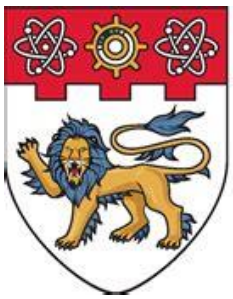

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE/CZ4046: Intelligent Agents

Assignment 2: Repeated Prisoners' Dilemma

Ingale Omkar (U2020724H)

Submitted to:
Prof Bo AN
CE/CZ4046: Intelligent Agents
School of Computer Science and Engineering
Nanyang Technological University, Singapore

Table of Contents

1. Introduction

This project aims to explore the development of strategies for agents to cooperate with one another in the setting of the Prisoners' Dilemma. In the standard setting for Prisoners' Dilemma, the dominant strategy is for an agent to defect despite the fact that it is mutually beneficial for all agents to cooperate. This problem, however, seems to disappear in the setting of repeated prisoners' dilemma, where the group of agents plays the game in a repeated fashion, sometimes forever. In this scenario, the reward for a player (an agent, in this scenario) is the sum of rewards over all the rounds divided by the total number of rounds.

This report details the various strategies that were designed, implemented and tested in the game of three player repeated prisoners' dilemma. Finally, an agent is chosen based on the average rewards it gained over 3 tournaments as the final agent. This agent can be found in the IngaleOmkarPlayer.java file. Other agents that were tested are listed in the ThreePrisonersDilemma.java file. Outputs of rewards from each tournament can also be found in the results.xlsx file in this projects GitHub Repository.

1.1 Summary of Returns Based on Chosen Action

The following table lists the returns that an agent can expect based on its own actions and the actions of its opponents.

Player 1	Player 2	Player 3	Reward
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

Table 1: Rewards based on chosen action
0 = Cooperate, 1 = Defect

As can be seen from the table, the two highest returns correspond to Cooperate (6) and Defect (8). This, however, is only subject to the other opponents cooperating as well. As this behaviour cannot be guaranteed, we need to design an agent that cooperates when it is confident in its opponents intentions to cooperate as well. This ensures mutual benefit to all agents that are involved. This means our agent should be trusting but also not "gullible" to keep cooperating when its opponents keep defecting.

2. Implemented Agents

This section of the report, walks through all the agents that were designed and tested in the setting of three agent repeated prisoners' dilemma problem.

2.1 Suspicious Tit-For-Tat Player

The Suspicious Tit-For-Tat Player [1] is very similar to the Tit-For-Tat player. They major difference is that this player starts off by defecting in the first round instead of cooperating. This ensures that the players presents an aggressive stance in the first round. In the following rounds however, it imitates its opponents choice from the last round. This ensures that the player remains open to cooperation if both its opponents are willing to cooperate as well.

```
class SuspiciousT4TPlayer extends Player {
    // Defect on the first round and imitate thereafter
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if(n == 0) return 1; // defect on the first round
        if(oppHistory1[n-1] == 1 || oppHistory2[n-1] == 1) return 1;
        return 0;
    }
}
```

Figure 1: Suspicious Tit-For-Tat Player

2.2 Alternate Player

The Alternate Player does not take into account, the histories of any of its opponents. It simply alternates between cooperating and defecting. Although this strategy is extremely predictable, it does aim to "throw off" rival agents by not considering their history at all. This may work against simpler agents but may not perform as well when opponents are not cooperating and complex.

```
class AlternatePlayer extends Player {
    // Does not consider the history of the other players
    // Alternates between cooperation and defection
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        return n % 2;
    }
}
```

Fig 2: Alternate Player

2.3 Limited Forgiveness Player

The Limited Forgiveness Player is inspired by the TF2T strategy [1]. This involves setting a unique "betrayal tolerance". This strategy is aimed to make the agent cooperative and instil trust in its opponents. It starts by cooperating unconditionally irrespective of opponent choices. Even if the opponents defect, it continues to cooperate until its betrayal tolerance is breached. From this point on, it undertakes an aggressive stance and defects for the rest of the match.

```
class LimitedForgivenessPlayer extends Player {
    // Has a pre-set number of "forgiveness" to grant
    // Once the number has been reached, it defects

    int betrayalTolerance = 10;

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if(n == 0) return 0; // cooperate on first round
        int betrayals = 0;
        for(int i = 0 ; i < oppHistory1.length ; i++) {
            if(oppHistory1[i] == 1 || oppHistory2[i] == 1) betrayals++;
        }
        return (betrayals < betrayalTolerance) ? 0 : 1;
    }
}
```

Figure 3: Limited Forgiveness Player

2.4 Grim Trigger Player

The Grim Trigger Player is based on a strategy where the player is unforgiving once it has experienced defection. This means that the agent cooperates as long as its opponents have not defected. Once they do, it continues to defect for the rest of the match.

```
class GrimTriggerPlayer extends Player {
    // Cooperates until the opponent defects, then defects forever
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if(n == 0) return 0; // cooperate on first round
        for(int i = 0 ; i < oppHistory1.length ; i++) {
            if(oppHistory1[i] == 1 || oppHistory2[i] == 1) return 1;
        }
        return 0;
    }
}
```

Figure 4: Grim Trigger Player

2.5 Win-Stay-Lose-Shift Player

The Win-Stay-Lose-Shift Player or the Pavlov Player [1], works on a very simple principle: cooperate when everyone else does, defect otherwise. This player checks if the opponents both cooperated in the preceding round. If they did, it cooperates as well. This motivates all participating agents to cooperate as it ensures that the agent is seen as “trustworthy”. However, if either agent defects in the preceding round, it defects too. This strategy is different than the Grim Trigger or Limited Forgiveness as it does keep its doors open to cooperation irrespective of how many times it has been betrayed. If all opposing agents are willing to cooperate, it will cooperate as well.

```
class WinStayLoseShiftPlayer extends Player {
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if(n == 0) return 0; // cooperate on first round
        // cooperate if same decision was made by all players
        if(oppHistory1[n-1] == myHistory[n-1] && oppHistory2[n-1] == myHistory[n-1] && myHistory[n-1] == 0) return 0;
        // defect if different decision was made by all players
        return 1;
    }
}
```

Figure 5: Win-Stay-Lose-Shift Player

2.6 Increasing Greed Player

The Increasing Greed Player can be considered to be the opposite of the Limited Forgiveness Player. While the Limited Forgiveness Player has a “betrayal tolerance”, the Increasing Greed Player has a “kindness limit”. This limit ensures that the agent only cooperates a limited number of times and defects thereafter. As the kindness has a limit, it is only used when all opposing agents have cooperated in the previous round. Once the kindness limit is breached, the agent defects irrespective of its opponents history.

```
class IncreasingGreedPlayer extends Player {

    int kindnessLimit = 40;

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if(n == 0) return 0; // cooperate on first round
        int myKindness = 0;
        for(int i = 0 ; i < myHistory.length ; i++) {
            if(myHistory[i] == 0) myKindness++;
        }
        // kindness is scarce, use it wisely (only when opponents are kind too)
        if(oppHistory1[n-1] == 0 && oppHistory2[n-1] == 0 && myKindness < kindnessLimit) {
            return 0;
        } else {
            return 1;
        }
    }
}
```

Figure 6: Increasing Greed Player

2.7 Reversible Trigger Player

The Reversible Trigger Player is a modification of the Trigger Player. This strategy is similar to the Trigger Player as it defects if any of the opponents have defected in the preceding round. It continues to defect until both the opponents have proven their loyalty. The loyalty for a player is proven if it has decided to cooperate in the previous two rounds. This high bar where both opposing agents have to cooperate together in the two preceding rounds ensures that the agents cooperation is hard earned and rewards the other agents with cooperation as well. Similarly the low burden of causing defection (defection by either opponent in the preceding round) ensures that the agent’s cooperation will always be hard earned.

```
class ReversibleTriggerPlayer extends Player {
    // Cooperates until the player is berayed.
    // It then defects (like GrimTrigger) but restarts cooperation after show of loyalty

    boolean betrayed = false;

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if(n == 0) return 0; // cooperate on first round
        if(!betrayed){
            // the player hasn't been betrayed so far
            if(oppHistory1[n-1] == 0 && oppHistory2[n-1] == 0) {
                // the two opponents are cooperating
                return 0;
            } else {
                // the player has been betrayed
                betrayed = true;
                return 1;
            }
        } else {
            // the player has been betrayed previously
            if(opp1HasReprovenLoyalty(n, oppHistory1) && opp2HasReprovenLoyalty(n, oppHistory2)) {
                // the opponents have reproven loyalty
                betrayed = false;
                return 0;
            } else {
                // the opponents have not reproven loyalty
                return 1;
            }
        }

    }

    boolean opp1HasReprovenLoyalty(int n, int[] oppHistory1) {
        return (oppHistory1[n-1] == 0 && oppHistory1[n-2] == 0) ? true : false;
    }

    boolean opp2HasReprovenLoyalty(int n, int[] oppHistory2) {
        return (oppHistory2[n-1] == 0 && oppHistory2[n-2] == 0) ? true : false;
    }
}
```

Figure 7: Reversible Trigger Player

2.8 Limited Reversible Trigger Player

This is an amendment to the Reversible Trigger Strategy by introducing an element of Limited Forgiveness Player. Although the opposing players still have to prove their loyalty by cooperating in the two preceding rounds before earning the agent's trust, they can only afford to lose it a fixed number of times (betrayal tolerance). When this tolerance limit is breached, the agent will not cooperate irrespective of how many times the opposing agents prove their loyalty.

```
class LimitedReversibleTriggerPlayer extends Player {
    // Cooperates until the player is berayed.
    // It then defects (like GrimTrigger) but restarts cooperation after show of loyalty

    boolean betrayed = false;
    int betrayalTolerance = 10;

    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if(n == 0) return 0; // cooperate on first round
        if(betrayalToleranceBreached(n, oppHistory1, oppHistory2)) {
            betrayed = true;
            return 1;
        }
        if(!betrayed){
            // the player hasn't been betrayed so far
            if(oppHistory1[n-1] == 0 && oppHistory2[n-1] == 0) {
                // the two opponents are cooperating
                return 0;
            } else {
                // the player has been betrayed
                betrayed = true;
                return 1;
            }
        } else {
            // the player has been betrayed previously
            if(opp1HasReprovenLoyalty(n, oppHistory1) && opp2HasReprovenLoyalty(n, oppHistory2)) {
                // the opponents have reproven loyalty
                betrayed = false;
                return 0;
            } else {
                // the opponents have not reproven loyalty
                return 1;
            }
        }

    }

    boolean betrayalToleranceBreached(int n, int[] oppHistory1, int[] oppHistory2) {
        int betrayals = 0;
        for(int i = 0 ; i < oppHistory1.length ; i++) {
            if(oppHistory1[i] == 1 || oppHistory2[i] == 1) betrayals++;
        }
        return (betrayals > betrayalTolerance) ? true : false;
    }

    boolean opp1HasReprovenLoyalty(int n, int[] oppHistory1) {
        return (oppHistory1[n-1] == 0 && oppHistory1[n-2] == 0) ? true : false;
    }

    boolean opp2HasReprovenLoyalty(int n, int[] oppHistory2) {
        return (oppHistory2[n-1] == 0 && oppHistory2[n-2] == 0) ? true : false;
    }
}
```

Figure 8: Limited Reversible Trigger Player

2.9 Cooperative Tit-For-Tat Player

This player is extremely similar to the tit-for-tat player. If any of the opposing players defected in the preceding round, the agent defects as well. However, the agent is designed to be forgiving and strives to re-establish cooperation by voting to cooperate if it defected in the previous round. This ensures that the agent is amenable and open to cooperation.

```
class CooperativeT4TPlayer extends Player {
    // Defects if the opponent has defected
    // Strives to reestablish cooperation after defection
    int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
        if(n == 0) return 0; // cooperate on first round
        if(myHistory[n - 1] == 1) return 0; // if we defected, cooperate irrespective of opponent choice
        if(oppHistory1[n - 1] == 1 || oppHistory2[n - 1] == 1) return 1;
        return 0;
    }
}
```

Figure 9: Cooperative Tit-For-Tat Player

3. Results

To determine the best strategy, all the agents were tested. The following table presents their scores in 3 tournaments and the average score taken after the tournaments. The best agent will be judged based on the highest average score.

Player Name	Run 1	Run 2	Run 3	Average
LimitedForgivenessPlayer	635.3216	650.9859	646.90735	644.40495
GrimTriggerPlayer	638.7538	634.5513	633.16266	635.4892533
ReversibleTriggerPlayer	620.7099	631.3717	632.73804	628.2732133
LimitedReversibleTriggerPlayer	614.7262	632.8019	623.9081	623.8120667
TolerantPlayer	613.23956	627.14514	630.73834	623.70768
T4TPlayer	612.2927	624.2613	624.44666	620.3335533
CooperativeT4TPlayer	613.81213	588.54443	595.17395	599.1768367
WinStayLoseShiftPlayer	591.1075	608.5266	594.4273	598.0204667
NicePlayer	562.65216	584.6289	567.67114	571.6507333
IncreasingGreedPlayer	551.75696	563.4829	559.7604	558.33342
FreakyPlayer	491.00644	527.12244	504.9864	507.7050933
SuspiciousT4TPlayer	467.9691	496.2118	472.10385	478.7615833
NastyPlayer	483.98517	474.2453	470.54874	476.2597367
AlternatePlayer	427.2762	427.06464	429.7836	428.04148
RandomPlayer	437.05334	418.3569	415.52768	423.6459733

Table 2: Scores for tournaments

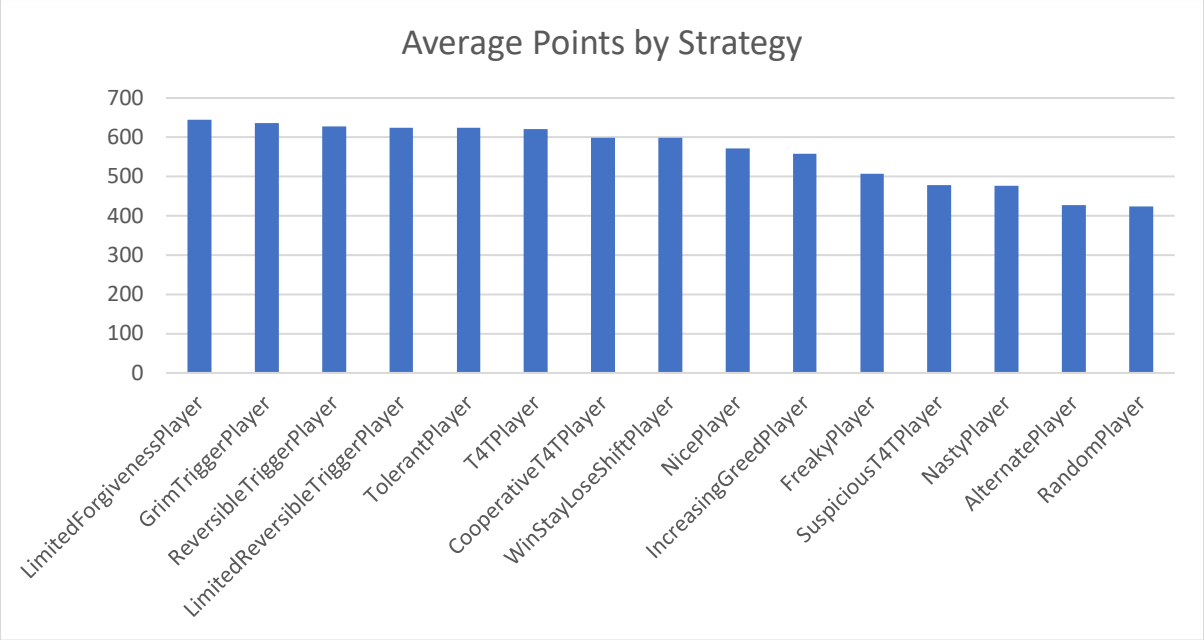

Figure 10: Average scores for the tournament

4. Conclusion

Based on the results of the tournament, the best strategy for the agent in the setting of repeated prisoners' dilemma is Limited Forgiveness. An agent that follows this strategy ensures that it seeks cooperation from its opponents by showing how it is willing to cooperate initially even if they choose to defect. However, once its betrayal tolerance is breached, it will defect for the rest of the match. This ensures that the agent present ample opportunities for its opponents to cooperate with it but also ensures that it will take a firmer stance with its opponents if they choose to defect. As such, this agent is chosen as the best agent.

5. References

[1] Strategies for the Iterated Prisoner's Dilemma (Stanford Encyclopaedia of Philosophy). Plato.stanford.edu, plato.stanford.edu/entries/prisoner-dilemma/strategy-table.html.

6. Appendix

The GitHub Repository for the project: https://github.com/IngaleOmkar/CZ4046-Assignment-2