float[] calcScore(int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        int numRounds = myHistory.length;
        float score1 = 0, score2 = 0, score3 = 0;

        for (int i = 0; i < numRounds; i++) {
            score1 = score1 + payoff[myHistory[i]][oppHistory1[i]][oppHistory2[i]];
            score2 = score2 + payoff[oppHistory1[i]][oppHistory2[i]][myHistory[i]];
            score3 = score3 + payoff[oppHistory2[i]][myHistory[i]][oppHistory1[i]];
        }

        float[] result = { score1 / numRounds, score2 / numRounds, score3 / numRounds };
        return result;
    }
}
```

Figure 11. Combined Implementation

3.1.4 Overall

The final overall strategy involves the agent finding the opponents' past majority moves. If both opponents have mostly defected, the agent will choose to defect. If both opponents have mostly cooperated, the agent will choose to cooperate. If the opponents have different majority moves, where one mostly defected while the other mostly cooperated, the agent will choose to adopt the Combined strategy.



```
class Jolene_Tan_Player extends CombinedPlayer {
    int oppDef1 = 0;
    int oppDef2 = 0;

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if (n == 0)
            return 0;

        else {
            oppDef1 += oppHistory1[n - 1];
            oppDef2 += oppHistory2[n - 1];

            if (oppDef1 <= n / 2 && oppDef2 <= n / 2)
                return 0;

            if (oppDef1 > n / 2 && oppDef2 > n / 2)
                return 1;

            else {
                return super.selectAction(n, myHistory, oppHistory1, oppHistory2);
            }
        }
    }
}
```

Figure 12. Agent Implementation

3.3 Agent Evaluation

3.3.1 Match with 15 Agents

Metric	Results

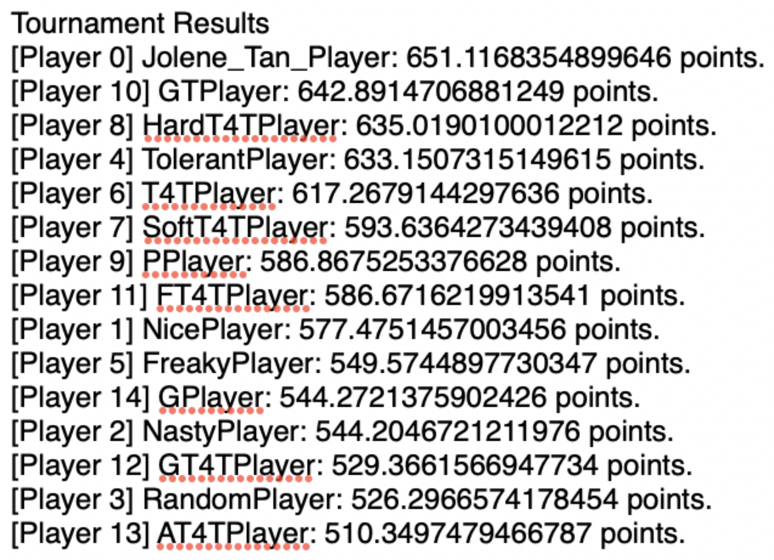
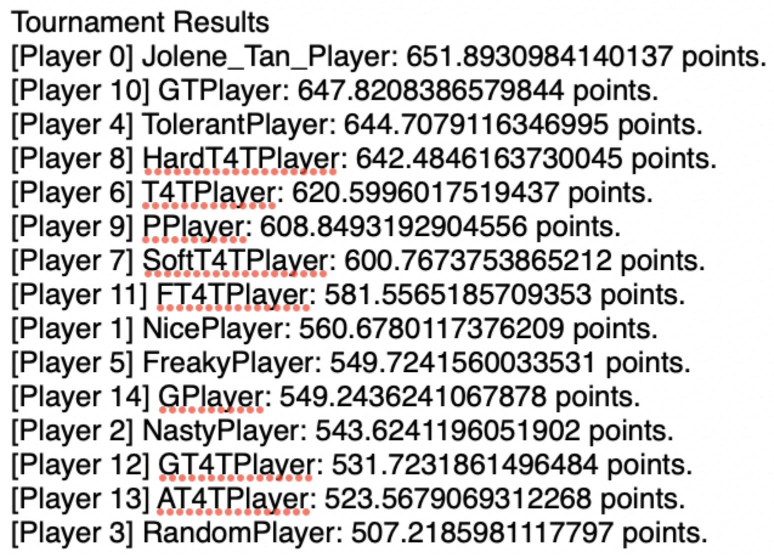
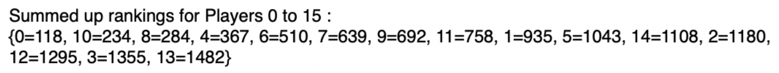
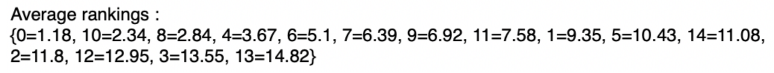
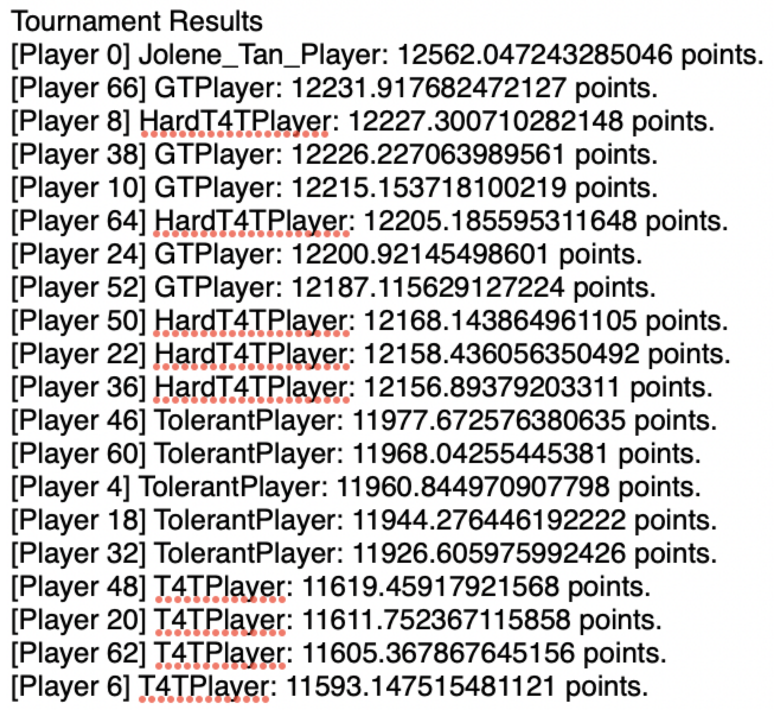
Round 99	 Tournament Results [Player 0] Jolene_Tan_Player: 651.1168354899646 points. [Player 10] GTPlayer: 642.8914706881249 points. [Player 8] HardT4TPlayer: 635.0190100012212 points. [Player 4] TolerantPlayer: 633.1507315149615 points. [Player 6] T4TPlayer: 617.2679144297636 points. [Player 7] SoftT4TPlayer: 593.6364273439408 points. [Player 9] PPlayer: 586.8675253376628 points. [Player 11] FT4TPlayer: 586.6716219913541 points. [Player 1] NicePlayer: 577.4751457003456 points. [Player 5] FreakyPlayer: 549.5744897730347 points. [Player 14] GPlayer: 544.2721375902426 points. [Player 2] NastyPlayer: 544.2046721211976 points. [Player 12] GT4TPlayer: 529.3661566947734 points. [Player 3] RandomPlayer: 526.2966574178454 points. [Player 13] AT4TPlayer: 510.3497479466787 points.
Round 100	 Tournament Results [Player 0] Jolene_Tan_Player: 651.8930984140137 points. [Player 10] GTPlayer: 647.8208386579844 points. [Player 4] TolerantPlayer: 644.7079116346995 points. [Player 8] HardT4TPlayer: 642.4846163730045 points. [Player 6] T4TPlayer: 620.5996017519437 points. [Player 9] PPlayer: 608.8493192904556 points. [Player 7] SoftT4TPlayer: 600.7673753865212 points. [Player 11] FT4TPlayer: 581.5565185709353 points. [Player 1] NicePlayer: 560.6780117376209 points. [Player 5] FreakyPlayer: 549.7241560033531 points. [Player 14] GPlayer: 549.2436241067878 points. [Player 2] NastyPlayer: 543.6241196051902 points. [Player 12] GT4TPlayer: 531.7231861496484 points. [Player 13] AT4TPlayer: 523.5679069312268 points. [Player 3] RandomPlayer: 507.2185981117797 points.
Summed Up Rankings	 Summed up rankings for Players 0 to 15 : {0=118, 10=234, 8=284, 4=367, 6=510, 7=639, 9=692, 11=758, 1=935, 5=1043, 14=1108, 2=1180, 12=1295, 3=1355, 13=1482}
Average Rankings	 Average rankings : {0=1.18, 10=2.34, 8=2.84, 4=3.67, 6=5.1, 7=6.39, 9=6.92, 11=7.58, 1=9.35, 5=10.43, 14=11.08, 2=11.8, 12=12.95, 3=13.55, 13=14.82}

Table 4. Match 3 Results

In this match, the agent is pitted against 1 of each given and additional agent. From the results shown in Table 4 above, Jolene_Tan_Player (my agent), GTPlayer and HardT4TPlayer are the top 3 performing players out of the given agents. Jolene_Tan_Player, my agent, performs the best which suggests that the adopted Combined Majority and Expected Utility strategy is effective. This strategy combines a safer Majority strategy with a riskier Expected Utility strategy, allowing the agent to alternate between the two approaches. As a result, the agent is willing to take risks when it has the lowest score in order to catch up with its opponents, while still playing it safe in other cases to avoid falling behind in the standings. It is noted that the agent outperforms even the Grim Trigger strategy which suggests that despite the positive attributes mentioned in Section 2.2.9 above, the Grim Trigger strategy can also lead to a lower overall payoff if the other players are able to cooperate effectively.

3.3.2 Match with 71 Agents

Metric	**Results**
Round 99	 Tournament Results [Player 0] Jolene_Tan_Player: 12562.047243285046 points. [Player 66] GTPlayer: 12231.917682472127 points. [Player 8] HardT4TPlayer: 12227.300710282148 points. [Player 38] GTPlayer: 12226.227063989561 points. [Player 10] GTPlayer: 12215.153718100219 points. [Player 64] HardT4TPlayer: 12205.185595311648 points. [Player 24] GTPlayer: 12200.92145498601 points. [Player 52] GTPlayer: 12187.115629127224 points. [Player 50] HardT4TPlayer: 12168.143864961105 points. [Player 22] HardT4TPlayer: 12158.436056350492 points. [Player 36] HardT4TPlayer: 12156.89379203311 points. [Player 46] TolerantPlayer: 11977.672576380635 points. [Player 60] TolerantPlayer: 11968.04255445381 points. [Player 4] TolerantPlayer: 11960.844970907798 points. [Player 18] TolerantPlayer: 11944.276446192222 points. [Player 32] TolerantPlayer: 11926.605975992426 points. [Player 48] T4TPlayer: 11619.45917921568 points. [Player 20] T4TPlayer: 11611.752367115858 points. [Player 62] T4TPlayer: 11605.367867645156 points. [Player 6] T4TPlayer: 11593.147515481121 points.

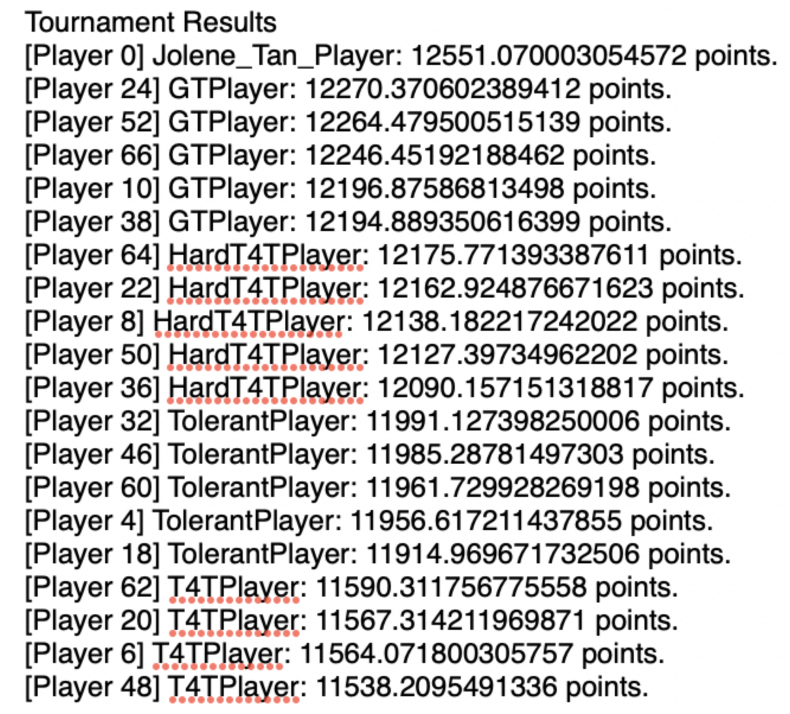
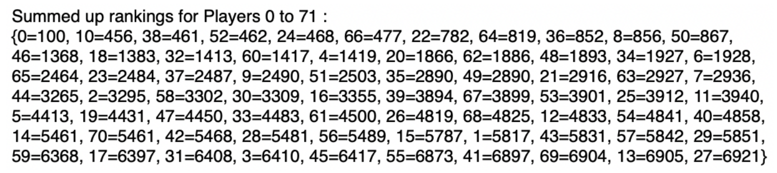
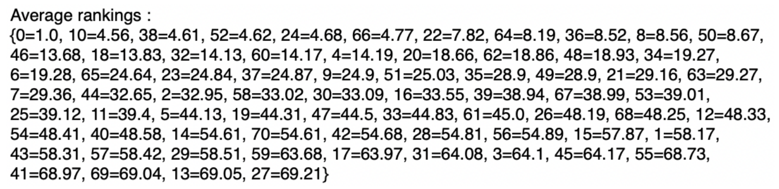
Round 100	 Tournament Results [Player 0] Jolene_Tan_Player: 12551.070003054572 points. [Player 24] GTPlayer: 12270.370602389412 points. [Player 52] GTPlayer: 12264.479500515139 points. [Player 66] GTPlayer: 12246.45192188462 points. [Player 10] GTPlayer: 12196.87586813498 points. [Player 38] GTPlayer: 12194.889350616399 points. [Player 64] HardT4TPlayer: 12175.771393387611 points. [Player 22] HardT4TPlayer: 12162.924876671623 points. [Player 8] HardT4TPlayer: 12138.182217242022 points. [Player 50] HardT4TPlayer: 12127.39734962202 points. [Player 36] HardT4TPlayer: 12090.157151318817 points. [Player 32] TolerantPlayer: 11991.127398250006 points. [Player 46] TolerantPlayer: 11985.28781497303 points. [Player 60] TolerantPlayer: 11961.729928269198 points. [Player 4] TolerantPlayer: 11956.617211437855 points. [Player 18] TolerantPlayer: 11914.969671732506 points. [Player 62] T4TPlayer: 11590.311756775558 points. [Player 20] T4TPlayer: 11567.314211969871 points. [Player 6] T4TPlayer: 11564.071800305757 points. [Player 48] T4TPlayer: 11538.2095491336 points.
Summed Up Rankings	 Summed up rankings for Players 0 to 71 : {0=100, 10=456, 38=461, 52=462, 24=468, 66=477, 22=782, 64=819, 36=852, 8=856, 50=867, 46=1368, 18=1383, 32=1413, 60=1417, 4=1419, 20=1866, 62=1886, 48=1893, 34=1927, 6=1928, 65=2464, 23=2484, 37=2487, 9=2490, 51=2503, 35=2890, 49=2890, 21=2916, 63=2927, 7=2936, 44=3265, 2=3295, 58=3302, 30=3309, 16=3355, 39=3894, 67=3899, 53=3901, 25=3912, 11=3940, 5=4413, 19=4431, 47=4450, 33=4483, 61=4500, 26=4819, 68=4825, 12=4833, 54=4841, 40=4858, 14=5461, 70=5461, 42=5468, 28=5481, 56=5489, 15=5787, 1=5817, 43=5831, 57=5842, 29=5851, 59=6368, 17=6397, 31=6408, 3=6410, 45=6417, 55=6873, 41=6897, 69=6904, 13=6905, 27=6921}
Average Rankings	 Average rankings : {0=1.0, 10=4.56, 38=4.61, 52=4.62, 24=4.68, 66=4.77, 22=7.82, 64=8.19, 36=8.52, 8=8.56, 50=8.67, 46=13.68, 18=13.83, 32=14.13, 60=14.17, 4=14.19, 20=18.66, 62=18.86, 48=18.93, 34=19.27, 6=19.28, 65=24.64, 23=24.84, 37=24.87, 9=24.9, 51=25.03, 35=28.9, 49=28.9, 21=29.16, 63=29.27, 7=29.36, 44=32.65, 2=32.95, 58=33.02, 30=33.09, 16=33.55, 39=38.94, 67=38.99, 53=39.01, 25=39.12, 11=39.4, 5=44.13, 19=44.31, 47=44.5, 33=44.83, 61=45.0, 26=48.19, 68=48.25, 12=48.33, 54=48.41, 40=48.58, 14=54.61, 70=54.61, 42=54.68, 28=54.81, 56=54.89, 15=57.87, 1=58.17, 43=58.31, 57=58.42, 29=58.51, 59=63.68, 17=63.97, 31=64.08, 3=64.1, 45=64.17, 55=68.73, 41=68.97, 69=69.04, 13=69.05, 27=69.21}

Table 5. Match 4 Results

In this match, the agent is pitted against 5 of each given and additional agent. The rationale behind doing so is to create a more dynamic and realistic simulation. This can provide insights into the competitive and cooperative dynamics of a group of agents. Furthermore, incorporating more agents with random variables in their strategies, such as the GT4TPlayer and GPlayer, can introduce random noise into the game and test the chosen strategy's robustness.

Note that in Table 5 above, only the Top 20 performing agents are shown in Rounds 99 and 100. From the results shown, Jolene_Tan_Player (my agent) and two variations of GTPlayer are the top 3 performing players out of the given agents. Jolene_Tan_Player, my agent, performs the best which suggests that the adopted Combined Majority and Expected Utility strategy is realistic, robust and able to adapt to different environments.

4 Conclusion

In conclusion, the significance of game theory and experimentation when developing effective strategies for agents in complex scenarios is highlighted.Developing a successful strategy for an agent in a three player repeated prisoners' dilemma requires careful consideration of various factors such as the nature of the game, the opponents’ behavior, and the agent's performance in the game. By leveraging game theory and conducting experiments, it was demonstrated that employing a combined strategy that switches between a safer majority approach and a riskier expected utility approach can result in optimal outcomes. This approach considers the agent's current standing in the game and enables it to take calculated risks to close the gap with opponents, thus creating a reasonable balance between safety and risk-taking.
experimentation in developing effective strategies for agents in complex situations like the repeated prisoners' dilemma.