Iterated Prisoner’s Dilemma

Aman Sunesh (as18181)

STEP 1: Problem Identification and Statement

The objective of this assignment is to design and implement a software simulation of the Iterated Prisoner's Dilemma (IPD) game, a classic from game theory. The program uses information such as the player strategies and the number of rounds to play the game . The software prints the moves of each player in each round and displays the final result (winner information) at the end of the game. It also allows the user to reset the game, add new players, and drop existing players.

STEP 2: Gathering of Information and Input and Output Description

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.

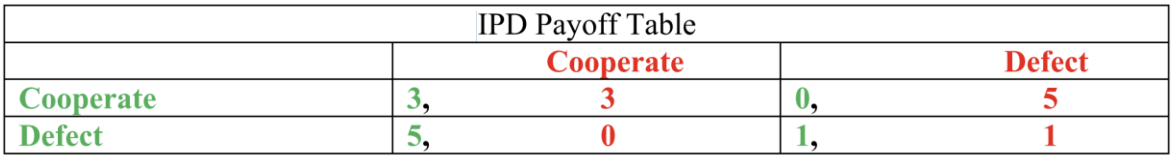
The name “prisoner’s dilemma” is derived from the following situation: Two prisoners are serving time for a minor offense. However, both are suspected of having committed a far more serious crime. The police approach each prisoner, privately, with the same deal. Each is given the choice between:

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. The consequences of these choices, when applied iteratively, set the stage for a dynamic and strategic game. This situation is the basis for interesting research in many areas, including political science, biology, and economics.

In the iterated version of the dilemma, players engage in repeated interactions with their opponent, with the first move made without knowledge of the opponent's response. Subsequent moves allow players to adapt their strategy based on their opponent's last move, introducing an element of complexity and strategic foresight.

The objective of the game is to accumulate the maximum points over a series of moves (N). The payoff table, a crucial component of the IPD, outlines the rewards for different combinations of cooperative and defective moves. Strategic nuances arise from the interplay of decisions, presenting players with the challenge of balancing self-interest with the potential for mutual cooperation.



IPD Payoff Table

	Cooperate	Defect
Cooperate	3, 3	0, 5
Defect	5, 0	1, 1

However, the complexity deepens as repeated interactions introduce the need for strategic thinking, fostering the development of cooperative strategies like "tit for tat" or exploring alternatives such as "Evil," "Random," "Cooperate," and "Forgiving."

Game Strategies

Examining the payoff table reveals intriguing dynamics, such as the allure of the highest payoff when one player defects while the other cooperates. However, if both players cooperate, each player will receive a higher payoff than if both had defected. In a single-move game, a rational choice might be to defect for the best chance of winning. However, in multiple encounters, forming a cooperative relationship with your opponent can be more advantageous. Developing a cooperative strategy, while guarding against defectors, is what makes this game an interesting behaviour 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 “Tit for Tat”.

1. **Evil:**

   The "Evil" strategy is straightforward and selfish. It always chooses to defect, betraying its opponent regardless of the opponent's previous move. This strategy aims to maximize its own score at the expense of the opponent.

2. **Cooperate:**

   The "Cooperate" strategy is cooperative and trusting. It always chooses to cooperate, regardless of the opponent's previous move. This strategy aims to build mutual cooperation, fostering a positive outcome for both players over repeated interactions.

3. **Random:**

   The "Random" strategy is unpredictable. In each round, it randomly chooses to cooperate or defect with a 50-50 chance. This strategy introduces an element of uncertainty, making it challenging for opponents to anticipate its moves.

4. **Tit for Tat:**

   The "Tit for Tat" strategy is reciprocal and forgiving. It starts by cooperating or defecting (as per player decision) and then mirrors its opponent's last move in subsequent rounds. If the opponent cooperates, it cooperates; if the opponent defects, it defects. This strategy encourages cooperation by reciprocating the opponent's behaviour.

Input Description:

The program for this problem would consist of classes and functions to create and manage players, define game strategies, and execute the game. The main functionalities include adding/dropping players, setting the number of rounds, choosing game strategies, and initiating the game simulation.

The data relevant to this problem consists of integer values, character values, string values, and boolean values.

Beginning with choice1 (character value), which plays a vital role in navigating the program’s menu.

1) If User Chooses 1) (Add/Drop Players) – Game class is called:
   Inputs: choice2, numOfPlayers, pID (ID of player to be dropped), playerName, addPlayers

2) If User Chooses 2) (Set Number of Rounds) – Game class is called:
   Inputs: numOfRounds

3) If User chooses 3) (Choose game strategy) – Game class is called:
   Inputs: choice3(game strategy of each player), defaultChar

Output Description:

1. Player 1 Move (for each round)

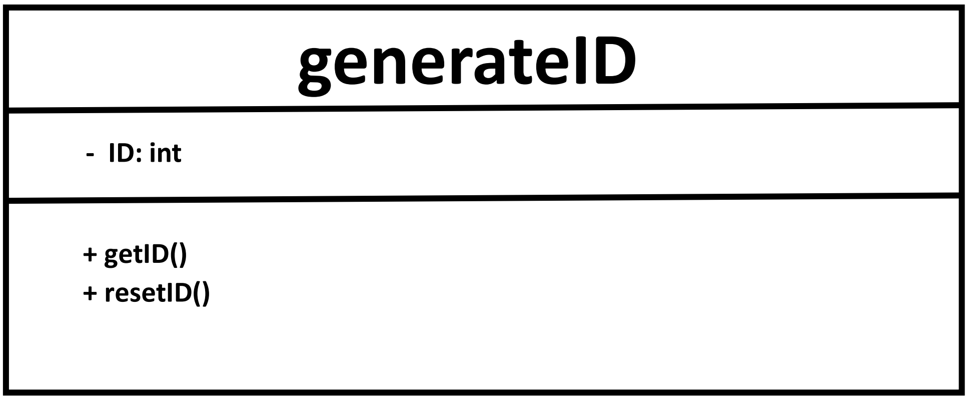
2. Player 2 Move (for each round)

3. Player Information (Name, ID , Strategy ,Score, Moves)

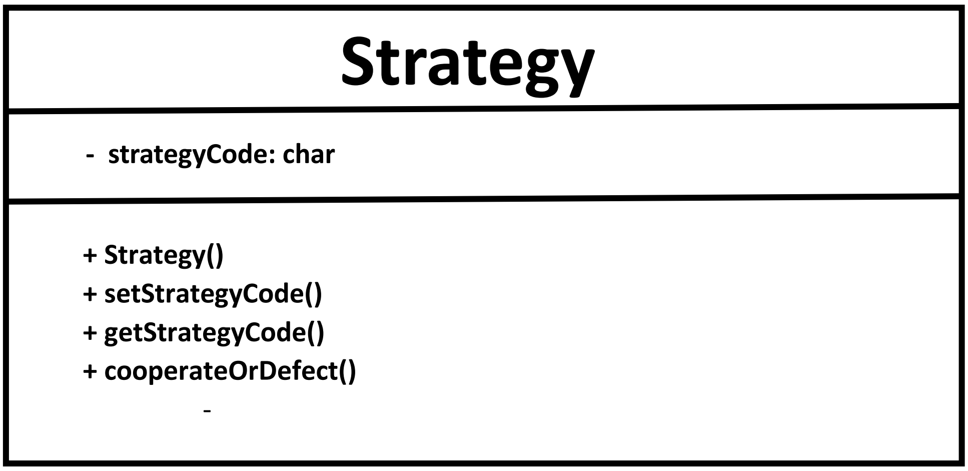
4. Winner Information (Name, ID, Score)

Class Diagrams:

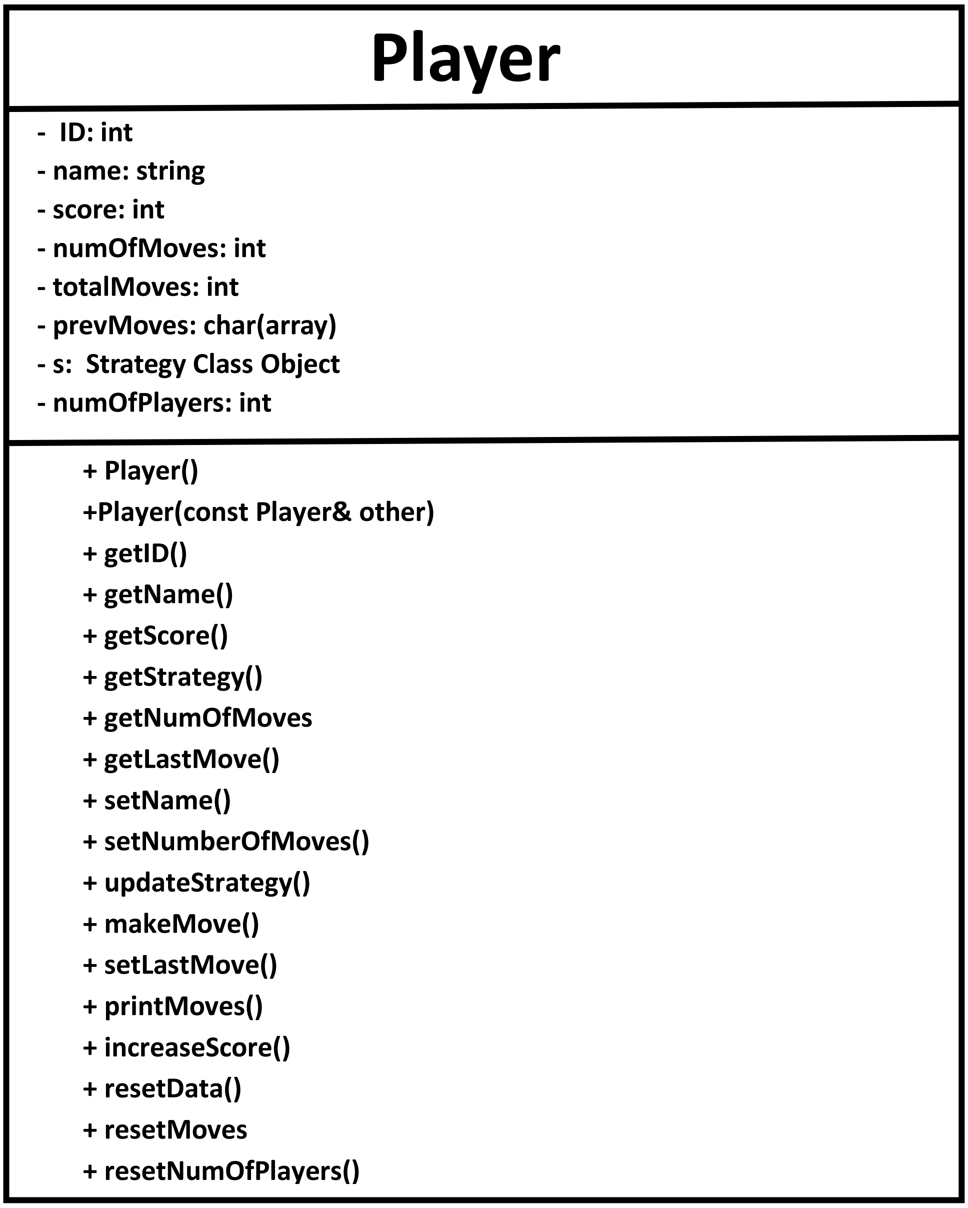
(i)



(i)



(iii)

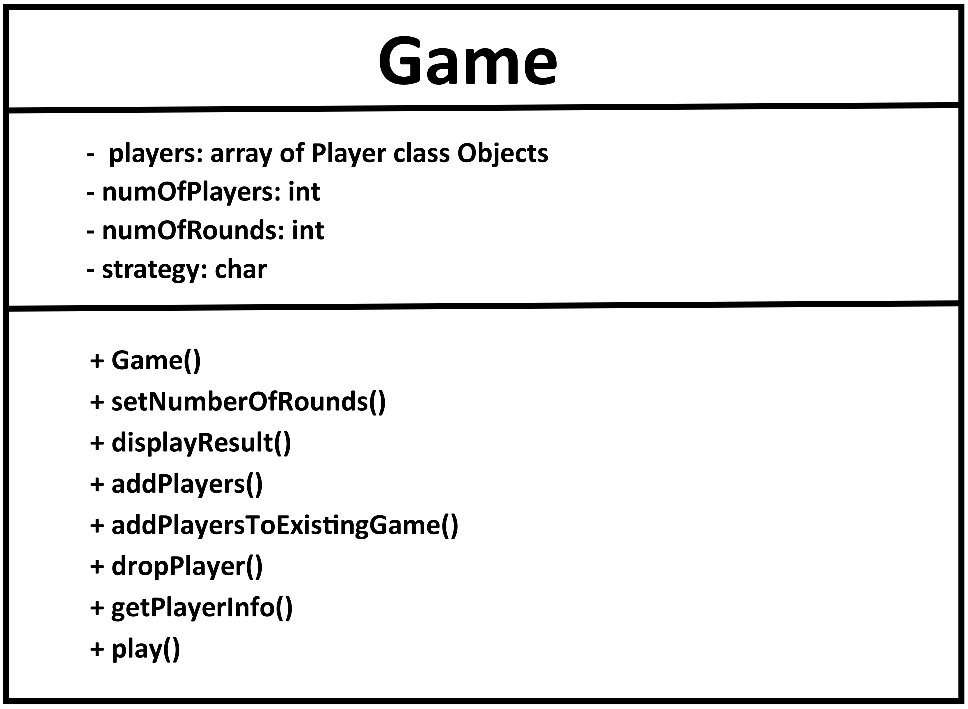


Player

- ID: int
- name: string
- score: int
- numOfMoves: int
- totalMoves: int
- prevMoves: char(array)
- s: Strategy Class Object
- numOfPlayers: int

+ Player()
+Player(const Player& other)
+ getID()
+ getName()
+ getScore()
+ getStrategy()
+ getNumOfMoves
+ getLastMove()
+ setName()
+ setNumberOfMoves()
+ updateStrategy()
+ makeMove()
+ setLastMove()
+ printMoves()
+ increaseScore()
+ resetData()
+ resetMoves
+ resetNumOfPlayers()

(iv)



Game

- players: array of Player class Objects
- numOfPlayers: int
- numOfRounds: int
- strategy: char

+ Game()
+ setNumberOfRounds()
+ displayResult()
+ addPlayers()
+ addPlayersToExistingGame()
+ dropPlayer()
+ getPlayerInfo()
+ play()

The I/O diagram for this problem is illustrated below:

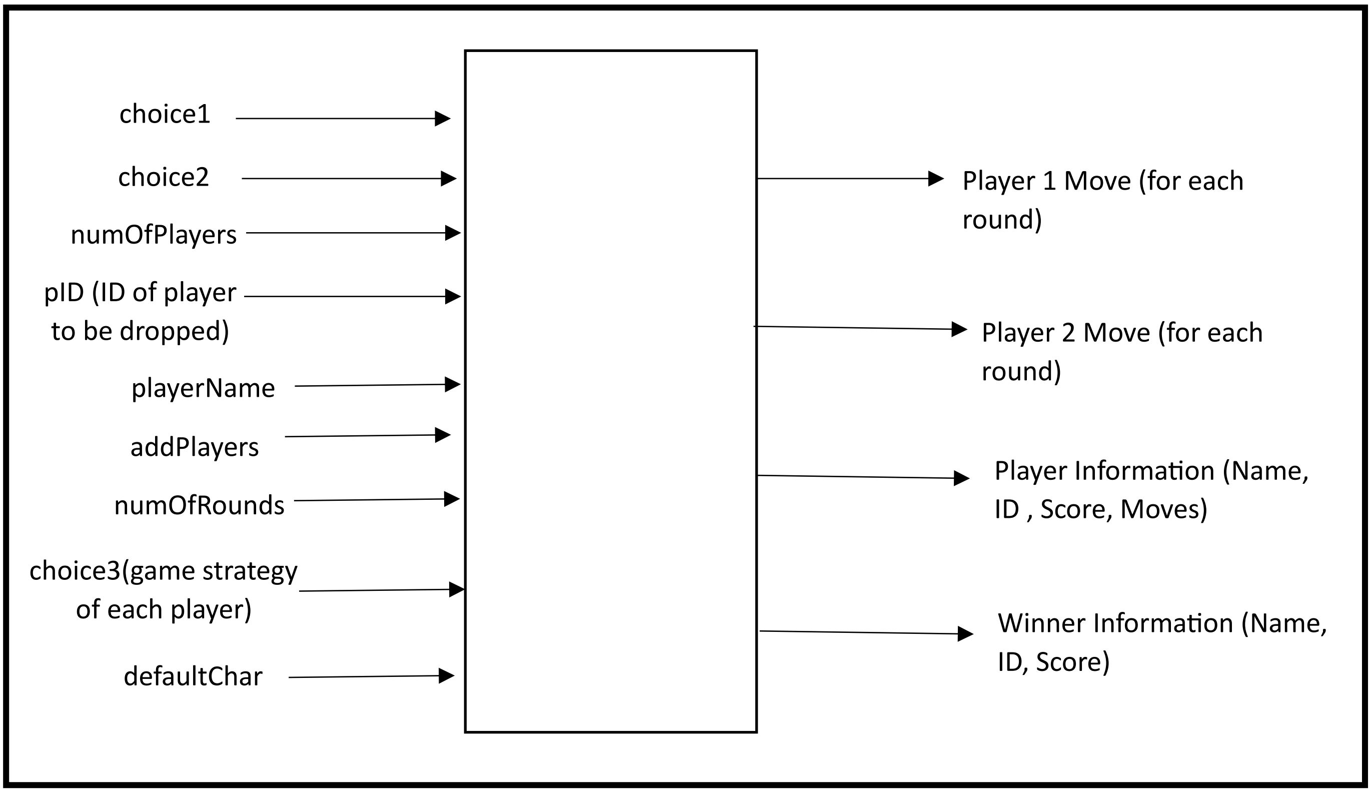

choice1
choice2
numOfPlayers
pID (ID of player to be dropped)
playerName
addPlayers
numOfRounds
choice3(game strategy of each player)
defaultChar
Player 1 Move (for each round)
Player 2 Move (for each round)
Player Information (Name, ID , Score, Moves)
Winner Information (Name, ID, Score)

Figure 0.1: I/O Diagram

Description of Menu and Output Messages:

1. **Menu Options:**

- Option 1: Add/Drop Players
- Option 2: Set Number of Rounds
- Option 3: Choose your Game Strategy
- Option 4: Start the Game
- Option 5: Exit

2. **User Interaction and Input Validation:**

- The program begins by displaying the menu options. The user is prompted to enter their choice (1, 2, 3, 4, or 5).
- If the user enters a choice other than 1, 2, 3, 4, or 5, the program displays an appropriate error message and prompts the user to re-enter a valid choice.
- Each option's execution is dependent on the completion of certain steps. For example, option 1 requires players to be created before dropping, option 2 requires players to be created, and option 3 requires players and the number of rounds to be set.

3. **Option 1: Add/Drop Players**

- Sub-options:
  - **1: Add New Players to the Game (Reset Game)**
    - The user is prompted to enter the number of players (exactly 2 players required).
    - Input validation ensures that the entered value is a positive integer and exactly 2 players are added.
    - Names for each player are entered with input validation for string values.

  - **2. Add players to the Existing Game**
    - Ensure that players are already created (Option 1, Sub-option 1).
    - The user is prompted to enter the number of players to add to the existing game.
    - Input validation ensures a positive integer value for the number of players to add, and the total number of players does not exceed the limit.
    - Names for each additional player are entered with input validation for string values.

  - **3: Drop Players**
    - The user is prompted to enter the ID of the player to drop.
    - Input validation ensures a non-negative integer value for the player ID.

4. **Option 2: Set Number of Rounds**

- The user is prompted to enter the number of rounds.
- Input validation ensures that the entered value is a positive integer.
- The number of rounds is set for the game, and memory is allocated for each player's moves based on the number of rounds.

5. **Option 3: Choose your Game Strategy**

- The user is prompted to choose a strategy (r, c, e, t) for each player.
- Input validation ensures valid strategy codes are entered.
- The chosen strategy is set for each player.

6. **Option 4: Start the Game**

- The game is played through multiple rounds.
- Players make moves based on their strategies and opponents' previous moves.
- Scores are calculated for each player based on the interactions.
- The winner is determined by the highest cumulative score.
- The final result is displayed, indicating the winner or a tie.

7. **Option 5: Exit**

- Displays a thank you message and terminates the program.

STEP 3: Test Cases and Algorithm Design

Test Cases:

```
Press 1 to Add/Drop Players
Press 2 to Set Number of Rounds
Press 3 to Choose your Game Strategy
Press 4 to Start the Game
Press 5 to quit the Program

Please enter your choice:
```

1.1) User Selects Option 1

1.1.1) User Selects Option 1 and Enters valid Input

Output:

```
Press 1 to Add New Players to the Game (Begin/Reset Game)
Press 2 to Add Players to the Existing Game
Press 3 to Drop Players

Please enter your choice:1

Please enter the number of players: 2

Enter the name for Player1: A
Players added successfully!

Enter the name for Player2: B
Players added successfully!
```

1.1.2) User Selects Option 1 and Enters Invalid Input

Output:

```
Press 1 to Add New Players to the Game (Begin/Reset Game)
Press 2 to Add Players to the Existing Game
Press 3 to Drop Players

Please enter your choice:1

Please enter the number of players: 3
Error! Please enter exactly 2 players:
```

1.1.3) User Selects Option 2 (After having added 2 players)

Output:

```
Press 1 to Add New Players to the Game (Begin/Reset Game)
Press 2 to Add Players to the Existing Game
Press 3 to Drop Players

Please enter your choice:1
Please enter the number of players: 2

Enter the name for Player1: A
Players added successfully!

Enter the name for Player2: B
Players added successfully!

Press 1 to Add New Players to the Game (Begin/Reset Game)
Press 2 to Add Players to the Existing Game
Press 3 to Drop Players

Please enter your choice:2
Error! Max Limit Reached! Could not add new players
```

1.1.4) User Selects Option 2 (After having added 2 players and then Dropping 1 Player (Option 3))

Output:

```
Press 1 to Add New Players to the Game (Begin/Reset Game)
Press 2 to Add Players to the Existing Game
Press 3 to Drop Players

Please enter your choice:1

Please enter the number of players: 2

Enter the name for Player1: A
Players added successfully!

Enter the name for Player2: B
Players added successfully!

Press 1 to Add New Players to the Game (Begin/Reset Game)
Press 2 to Add Players to the Existing Game
Press 3 to Drop Players

Please enter your choice:3
Enter the ID of the player you want to drop: 1
Player 1 successfully deleted


Press 1 to Add New Players to the Game (Begin/Reset Game)
Press 2 to Add Players to the Existing Game
Press 3 to Drop Players

Please enter your choice:2
Please enter the number of players to add to the game: 1
Enter the name of Player: C
Players added successfully!
```

1.2) User Selects Option 2

1.2.1) User has not Selected Option1 before Option 2
Output: *Error: Create the players first (Option 1)!*

1.2.2) User has Selected Option1 before Option 2
Output: *Please enter the number of rounds: 5*

1.3) User Selects Option 3

1.3.1) User has not Selected Option 1 or 2 before Option 3
Output: *Error: Create the players, and set the number of rounds first (Options 1, and 2)!*

1.3.2) User has Selected Option 1 and 2 before Option 3 (User enters invalid input)
Output:

```
=============================
               GAME STRATEGIES
=============================
Enter r for Random
Enter c for Cooperate
Enter e for Evil
Enter t for Tit for Tat

Player 1, Please choose your strategy: g
Please either 'r', 'c', 'e', or 't' for your strategy :
```

1.3.3) User has Selected Option 1 and 2 before Option 3 (User enters valid input)

Output:

```
====================================
                GAME STRATEGIES
====================================
Enter r for Random
Enter c for Cooperate
Enter e for Evil
Enter t for Tit for Tat

Player 1, Please choose your strategy: c
Player 2, Please choose your strategy: e
```

1.4) **User Selects Option 4**

1.4.1) **Player 1 Strategy: Cooperate, Player 2 Strategy: Evil**

```
 Press 1 to Add New Players to the Game (Begin/Reset Game)
 Press 2 to Add Players to the Existing Game
 Press 3 to Drop Players

 Please enter your choice:1

 Please enter the number of players: 2

 Enter the name for Player1: A
 Players added successfully!

 Enter the name for Player2: B
 Players added successfully!

Please enter the number of rounds: 5

Player 1, Please choose your strategy: c
Player 2, Please choose your strategy: e
```

Output:

Round	*Player 2*	*Player 2 Score*	*Player 1*	*Player 1 Score*
1	*defect*	*5*	*cooperate*	*0*
2	*defect*	*5+5 = 10*	*cooperate*	*0 + 0 = 0*
3	*defect*	*10 + 5 = 15*	*cooperate*	*0 + 0 = 0*
4	*defect*	*15 + 5 = 20*	*cooperate*	*0 + 0 = 0*
5	*defect*	*20 + 5 = 25*	*cooperate*	*0 + 0 = 0*
Score:		***25***		***0***

Result:
WinnerID: 2
Name: B
Score: 25

1.4.2) **Player 1 Strategy: Cooperate, Player 2 Strategy: Tit for tat**

```
 Press 1 to Add New Players to the Game (Begin/Reset Game)
 Press 2 to Add Players to the Existing Game
 Press 3 to Drop Players

 Please enter your choice:1

 Please enter the number of players: 2

 Enter the name for Player1: A
 Players added successfully!

 Enter the name for Player2: B
 Players added successfully!

Please enter the number of rounds: 7

Player 1, Please choose your strategy: c
Player 2, Please choose your strategy: t
```

Output:

Player 2, please enter your first move: Enter 'c' for cooperate or 'd' for defect: d

Round	*Player 2*	*Player 2 Score*	*Player 1*	*Player 1 Score*
1	*defect*	*5*	*cooperate*	*0*
2	*cooperate*	*5 + 3 = 8*	*cooperate*	*0 + 3 = 3*
3	*cooperate*	*8 + 3 = 11*	*cooperate*	*3 + 3 = 6*
4	*cooperate*	*11 + 3 = 14*	*cooperate*	*6 + 3 = 9*
5	*cooperate*	*14 + 3 = 17*	*cooperate*	*9 + 3 = 12*
6	*cooperate*	*17 + 3 = 20*	*cooperate*	*12 + 3 = 15*
7	*cooperate*	*20 + 3 = 23*	*cooperate*	*15 + 3 = 18*
Score:		***23***		***18***

Result:
WinnerID: 2
Name: B
Score: 23

1.4.3) Game 1: **Player 1 Strategy: Tit for Tat, Player 2 Strategy: Tit for tat**
Game 2: **Player 1 Strategy: Tit for Tat, Player 2 Strategy: Random**

Press 1 to Add New Players to the Game (Begin/Reset Game)
Press 2 to Add Players to the Existing Game
Press 3 to Drop Players

Please enter your choice:1

Please enter the number of players: 2

Enter the name for Player1: Mr John
Players added successfully!

Enter the name for Player2: Aman
Players added successfully!

Please enter the number of rounds: 10

Player 1, Please choose your strategy: t
Player 2, Please choose your strategy: t

Output:

```
Player 1, please enter your first move: Enter ‘c’ for cooperate or ‘d’ for defect: d
Player 2, please enter your first move: Enter ‘c’ for cooperate or ‘d’ for defect: c
```

Round	*Player 2*	*Player 2 Score*	*Player 1*	*Player 1 Score*
1	*cooperate*	*0*	*defect*	*5*
2	*defect*	*0 + 5 = 5*	*cooperate*	*5 + 0 = 5*
3	*cooperate*	*5 + 0 = 5*	*defect*	*5 + 5 = 10*
4	*defect*	*5 + 5 = 10*	*cooperate*	*10 + 0 = 10*
5	*cooperate*	*10 + 0 =10*	*defect*	*10 + 5 = 15*
6	*defect*	*10 + 5 = 15*	*cooperate*	*15 + 0 = 15*
7	*cooperate*	*15 + 0 = 15*	*defect*	*15 + 5 = 20*
8	*defect*	*15 + 5 = 20*	*cooperate*	*20 + 0 = 20*
9	*cooperate*	*20 + 0 = 20*	*defect*	*20 + 5 = 25*
10	*defect*	*20 + 5 = 25*	*cooperate*	*25 + 0 = 25*
Score:		***25***		***25***

Result:

```
It’s a tie

Tied Players:
Player 1
Player 2

Score of each player: 25
```

Round 2 (Continue running the code for the second time with new players):

```
 Press 1 to Add New Players to the Game (Begin/Reset Game)
 Press 2 to Add Players to the Existing Game
 Press 3 to Drop Players

 Please enter your choice:1

 Please enter the number of players: 2

 Enter the name for Player1: Alex
 Players added successfully!

 Enter the name for Player2: Mary
 Players added successfully!
```

```
Please enter the number of rounds: 5

Player 1, Please choose your strategy: r
Player 2, Please choose your strategy: c
```

Output:

Since Player 1 has chosen a random strategy, we cannot precisely predict the final scores of each player. However, given that Player 2 has opted for a cooperative strategy, we can anticipate that Player 1 is likely to win the game. If Player 1's move is 'c' through the random strategy, both players would receive 3 points. On the other hand, if Player 1's move is 'd' through the random strategy, Player 1 would gain 5 points, and Player 2 would receive 0 points. Therefore, we can logically infer that Player 1 is positioned to win.

Result:
WinnerID: 1
Name: Alex
Score: (depends on moves made)

1.5) User Selects Option 5
Output: *Thank you for playing!*
End of the program!

1) Menu Options

Algorithm Design:

```
Declare and Initialize Max_Players to 2


Define class generateID
    Define private variables: static variable ID;


    Define static function getID()
        Increment ID by 1
        Return ID


    Define static function resetID()
        Set ID to 0


Initialize static variable ID of generateID class to 0
```

```
Define class Strategy

    Define private variables: strategyCode


    Define default contructor Strategy()
        Set strategyCode to ‘r’

    Define function setStrategyCode(char code), with parameter code
        Set strategyCode to code

    Define function getStrategyCode()
        Return strategyCode

    Define function cooperateOrDefect(char opponentLastMove), with parameter opponentLastMove

        Switch based on strategyCode
            If strategyCode is equal to ‘r’
                If rand()%2 is equal to 0
                    Return ‘c’
                Otherwise
                    Return ‘d’

            If strategyCode is equal to ‘c’
                Return ‘c’

            If strategyCode is equal to ‘d’
                Return ‘d’

            If strategyCode is equal to ‘t’
                Return opponentLastMove

            default case
                Print “Error: Invalid Strategy Code”, newline
                Return ‘\0’


Define class Player
    Define private variables: ID, name, score, numOfMoves, totalMoves, prevMoves, s, static variable numOfPlayers


    Define default constructor Player()
        Set ID using getID() function of generateID class
        Set name to “”
        Set score to 0
        Set numOfMoves to 0
        Set totalMoves to 0
        Set prevMoves to nullptr
        Increment numOfPlayers by 1


    Define copy constructor Player(const Player& other), with parameter Player object other
        Set ID to other.ID
        Set name to other.name
        Set score to other.score
        Set numOfMoves to other.numOfMoves
        Set totalMoves to other.totalMoves
        If other.prevMoves is not equal to nullptr
            Set prevMoves = other.prevMoves

        Otherwise
            Set prevMoves to nullptr

        Set s = other.s

    Define function getID()
        Return ID

    Define function getName()
        Return name

    Define function getScore()
        Return score

    Define function getStrategy()
        Return s.getStrategyCode()

    Define function getNumOfMoves()
        Return numOfMoves

    Define function getLastMove(int x), with parameter x

        If prevMoves is not equal to nullptr and x >= 0 and x < numOfMoves
            return prevMoves[numOfMoves – (numOfPlayers – 1) + x]


        Otherwise
            Print “Error! Cannot access last move”


    Define function setName(string playerName), with parameter playerName
        Set name to playerName

    Define function setNumberOfMoves(int initTotalMoves), with parameter initTotalMoves
        Set totalMoves to initTotalMoves * (numOfPlayers – 1)
        Allocate memory for prevMoves array with size totalMoves

    Define function updateStrategy(char code), with parameter code
        Call s.setStrategyCode(code)

    Define function makeMove(char opponentMove)
        Define and Initialize move to s.cooperateOrDefect(opponentMove)

        Call printMoves(move), with argument move

        If numOfMoves less than or equal to totalMoves
            Call setLastMove(move), with argument move

        Otherwise
            Print “Error: Too many moves made. Move index out of bounds.”, newline

        Return move
    Define function setLastMove(char newMove), with parameter newMove
        Set prevMoves[numOfMoves] to newMove
        Increment numOfMoves by 1

    Define function printMoves(char move), with parameter move
        If move is equal to ‘c’
            Print “Player “, ID, “ Move: Cooperate”, newline
            Print newline

        Otherwise
            Print “Player “, ID, “ Move: Defect”, newline
            Print newline

    Define function increaseScore(int newScore), with parameter newScore
        Increment score by newScore

    Define function resetData()
        Set score to 0
        Set numOfMoves to 0

    Define function resetMoves()
        Return prevMoves

    Define static function resetNumOfPlayers()
        Set numOfPlayers to 0


    Define Destructor of Player class
        Deallocate memory for prevMoves array


Initialize static variable numOfPlayers of Player class to 0

Define class Game
    Define private variables: players, numOfPlayers, numOfRounds, strategy

    Define default constructor Game()
        Set players to nullptr
        Set numOfPlayers to 0
        Set numOfRounds to 0


    Define function setNumberOfRounds(int rounds) , with parameter rounds
        Set numOfRounds to rounds

        Declare and Initialize i to 0

        Repeat while i less than numOfPlayers
            Call players[i].resetData()
            Call players[i].setNumberOfMoves(rounds), with argument rounds
            Increment i
```

```
Define function displayResult()
    Declare and initialize winnerID to -1, highestScore to -1
    Declare tiedPlayers array with size numOfPlayers
    Declare and Initialize numTiedPlayers to 0
    Declare winner


    Print newline
    Print “*********************************”
    Print newline

    Declare and initialize i to 0

    Repeat while i less than numOfPlayers
        Print “Player ID: “, player ID,  newline
        Print “Name: “, player name, newline
        Print “Score: “, player score, newline
        Print newline


        If player score greater than highestScore
            Set numTiedPlayers to 0
            Set tiedPlayers[numTiedPlayers] to player ID
            Increment numTiedPlayers by 1
            Set winnerID to player ID
            Set winner to player name
            Set highestScore to player score

        Otherwise if player score is equal to highestScore

            If numTiedPlayers is equal to 0
                Create tiedPlayers array with size numOfPlayers

            Set tiedPlayers[numTiedPlayers] to player ID
            Increment numTiedPlayers by 1

        Increment i by 1


    Print “*********************************”, newline
    Print newline

    Print “------------------------------------”, newline
    Print “|              RESULT              |”, newline
    Print “------------------------------------”, newline

    If numTiedPlayers is equal to 1
        Print “|WinnerID: “, winnerID, “            |”, newline
        Print “|Name: “, winner, “                  |”, newline
        Print “|Score: “, highestScore, “           |”, newline

    Otherwise
        Print “|It’s a tie                          |”, newline
        Print “|                                    |”, newline
        Print “|Tied Players:                       |”, newline
        Declare and initialize i to 0

        Repeat while i less than numTiedPlayers
            Print “|Player “, tiedPlayers[i], “          |”, newline
            Increment i by 1

        Print “|                                        |”, newline

        Print “|Score of each player: “, highestScore, “        |”, newline

    Print “-----------------------------------”, newline

    Print newline


    Deallocate memory for tiedPlayers array


Define function addPlayers(int numPlayers), with parameter numPlayers

    If players is not equal to nullptr
        Deallocate memory for players
        Call resetID() function of generateID class
        Call resetNumOfPlayers() function of Player class


    Allocate memory for players array with size numPlayers
    Set numOfPlayers to numPlayers

    Ignore any remaining characters in the input buffer

    Declare and initialize i to 0

    Repeat while i less than numOfPlayers
        Declare playerName

        Print “Enter the name for Player”, i + 1, “: “
        Read value into playerName


        Repeat while input is invalid
            Print “Please enter a string value for the name of the player: “
            Read value into playerName

        Call players[i].setName(playerName), with argument playerName

        Print “Players added successfully!”, newline
        Print newline

        Increment i by 1
```

```
Define function addPlayersToExistingGame(int numOfPlayersToAdd), with parameter numOfPlayersToAdd
    If players is not equal to nullptr
        If (numOfPlayers + numOfPlayersToAdd) less than or equal to Max_Players
            Declare updatePlayers array of Player class with size (numOfPlayers +
            numOfPlayersToAdd)

            Declare and initialize i to 0

            Repeat while i less than numOfPlayers
                Set updatePlayers[i] to players[i]
                Call updatePlayers[i].resetData()
                Increment i

            Deallocate memory for players array

            Set players to updatedPlayers

            Increment numOfPlayers by numOfPlayersToAdd


        Otherwise
            Print "Error! Max Limit Reached! Could not add new players", newline

    Ignore any remaining characters in the input buffer

    Declare and initialize i to (numOfPlayers – numOfPlayersToAdd)

    Repeat while i less than numOfPlayers
        Declare playerName

        Print "Enter the name for Player: "
        Read value into playerName


        Repeat while input is invalid
            Print "Please enter a string value for the name of the player: "
            Read value into playerName

        Call players[i].setName(playerName), with argument playerName

        Print "Players added successfully!", newline
        Print newline

        Increment i by 1



Define function dropPlayer(int playerID), with parameter playerID
    Declare and initialize flag to false

    Declare and initialize i to 0
    Repeat while i less than numOfPlayers
        If player ID is equal to playerID
            Declare and initialize j to i
            Repeat while j less than (numOfPlayers – 1)
                Set players[j] to players[j+1]
                Incrment j by 1
            Set players[numOfPlayers – 1] to Player()
            Decrement numOfPlayers by 1

            Set flag to true
            Print “Player ”, playerID, “ successfully deleted”, newline
            Break

        Increment i by 1


    If flag is equal to false
        Print “Player “, playerID, “ not found”, newline


Define function getPlayerInfo()
    Return players


Define function play()

    Declare and initialize i to 0
    Repeat while i less than numOfRounds
        Declare and initialize j to 0
        Repeat while j less than numOfPlayers
            Declare and initialize x to 0
            Declare and initialize k to 0
            Repeat while k less than j
                Print “------------------------------------”, newline
                Print “            Round ”, i + 1, “   ”, newline
                Print “------------------------------------”, newline
                Print newline

                Declare and initialize playerOne to ‘c’
                Declare and initialize playerTwo to ‘c’

                If i is equal to 0
                    Declare and initialize defaultChar to ‘c’

                    If player[j]’s strategy is equal to ‘t’
                        Print “Player “, j+1, “, please enter your first move:
                        Enter ‘c’ for cooperate or ‘d’ for defect: ”

                        Read value into defaultChar

                        Repeat while input is invalid or (defaultChar is not
                        equal to ‘c’ and defaultChar is not equal to ‘d’)

                            Print “Please enter either ‘c’ or ‘d’ for your
                            first move”

                            Read value into defaultChar

                        Set playerOne to players[j].makeMove(defaultChar)

                    Otherwise
                        Set playerOne to players[j].makeMove(defaultChar)
                    If player[k]’s strategy is equal to ‘t’
                        Print “Player “, k+1, “, please enter your first move:
                        Enter ‘c’ for cooperate or ‘d’ for defect: ”

                        Read value into defaultChar

                        Repeat while input is invalid or (defaultChar is not
                        equal to ‘c’ and defaultChar is not equal to ‘d’)

                            Print “Please enter either ‘c’ or ‘d’ for your
                            first move”

                            Read value into defaultChar

                        Set playerTwo to players[k].makeMove(defaultChar)

                    Otherwise
                        Set playerTwo to players[k].makeMove(defaultChar)



                Otherwise
                    Declare and initialize last_Move2 to players[k].getLastMove(x)
                    Declare and initialize last_Move1 to players[j].getLastMove(x)

                    Set playerOne to players[j].makeMove(last_Move2)
                    Set playerTwo to players[k].makeMove(last_Move1)

                Increment x by 1

                If playerOne is equal to ‘c’ and playerTwo is equal to ‘c’
                    Call players[j].increaseScore(3), with argument 3
                    Call players[k].increaseScore(3), with argument 3

                Otherwise if playerOne is equal to ‘c’ and playerTwo is equal to ‘d’
                    Call players[j].increaseScore(0), with argument 0
                    Call players[k].increaseScore(5), with argument 5

                Otherwise if playerOne is equal to ‘d’ and playerTwo is equal to ‘c’
                    Call players[j].increaseScore(5), with argument 5
                    Call players[k].increaseScore(0), with argument 0

                Otherwise
                    Call players[j].increaseScore(1), with argument 1
                    Call players[k].increaseScore(1), with argument 1

                Increment k

            Increment j

        Increment i

    Call displayResult()

Define Destructor of Game class
    Deallocate memory for players array
```

```
Main Function:
    Seed random number generator

    Declare and initialize choice1 = 0, choice2 = 0
    Declare choice3
    Declare and initialize numOfPlayers to 0
    Declare and initialize numOfRounds to 0
    Declare and initialize addPlayers to 0
    Declare and initialize name to “”

    Declare and initialize playersCreated to false
    Declare and initialize setNumberOfRounds to false
    Declare and initialize setStrategy to false

    Create Game object G

    Repeat while choice1 is not equal to 5
        Print newline
        Print “------------------------------------------------------”, newline
        Print “       ITERATED PRISONER’S DILEMMA       ”, newline
        Print “------------------------------------------------------”, newline
        Print “1. Add/Drop Players”, newline
        Print “2. Set Number of Rounds”, newline
        Print “3. Choose your Game Strategy”, newline
        Print “4. Start the Game”, newline
        Print “5. Exit”, newline
        Print “Enter your choice (1-5): “
        Read value into choice1
        Print newline

        Switch based on choice1

            If choice1 is equal to 1
                Print newline
                Print “1. Add New Players to the Game (Begin/Reset Game)”, newline
                Print “2. Add Players to the Existing Game”, newline
                Print “3. Drop Players”, newline
                Print “Enter your choice (1-3): ”
                Read value into choice2

                Repeat while input is invalid or (choice2 is not equal to 1 and
                choice2 is not equal to 2 and choice2 is not equal to 3)

                    Print “Invalid Choice! Please enter either 1, 2 or 3: ”
                    Read value into choice2


                if choice2 is equal to 1
                    Print “Enter the number of players (exactly 2 players required): ”, newline
                    Read value into numOfPlayers

                    Repeat while input is invalid or numOfPlayers is not equal to
                    Max_Players)

                        Print “Error! Please enter exactly 2 players: ”
                        Read value into numOfPlayers

                    Call G.addPlayers(numOfPlayers), with argument numOfPlayers
    Set playersCreated to true


Otherwise if choice2 is equal to 2
    If not playersCreated
        Print “Error: Create the players first (Option 1)!”, newline
        Break

    If numOfPlayers is equal to Max_Players
        Print “Error! Max Limit Reached! Could not add new players”
        Break

    Declare and initialize maxReached to false

    Print “Enter the number of players to add to the game”, newline
    Read value into addPlayers

    Repeat while input is invalid or addPlayers less than 1 or
    (numOfPlayers + addPlayers) greater than 2

        If addPlayers less than 1
            Print “Please enter at least one player
            to add:”, newline
            Read value into addPlayers

        Otherwise if (numOfPlayers + addPlayers) greater than 2
            Print “Please enter a maximum of 2 players to start
            the game(enter exactly 2 players):”

            Read value into addPlayers
            Set maxReached to true
            Break


    If not maxReached
        Call G.addPlayersToExistingGame(addPlayers) with
        argument addPlayers

        Increment numOfPlayers by addPlayers

    Otherwise
        Continue

    Break



Otherwise if choice2 is equal to 3
    If not playersCreated
        Print “Error: Create the players first (Option 1)!”, newline
        Break

    Declare pID

    Print “Enter the ID of the player you want to drop:”, newline
    Read value into pID
```

```
        Repeat while input is invalid or PID < 0
            Print “Please enter a non-negative integer value for ID: ”
            Read value into pID

        Call G.dropPlayer(pID) with argument pID
        Decrement numOfPlayers by 1

    Break


If choice1 is equal to 2
    If not playersCreated
        Print “Error: Create the players first (Option 1)!”, newline
        Break

    Declare and initialize flag to true

    If not flag
        Declare and initialize i to 0
        Repeat while i less than numOfPlayers
            Deallocate memory for G.getPlayerInfo()[i].resetMoves()
            Increment i
        Set flag to true

    Print “Enter the number of rounds:”
    Read value into numOfRounds

    Repeat while input is invalid or numOfRounds less than or equal to 0
        Print “Please enter a positive value for number of rounds: ”
        Read value into numOfRounds

    Call G.setNumberOfRounds(numberOfRounds), with argument numberOfRounds

    Set setNumberOfRounds to true
    Set flag to false
    Break


If choice1 is equal to 3
    If not playersCreated or not setNumberOfRounds
        Print “Error: Create the players, and set the number of rounds first
        (Options 1, and 2)!”, newline

        Break


    Print newline
    Print “====================”, newline
    Print “           GAME STRATEGIES          ”, newline
    Print “====================”, newline
    Print “Enter r for Random”, newline
    Print “Enter c for Cooperate”, newline
    Print “Enter e for Evil”, newline
    Print “Enter t for Tit for Tat”, newline
    Print newline
```

```
                Declare and initialize i to 0

                Repeat while i less than numOfPlayers
                    Repeat while input is invalid or (choice3 is not equal to ‘r’ and
                    choice3 is not equal to ‘c’ and choice3 is not equal to ‘e’ and
                    choice 3 is not equal to ‘t’)

                        Print “Player “, i + 1, “ , Please choose your strategy: “
                        Read value into choice3

                        If choice3 is not equal to ‘r’ and choice3 is not equal to ‘c’ and
                        choice3 is not equal to ‘e’ and choice3 is not equal to ‘t’

                            Print “Please either 'r', 'c', 'e', or 't' for your
                            strategy : ”, newline


                        Clear Input Buffer

                        Ignore input until newline


                    Call G.getPlayerInfo()[i].updateStrategy(choice3) with argument choice3

                    Increment i by 1

                Set setStrategy to true
                Break



            If choice1 is equal to 4
                If not playersCreated or not setNumberOfRounds or not setStrategy
                    Print “Error: Create the players, set the number of rounds, and choose
                    strategy first (Options 1, 2, and 3)!”, newline
                    Break

                Call G.play()

                Break

            If choice1 is equal to 5
                Print “Thank you for playing!”, newline

                Print newline

                Print “End of the program!”, newline

                Break


            default case
                Print “Invalid Choice! Please enter either 1, 2, 3, 4, or 5.”

                Clear Input Buffer

                Ignore input until newline

                Continue

    Exit with code 0
```

```
Define function isInvalidInput()
      Declare and initialize invalidInput to false

      If input fails
            Assign true to invalidInput
            Print "Error! Invalid Input! Please enter a valid numeric input!", newline
            Print newline

            Clear input buffer

            Ignore input until newline

      Otherwise
            Assign false to invalidInput

      Return invalidInput
```

STEP 4: Implementation

```
/*--------------------------------------------------*/
/* Name: Aman Sunesh, NetID: as18181                */
/* Date: December 3, 2023                           */
/* Program: IteratedPrisonersDilemma.cpp            */
/* Description: This program implements the Iterated */
/* Prisoner's Dilemma (IPD) game, a classic scenario */
/* in game theory. Players, represented by  distinct */
/* strategies,  repeatedly  choose  to  cooperate or */
/* defect  based on  their opponent'  previous move. */
/* The strategies include Random (r), Cooperate (c), */
/* Evil (e), and Tit for Tat (t).The game progresses */
/* through multiple  rounds, calculating  scores for */
/* each player based on the interactions. The winner */
/* is determined by the highest cumulative score.The */
/* program provides a menu-driven interface for user */
/* interactions, allowing them to  add/drop players, */
/* set the number  of rounds, choose strategies, and */
/* initiate  the   game  simulation.  Proper  memory */
/* management  is  ensured  for  a  clean execution. */
/*--------------------------------------------------*/


#include <iostream>
#include <cstdlib>
```

```
#include <string>
#define Max_Players 2

using namespace std;

//Function prototype for input validation.
bool isInvalidInput();

//Class for generating unique IDs
class generateID
{
private:
        static int ID;

public:

        //Get the next available unique ID.
        static int getID()
        {
                ID++;
                return ID;
        }

        //Reset the static ID variable in the generateID class back to zero.
        static void resetID()
        {
                ID = 0;
        }
};

//Initializing static member ID of generateID class
int generateID::ID = 0;

//Class defining different strategies for the players
class Strategy
{
private:
        char strategyCode;


public:
        //Default Constructor
        Strategy()
        {
                strategyCode = 'r'; //Default strategy is Random
        }


        //Setter for strategyCode
        void setStrategyCode(char code)
        {
                strategyCode = code;
        }

        //Getter for strategyCode
        char getStrategyCode()
        {
                return strategyCode;
        }

        //Function to determine the move based on the strategy
        char cooperateOrDefect(char opponentLastMove)
        {
                switch (strategyCode)
                {


                        case 'r':  // Random
                        {
                                if (rand() % 2 == 0) {
                                        return 'c';
                                }
                                else {
                                        return 'd';
                                }
                        }

                        case 'c':  // Cooperate
                                return 'c';

                        case 'e':  // Evil
                                return 'd';

                        case 't':  // Tit for Tat
                                return opponentLastMove;



                        default:
                        {
                                cout << "Error: Invalid Strategy Code" << endl;
                                return '\0';
                        }
                }
        }

};


//Class representing player in the game.
class Player
{
private:
        int ID;
        string name;
        int score;
        int numOfMoves;
        int totalMoves;
        char* prevMoves;
        Strategy s;
        static int numOfPlayers;
```

```
public:
        //Default Constructor
        Player()
        {
                ID = generateID::getID();
                name = "";
                score = 0;
                numOfMoves = 0;
                totalMoves = 0;
                prevMoves = nullptr;
                numOfPlayers++;
        }


        //Copy Constructor
        Player(const Player& other)
        {
                //Copy primitive data members
                ID = other.ID;
                name = other.name;
                score = other.score;
                numOfMoves = other.numOfMoves;
                totalMoves = other.totalMoves;

                if (other.prevMoves != nullptr)
                {
                        prevMoves = other.prevMoves;
                }




                else
                {
                        prevMoves = nullptr;
                }

                //Copy strategy (assuming that Strategy has an appropriate copy
      constructor)
                s = other.s;
        }

        //Accessors
        int getID()
        {
                return ID;
        }

        string getName()
        {
                return name;
        }

        int getScore()
        {
                return score;
        }
```

```
char getStrategy()
{
        return s.getStrategyCode();
}

int getNumOfMoves() {
        return numOfMoves;
}


char getLastMove(int x)
{
        if (prevMoves != nullptr && x >= 0 && x < numOfMoves)
        {
                return prevMoves[numOfMoves - (numOfPlayers - 1) + x];
        }
        else
        {
                cout << "Error! Cannot access last move";
        }
}


//Modifiers
void setName(string playerName)
{
        name = playerName;
}




void setNumberOfMoves(int initTotalMoves)
{
        //Calculating total moves considering we do not know that the
        number of players is 2
        totalMoves = initTotalMoves * (numOfPlayers - 1);

        prevMoves = new char[totalMoves];

}


void updateStrategy(char code)
{
        s.setStrategyCode(code);
}

char makeMove(char opponentMove)
{
        char move = s.cooperateOrDefect(opponentMove);

        printMoves(move);

        if (numOfMoves <= totalMoves)
        {
```

```cpp
                setLastMove(move);
        }
        else
        {
                cout << "Error: Too many moves made. Move index out of bounds." << endl;
        }
        return move;
}


void setLastMove(char newMove)
{
        prevMoves[numOfMoves] = newMove;
        numOfMoves++;
}


//Functions
void printMoves(char move)
{
        if (move == 'c')
        {
                cout << "Player " << ID << " Move: Cooperate" << endl;
                cout << endl;
        }

        else
        {
                cout << "Player " << ID << " Move: Defect" << endl;
                cout << endl;
        }
}

void increaseScore(int newScore)
{
        score += newScore;
}

void resetData()
{
        score = 0;
        numOfMoves = 0;
}

char* resetMoves()
{
        return prevMoves;
}

static void resetNumOfPlayers()
{
        numOfPlayers = 0;
}

//Destructor
~Player()
```

```
        {
                delete[] prevMoves; // Deallocate memory for prevMoves array
        }
};


//Initializing static member numOfPlayers of Player class
int Player::numOfPlayers = 0;


//Class representing the game and its operations.
class Game
{
private:
        Player* players; //Array to store player objects
        int numOfPlayers;
        int numOfRounds;
        char strategy;

public:

        //Default Constructor
        Game()
        {
                players = nullptr;
                numOfPlayers = 0;
                numOfRounds = 0;
        }

        //Set the number of rounds and allocate memory for each player's moves
        void setNumberOfRounds(int rounds)
        {
                numOfRounds = rounds;


                //Allocate memory for each player's moves
                for (int i = 0; i < numOfPlayers; i++)
                {
                        // Reset player-specific data
                        players[i].resetData();


                        // Set the number of moves for each player based on
                        the specified rounds
                        players[i].setNumberOfMoves(rounds);
                }
        }


        //Display the result of the game
        void displayResult()
        {
                int winnerID = -1, highestScore = -1;
                int* tiedPlayers = new int[numOfPlayers];  //Dynamic array to
                store IDs of tied players
                int numTiedPlayers = 0;
                string winner;
```

```
cout << endl;
cout << "**********************************" << endl;
cout << endl;

for (int i = 0; i < numOfPlayers; i++)
{
        //Display player information
        cout << "Player ID: " << players[i].getID() << endl;
        cout << "Name: " << players[i].getName() << endl;
        cout << "Score: " << players[i].getScore() << endl;
        cout << endl;

        //Check for the winner
        if (players[i].getScore() > highestScore)
        {
                //Reset tied player count if a new highest score
                is found
                numTiedPlayers = 0;
                tiedPlayers[numTiedPlayers++] =
                players[i].getID();
                winnerID = players[i].getID();
                winner = players[i].getName();
                highestScore = players[i].getScore();
        }

        else if (players[i].getScore() == highestScore)
        {
                if (numTiedPlayers == 0)  //Allocate memory only
                if it's the first tie
                        tiedPlayers = new int[numOfPlayers];

                //Store the ID of a tied player
                tiedPlayers[numTiedPlayers++] =
                players[i].getID();
        }

}


cout << "**********************************" << endl;
cout << endl;

//Display Result
cout << "-------------------------------------" << endl;
cout << "|              RESULT              |" << endl;
cout << "-------------------------------------" << endl;

//Check if there is a single winner or a tie
if (numTiedPlayers == 1)
{
        //Display information for a single winner
        cout << "|WinnerID: " << winnerID << "
        |" << endl;

        cout << "|Name: " << winner << "
        |" << endl;
```

```
                        cout << "|Score: " << highestScore << "
                        |" << endl;
                }

                else
                {
                        //Display information for a tie
                        cout << "|It's a tie                        |" << endl;
                        cout << "|                                  |" << endl;
                        cout << "|Tied Players:                     |" << endl;

                        for (int i = 0; i < numTiedPlayers; i++)
                        {

                                cout << "|Player " << tiedPlayers[i] << "
                                |" << endl;
                        }

                        cout << "|                                  |" << endl;

                        cout << "|Score of each player: " << highestScore << "
                        |" << endl;
                }

                cout << "------------------------------------" << endl;

                cout << endl;

                //Deallocate memory for tiedPlayers array
                delete[] tiedPlayers;
        }






        //Add players to the game
        void addPlayers(int numPlayers)
        {
                //Release memory if players array is not null
                if (players != nullptr)
                {
                        delete[] players;
                        generateID::resetID();
                        Player::resetNumOfPlayers();

                }

                players = new Player[numPlayers];
                numOfPlayers = numPlayers;

                //Clear any remaining newline characters in the input buffer.
                cin.ignore();
```

```
    for (int i = 0; i < numOfPlayers; i++)
    {
        string playerName;

        cout << "Enter the name for Player " << i + 1 << ": ";

        //Use getline to read the entire line, allowing names with spaces.
        getline(cin, playerName);


        while (isInvalidInput())
        {
            cout << "Please enter a string value for the name of the player: ";

            //Use getline to read the entire line, allowing names with spaces.
            getline(cin, playerName);
        }

        players[i].setName(playerName);

        cout << "Players added successfully!" << endl;
        cout << endl;
    }
}


void addPlayersToExistingGame(int numOfPlayersToAdd)
{
    //Ensure that game already exists
    if (players != nullptr)
    {

        //Check if the total number of players after adding new players is within the allowable limit
        if ((numOfPlayers + numOfPlayersToAdd) <= Max_Players)
        {
            //Create a temporary array to hold the updated players
            Player* updatePlayers = new Player[numOfPlayers + numOfPlayersToAdd];

            //Copy existing players data to the updated array and reset their individual data
            for (int i = 0; i < numOfPlayers; i++)
            {
                updatePlayers[i] = players[i];
                updatePlayers[i].resetData();

            }

            //Deallocate the memory for players array
            delete[] players;
            //Assign the updated array to Players
            players = updatePlayers;

            //Update the total number of players
            numOfPlayers += numOfPlayersToAdd;
        }

        else
        {
            cout << "Error! Max Limit Reached! Could not add
            new players" << endl;
        }

    }

    //Ignore any remaining characters in the input buffer
    cin.ignore();

    for (int i = (numOfPlayers - numOfPlayersToAdd); i < numOfPlayers
    ; i++)
    {
        string playerName;

        cout << "Enter the name of Player: ";

        //Use getline to read the entire line, allowing names with
        spaces.
        getline(cin, playerName);


        while (isInvalidInput())
        {
            cout << "Please enter a string value for the name
            of the player: ";

            //Use getline to read the entire line, allowing
            names with spaces.
            getline(cin, playerName);
        }

        players[i].setName(playerName);

        cout << "Players added successfully!" << endl;
        cout << endl;
    }


}

//Drop a player from the game
void dropPlayer(int playerID)
{
    bool flag = false;

    for (int i = 0; i < numOfPlayers; ++i) {
        if (players[i].getID() == playerID) {
            for (int j = i; j < numOfPlayers - 1; j++)
```

```
                        {
                                players[j] = players[j + 1];
                        }

                        //Initialize last player object to zero in order
                        to remove duplicate player objects
                        players[numOfPlayers - 1] = Player();
                        numOfPlayers--;

                        flag = true;
                        cout << "Player " << playerID << " successfully
                        deleted" << endl;

                        //Dynamic memory allocated to the dropped player
                        object is deallocated at the end of main

                        break;
                }
        }

        if (flag == false)
        {
                cout << "Player " << playerID << " not found" << endl;
        }
}


//Get information about the players
Player* getPlayerInfo() {
        return players;
}




//Start the game
void play()
{

        for (int i = 0; i < numOfRounds; i++)
        {
                for (int j = 0; j < numOfPlayers; j++)
                {
                        int x = 0;

                        for (int k = 0; k < j; k++)
                        {
                                cout << "-------------------------------
                                --" << endl;
                                cout << "             Round " << i + 1
                                << "            " << endl;
                                cout << "-------------------------------
                                --" << endl;
                                cout << endl;

                                //Stores Player Moves
                                char playerOne = 'c';
```

```
                                           char playerTwo = 'c';

                                           if (i == 0)
                                           {
                                                    //For the first round, the
                                                    opponent's last move is set as
                                                    'c'.

                                                    //This does not impact the
                                                    outcome since we only need the
                                                    last move if the user chooses
                                                    "tit for tat."

                                                    //If the user chooses "tit for
                                                    tat," the program prompts the
                                                    user for their first move.

                                                    //Otherwise, the first move is
                                                    determined based on the player's
                                                    selected strategy.

                                                    char defaultChar = 'c';


                                                    if (players[j].getStrategy() ==
                                                    't')

                                                    {
                                                            cout << "Player " << j +
1 << ", please enter your first move: Enter 'c' for cooperate or 'd' for defect: ";
                                                            cin >> defaultChar;



                                                            while (isInvalidInput()
|| (defaultChar != 'c' && defaultChar != 'd'))
                                                            {
                                                                    cout << "Please
enter either 'c' or 'd' for your first move: ";
                                                                    cin >>
defaultChar;
                                                            }



                                                            playerOne =
players[j].makeMove(defaultChar);
                                                    }

                                                    else
                                                    {
                                                            playerOne =
players[j].makeMove(defaultChar); //If selected strategy is not tit for tat
                                                    }



                                                    if (players[k].getStrategy() ==
't')
```

```
                                                {
                                                        cout << "Player " << k +
1 << ", please enter your first move: Enter 'c' for cooperate or 'd' for defect: ";
                                                        cin >> defaultChar;

                                                        while (isInvalidInput()
|| (defaultChar != 'c' && defaultChar != 'd'))
                                                        {
                                                                cout << "Please
enter either 'c' or 'd' for your first move: ";
                                                                cin >>
defaultChar;
                                                        }

                                                        playerTwo =
players[k].makeMove(defaultChar);
                                                }

                                                else
                                                {
                                                        playerTwo =
players[k].makeMove(defaultChar); //If selected strategy is not tit for tat
                                                }

                                        }


                                        else
                                        {
                                                int last_Move2 =
players[k].getLastMove(x);
                                                int last_Move1 =
players[j].getLastMove(x);

                                                playerOne =
                                                players[j].makeMove(last_Move2);
                                                playerTwo =
                                                players[k].makeMove(last_Move1);
                                        }

                                        x = x + 1;

                                        //Set Score based on moves
                                        if (playerOne == 'c' && playerTwo == 'c')
                                        {
                                                players[j].increaseScore(3);
                                                players[k].increaseScore(3);
                                        }
                                        else if (playerOne == 'c' && playerTwo == 'd')
                                        {
                                                players[j].increaseScore(0);
                                                players[k].increaseScore(5);
                                        }
                                        else if (playerOne == 'd' && playerTwo == 'c')
                                        {
                                                players[j].increaseScore(5);
                                                players[k].increaseScore(0);
```

```cpp
                                        }
                                        else // move1 == 'd' && move2 == 'd'
                                        {
                                                players[j].increaseScore(1);
                                                players[k].increaseScore(1);
                                        }

                                }
                        }


                }
                displayResult(); // Display the final result of the game
        }

        //Destructor
        ~Game()
        {
                delete[] players; //Deallocate memory for players array
        }

};


//Main function
int main()
{
        //Seed the random number generator for generating random moves in the game
        srand(time(NULL));

        //Declare and Initialize Variables
        int choice1 = 0, choice2 = 0;
        char choice3;
        int numOfPlayers = 0;
        int numOfRounds = 0;
        int addPlayers = 0;
        string name = "";

        //Flags to track which steps have been completed
        bool playersCreated = false;
        bool setNumberOfRounds = false;
        bool setStrategy = false;

        //Create a Game Object
        Game G;


        //User Interface
        do
        {
                //Display menu options
                cout << endl;
                cout << "----------------------------------------------" << endl;
                cout << "          ITERATED PRISONER'S DILEMMA         " << endl;
                cout << "----------------------------------------------" << endl;
                cout << "1. Add/Drop Players" << endl;
                cout << "2. Set Number of Rounds" << endl;
                cout << "3. Choose your Game Strategy" << endl;
                cout << "4. Start the Game" << endl;
                cout << "5. Exit" << endl;
                cout << "Enter your choice (1-5): ";
                cin >> choice1;
                cout << endl;


                switch (choice1)
                {

                        case 1:
                        {
                                //Option 1: Add/Drop Players

                                cout << endl;
                                cout << "1. Add New Players to the Game
                                (Begin/Reset Game)" << endl;
                                cout << "2. Add Players to the Existing Game" <<
                                endl;
                                cout << "3. Drop Players" << endl;
                                cout << "Enter your choice (1-3): ";
                                cin >> choice2;

                                while (isInvalidInput() || (choice2 != 1 &&
                                choice2 != 2 && choice2 != 3))
                                {
                                        cout << "Invalid Choice! Please enter
                                        either 1,2 or 3: ";
                                        cin >> choice2;
                                }


                                if (choice2 == 1)
                                {
                                        //Add Players
                                        cout << "Enter the number of players
                                        (exactly 2 players required): " << endl;
                                        cin >> numOfPlayers;

                                        while (isInvalidInput() || numOfPlayers
                                        != Max_Players)
                                        {
                                                cout << "Error! Please enter
                                                exactly 2 players: ";
                                                cin >> numOfPlayers;
                                        }

                                        //Number of players can be increased by
                                        modifying the condition of the while loop

                                        //Add new players to the game
                                        G.addPlayers(numOfPlayers);
                                        playersCreated = true;
                                }
```

```
                                else if (choice2 == 2)
                                {
                                        if (!playersCreated)
                                        {
                                                cout << "Error: Create the
                                                players first (Option 1)!" <<
                                                endl;
                                                break;
                                        }


                                        if (numOfPlayers == Max_Players)
                                        {
                                                cout << "Error! Max Limit
                                                Reached! Could not add new
                                                players";
                                                break;
                                        }

                                        bool maxReached = false;

                                        cout << "Enter the number of players to
                                        add to the game" << endl;
                                        cin >> addPlayers;

                                        while (isInvalidInput() || addPlayers < 1
                                        || (numOfPlayers + addPlayers) > 2)
                                        {
                                                if (addPlayers < 1)
                                                {
                                                        cout << "Please enter at
                                                        least one player to add:
                                                        " << endl;
                                                        cin >> addPlayers;
                                                }

                                                else if ((numOfPlayers +
                                                addPlayers) > 2)
                                                {
                                                        //For the context of
this assignment, the number of players has to be 2
                                                        cout << "Please enter a
maximum of 2 players to start the game(enter exactly 2 players): ";

                                                        cin >> addPlayers;
                                                        maxReached = true;
                                                        break;

                                                }
                                        }

                                        if (!maxReached) //Executes if and only
                                        if number of players is within the
                                        allowed limit
                                        {
                                                //Add additional players to the
                                                game
```

```
                 G.addPlayersToExistingGame
                (addPlayers);

                numOfPlayers += addPlayers;
            }

            else
            {
                continue;
            }

            break;
        }


        else if (choice2 == 3)
        {
            //Drop Players
            if (!playersCreated)
            {
                cout << "Error: Create the
                players first (Option 1)!" <<
                endl;
                break;
            }

            int pID;

            cout << "Enter the ID of the player you
            want to drop: " << endl;
            cin >> pID;

            while (isInvalidInput() || pID < 0)
            {
                cout << "Please enter a non-
                negative integer value for ID:
                ";
                cin >> pID;
            }

            //Drop the specified player
            G.dropPlayer(pID);
            numOfPlayers--;
        }

        break;
    }

    case 2:
    {
        //Option 2: Set Number of Rounds

        //Ensure players are created before setting the
       number of rounds
        if (!playersCreated)
        {
                cout << "Error: Create the players first
                (Option 1)!" << endl;
                break;
        }

        bool flag = true;


        if (!flag)
        {
                for (int i = 0; i < numOfPlayers; i++)
                {
                        //Deallocate memory for the
player's previous moves
                        delete[] G.getPlayerInfo()
                        [i].resetMoves();

                }

                flag = true;

        }

        cout << "Enter the number of rounds: ";
        cin >> numOfRounds;

        while (isInvalidInput() || numOfRounds <= 0)
        {
                cout << "Please enter a positive value
for number of rounds: ";
                cin >> numOfRounds;
        }

        //Set the number of rounds in the game
        G.setNumberOfRounds(numOfRounds);

        setNumberOfRounds = true;
        flag = false;
        break;
}


case 3:
{
        //Option 3: Choose your game strategy

        //Ensure players and the number of rounds are set
        before choosing strategies.




        if (!playersCreated || !setNumberOfRounds)
        {
```

```
                                          cout << "Error: Create the players, and
                                          set the number of rounds first (Options
                                          1, and 2)!" << endl;
                                          break;
                                  }

                                  //Display available game strategies
                                  cout << endl;
                                  cout << "==================================" << endl;
                                  cout << "          GAME STRATEGIES         " << endl;
                                  cout << "==================================" << endl;
                                  cout << "Enter r for Random" << endl;
                                  cout << "Enter c for Cooperate" << endl;
                                  cout << "Enter e for Evil" << endl;
                                  cout << "Enter t for Tit for Tat" << endl;
                                  cout << endl;


                                  //Prompt each player to choose a strategy
                                  for (int i = 0; i < numOfPlayers; i++)
                                  {
                                          do
                                          {
                                                  cout << "Player " << i + 1 << "
, Please choose your strategy: ";
                                                  cin >> choice3;

                                                  if (choice3 != 'r' && choice3 !=
'c' && choice3 != 'e' && choice3 != 't')
                                                  {
                                                          cout << "Please either
'r', 'c', 'e', or 't' for your strategy : " << endl;
                                                  }

                                                  cin.clear();

                           cin.ignore(numeric_limits<streamsize>::max(), '\n');

                                          } while (isInvalidInput() || (choice3 !=
'r' && choice3 != 'c' && choice3 != 'e' && choice3 != 't'));

                                          //Set the chosen strategy for the player

        G.getPlayerInfo()[i].updateStrategy(choice3);

                                  }

                                  setStrategy = true;
                                  break;
                         }




                         case 4:
                         {
```

```
                                  //Option 4: Start the game

                                  //Ensure players, the number of rounds, and
                                 strategies are set before starting the game.
                                  if (!playersCreated || !setNumberOfRounds ||
                                 !setStrategy)
                                  {
                                          cout << "Error: Create the players, set
                                          the number of rounds, and choose strategy
                                          first (Options 1, 2, and 3)!" << endl;
                                          break;
                                  }

                                  //Start the game
                                  G.play();

                                  break;
                         }


                         case 5:
                         {
                                  //Option 5: Exit

                                  cout << "Thank you for playing!" << endl;

                                  cout << endl;

                                  cout << "End of the program!" << endl;
                                  break;
                         }


                         default:
                         {
                                  //Handle invalid choices
                                  cout << "Invalid Choice! Please enter either 1,
                                 2, 3, 4, or 5. ";

                                  //Clears the input buffer
                                  cin.clear();

                                  //To ensure that error message is printed only
                                 once
                                  cin.ignore(numeric_limits<streamsize>::max(),
       '\n');

                                  continue;
                         }
                 }

        } while (choice1 != 5);

        return 0;
}

//Function to check if the input is valid
bool isInvalidInput()
```

```
{
        bool invalidInput = false;

        //Check for invalid input
        if (cin.fail())
        {
                invalidInput = true;
                cout << "Error! Invalid Input! Please enter a valid numeric
                input!" << endl;
                cout << endl;

                //Clears the input buffer
                cin.clear();

                //To ensure that error message is printed only once
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }

        else
        {
                invalidInput = false;
        }

        return invalidInput;
}
```

STEP 5: Test and Verification (and Debugging)

Test Cases 1.1.1-1.1.2:
The output for each of these test cases from the program is in agreement with the test case expected output. This can be seen from the Sample Output section that follows this section.

Therefore, we can conclude that the program is functioning correctly.

Test Case 1.1.3: ***User Selects Option 2 (After having added 2 players)***

Error! Max Limit Reached! Could not add new players

,which is in agreement with the test case expected output.

Therefore, we can conclude that the program is functioning correctly.

Test Case 1.1.4:
The output for this test cases from the program is in agreement with the test case expected output. This can be seen from the Sample Output section that follows this section.

Therefore, we can conclude that the program is functioning correctly.

Test Case 1.2.1: User has not Selected Option1 before Option 2

```
Error: Create the players first (Option 1)!
```

,which is in agreement with the test case expected output.

Therefore, we can conclude that the program is functioning correctly.

Test Case 1.2.2: User has Selected Option1 before Option 2

```
Enter the number of rounds: 5
```

,which is in agreement with the test case expected output.

Therefore, we can conclude that the program is functioning correctly.

Test Cases 1.3.1-1.3.3:
The output for each of these test cases from the program is in agreement with the test case expected output. This can be seen from the Sample Output section that follows this section.

Therefore, we can conclude that the program is functioning correctly.

Test Case 1.4.1: Player 1 Strategy: Cooperate, Player 2 Strategy: Evil

```
**********************************

Player ID: 1
Name: A
Score: 0

Player ID: 2
Name: B
Score: 25

**********************************


-----------------------------------
|            RESULT               |
-----------------------------------
|WinnerID: 2                      |
|Name: B                          |
|Score: 25                        |
-----------------------------------
```

,which is in agreement with the test case expected output.

Therefore, we can conclude that the program is functioning correctly.

Test Case 1.4.2: Player 1 Strategy: Cooperate, Player 2 Strategy: Tit for tat

```
*********************************

          Player ID: 1
            Name: A
           Score: 18

          Player ID: 2
            Name: B
           Score: 23

*********************************
```

```
------------------------------------
|              RESULT              |
------------------------------------
|WinnerID: 2                       |
|Name: B                           |
|Score: 23                         |
------------------------------------
```

,which is in agreement with the test case expected output.

Therefore, we can conclude that the program is functioning correctly.

Test Case 1.4.3: Game 1: Player 1 Strategy: Tit for Tat, Player 2 Strategy: Tit for tat
Game 2: Player 1 Strategy: Tit for Tat, Player 2 Strategy: Random

Game 1:

```
*********************************

          Player ID: 1
         Name: Mr John
           Score: 25

          Player ID: 2
          Name: Aman
           Score: 25

*********************************
```

```
-----------------------------------
|            RESULT               |
-----------------------------------
|It's a tie                       |
|                                 |
|Tied Players:                    |
|Player 1                         |
|Player 2                         |
|                                 |
|Score of each player: 25         |
-----------------------------------
```

Game 2:

```
*********************************

         Player ID: 1
         Name: Alex
         Score: 23

         Player ID: 2
         Name: Mary
         Score: 3

*********************************
```

```
-----------------------------------
|            RESULT               |
-----------------------------------
|WinnerID: 1                      |
|Name: Alex                       |
|Score: 23                        |
-----------------------------------
```

,which is in agreement with the test case expected output.

Therefore, we can conclude that the program is functioning correctly.

Test Case 1.5 : User Selects Option 5

Thank you for playing!

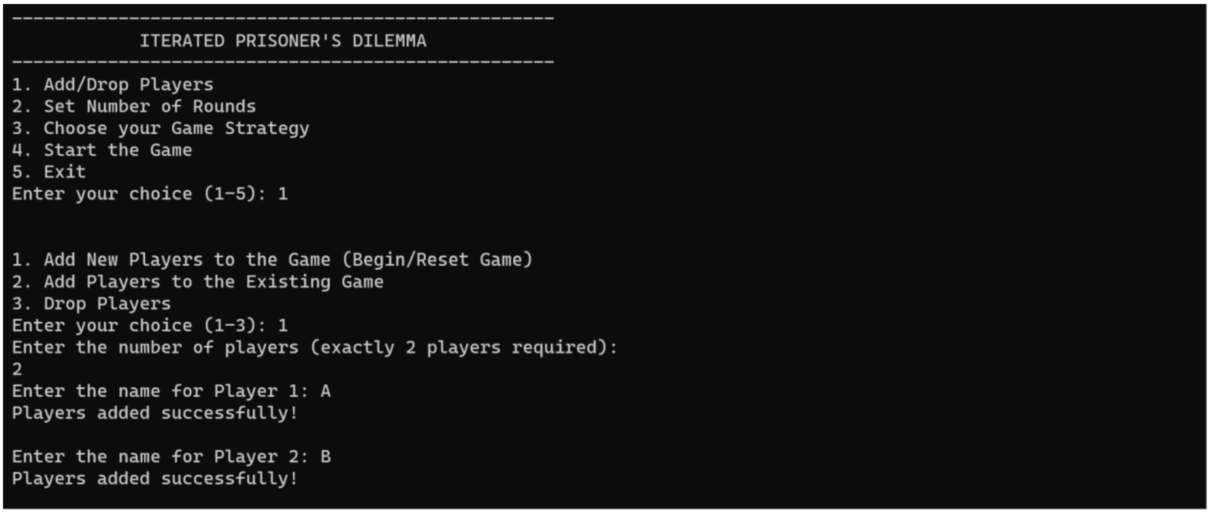
End of the program!

,which is in agreement with the test case expected output.

Therefore, we can conclude that the program is functioning correctly.

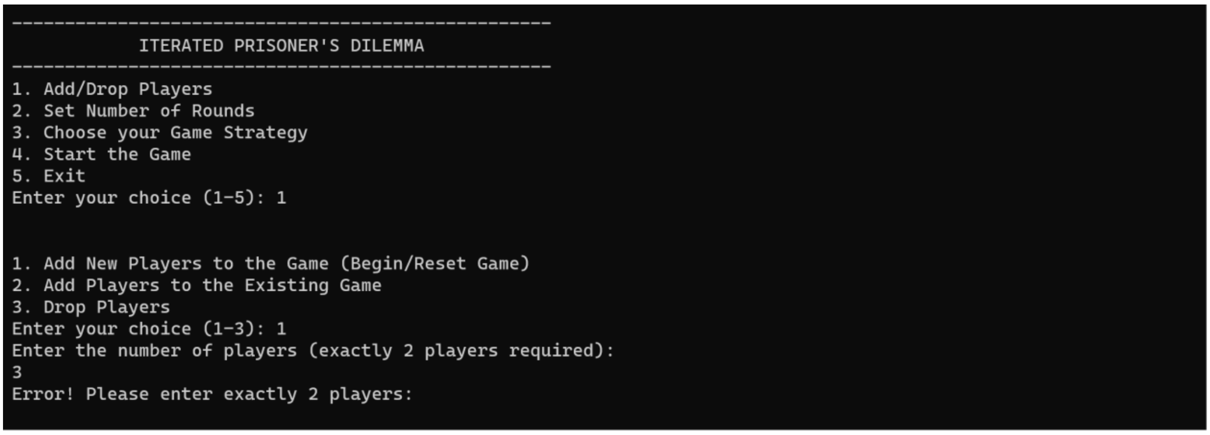
Sample Outputs –

1.1.1)



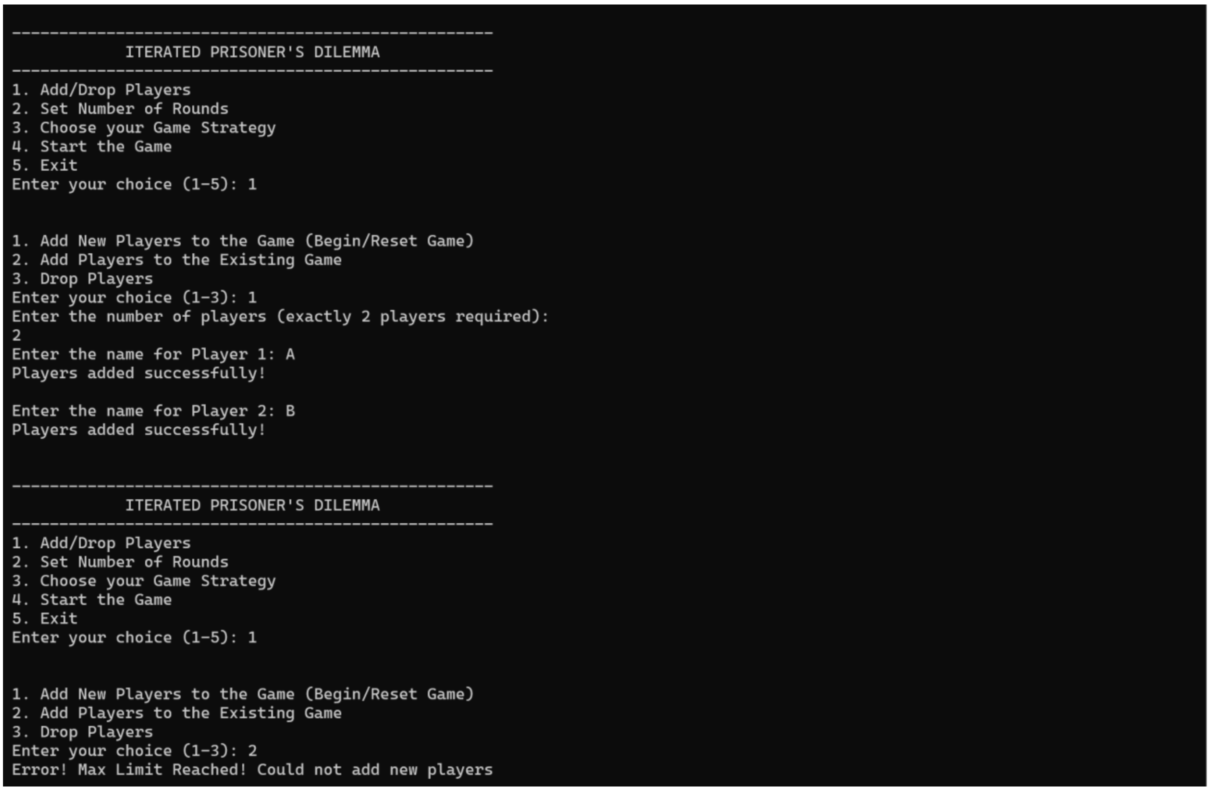
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 1


1. Add New Players to the Game (Begin/Reset Game)
2. Add Players to the Existing Game
3. Drop Players
Enter your choice (1-3): 1
Enter the number of players (exactly 2 players required):
2
Enter the name for Player 1: A
Players added successfully!

Enter the name for Player 2: B
Players added successfully!
```

1.1.2)



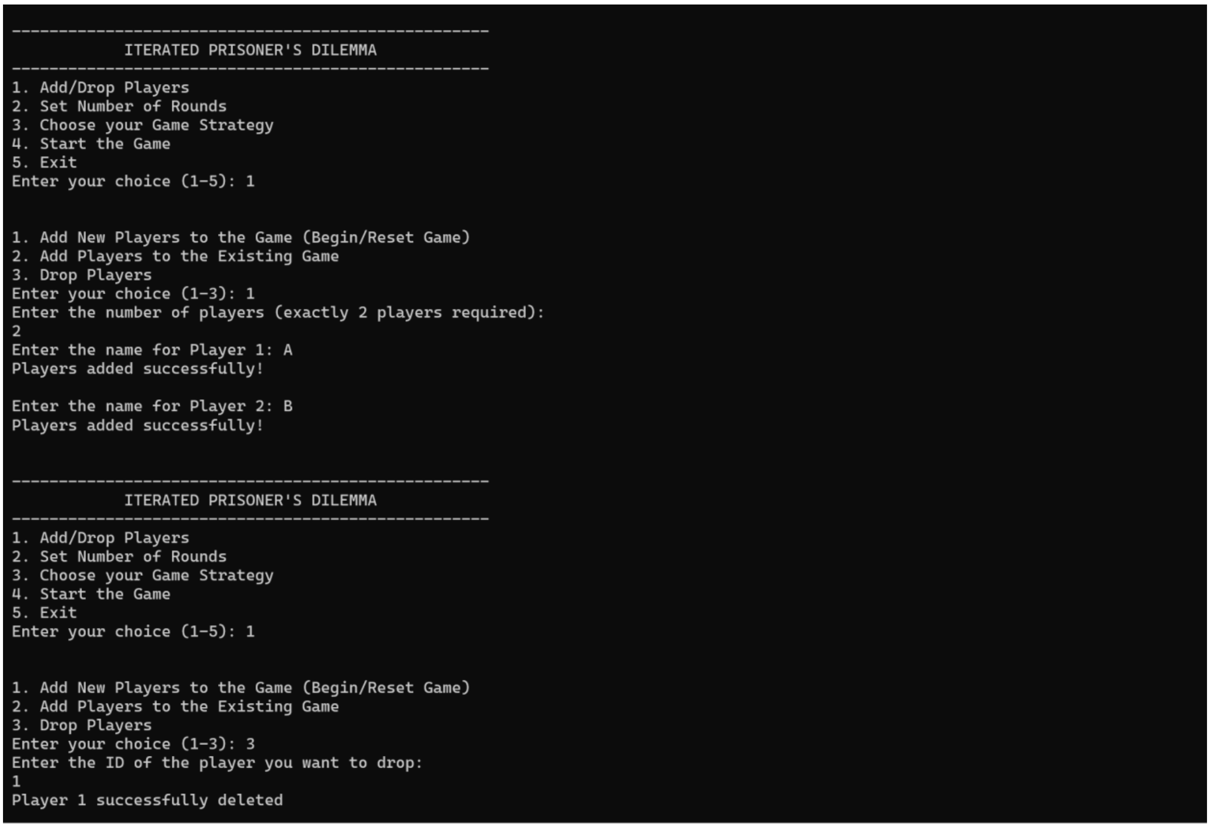
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 1


1. Add New Players to the Game (Begin/Reset Game)
2. Add Players to the Existing Game
3. Drop Players
Enter your choice (1-3): 1
Enter the number of players (exactly 2 players required):
3
Error! Please enter exactly 2 players:
```

1.1.3)

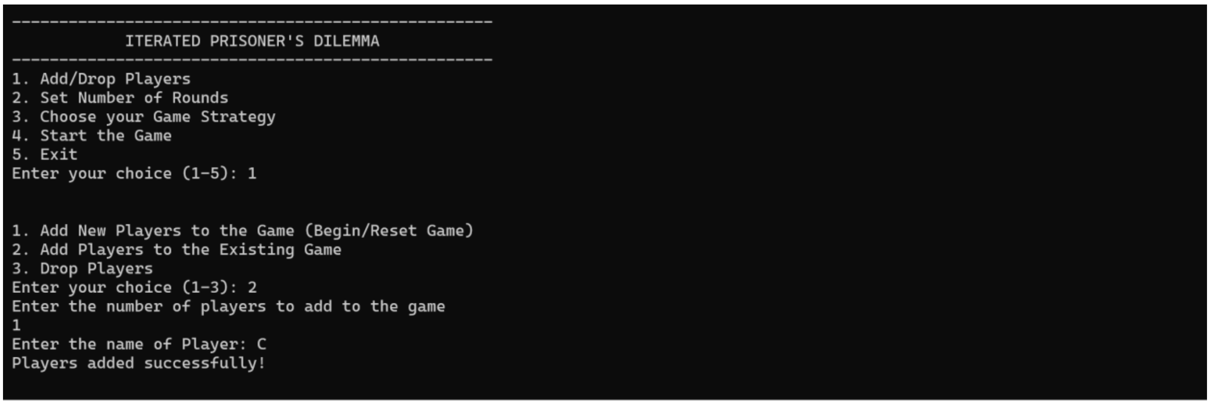


```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 1


1. Add New Players to the Game (Begin/Reset Game)
2. Add Players to the Existing Game
3. Drop Players
Enter your choice (1-3): 1
Enter the number of players (exactly 2 players required):
2
Enter the name for Player 1: A
Players added successfully!

Enter the name for Player 2: B
Players added successfully!


---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 1


1. Add New Players to the Game (Begin/Reset Game)
2. Add Players to the Existing Game
3. Drop Players
Enter your choice (1-3): 2
Error! Max Limit Reached! Could not add new players
```

1.1.4)

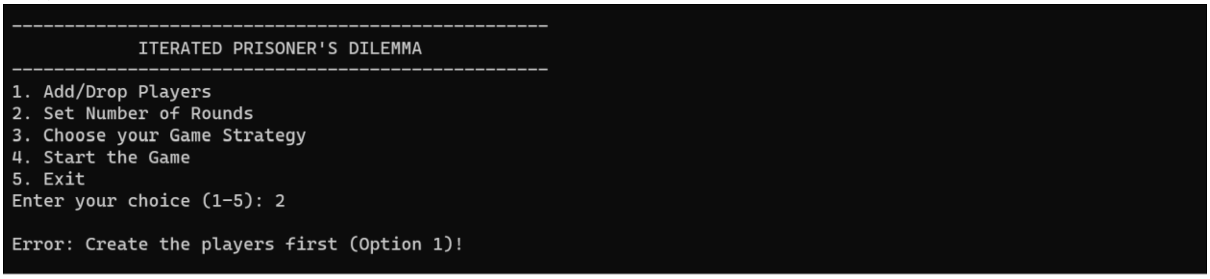


```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 1


1. Add New Players to the Game (Begin/Reset Game)
2. Add Players to the Existing Game
3. Drop Players
Enter your choice (1-3): 1
Enter the number of players (exactly 2 players required):
2
Enter the name for Player 1: A
Players added successfully!

Enter the name for Player 2: B
Players added successfully!


---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 1


1. Add New Players to the Game (Begin/Reset Game)
2. Add Players to the Existing Game
3. Drop Players
Enter your choice (1-3): 3
Enter the ID of the player you want to drop:
1
Player 1 successfully deleted
```



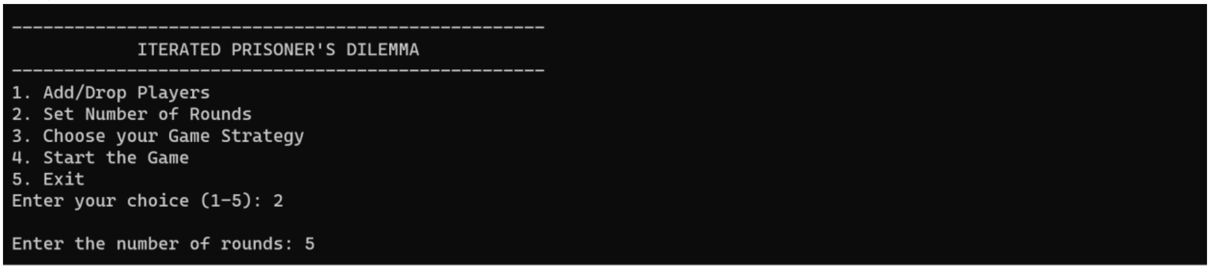
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 1


1. Add New Players to the Game (Begin/Reset Game)
2. Add Players to the Existing Game
3. Drop Players
Enter your choice (1-3): 2
Enter the number of players to add to the game
1
Enter the name of Player: C
Players added successfully!
```

1.2.1)



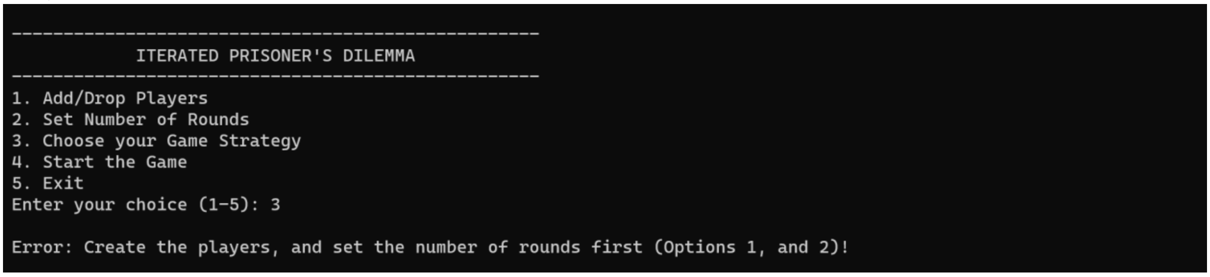
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 2

Error: Create the players first (Option 1)!
```

1.2.2)



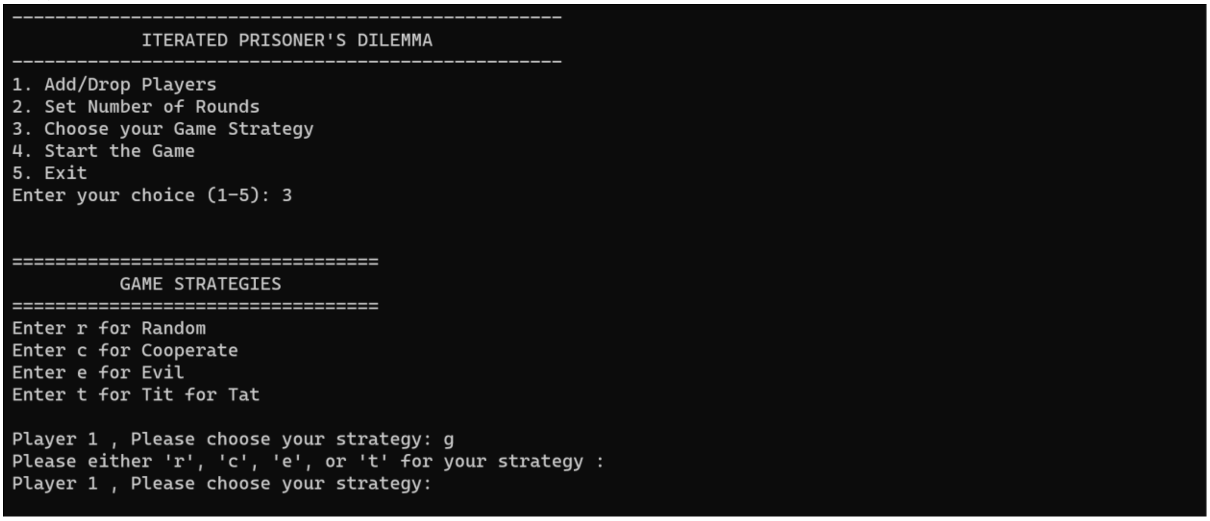
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 2

Enter the number of rounds: 5
```

1.3.1)



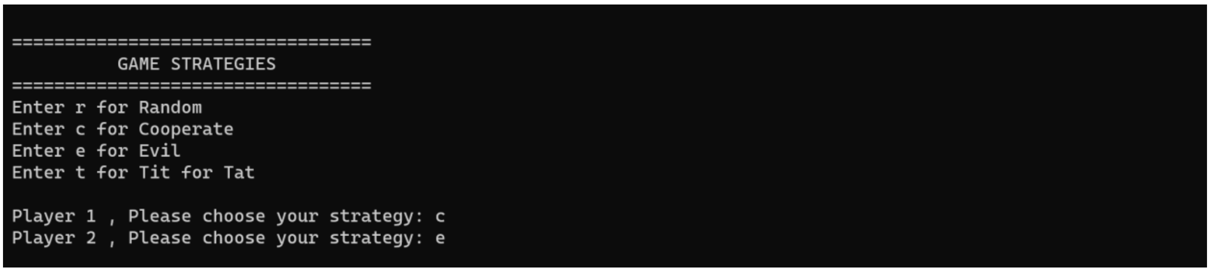
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 3

Error: Create the players, and set the number of rounds first (Options 1, and 2)!
```

1.3.2)



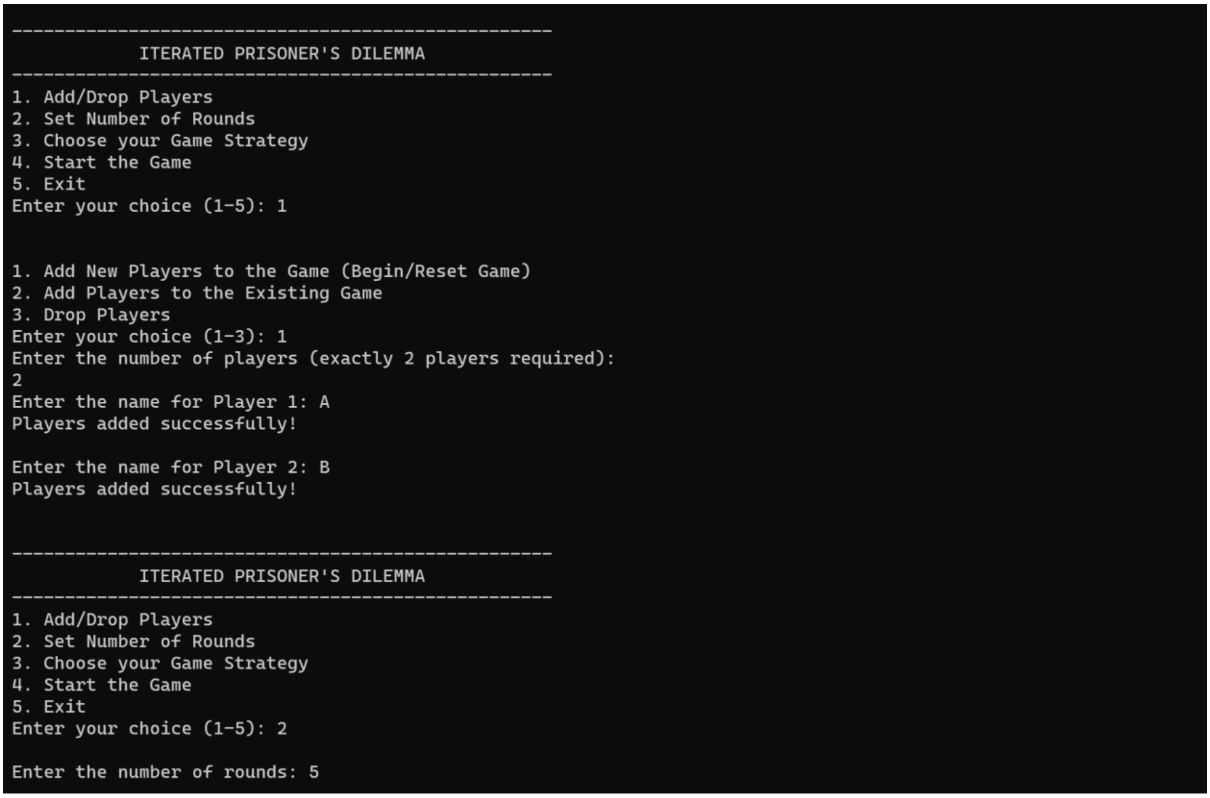
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 3


==================================
          GAME STRATEGIES
==================================
Enter r for Random
Enter c for Cooperate
Enter e for Evil
Enter t for Tit for Tat

Player 1 , Please choose your strategy: g
Please either 'r', 'c', 'e', or 't' for your strategy :
Player 1 , Please choose your strategy:
```

1.3.3)

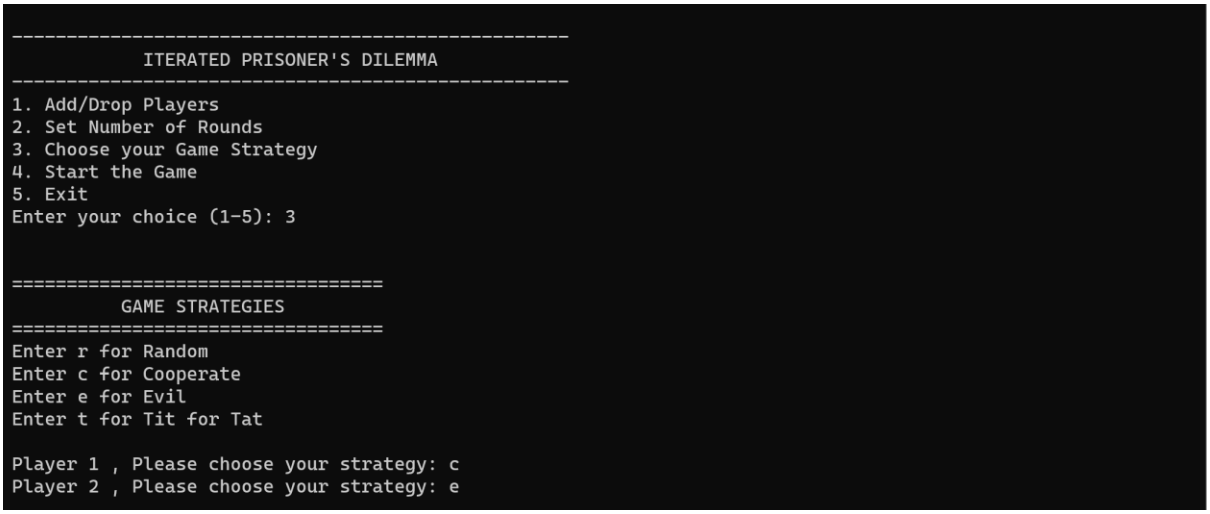


```
==================================
          GAME STRATEGIES
==================================
Enter r for Random
Enter c for Cooperate
Enter e for Evil
Enter t for Tit for Tat

Player 1 , Please choose your strategy: c
Player 2 , Please choose your strategy: e
```

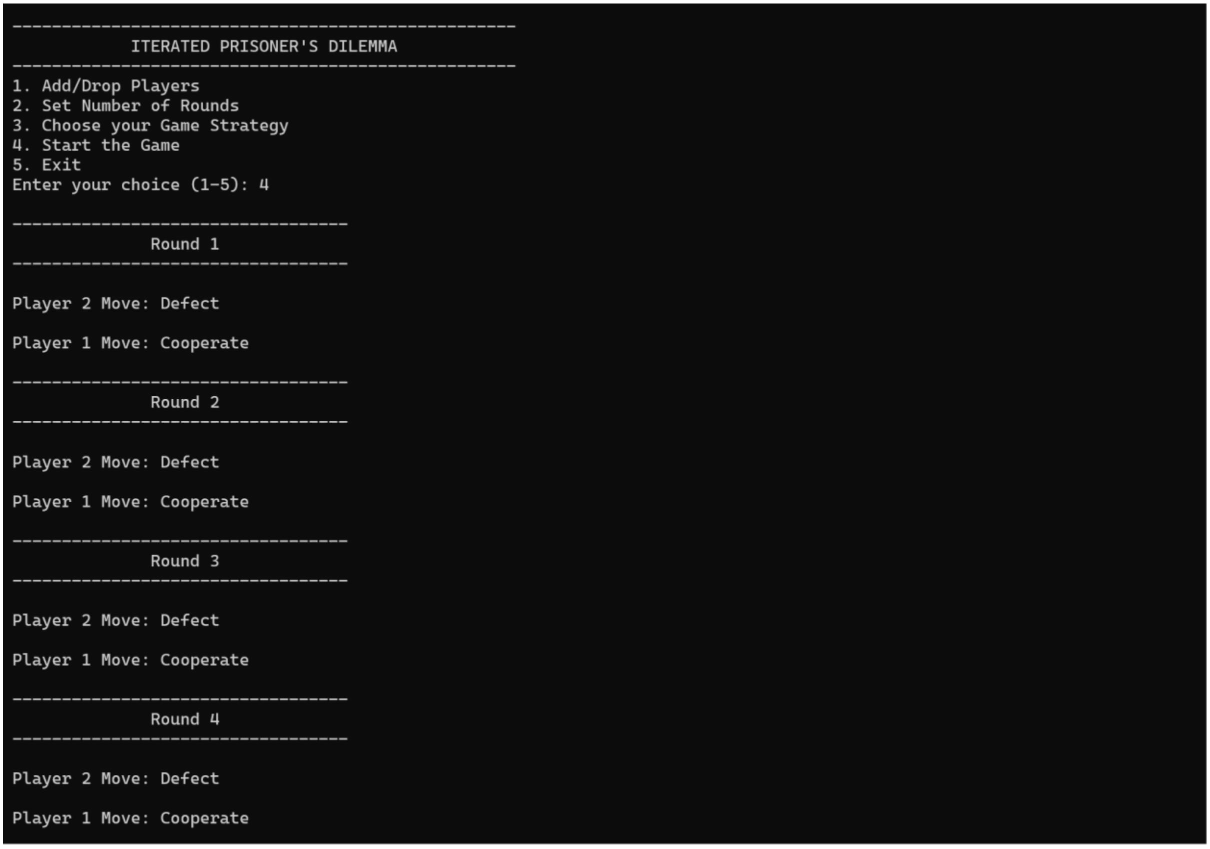
1.4.1)



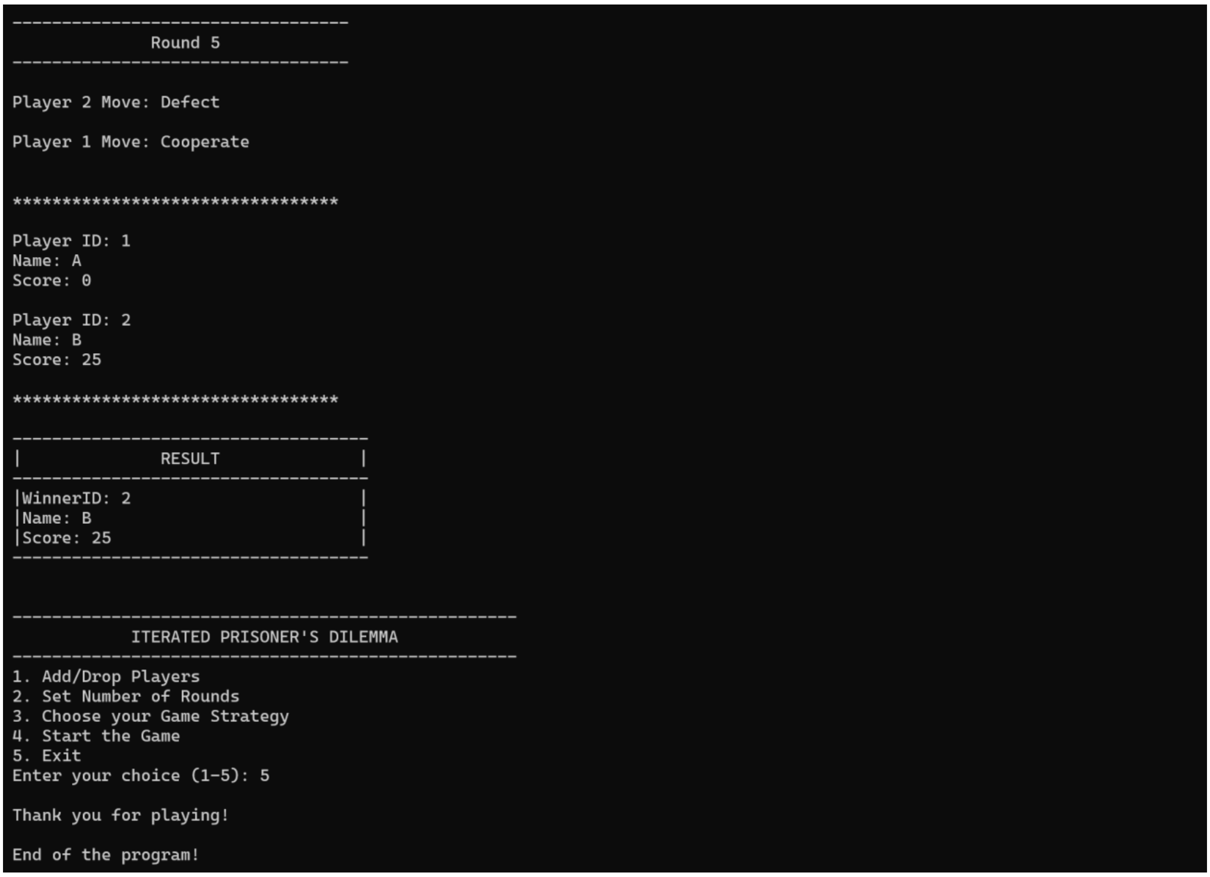
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 1


1. Add New Players to the Game (Begin/Reset Game)
2. Add Players to the Existing Game
3. Drop Players
Enter your choice (1-3): 1
Enter the number of players (exactly 2 players required):
2
Enter the name for Player 1: A
Players added successfully!

Enter the name for Player 2: B
Players added successfully!


---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 2

Enter the number of rounds: 5
```



```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 3


==================================
          GAME STRATEGIES
==================================
Enter r for Random
Enter c for Cooperate
Enter e for Evil
Enter t for Tit for Tat

Player 1 , Please choose your strategy: c
Player 2 , Please choose your strategy: e
```

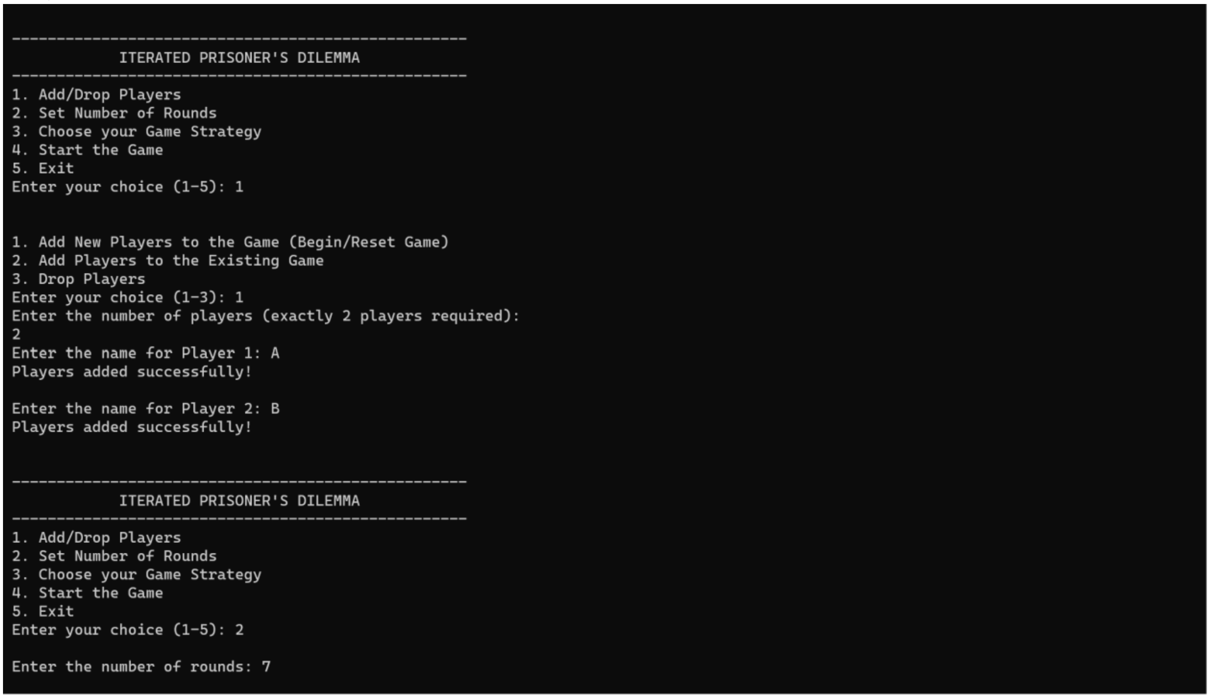


```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 4

----------------------------------
              Round 1
----------------------------------

Player 2 Move: Defect

Player 1 Move: Cooperate

----------------------------------
              Round 2
----------------------------------

Player 2 Move: Defect

Player 1 Move: Cooperate

----------------------------------
              Round 3
----------------------------------

Player 2 Move: Defect

Player 1 Move: Cooperate

----------------------------------
              Round 4
----------------------------------

Player 2 Move: Defect

Player 1 Move: Cooperate
```

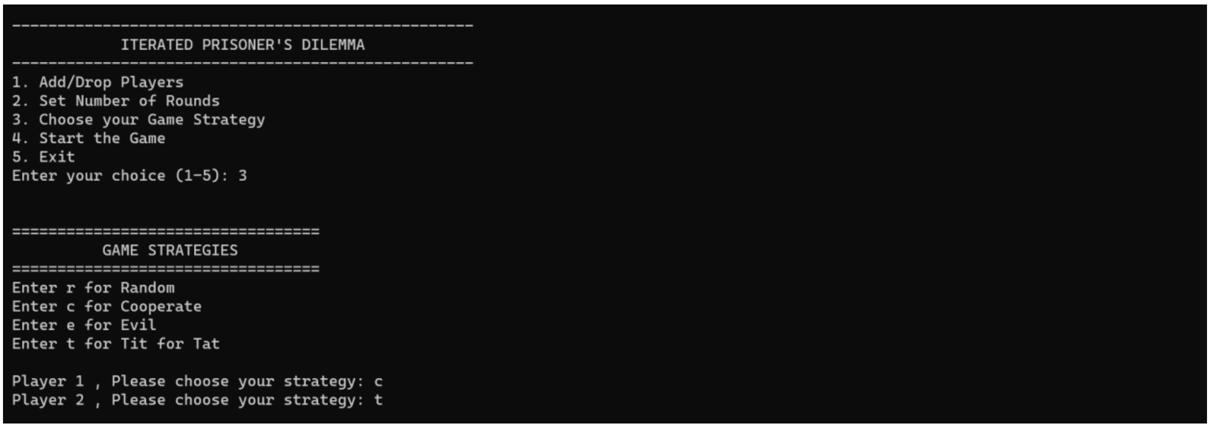


```
----------------------------------
              Round 5
----------------------------------

Player 2 Move: Defect

Player 1 Move: Cooperate


*********************************

Player ID: 1
Name: A
Score: 0

Player ID: 2
Name: B
Score: 25

*********************************

------------------------------------
|              RESULT              |
------------------------------------
|WinnerID: 2                       |
|Name: B                           |
|Score: 25                         |
------------------------------------


---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 5

Thank you for playing!

End of the program!
```

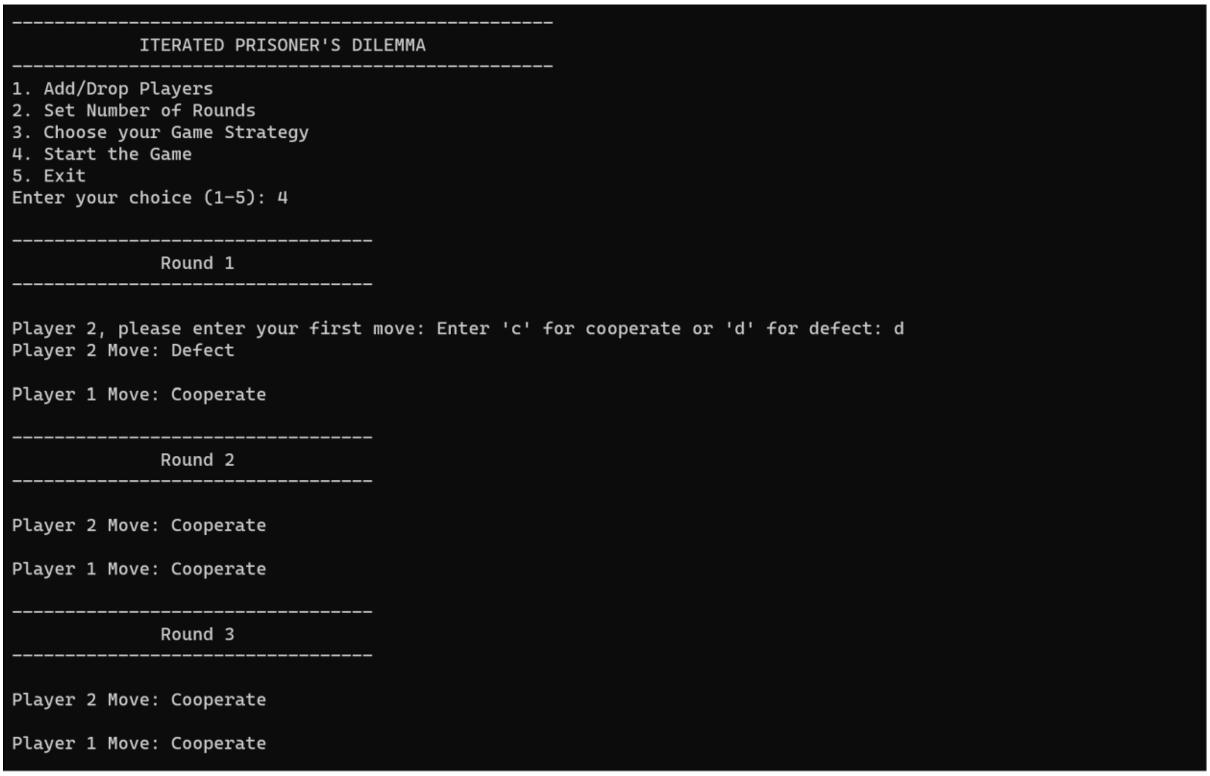
1.4.2)



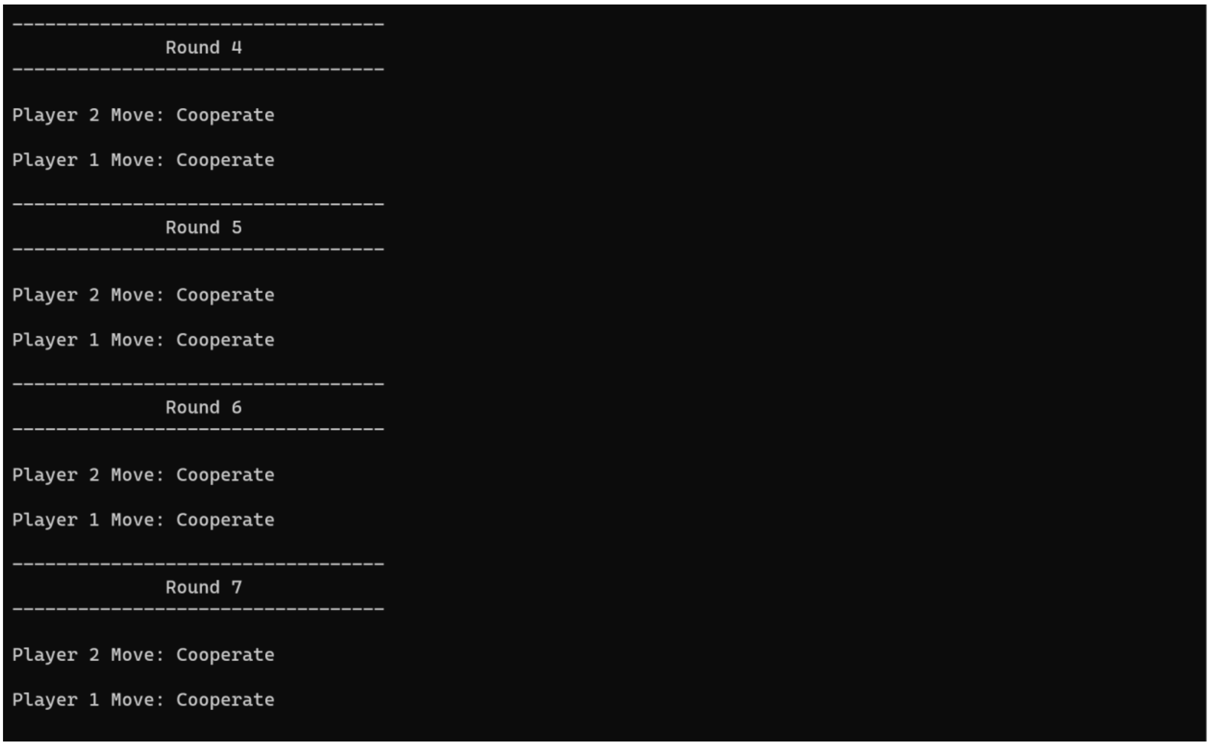
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 1


1. Add New Players to the Game (Begin/Reset Game)
2. Add Players to the Existing Game
3. Drop Players
Enter your choice (1-3): 1
Enter the number of players (exactly 2 players required):
2
Enter the name for Player 1: A
Players added successfully!

Enter the name for Player 2: B
Players added successfully!


---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 2

Enter the number of rounds: 7
```



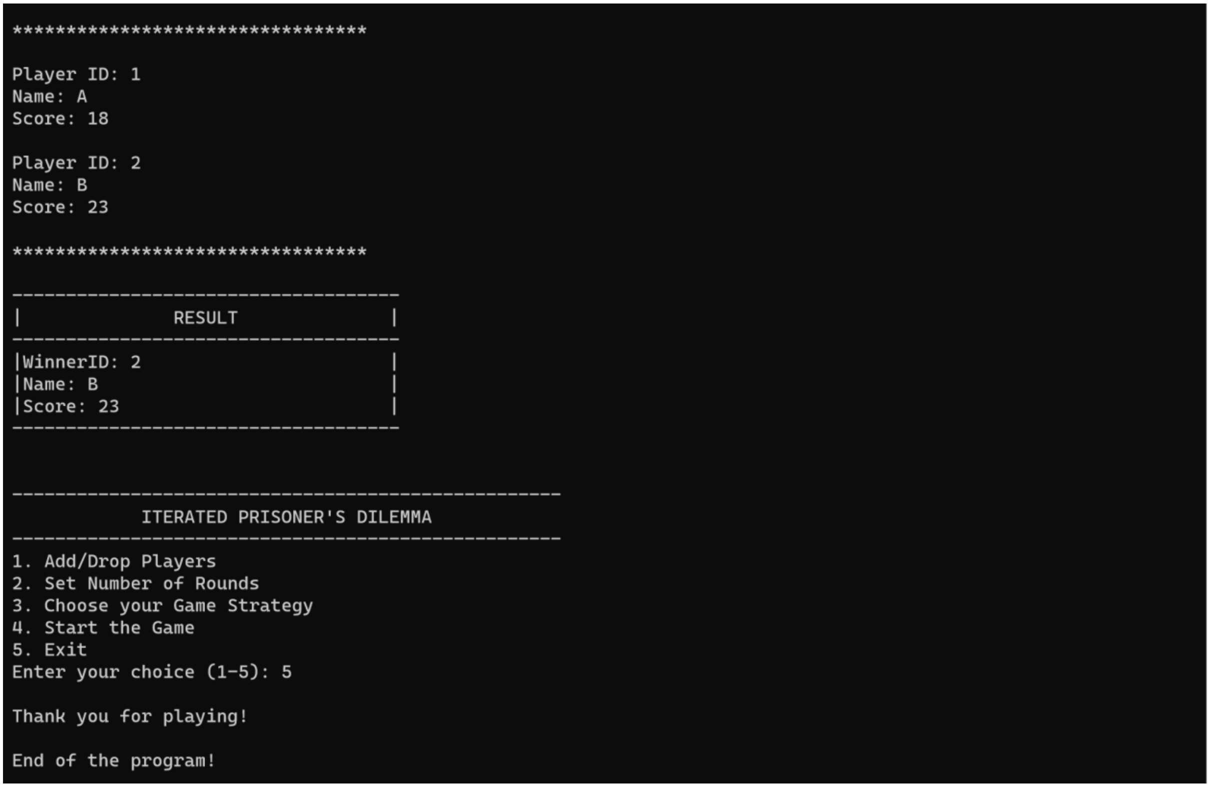
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 3


==================================
          GAME STRATEGIES
==================================
Enter r for Random
Enter c for Cooperate
Enter e for Evil
Enter t for Tit for Tat

Player 1 , Please choose your strategy: c
Player 2 , Please choose your strategy: t
```



```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 4

----------------------------------
              Round 1
----------------------------------

Player 2, please enter your first move: Enter 'c' for cooperate or 'd' for defect: d
Player 2 Move: Defect

Player 1 Move: Cooperate

----------------------------------
              Round 2
----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Cooperate

----------------------------------
              Round 3
----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Cooperate
```

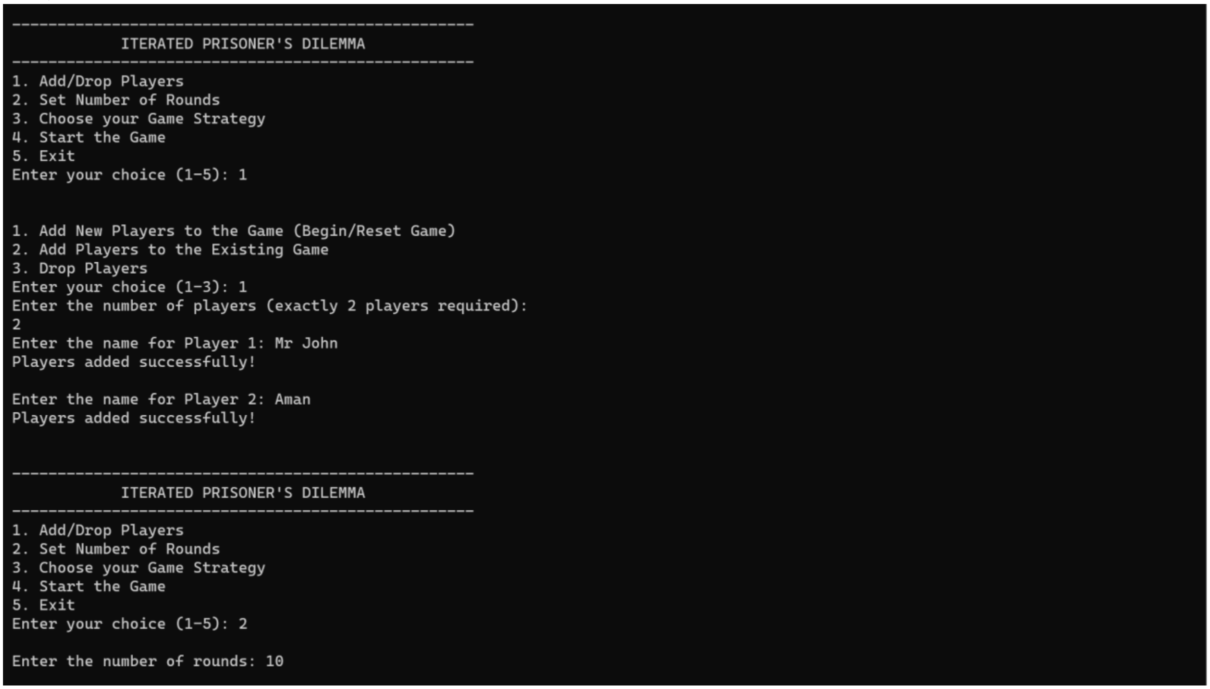


```
-----------------------------------
              Round 4
-----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Cooperate

-----------------------------------
              Round 5
-----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Cooperate

-----------------------------------
              Round 6
-----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Cooperate

-----------------------------------
              Round 7
-----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Cooperate
```

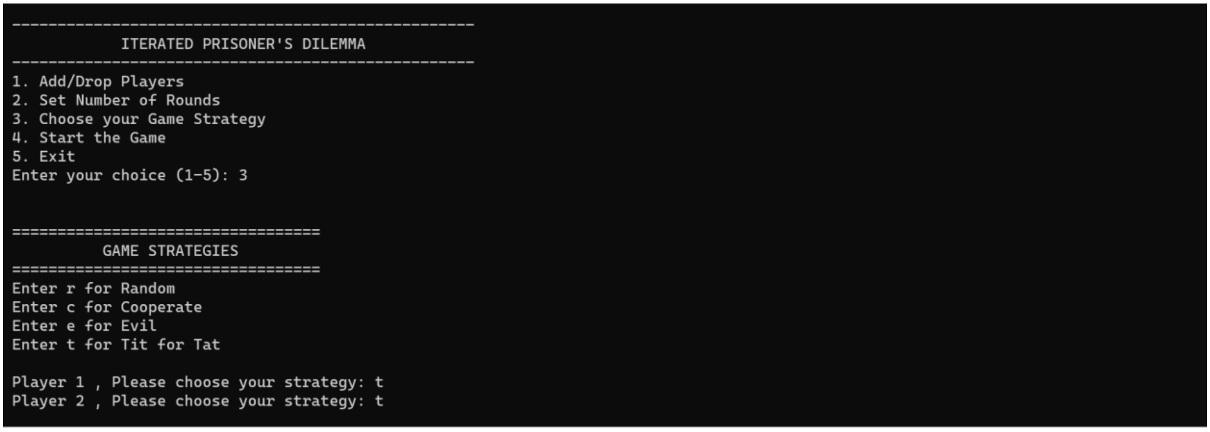


```
*********************************

Player ID: 1
Name: A
Score: 18

Player ID: 2
Name: B
Score: 23

*********************************

------------------------------------
|              RESULT              |
------------------------------------
|WinnerID: 2                       |
|Name: B                           |
|Score: 23                         |
------------------------------------


---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 5

Thank you for playing!

End of the program!
```

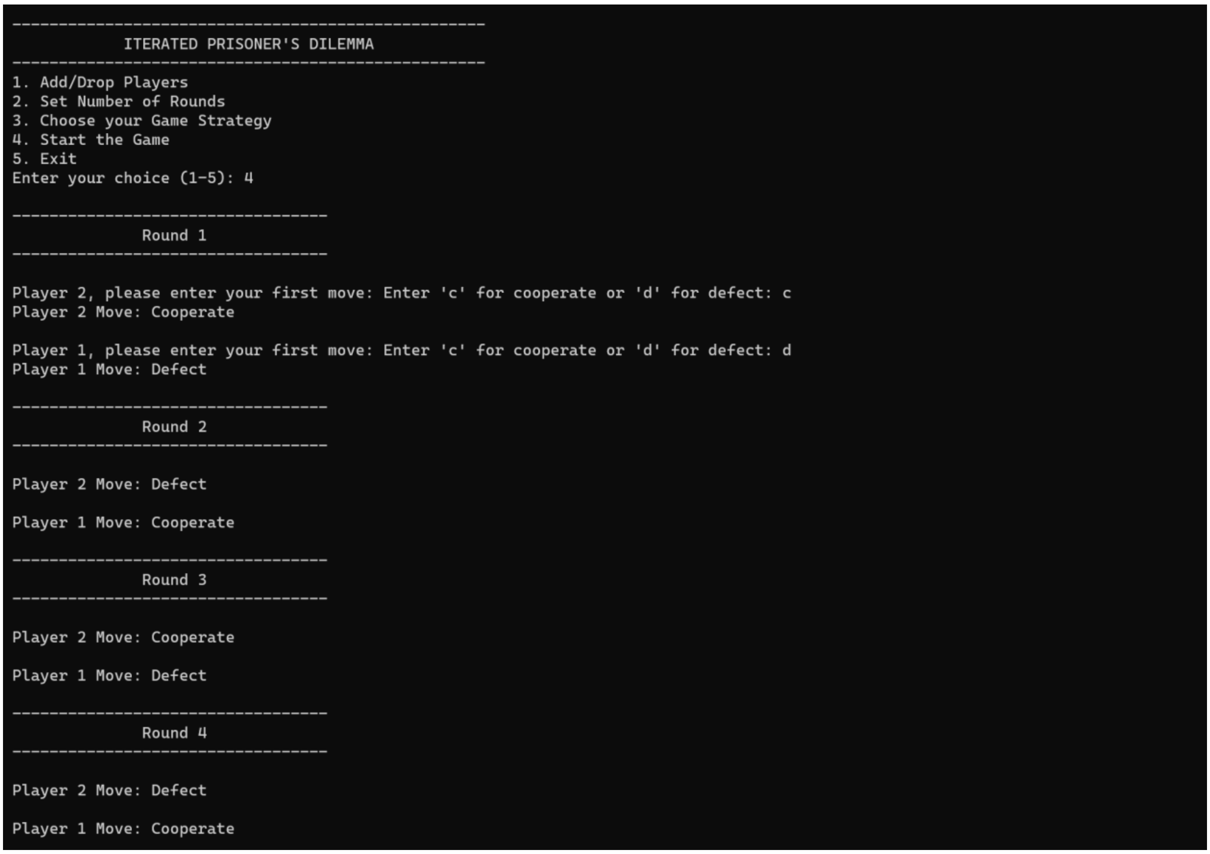
1.4.3)



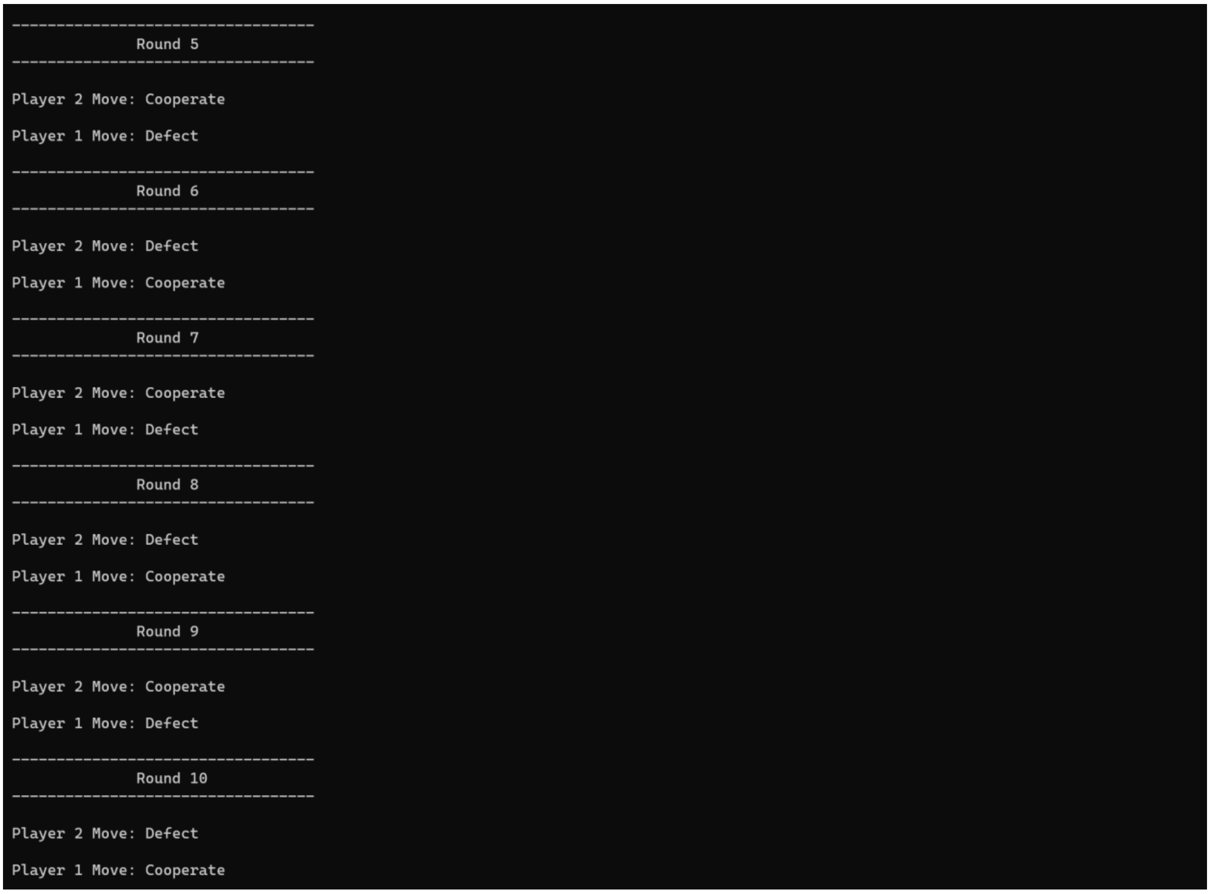
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 1


1. Add New Players to the Game (Begin/Reset Game)
2. Add Players to the Existing Game
3. Drop Players
Enter your choice (1-3): 1
Enter the number of players (exactly 2 players required):
2
Enter the name for Player 1: Mr John
Players added successfully!

Enter the name for Player 2: Aman
Players added successfully!


---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 2

Enter the number of rounds: 10
```



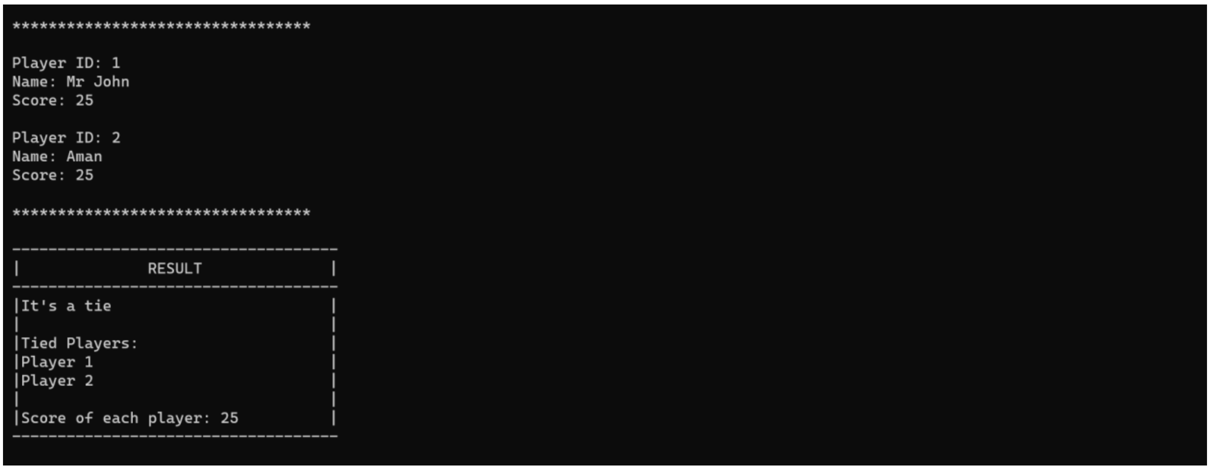
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 3


==================================
          GAME STRATEGIES
==================================
Enter r for Random
Enter c for Cooperate
Enter e for Evil
Enter t for Tit for Tat

Player 1 , Please choose your strategy: t
Player 2 , Please choose your strategy: t
```



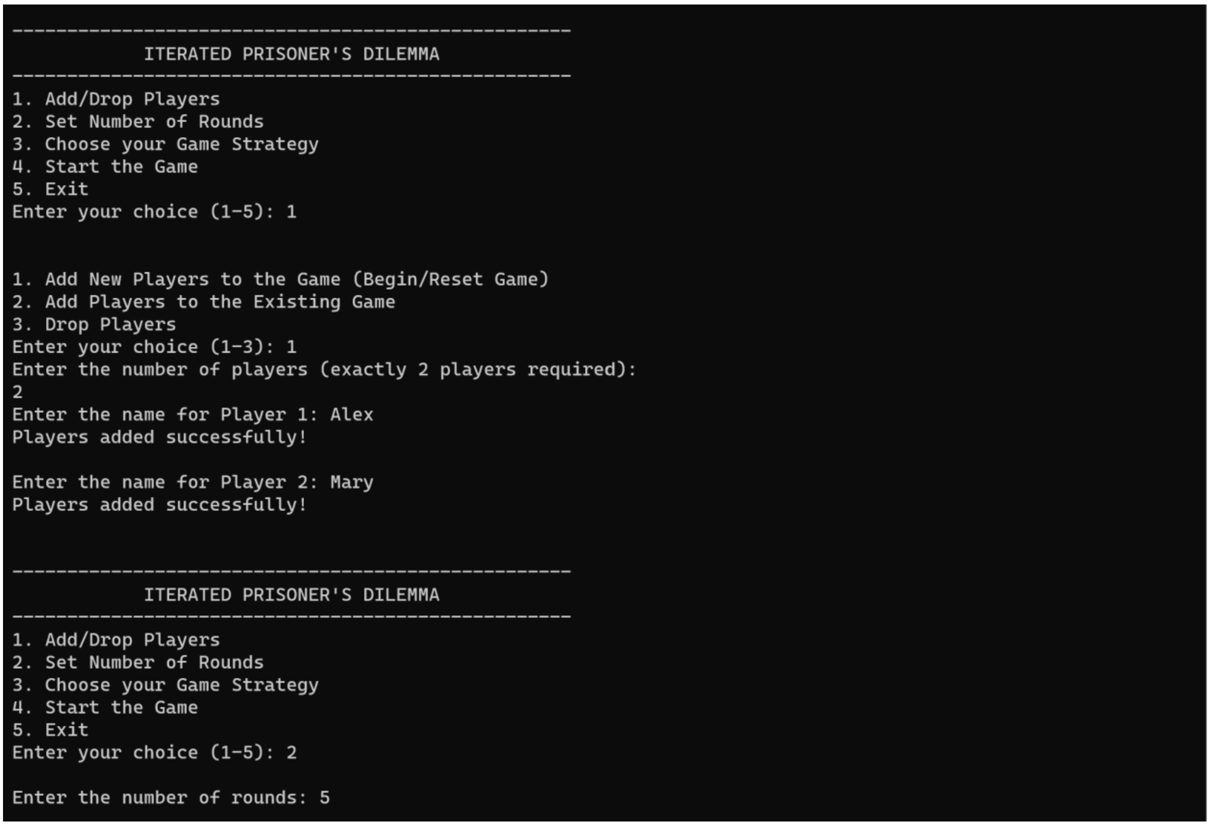
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 4

----------------------------------
              Round 1
----------------------------------

Player 2, please enter your first move: Enter 'c' for cooperate or 'd' for defect: c
Player 2 Move: Cooperate

Player 1, please enter your first move: Enter 'c' for cooperate or 'd' for defect: d
Player 1 Move: Defect

----------------------------------
              Round 2
----------------------------------

Player 2 Move: Defect

Player 1 Move: Cooperate

----------------------------------
              Round 3
----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Defect

----------------------------------
              Round 4
----------------------------------

Player 2 Move: Defect

Player 1 Move: Cooperate
```



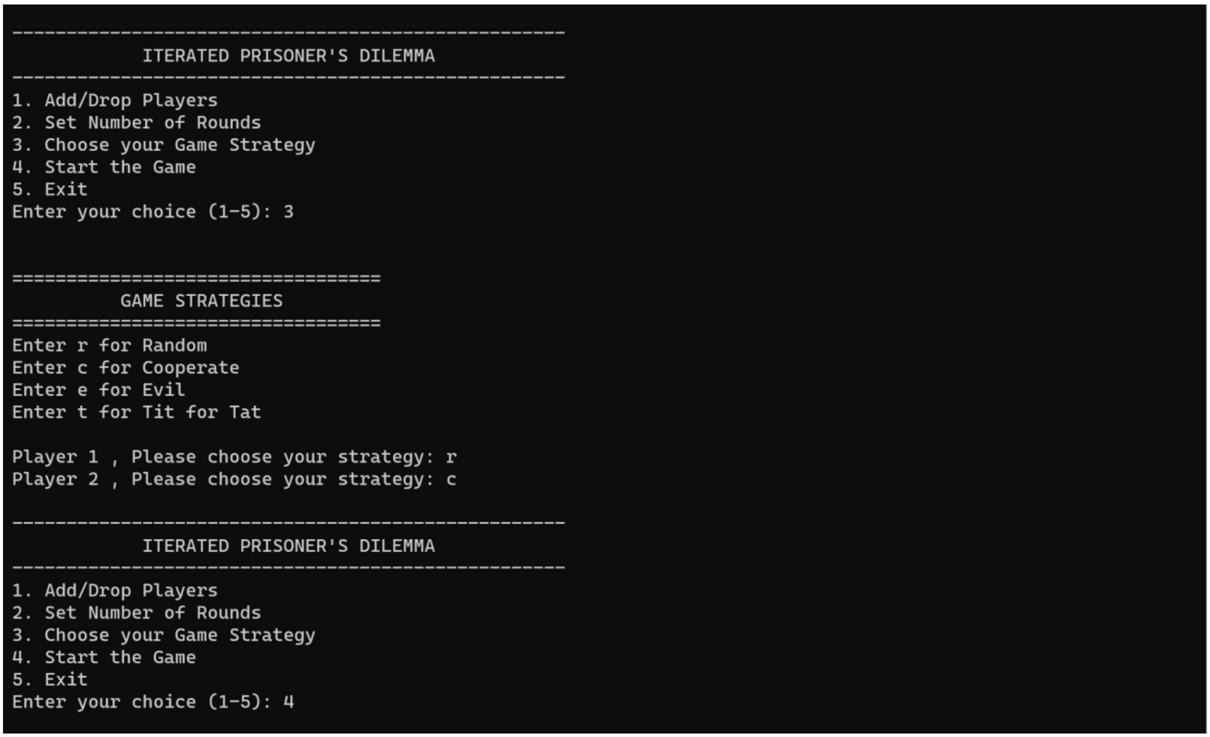
```
----------------------------------
              Round 5
----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Defect

----------------------------------
              Round 6
----------------------------------

Player 2 Move: Defect

Player 1 Move: Cooperate

----------------------------------
              Round 7
----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Defect

----------------------------------
              Round 8
----------------------------------

Player 2 Move: Defect

Player 1 Move: Cooperate

----------------------------------
              Round 9
----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Defect

----------------------------------
              Round 10
----------------------------------

Player 2 Move: Defect

Player 1 Move: Cooperate
```



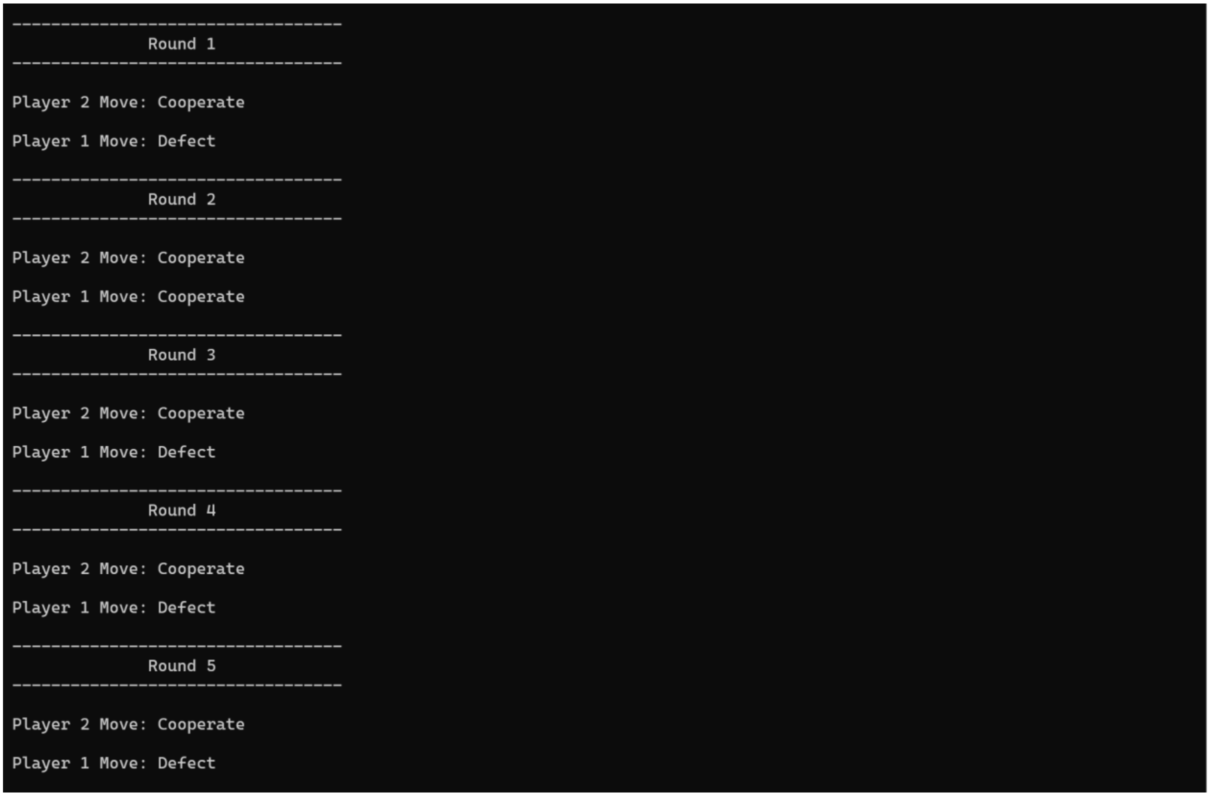
```
*********************************

Player ID: 1
Name: Mr John
Score: 25

Player ID: 2
Name: Aman
Score: 25

*********************************

------------------------------------
|             RESULT               |
------------------------------------
|It's a tie                        |
|                                  |
|Tied Players:                     |
|Player 1                          |
|Player 2                          |
|                                  |
|Score of each player: 25          |
------------------------------------
```



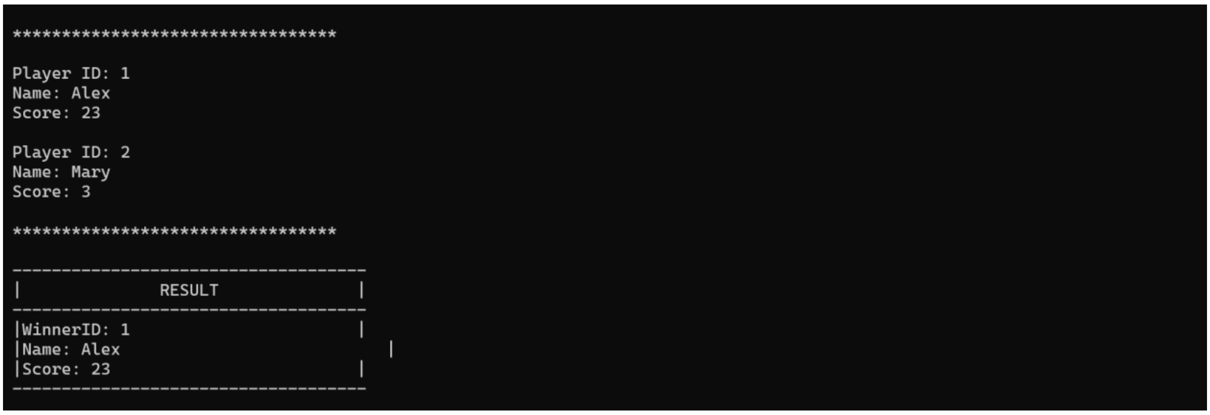
```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 1


1. Add New Players to the Game (Begin/Reset Game)
2. Add Players to the Existing Game
3. Drop Players
Enter your choice (1-3): 1
Enter the number of players (exactly 2 players required):
2
Enter the name for Player 1: Alex
Players added successfully!

Enter the name for Player 2: Mary
Players added successfully!


---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 2

Enter the number of rounds: 5
```



```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 3


==================================
          GAME STRATEGIES
==================================
Enter r for Random
Enter c for Cooperate
Enter e for Evil
Enter t for Tit for Tat

Player 1 , Please choose your strategy: r
Player 2 , Please choose your strategy: c

---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 4
```

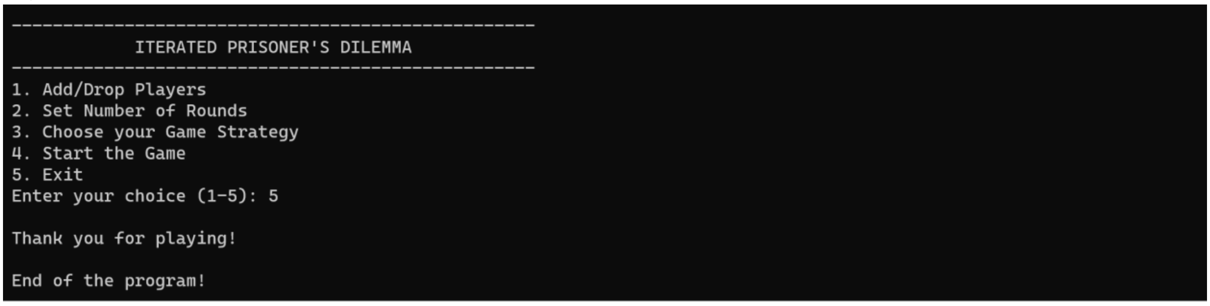


```
----------------------------------
              Round 1
----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Defect

----------------------------------
              Round 2
----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Cooperate

----------------------------------
              Round 3
----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Defect

----------------------------------
              Round 4
----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Defect

----------------------------------
              Round 5
----------------------------------

Player 2 Move: Cooperate

Player 1 Move: Defect
```



```
*********************************

Player ID: 1
Name: Alex
Score: 23

Player ID: 2
Name: Mary
Score: 3

*********************************

------------------------------------
|              RESULT              |
------------------------------------
|WinnerID: 1                       |
|Name: Alex                           |
|Score: 23                         |
------------------------------------
```

1.5)



```
---------------------------------------------------
            ITERATED PRISONER'S DILEMMA
---------------------------------------------------
1. Add/Drop Players
2. Set Number of Rounds
3. Choose your Game Strategy
4. Start the Game
5. Exit
Enter your choice (1-5): 5

Thank you for playing!

End of the program!
```

User Guide

The Iterated Prisoner's Dilemma (IPD) simulation is a program that implements the classic scenario from game theory. In this game, players repeatedly choose to cooperate or defect based on their opponent's previous moves. The goal is to accumulate the maximum number of points in a given number of moves. The simulation allows users to add and drop players, set the number of rounds, choose game strategies, and initiate the game simulation.

1) **Adding/Dropping Players (Option 1):**

- Choose option 1 from the main menu.
- Select sub-option 1 to add new players to the game or reset the game.
- Enter the number of players (exactly 2 players required).
- Input names for each player when prompted.

2) **Adding Players to the Existing Game (Option 1):**

- **Choose option 1 from the main menu.**
- Select sub-option 2 to add players to the existing game.
- Enter the number of players to add.
- Input names for each additional player when prompted.

3) **Dropping Players (Option 1):**

- Choose option 1 from the main menu.
- Select sub-option 3 to drop players.
- Enter the ID of the player you want to drop.

4) **Setting Number of Rounds (Option 2):**

- Choose option 2 from the main menu.
- Enter the desired number of rounds for the game.

5) **Choosing Game Strategy (Option 3):**

- Choose option 3 from the main menu.
- Enter the strategy code ('r' for Random, 'c' for Cooperate, 'e' for Evil, 't' for Tit for Tat) for each player when prompted.

6) **Starting the Game (Option 4):**

- Choose option 4 from the main menu.
- The game will simulate interactions between players over the specified number of rounds.
- The final results, including the winner or tied players, will be displayed.

7) **Exiting the Program (Option 5):**

- Choose option 5 to exit the program.