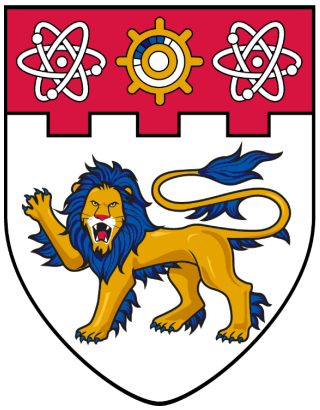

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4046: Intelligent Agents

ASSIGNMENT 2

Prepared by:

GUCON NAILAH GINYLLE PABILONIA

U2021643H

Table of Content

Introduction

The Repeated Prisoner's Dilemma is a variation of the classic Prisoner's Dilemma game in game theory where a given game is played multiple times by the same set of players. In the case of this assignment, triples of players will play with each other repeatedly in a ‘match’. Each player has the opportunity to cooperate or defect in each round of the game. The players may use their knowledge of each other's previous choices to inform their decisions in subsequent rounds.

There are various strategies that players can use to play the Repeated Prisoner's Dilemma, including "tit-for-tat," "grim trigger," and many others. These strategies are designed to incentivise cooperation and punish defection over multiple rounds of play.

The results of the match to the Repeated Prisoner's Dilemma depends on the strategies employed by the players, as well as the number of rounds played and the payoffs associated with different outcomes. Like in the two-player version, cooperation and communication between players can lead to mutually beneficial outcomes, while repeated defection can result in suboptimal outcomes for all players.

Overall, the Repeated Prisoner's Dilemma highlights the importance of trust and cooperation in repeated interactions between players.

Agent Design

After exploring a range of strategies, I developed a mixed strategy that strikes a balance between cooperation and defection, while also adapting to the specific behaviour of the opponents. The key strategies used with explanations are listed below.

Joss

Joss's strategy is very similar to Tit-for-Tat, starting with cooperation and responding to defection with defection and cooperation with cooperation. The difference is that Joss makes a "surprise defection" decision with a 10% chance after the opponent cooperates.

I took inspiration from the Joss strategy and designed my agent to take a similar but more refined approach by considering the cooperation ratio of each opponent before deciding whether to cooperate or defect. If both opponents have a high cooperation ratio (above or equal to 0.8), then the player cooperates. If either opponent has a low cooperation ratio, then the player cooperates with a 90% probability.

Implementing this strategy in the initial 10 rounds can create a perception of my agent as a cooperative player, while also allowing for a strategic "surprise defection" in response to an opponent with a low cooperation ratio, without fully revealing my agent's overall approach.

Tit-for-Tat

Tit-for-Tat is one of the most effective strategies for iterated games like the Prisoner's Dilemma where a player starts by cooperating and then replicates the previous move of the opponent in subsequent rounds. In other words, if the opponent cooperated in the previous round, the player will cooperate in the current round, and if the opponent defected, the player will defect in the current round.

I integrated Tit-for-Tat into my agent code with some enhancements. My agent not only responds to cooperation with cooperation but also considers the defection rate of each opponent and adjusts its strategy accordingly. It also includes some punishment and forgiveness mechanisms, such as immediately punishing any opponent who defects, and forgiving opponents who have shown low defection rates in the past.

This gives my agent a nice balance between being nice and being retaliatory.

Agent Implementation

Variable Initialisation

In my agent's strategy, it needs to know the past behaviour of its opponents to calculate its next move.

To achieve this, the variables `oppDefections1` and `oppDefections2` are initialised to count the number of times each opponent has defected in arrays `oppHistory1` and `oppHistory2`, respectively.

Similarly, `oppCooperations1` and `oppCooperations2` are initialised to count the number of times each opponent has cooperated in arrays `oppHistory1` and `oppHistory2`, respectively.

Finally, the variables oppCooperationRatio1 and oppCooperationRatio2 are calculated using the formula:

```
1.0 * oppCooperations / (oppCooperations + oppDefections)
```

which represents the cooperation ratio of each opponent. This ratio is used by my agent to decide its strategy in the first 10 rounds of the game.

```
// Variable Declarations
int oppDefections1 = 0, oppDefections2 = 0, oppCooperations1 = 0, oppCooperations2 = 0;

// Count the number of times the opponents defected and cooperated in the past
for (int i = 0; i < n; i++) {
    if (oppHistory1[i] == 1) oppDefections1++;
    else oppCooperations1++;
    if (oppHistory2[i] == 1) oppDefections2++;
    else oppCooperations2++;
}

// Calculate the cooperation ratio of the opponents
double oppCooperationRatio1 = 1.0 * oppCooperations1 / (oppCooperations1 +
oppDefections1);
double oppCooperationRatio2 = 1.0 * oppCooperations2 / (oppCooperations2 +
oppDefections2);
...
```

Figure 1: Variable Declaration Code

First Round

My agent always cooperates in the first round of every match. Assuming that other agents follow this same rule, this leads to a reciprocal cooperation that ensures a high payoff throughout the game.

Although it may be tempting to defect in the first round to achieve the highest immediate payoff, this strategy is unlikely to succeed in the long run since defection is likely to be punished in subsequent rounds, leading to lower payoffs overall. Therefore, my agent chooses to cooperate from the start, which leads to better outcomes in the long term.

```
if (n == 0) { return 0; }
...
```

Figure 2: First Round Code

First 10 Rounds

For the first 10 rounds (with the exception of the first round) of the match, my agent makes use of a refined version of Joss strategy that I mention in the “Agent Design” section above, where the cooperation ratio of each opponent is considered before deciding whether to cooperate or defect with a certain probability.

If both opponents have an oppCooperationRatio value of more than 0.8 (i.e. cooperated in at least 80% of the previous rounds), it will cooperate by returning 0. This is because it assumes that its opponents are trustworthy and that cooperation will result in mutual benefit.

In any other scenario, like for example, one of the opponents have a oppCooperationRatio of less than 0.8, my agent has a 90% chance of cooperating (returning a 0) and a 10% chance of defecting (returning a 1). This allows for some random variation in my agent's decision-making process, which can prevent opponents from predicting its strategy and taking advantage of it.

```
...
else if (n <= 10) {
                if (oppCooperationRatio1 >= 0.8 && oppCooperationRatio2 >= 0.8) {
                    return 0;
                } else {
                    return Math.random() < 0.9 ? 0 : 1;
                }
            }
...
```

Figure 3: First 10 rounds Code

All Subsequent Rounds

In the subsequent rounds, my agent makes use of an enhanced Tit-for-Tat strategy that I mentioned in the “Agent Design” section above.

My agent first calculates the defection rate of each opponent by dividing the number of times each opponent has defected (stored in `oppDefections1` and `oppDefections2`) by the total number of rounds played so far (`n`).

```
} else {
    double oppDefectRate1 = 1.0 * oppDefections1 / n;
    double oppDefectRate2 = 1.0 * oppDefections2 / n;
```

Figure 4: All Subsequent Rounds Code Snippet 1

If any of the opponents has defected more than 20% of the time (see “Evaluating Defection Rates” section for defection percentage threshold explanation), my agent perceives the opponents as untrustworthy or likely to defect, and so it chooses to defect in order to protect itself from losing points in the future.

Hence, this code snippet checks if either opponent's defection rate is greater than or equal to `0.2`. If either has, the agent defects by returning 1.

```
if (oppDefectRate1 >= 0.2 || oppDefectRate2 >= 0.2) {
    return 1;
}
```

Figure 5: All Subsequent Rounds Code Snippet 2

If both opponents have defected less than 5% of the time (see “Evaluating Defection Rates” section for defection percentage threshold explanation), it is likely that they are playing a cooperative strategy. In this case, it is more beneficial for my agent to also cooperate, establishing a cooperative relationship with both opponents, which can lead to mutual gains in the long term.

Hence, This code snippet checks if both opponents have a defection rate of less than `0.05`. If yes, my agent cooperates by returning 0.

```
if (oppDefectRate1 < 0.05 && oppDefectRate2 < 0.05) {
    return 0;
}
```

Figure 6: All Subsequent Rounds Code Snippet 3

If both opponents have defection rates between `0.05` and `0.2`, my agent has to make a decision based on the recent behaviour of its opponents. If both opponents cooperated in the previous round, it could indicate that they are trying to cooperate and reach a mutual benefit. However, if either opponent defected in the round before that, it suggests that they may not be trustworthy and are only cooperating intermittently, and my agent defects in response to protect itself and discourage future defections. If both opponents have been mostly cooperative, my agent assumes they are trying to cooperate and reach a mutual benefit, and it cooperates as well. Finally, if either opponent defected in the previous round, it suggests that they are not trustworthy or cooperative, and my agent defects as a way of punishing them for their behaviour. This behaviour incentivizes both opponents to cooperate in future rounds, promoting mutually beneficial outcomes.

Hence, this code snippet checks if both opponents cooperated in the previous round. If they did, my agent checks if either opponent defected in the round before that. If either did, the agent defects by returning 1. Otherwise, the agent cooperates by returning 0. If at least one of the opponents did not cooperate in the previous round, the agent defects by returning 1.

```
if (oppHistory1[n - 1] == 0 && oppHistory2[n - 1] == 0) {
    if (oppHistory1[n - 2] == 1 || oppHistory2[n - 2] == 1) {
        return 1;
    } else {
        return 0;
    }
} else {
    return 1;
}
...
```

Figure 7: All Subsequent Rounds Code Snippet 4

Evaluating Defection Rates

Threshold values for the defection rates of the agents can be useful because they provide a simple way to determine whether the opponents are behaving cooperatively or uncooperatively, and to adjust my agent's strategy accordingly, making more informed and effective decisions.

Deciding on a specific threshold value depends on the specific context and assumptions of the match being played. If a good threshold value for defection rate is chosen, it can be useful in indicating if an opponent is not trustworthy and is likely to defect again in the future. However, if a bad threshold value for defection rate is chosen, it may result in overly cautious behaviour or missed opportunities for cooperation.

Hence, a simple experiment involving 5 different threshold values of defection rates (`0.05, 0.10, 0.15, 0.20, 0.25`) and the strategies used by my agent's opponents in a series of tournaments are observed.

Based on the results obtained, the following conclusions have been made:

- ➔ If an opponent's defection rate is more than or equals to `0.2`, it is deemed to be not trustworthy as it has been defecting for the majority of the rounds held thus far, and will likely continue to defect in future rounds.
- ➔ If an opponent's defection rate is between `0.05` and `0.2`, it is deemed to have a less predictable strategy and my agent will have to depend on other factors to decide its next move.
- ➔ If an opponent's defection rate is less than `0.05`, it is deemed to be trustworthy as it has been cooperating for the majority of the rounds held thus far, and will likely continue to cooperate in future rounds.

The defection rates of my agent's opponents will likely change as the match progresses, catering to the possibility of opponents switching strategies mid-match.

Overall, these threshold values are chosen based on a balance between being strict enough to avoid cooperating with untrustworthy opponents too often, and being lenient enough to cooperate with trustworthy opponents as much as possible.

Performance Comparison and Analysis

In this section, we do a comparison and analysis of tournaments that were held between my agent, Gucon_Nailah_Player, and the given strategy agents to evaluate the effectiveness of my agent's design, followed by an All Agents Tournament which involved all the agents.

For both the individual and All Agents tournaments, instead of using the raw scores, the code was slightly edited, as seen in Figure X, so that the comparison will be done using the average payoff. Using the average payoff across multiple rounds is a better indication of an agent's performance in the tournaments because it takes into account the cumulative effect of an agent's decisions over the different rounds in the tournaments. Then the average payoff for each tournament for each agent will be summed up and produce the final verdict.

```
System.out.println("Tournament Results");
       for (int i = 0; i < numPlayers; i++)
           System.out.println(makePlayer(sortedOrder[i]).name() + ": "
                   + totalScore[sortedOrder[i]]/ numMatches[i] + " points.");
```

Figure 8: Main temporary change made to main code

Gucon_Nailah_Player VS Individual Given Strategy Agents

Table 1: Match Scores against individual given strategy agents

Opponent	Opponent Score	My Score
NicePlayer	600.0	600.0
NastyPlayer	217.0	401.1
RandomPlayer	318.5	501.8
TolerantPlayer	600.0	600.0
FreakyPlayer	303.2	450.2
T4TPlayer	600.0	600.0

A total of 600 tournaments were held (100 for each given strategy agent) and the summed average scores are listed in Table 1.

Observations

When my agent goes up against NicePlayer, TolerantPlayer and T4TPlayer, it outputs similar scores due to the fact that there's mutual cooperation throughout the tournaments. There is no defection as all the agents will never be the first to defect, hence guaranteeing cooperation.

Since the NastyPlayer always defects, my agent would always defect in response, resulting in mutual defection. However, my agent was able to surpass NastyPlayer when playing with itself due to mutual cooperation, which has a much higher payoff.

Similarly, my agent was able to surpass RandomAgent and FreakyAgent when playing with itself due to mutual cooperation. For RandomPlayer, it picks its action randomly, my agent would have to take into account other factors such as RandomPlayer's recent actions to determine whether or not it should cooperate or defect. For FreakyPlayer, it randomly chooses whether to cooperate or defect at the start of the match, and then sticks to that decision throughout the game. If FreakyPlayer cooperates, my agent would cooperate, and if it defects, my agent would also defect.

All Agents Tournament

Table 2: All Agents Tournament Results

Rank	Opponent	Summed Average Score
1	**Gucon_Nailah_Player**	464.1857992
2	**TolerantPlayer**	457.6997843
3	**T4TPlayer**	455.4015683
4	**NicePlayer**	422.1192199
5	**FreakyPlayer**	412.1927742
6	**NastyPlayer**	395.3001105
7	**RandomPlayer**	379.468969

A total of 100 tournaments were held and the summed average scores are listed in Table 2.

Observations

It can be observed that my agent was able to outperform every other agent in the All Agents Tournament, likely because its strategy takes into account multiple factors when making decisions. This makes it more likely to perform well against a wide range of opponents, including those that are more basic or predictable.

My agent did face challenges when playing against certain opponents. When facing an aggressive opponent like the NastyPlayer, which always defects, my agent had to respond with defection in order to avoid exploitation and mutual defection. Similarly, when facing a cooperative opponent like TolerantPlayer, which is very tolerant of cooperation and can easily take advantage of other cooperative agents, my agent needs to adjust its strategy to cooperate more often and establish a cooperative relationship with the opponent, while still being prepared to punish the opponent if they defect. Despite these hurdles, my agent was able to effectively adapt its strategy and still achieve better payoffs against this opponent for the majority of the tournaments and come up on top overall.

Conclusion

In summary, my agent displayed strong performances against the given strategy agents in both the individual and the All Agents tournaments, demonstrating its ability to effectively navigate the complex dynamics of the Prisoner's Dilemma game. The combination of historical information tracking and decision making strategies allowed my agent to adjust its actions based on the actions of the opponents, leading to more successful outcomes that benefited both sides.

It appears that reciprocal cooperation, and the avoidance of being the first to defect, is a winning strategy. This is evident in Table 2 where agents that adhere to these principles, such as my agent, TolerantPlayer, T4TPlayer and NicePlayer, outperformed agents that may initiate a defection on their own, such as FreakyPlayer, NastyPlayer and RandomPlayer. This highlights the importance of cooperation in achieving positive outcomes in cooperative game scenarios like Prisoner’s Dilemma.