CZ4046 Intelligent Agents

Repeated Prisoners Dilemma – Assignment 2

Name: Kyle Huang Junyuan

Matriculation: U1721717G

Table of Contents

Introduction

The Prisoner's Dilemma is a thought experiment that can show why separate individuals may not choose to cooperate, even if it appears beneficial for both parties to do so.

In this assignment, a strategy is developed for a three-player repeated Prisoner's Dilemma of about 100 rounds.

The following Table 1 lists all possible combinations along with their respective payoff. Cooperation is denoted using the integer 0 while defection using the integer 1.

You	Opponent 1	Opponent 2	Payoff
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: Possible combinations and their payoff

There are a few well-known strategies for a two-player Prisoner's Dilemma. One of which is Tit-for-Tat (T4T) – where an agent chooses to cooperate in the first round and subsequently mimics the opponent's action.

However, it is worth noting that the T4T strategy plays to tie the game and not to win it. Such a technique may also not translate very well into a three-player strategy. That is, which of the two opponents should the agent mimic if playing the T4T strategy?

Thought Process

The problem with always-defect

When playing against a single opponent in a two-player Prisoner's Dilemma, being an always-defect agent would always guarantee a win or a tie. This is because the opponent would never have a chance to gain points from the current player.

However, in a three-player Prisoner's Dilemma, the dynamics can change against an always-defect agent. This happens when the other two players team up against the always-defect player.

The following Figure 1 illustrates this problem with an example of a tournament with 1x NastyPlayer (always-defect) and 2x T4TPlayer (Tit-for-Tat). Even though NastyPlayer technically won every match against the T4TPlayer, the NastyPlayer only achieved about a total of 20 points compared to the T4TPlayer achieving about a total of 44 points – more than double. As such, **it is still important to cooperate with other players whenever possible**.

```
NastyPlayer scored 2.0 points, NastyPlayer scored 2.0 points, and NastyPlayer
scored 2.0 points.
NastyPlayer  scored  2.029703  points,  NastyPlayer  scored  2.029703  points,  and
T4TPlayer scored 1.980198 points.
NastyPlayer  scored  2.032258  points,  NastyPlayer  scored  2.032258  points,  and
T4TPlayer scored 1.9784946 points.
NastyPlayer  scored  2.060606  points,  T4TPlayer  scored  2.010101  points,  and
T4TPlayer scored 2.010101 points.
NastyPlayer  scored  2.185567  points,  T4TPlayer  scored  2.030928  points,  and
T4TPlayer scored 2.030928 points.
NastyPlayer  scored  2.15625  points,  T4TPlayer  scored  2.0520833  points,  and
T4TPlayer scored 2.0 points.
T4TPlayer scored 6.0 points, T4TPlayer scored 6.0 points, and T4TPlayer scored
6.0 points.
T4TPlayer scored 6.0 points, T4TPlayer scored 6.0 points, and T4TPlayer scored
6.0 points.
T4TPlayer scored 6.0 points, T4TPlayer scored 6.0 points, and T4TPlayer scored
6.0 points.
T4TPlayer scored 6.0 points, T4TPlayer scored 6.0 points, and T4TPlayer scored
6.0 points.

Tournament Results
T4TPlayer: 44.061504 points.
T4TPlayer: 44.031326 points.
NastyPlayer: 20.526344 points.
```

Figure 1: NastyPlayer vs T4T vs T4T

Defection should still be the dominant choice

A fine line between acting as an always-defect agent and sometimes cooperating needs to be found as the previous section emphasises the flaw of always defecting.

But defection should still be the dominant choice. A closer look at the table of all possible combinations and their payoff (but now sorted by payoff) in the following Table 2 reveals that the best action remains at defection – highlighted by the green rows. It can also be deduced that it is better for the current player if more opponents to choose cooperation.

You	**Opponent 1**	**Opponent 2**	**Payoff** (↓ Desc)
1	0	0	**8**
0	0	0	**6**
1	0	1	**5**
1	1	0	**5**
0	0	1	**3**
0	1	0	**3**
1	1	1	**2**
0	1	1	**0**

Table 2: Possible combinations and their payoff (Sorted)

The following scenarios in Table 3 can then be concluded:

Possible Scenarios	**Best Action**	**Payoff**
If both opponents choose defect	1 (Defect)	2
If one opponent chooses a different action from the other opponent	1 (Defect)	5
If both opponents choose to cooperate	0 (Cooperate) or 1 (Defect)	6 or 8

Table 3: Best-case scenario

Implementation

Discovery

Measuring the Opponents' Score

With the history of each opponent provided as the arguments of the *selectAction()* function, measuring the current total score of each opponent is possible.

The following code segment in Figure 2 shows how it can be done. By iterating through a total of *n* rounds, we can use the payoff matrix to calculate the past scores of *myHistory* as well as *oppHistory1* and *oppHistory2*. These scores are then accumulated into a *scores[]* array of size 3 containing the total scores respectively.

```
int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
    int[] scores = new int[3];
    for (int i = 0; i < n; i++) {
        scores[0] += payoff[myHistory[i]][oppHistory1[i]][oppHistory2[i]];
        scores[1] += payoff[oppHistory1[i]][oppHistory2[i]][myHistory[i]];
        scores[2] += payoff[oppHistory2[i]][myHistory[i]][oppHistory1[i]];
    }
    // ...
}
```

Figure 2: Measuring opponents' score

Guessing the Opponents' Strategy

It was also considered whether it was possible to detect the opponent's strategy using the provided history array. This would be highly effective if the designed strategy in this opponent only plays against the 6 provided players: NicePlayer, NastyPlayer, RandomPlayer, TolerantPlayer, FreakyPlayer and T4tPlayer.

However, based on the evaluation of this assignment, the strategy proposed in this document will be competing against agents designed by other students (in all combinations). These other strategies can be designed in an infinite number of ways and it would be virtually impossible to discover all of them.

As such, it would not be feasible to guess the opponents' strategy or to design a strategy that does well against the 6 provided players but fails terribly against unknown players. It is paramount to design a more robust and flexible strategy.

Trialled Strategies

A few strategies were trialled in this assignment to see which performed the best against the 6 provided players. But they were carefully designed to be more generic instead of beating their strategies specifically.

These were mainly trialled:

1. Reacting on opponents' score
2. Reacting on the most popular action
3. If one opponent has a fixed action – play T4T with the other

Approach 1 – Reacting on opponents' score
The first approach was simply reactive to the opponent's score. If either *oppHistory1* or *oppHistory2* had a higher score than *myHistory*, it would defect straight away. Else it would cooperate.

This approached worked rather well with around a 48% to 57%[1] chance of obtaining the first place out of all players.

Approach 2 – Reacting on most popular action
The second approach is an attempt to further improve on the first approach by adding another check. It would calculate the percentage of *oppHistory1* and *oppHistory2* cooperating and act upon it. That is, if both opponents are cooperating more than 90% of the time, it would choose to cooperate. Else it would defect.

This approached worked a little better with around a 58% to 60%[1] chance of obtaining the first place out of all players.

Approach 3 – If one opponent has a fixed action
The third approach will attempt to detect if an opponent has a fixed action. If so, it will then play Tit-for-tat (T4T) with the other opponent.

It is worth noting that the T4T strategy plays to tie the game and not to win it. Moreover, it is unlikely for other students to design a fixed action in their strategy as well.

With these considerations in mind, approach 3 was not used.

[1] How these trialled strategies were evaluated is explained in the *Evaluation* section.

Selected Strategy

The final selected strategy combines a few features from the above approaches and consists of just 4 main steps:

1. 1st round: Cooperate
2. Calculate the percentage of *oppHistory1* and *oppHistory2* cooperating
3. If both opponents are cooperating > 90%:
    a. Choose to cooperate
    b. If round > 95 – 100:
        i. Selfishly defect at the last moment
4. Default: Defect

The following code segment in Figure 3 shows the implementation of the final selected strategy.

```
int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {
    if (n == 0)
        return 0; // First round: Cooperate

    /* 1. Calculate percentage of cooperation */
    float perOpp1Coop = calCoopPercentage(oppHistory1);
    float perOpp2Coop = calCoopPercentage(oppHistory2);

    /* 2. If both players are mostly cooperating */
    if (perOpp1Coop > 90 && perOpp2Coop > 90) {
        int range = (10 - 5) + 1; // Max: 10, Min: 5
        int random = (int) (Math.random() * range) + 5;
        if (n > (90 + random))  // Selfish: Last min defect
            return 1;
        else
            return 0;   // First ~90 rounds: Cooperate
    }

    /* 3. Defect by default */
    return 1;
}
```

Figure 3: Code snippet of the selected strategy

In the first round, it will choose to cooperate. This is done in the hopes of informing the other opponents that this strategy is open for cooperation with other players.

Subsequently, the percentage of cooperation is calculated using a helper function called *calCoopPercentage()* that is shown in the following Figure 4. The percentages are then compared with both *oppHistory1* and *oppHistory2* to see if both opponents are cooperating more than 90% of the time. This is to target and maintain the possibility of cooperation between "nicer" players as described in the *Thought Process* section above.

Only at the very last moment between the 95th round and above, the strategy will defect at the very last minute to try and garner more points (i.e. 8 instead of just 6 when cooperating). This uncertainty is designed with a *Math.random()* value that will add between 5 – 10 to the 90th round.

Other than this specific condition, the strategy will default to defection.

```java
float calCoopPercentage(int[] history) {
    int cooperates = 0;
    int length = history.length;

    for (int i = 0; i < length; i++)
        if (history[i] == 0)
            cooperates++;

    return (float) cooperates / length * 100;
}
```

Figure 4: Code snippet of calCoopPercentage()

Evaluation

This section aims to describe the process of how the strategies discussed above were evaluated. The main idea was to run up to 200 tournaments and calculate the average probability of the final strategy obtaining 1st, 2nd or 3rd rank.

Firstly, in the main program, the *runTournament()* function was encapsulated in a for loop that repeats 200 times as shown in the following code snippet in Figure 5.

Secondly, the *runTournament()* function is also slightly modified to return the final rank of the specified player. This rank is then added to an array called *ranks[]* which will keep track of the number of times the 1st, 2nd or 3rd rank is achieved.

Finally, the average probability of obtaining each rank is printed after all 200 tournaments have completed.

```java
public static void main(String[] args) {
    int[] ranks = new int[7];
    int numTournaments = 200;

    for (int i = 1; i <= numTournaments; i++) {
        System.out.println("Tournament: " + i);
        ThreePrisonersDilemma instance = new ThreePrisonersDilemma();
        int rank = instance.runTournament("Huang_KyleJunyuan_Player");
        System.out.println();

        /* Update rankings */
        ranks[rank]++;
    }

    System.out.println("============ Ranking Probabilities ============");
    System.out.println("1st: " + ranks[0] / (float) numTournaments * 100 + "%");
    System.out.println("2nd: " + ranks[1] / (float) numTournaments * 100 + "%");
    System.out.println("3rd: " + ranks[2] / (float) numTournaments * 100 + "%");
}
```

Figure 5: Code snippet of multiple tournaments

Trialled Strategies

The following Figure 6 shows the final evaluation results of Approach 1 as described in the *Trialled Strategies* section. It averages about 48% to 57% chance of obtaining the 1st place. It can also be deduced that it has a 97% chance of obtaining the top 3 position when played against the 6 provided players.

```
============ Ranking Probabilities ============
1st: 57.5%
2nd: 21.5%
3rd: 18.0%
```

Figure 6: Evaluation results of Approach 1

The following Figure 7 shows the final evaluation results of Approach 2 as described in the *Trialled Strategies* section. It averages about 58% to 60% chance of obtaining the 1st place. It can also be deduced that it has a 99% chance of obtaining the top 3 positions when played against the 6 provided players.

```
============ Ranking Probabilities ============
1st: 60.000004%
2nd: 27.000002%
3rd: 12.0%
```

Figure 7: Evaluation results of Approach 2

Selected Strategy

The following Figure 8 shows the final evaluation results of the final selected strategy. It averages about 66% to 71% chance of obtaining the 1st place. It can also be deduced that it has a **99.5% chance of obtaining the top 3 position** when played against the 6 provided players.

```
============ Ranking Probabilities ============
1st: 71.0%
2nd: 17.0%
3rd: 11.5%
```

Figure 8: Evaluation results of final selected strategy

Conclusion

In conclusion, developing an always winning strategy is a very difficult task due to the infinite number of ways a strategy can counteract one that has already been designed.

The final selected strategy proposed in this report has been designed to be as generic as possible – dominantly defecting but allowing for cooperation at times to try and achieve the highest possible score.