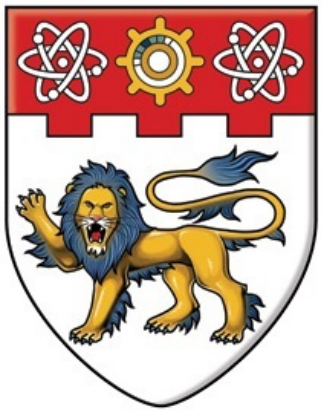


NANYANG TECHNOLOGICAL UNIVERSITY

CZ4046 Intelligent Agents

Assignment 2 – Three Prisoners Dilemma

Jervis Chan Jun Yong
U1821352H

Introduction to Repeated Prisoners Dilemma

The prisoner's dilemma showcases how two completely rational individuals might not cooperate, even if it appears that it is in their best interests to do so. It introduces a paradox in which 2 individuals acting in their own self-interests do not produce the optimal outcome for the collective parties. In the one-round Prisoner's Dilemma, defection strictly dominates cooperation for each player; causing mutual defection to be the dominant strategy for all players.

This changes in the Repeated Prisoner's Dilemma as the time horizon increases and players are now incentivised to maximise the total personal utility values received across multiple rounds of the Prisoner's Dilemma. In other words, players now have to consider the impact of their current decision on their opponents' future actions - which would in turn affect their future utility.

Three Player Repeated Prisoners Dilemma

In our experiment of a Repeated Prisoner's Dilemma for 3 players, triplets of players will play each other repeatedly in a match and make decisions based on their different strategies.

Payoff Matrix

Three-player Prisoner's Dilemma - Payoff Matrix						
Player 1	Cooperate	Player 2	Cooperate	Player 3	Cooperate	Player 1's Payoff: 6
Player 1	Cooperate	Player 2	Cooperate	Player 3	Defect	Player 1's Payoff: 3
Player 1	Cooperate	Player 2	Defect	Player 3	Cooperate	Player 1's Payoff: 3
Player 1	Cooperate	Player 2	Defect	Player 3	Defect	Player 1's Payoff: 0
Player 1	Defect	Player 2	Cooperate	Player 3	Cooperate	Player 1's Payoff: 8
Player 1	Defect	Player 2	Cooperate	Player 3	Defect	Player 1's Payoff: 5
Player 1	Defect	Player 2	Defect	Player 3	Cooperate	Player 1's Payoff: 5
Player 1	Defect	Player 2	Defect	Player 3	Defect	Player 1's Payoff: 2

The above payoff matrix illustrates the different combination of actions that can take place in a Three Player Prisoner's Dilemma round. As we can see, a player can receive a payoff value from 0 to 8 - depending on the combination of actions from the different players. The following code implements the payoff structure for the Three-player Prisoner's Dilemma.

```
static int[][][] payoff = {
            {{6, 3}, // payoffs when first and second players cooperate
            {3, 0}}, // payoffs when first player coops, second defects
            {{8, 5}, // payoffs when first player defects, second coops
            {5, 2}}}; // payoffs when first and second players defect
```

```
private static void showPayoff() {
    for (int i = 0; i < payoff.length; i++) {
        for (int j = 0; j < payoff[0].length; j++) {
             for (int k = 0; k < payoff[0][0].length; k++) {
                  System.out.printf(
                      "Player 1 %-10s Player 2 %-10s Player 3 %-10s --> "
                      + " Player 1's payoff: %d%n",
                      i == 0 ? "Cooperate," : "Defect,",
                      j == 0 ? "Cooperate," : "Defect,",
                      k == 0 ? "Cooperate" : "Defect",
                      payoff[i][j][k]);
             }
        }
    }
    System.out.println();
}
```

In each match, players will play each other repeatedly and continuously for 90-110 (implemented through randomisation) rounds. According to the code implementation below, accumulated scores of the players will be recorded to be used to evaluate performance of the agents and their respective strategies in the tournament.

```
float[] scoresOfMatch(Player A, Player B, Player C, int rounds) {
        int[] HistoryA = new int[0], HistoryB = new int[0], HistoryC = new
int[0];
        float ScoreA = 0, ScoreB = 0, ScoreC = 0;

        for (int i = 0; i < rounds; i++) {
            int PlayA = A.selectAction(i, HistoryA, HistoryB, HistoryC);
            int PlayB = B.selectAction(i, HistoryB, HistoryC, HistoryA);
            int PlayC = C.selectAction(i, HistoryC, HistoryA, HistoryB);

            ScoreA = ScoreA + payoff[PlayA][PlayB][PlayC];
            ScoreB = ScoreB + payoff[PlayB][PlayC][PlayA];
            ScoreC = ScoreC + payoff[PlayC][PlayA][PlayB];

            HistoryA = extendIntArray(HistoryA, PlayA);
            HistoryB = extendIntArray(HistoryB, PlayB);
            HistoryC = extendIntArray(HistoryC, PlayC);
        }

        float[] result = {ScoreA / rounds, ScoreB / rounds, ScoreC /
rounds};
        return result;
```

Agent Behaviour

In order to build a strong agent, the most common and successful strategies for the Repeated Prisoner's Dilemma have been curated for us to evaluate our agent against. The first 6 players are agents provided by the assignment,which we will use to serve as our "control" in our experimentation.

Sample Players for Performance Benchmarking

1) **Nice Player**
   - The Nice Player always cooperates in all situations.

2) **Nasty Player**
   - The Nasty Player always defects in all situations.

3) **Random Player**
   - The Random Player cooperates and defects randomly with a split 50-50% probability.

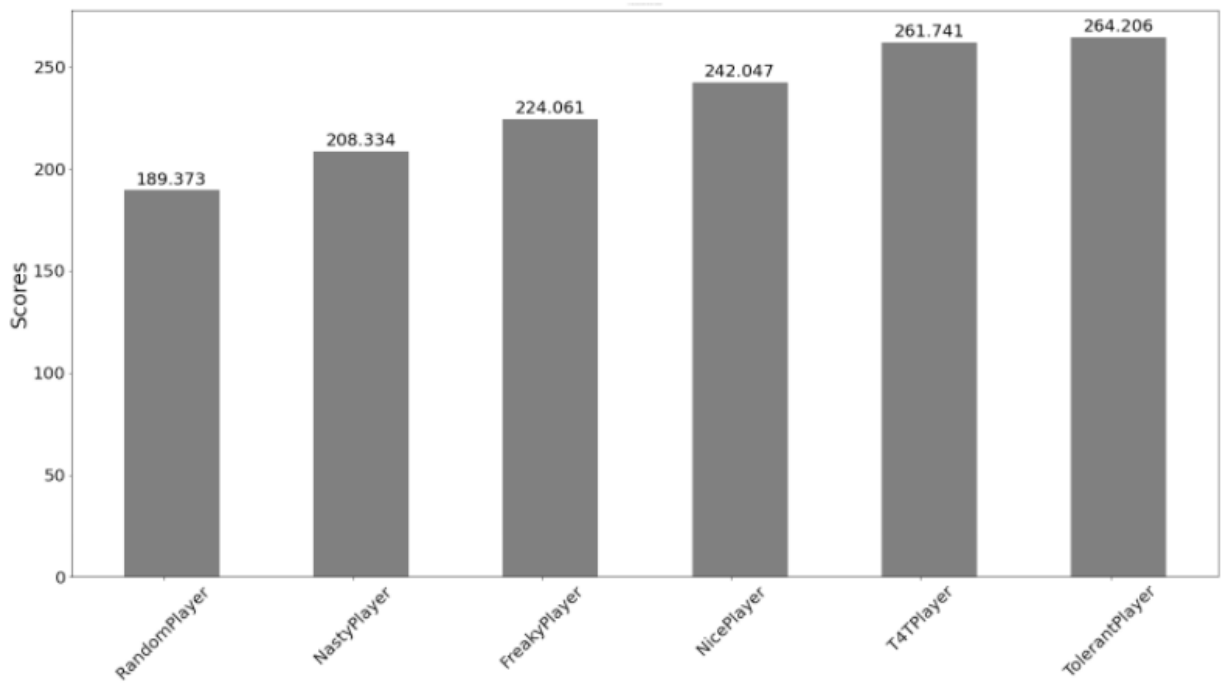
4) **Freaky Player**
   - The Freaky Player determines at the start of the match, either to always be nice (cooperate) or always be nasty (defects).

5) **Tolerant Player**
   - The Tolerant Player looks into his opponents' histories, and defects if at least half of the other players' actions have been defects.

6) **Tit-for-Tat Player**
   - The strategy "Tit-for-Tat" (TFT) cooperates on the first round and repeats the previous move of its opponent thereafter.

Pre-Agent Behaviour

Before introducing JervisChan, I placed the sample agents in a preliminary round of tournament for me to evaluate the effectiveness of different design elements. The tournaments were repeated 5000 times (5000 iterations) and the scores of the agents are averaged to minimise the variability of randomness.



From the results above, we can see that the 4 lowest-scoring agents share the common characteristic of being unreactive agents. All 4 of those agents will decide on a particular decision to either cooperate or defect without taking into account their opponent's past moves. This tells us that reactive agents are more effective than unreactive agents and we should utilise the past moves of our opponents in our agent design strategy.

Finally, we can see that the agents T4TPlayer and TolerantPlayer are successful - being placed at the top 2 positions. These agents share the common characteristic of playing nice when their opponents have exhibited a tendency to cooperate. Therefore, we learn that the willingness to cooperate when faced with "nice" opponents is effective in capitalising on mutual long-term partnership.

Agent Design

After gathering the above insights from our preliminary tournament between the sample agents, **t**he following rules describe the key principles used in the creation of our agent and will influence how the agent makes a decision to either cooperate or defect. As most of the rules established in our agent design is not mutually exclusive, they will be listed in decreasing importance, with the first rule being the most important guiding principle for the agent.

Key Rules:

1) **Cooperate at the start**
According to Axelrod, a program is known to be "nice" if it starts by cooperating. Empirical observations highlighted that the outcome of the game depends on whether the rule used by the program was nice or not. Although there is a risk that it may receive the sucker's payoff on the first round, this loss of utility is relatively small when compared to potential benefits that will be achieved in later rounds by means of mutual cooperation. Hence, our agent will adopt this rule and start the game by being nice.

2) **Cooperate in a cooperative environment**
In the nature of the Repeated Prisoner's Dilemma, all players will have access to the history of all past actions of their opponents before making a decision to cooperate or defect for the subsequent round. If it is observed that an opponent is playing nice by cooperating, we can deem that the opponent would want to partake in a long-term win-win partnership with us. With that insight, we will also cooperate to maximise our mutual long-term benefits.

3) **If someone betrays me, punish them harshly so they stop doing it.**
If we observe that an opponent has chosen to defect against us - we can deem that the opponent has decided to betray our mutual partnership and is optimising for their personal gain at our expense. When that happens, we should defect quickly to punish that behaviour with an aim of getting our opponents to cooperate for subsequent rounds.

4) **Forgive fast.**
After punishing our opponents when betrayed with rule 4, we should observe their future moves to check if they have learned their lessons. If they choose to remain defecting, we will then partake in the dominant strategy of mutual defection. On the other hand, if the opponent chooses to cooperate - indicating that they have learnt their lesson, we should give them a chance to re-engage in a mutually-benefitting win-win partnership by cooperating.

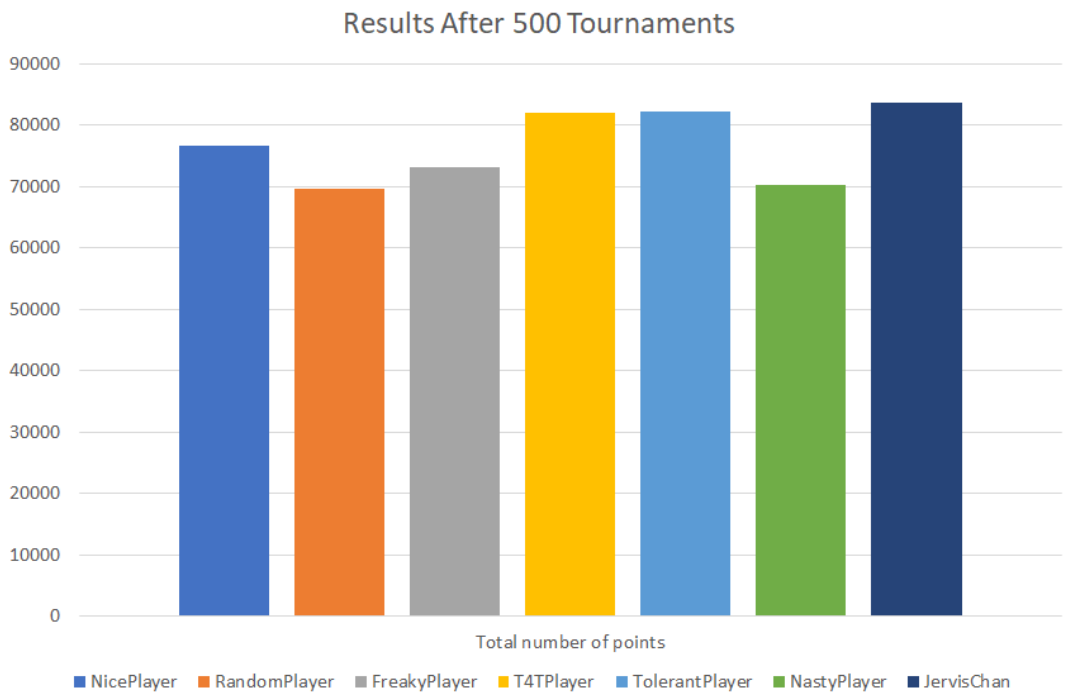
5) **"I will cooperate unless you defect 3 times in a row at which point I will defect forever."**

6) **If unsure, choose cooperation over defecting.**

Agent Implementation

```
class JervisChan extends Player {

        boolean trigger = false;

        int selectAction(int n, int[] myHistory, int[] oppHistory1, int[] oppHistory2) {

                // Rule 1 - Cooperate at the start
                if (n==0) {
                        return 0;
                }

                if (trigger==true) {
                        return 1;
                }

                // Rule 2 - Cooperate in a cooperative environment
                if(oppHistory1[n-1]==0 && oppHistory2[n-1]==0) {
                        return 0;
                }

                // Rule 3 - If someone betrays me, punish them harshly immediately
                if(oppHistory1[n-1]==1 || oppHistory2[n-1]==1) {
                        return 1;
                }

                // Rule 5 -  "I will cooperate unless you defect 3 times in a row at which point I will
defect forever."
                if (n>5) {
                if(oppHistory1[n-3]==1 && oppHistory2[n-3]==1 && oppHistory1[n-2]==1 &&
oppHistory2[n-2]==1 && oppHistory1[n-1]==1 && oppHistory2[n-1]==1) {
                        trigger=true;
                }

                }
        return 0; // Rule 6 - If unsure, choose cooperation over defecting
    }
}
```

Performance Evaluation of our Agent



The agents were made to play 500 tournaments which consisted of 90 - 110 rounds per tournament. The following result can be seen is as follows:
1st Place: JervisChan, 83928 points
2nd Place: TolerantPlayer, 82530
3rd Place: T4TPlayer, 82113

JervisChan has consistently achieved the top placing for each tournament, which allowed him to have the highest score after 500 tournaments.

Analysis of JervisChan Performance Against Others

JervisChan versus Nice Player

JervisChan always performs better than Nice Player no matter how many tournaments are played. This is simply because Nice Player is not able to react to betrayers.

JervisChan versus Nasty Player

JervisChan always performs better than Nasty Player no matter how many tournaments are played. This is because a nasty player is not able to adapt into a cooperative environment which will ultimately earn the most points for the players.

JervisChan versus Random Player

JervisChan always performs better than Random Player. This is because the random player does not know when is the right time to cooperate and when is the right time to defect.

JervisChan versus Freaky Player

Since the Freaky Player is either Nice or Nasty, JervisChan always performs better than the Freaky Player as well.

JervisChan versus T4T Player

JervisChan usually wins the T4T Player. This is likely because the tit-for-tat player picks a random opponent to base its tit for tat strategy on, when in actual fact, it should be considering both players.

JervisChan versus Tolerant Player

JervisChan usually wins the Tolerant Player. This is likely because the Tolerant Player is satisfied with being in an environment where one player cooperates while the other player defects. However, in such a situation, the tolerant player would stand to gain more if it chooses to defect.

Conclusion

In the repeated prisoner's dilemma, the optimal strategy changes rapidly depending on the strategies adopted by other opponents in the game. The interesting thing about the prisoner's dilemma is that whenever a strategy is highly adopted by its players, it opens up an opportunity for another strategy to exploit those strengths and turn them into weaknesses. For example, in a 2-player Prisoner's dilemma, the NastyPlayer will always outplay the Tit4TatPlayer as the NastyPlayer will always either tie with (D|D) or win with (D|C). However, in a 3-player Prisoner's dilemma with 2 Tit4TatPlayers and 1 NastyPlayer, we can see that even though the NastyPlayer has technically won both of its matches against its opponents, it would have earned a lower point total than the other two players - as the 2 Tit4TatPlayers would benefit from being able to cooperate with each other. This tells us that it is impossible to create a strategy with absolute certainty to prevail in a tournament, without first knowing the strategies adopted by other players.

Therefore in our agent design, we have chosen to stick to the core principles that give rise to successful strategies. We explored successful design elements of the prisoner's dilemma, and synergised them into one strategy that is flexible and generally effective.