Assignment 3: Computer Engineering Case Study – Iterated Prisoner's Dilemma

Available: Monday Nov. 20, 2023
Due: Sunday Dec. 3 @11:55pm (Brightspace submission system only!)

Programming aspects to get familiarized with:
- Object-Oriented Programming.

Background

The iterated prisoner's dilemma (IPD) is a popular game from game theory. The traditional form of the prisoner's dilemma game has two players. Each player must choose between one of two possible moves: defect or cooperate. The combination of your move and your opponent's move determines some form of payoff, usually represented as a score.

The name "prisoner's dilemma" is derived from the following situation: Two prisoners are serving time for a minor offense. However, both are suspected of having committed a far more serious crime. The police approach each prisoner, privately, with the same deal. Each is given the choice between:

1. Implicating the other prisoner (i.e., defect, relative to the other prisoner) and thereby getting paroled,
2. Not implicating the other prisoner (i.e., cooperate, relative to the other prisoner) and thereby continuing to serve time for the minor offense.

In this example, each suspect has only one move and one payoff. If both cooperate, each continues to serve their remaining time in prison. If both defect, each gets paroled; however, each is then convicted of the more serious crime and must serve a new, longer jail sentence. If one defects and the other cooperates, the defector goes free, and the cooperator spends a lot of time behind bars. What should you do in this situation? What do you hope your fellow prisoner does? How will you behave towards your fellow prisoner in the future if they implicate you in the crime? This situation is the basis for interesting research in many areas, including political science, biology, and economics.

In the implementation of the IPD game, you will have repeated interactions with your opponent, rather than just once, thus the name iterated prisoner's dilemma. Your first move in the game, cooperative or defect, will be decided with no knowledge of how your opponent will move. However, on all subsequent moves you can decide whether to cooperate or defect based on your opponent's last move. This is when the strategy becomes interesting.

You will be competing with your opponent. The goal of the game is to accumulate the maximum number of points in a given number of moves (N). The payoff table for the prisoner's dilemma is as follows:

IPD Payoff Table				
	Cooperate		**Defect**	
Cooperate	**3**,	**3**	**0**,	**5**
Defect	**5**,	**0**	**1**,	**1**

Game Strategies

If you look at table 1 carefully, you will notice that the highest payoff occurs when your opponent cooperates and you defect. However, if both players cooperate, each player will receive a higher payoff than if both had defected.

If the game consisted of only one move, then you could argue that it is rational to defect on the first move to give yourself the best chance of winning the game. However, since you will encounter your opponent multiple times, and your opponent will know your behavior, you may have a better chance of accumulating points if you attempt to form a cooperative relationship with your opponent. Developing a cooperative strategy, while protecting oneself from defectors, is what makes this game an interesting behavior model, and a challenging programming task. One very simple and effective strategy is called the "tit for tat". With this strategy, your next move is always your opponent's last move. There are other strategies such as the "Evil", "Random", "Cooperate", and "Forgiving". You can research the implementation of these strategies.

The Software

This assignment involves the design of a software to simulate the IPD game. The software implements and tests the following classes:

- A `Player` class: A Player class represents a player in the game. It has the following data attributes: a unique ID (automatically generated), a score, an array of all previous moves in the game, the number of moves made so far, a reference to a game `Strategy` object, and the total number of players (maximum of two players). In addition to the necessary constructor, destructor, accessor, and modifier functions, the player class implements functions such as `makeMove()` (based on the specific strategy and last players moves), `printMoves()`, `setLastMove()`, `updateStrategy`(), among other functions that you find useful.
- A `Strategy` class: The `Strategy` class implements multiple strategies for the players to use, such as the "Evil", "Random", "Cooperate", and "Forgiving" strategies. The Strategy class has a private data member to store the code for the current strategy (e.g. 0 for random, 1 for always defect, 2 for always cooperate, etc.). The `Strategy` class implements a function, called `cooperateOrDefect()`, to return the player's move based on the adopted strategy and the last moves of the two players. You may add more functions as needed (constructors, accessors and modifiers, etc.). This class must implement at least 4 strategies.
- A `Game` class: The `Game` class manages the game between two players. The `Game` class maintains reference to two `Player` objects, keep track of the number of players, and the total number of moves for the game (N). The `Game` class is responsible for setting up the game (by adding/dropping players to/from the game – for a maximum of two players), playing the game (interaction between the two players), calculating the scores for the players based on the specific game strategy, and reporting the results (scores) after completing the game (N moves).

You must implement a scenario to test the three classes in the `main()` function.

Write your report using the five steps model for software development:

a) Step 1: Problem Identification and Statement (5 points)
b) Step 2: Gathering Information (10 points)
c) Step 3: Test Cases and algorithm (35 points)
d) Step 4: Code or implementation (35 points)
e) Step 5: Test and Verification (15 points)