Step 1: Problem Identification and Statement

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., cooperating relative to the other prisoner) and 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.

In the implementation of the IPD game, you will have repeated interactions with your opponent, rather than just once, thus, the name iterated the prisoner’s dilemma. Your first move in the game, cooperative or defect, will be decided with no knowledge of how your opponent will move. However, on all subsequent moves, you can decide whether to cooperate or defect based on your opponent’s last move. You will be competing with your opponent. The goal of the game is to accumulate the maximum number of points in a given number of moves (N).

Step 2: Gathering of Information and Input/Output Description

Relevant information:

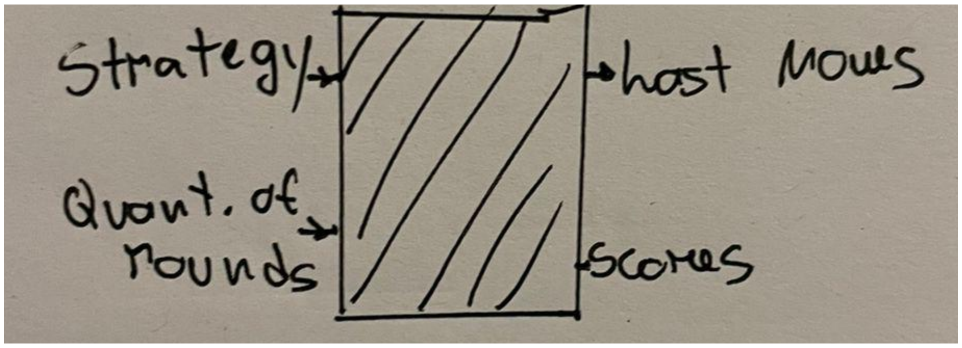
In the implementation of the IPD game, you will have repeated interactions with your opponent, rather than just once, thus the name iterated prisoner’s dilemma. Your first move in the game, cooperative or defect, will be decided with no knowledge of how your opponent will move. However, on all subsequent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

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

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

This situation is the basis for interesting research in many areas, including political science, biology, and economics.

Input/output Description:



The program only requires the strategy chosen by the two players and the number of rounds they want to play. The program outputs the Last Moves of both players and their scores.

Step 3: Design the algorithm and test cases

Test Case 1: Invalid Strategy

```
Welcome to the Prisioners Game

Please, choose the strategy you want to implement to compete with the
other prisioner:
1 - Random(The move Cooperate or Defect is chosen randomly)
2 - Evil(The move will be always defect)
3 - Cooperative(The move will be Cooperate)
4 - Tit for Tat(Cooperate on the first round and imitates opponent's
previous move thereafter)

To begin
Enter the number of rounds you want to play:
8
Player 1, please choose your strategy: 5
Invalid input, couldn't find that Strategy number'
Player 1, please choose your strategy:
```

Test Case 2: Negative Number of Rounds

```
Welcome to the Prisioners Game

Please, choose the strategy you want to implement to compete with the
other prisioner:
1 - Random(The move Cooperate or Defect is chosen randomly)
2 - Evil(The move will be always defect)
3 - Cooperative(The move will be Cooperate)
4 - Tit for Tat(Cooperate on the first round and imitates opponent's
previous move thereafter)

To begin
Enter the number of rounds you want to play:
-5
Please enter a positive number for the number of rounds:
-5
Please enter a positive number for the number of rounds:
```

Test Case 3: Both strategies are Cooperative

```
Welcome to the Prisioners Game

Please, choose the strategy you want to implement to compete with the
other prisioner:
1 - Random(The move Cooperate or Defect is chosen randomly)
2 - Evil(The move will be always defect)
3 - Cooperative(The move will be Cooperate)
4 - Tit for Tat(Cooperate on the first round and imitates opponent's
previous move thereafter)

To begin
Enter the number of rounds you want to play:
4
Player 1, please choose your strategy: 3

Player 2, please choose your strategy: 3

Game Scores:
Player 1 - Score: 12
Player 1: C C C C
Player 2 - Score: 12
Player 2: C C C C
Thank you for playing Prisioner's game! Hope you got back to play
again!
```

Test Case 4: One strategy Cooperative and the other tit-for-tat

```
Welcome to the Prisioners Game

Please, choose the strategy you want to implement to compete with the
other prisioner:
1 - Random(The move Cooperate or Defect is chosen randomly)
2 - Evil(The move will be always defect)
3 - Cooperative(The move will be Cooperate)
4 - Tit for Tat(Cooperate on the first round and imitates opponent's
previous move thereafter)

To begin
Enter the number of rounds you want to play:
4
Player 1, please choose your strategy: 3

Player 2, please choose your strategy: 4

Game Scores:
Player 1 - Score: 12
Player 1: C C C C
Player 2 - Score: 12
Player 2: C C C C
Thank you for playing Prisioner's game! Hope you got back to play
again!
```

Algorithm design:
Classes:

```
Define Class Strategy

  Assign currentStrategy as integer. (private member)

(Public)

Constructors

  Define Constructor Strategy()

    Assign currentStrategy with 0.

 Define Constructor Strategy(assign initialStrategy)

    Assign currentStrategy with initialStrategy.
  Define Destructor Strategy() (destructor)
  Define Function setStrategy(assign newStrategy)
    Assign currentStrategy with newStrategy.
    Return currentStrategy.
  Define Function cooperateOrDefect(assign enemylastMove)
    If currentStrategy is 1
      Return C or D based on random choice. (cooperate or defect)
    Otherwise if currentStrategy is 2
       Return D
    Otherwise if currentStrategy is 3
     Return C
    Otherwise if currentStrategy is 4
      If enemylastMove is blank
        Return C or D based on random choice.
      Otherwise
        Return enemylastMove.
    Otherwise
      Return 0.
Define Class Player
  Assign idCounter as static integer.
(private)
  Assign ID, score, numMoves as integers.
  Assign moves as a pointer to a character array.
Assign strategy as Strategy. (Strategy object)
  Define Constructor Player()
```

```
    Increment idCounter and assign to ID.
    Assign score and numMoves with 0.
    Assign moves with memory allocation for 1 character.
    Assign strategy with new Strategy.
  Define Constructor Player(assign initialStrategyNum)
    Increment idCounter and assign to ID.
    Assign score and numMoves with 0.
    Assign moves with memory allocation for 1 character.
    Assign strategy with new Strategy using initialStrategyNum.
  Define Destructor Player()
    Release memory assigned to moves.
(methods)
 Define Function makeMove(assign enemylastMove)
    Assign move with cooperation or defection decision.
    Assign newMoves with new memory allocation for numMoves + 1 characters.
Repeat while i equals to 0, i less than numMoves, increment i
    Copy elements from moves to newMoves.
    Assign move to newMoves at position numMoves.
    Release memory of moves.
    Assign moves with newMoves.
    Increment numMoves.
    Return move.
  Define Function printMoves()
Print "Player", ID
  Repeat while i equals to 0, i less than numMoves, increment i
```

```
    Print moves
  Print Newline
  Define Function setLastMove(assign lastMove)
    If numMoves is greater than 0
      Assign lastMove to the last position in moves.
      Return 0.
    Otherwise
      Return -1.
  Define Function updateStrategy(assign newStrategy)
    If strategy is successfully updated with newStrategy
      Return 0.
    Otherwise
      Return -1.
(Setters and Getters, Access and modify player attributes)
  Define Function getPlayerID()
    Return ID.
  Define Function getScore()
    Return score.
  Define Function getNumMoves()
    Return numMoves.
  Define Function getMoves() (Pointer to character)
    Return moves.
  Define Function updateScore(assign points)
Add points to final score.
```

Step 4: Implementation

```
/*******************************************************
Author: Pedro Felix Fernandes
Date Created: December 01, 2023
Description:
___________________________________________________
Assignment 3 - Iterated Prisoner's Dilemma
___________________________________________________
Computer Engineering Case Study – Iterated Prisoner's Dilemma:
The program simulates the iterated prisoner's dilemma (IPD), a popular game from game
theory.
The game accepts a number of maximum 2 players and each player must choose between one
of two possible moves: defect or cooperate.
Each combination of movie with your oponents' move will give you a payoff. Additionally,
on this game you can also choose one out of four strategies:

1 - Random (The move Cooperate or Defect is chosen randomly)
2 - Evil (The move will be always defect)
3 - Cooperative (The move will be Cooperate)
4 - Tit for Tat (Cooperate on the first round and imitates the opponent's previous move
thereafter)

The game ends, when the number of rounds ends.
**************************************************/


//including header files
#include <iostream>
using namespace std;
//Strategy Class
class Strategy
{
private:
   int currentStrategy;
   // stores the current strategy

public:
   // Constructors
   Strategy()
   {
      currentStrategy = 0;
   }
```

```
    Strategy(int initialStrategy) // Parameterized constructor to set an initial strategy
    {
        currentStrategy = initialStrategy;
    }
    ~Strategy() //Destructor
    {
    }
    //Methods
    int setStrategy(int newStrategy)  // set a new strategy.
    {
        currentStrategy = newStrategy;
        return currentStrategy;
    }
    char cooperateOrDefect(char enemylastMove)  //decide cooperation or defection based on
strategy
    {
        // Strategy implementation
        if (currentStrategy == 1)
        {
            return (rand() % 2 == 0) ? 'C' : 'D'; // Random strategy
        }
        else if (currentStrategy == 2)
        {
            return 'D'; // Always defect strategy
        }
        else if (currentStrategy == 3)
        {
            return 'C'; //Always cooperate strategy
        }
        else if (currentStrategy == 4)
        {
            if (enemylastMove == ' ')
            {
                if (rand() % 2 == 0)
                {
                    return 'C';
                }
                else
                {
                    return 'D';
                }
            }
            return enemylastMove; // Mimic last move of the opponent (Tit-for-Tat)
        }
        else
        {
            return 0;
        }
    }
};
//Player Class
class Player
{

private:
    int ID; // Unique ID for each player
    int score; // Score of the player
    char* moves; // Dynamic array to store moves
    int numMoves; // Number of moves made
    Strategy strategy; // Strategy object
    static int idcounter; // Static counter to assign unique IDs.

public:
    // Constructors
    Player()
    {
        ID = ++idcounter;
        score = 0;
        moves = new char[1];
        numMoves = 0;
        strategy = Strategy();
    }
    Player(int InitialStrategyNum)  // Constructor with initial strategy
    {
        ID = ++idcounter;
        score = 0;
        moves = new char[1];
        numMoves = 0;
        strategy = Strategy(InitialStrategyNum);
    }
    ~Player() // Destructor
    {
        delete[] moves;
    }

    // Methods: Include making moves, printing moves, updating last move, changing strategy,
and managing score.
    char makeMove(char enemylastMove) // make a move
```

```
{
    char move = strategy.cooperateOrDefect(enemylastMove); // Get move based on strategy

    char* newMoves = new char[numMoves + 1]; // Allocate memory for new moves
    for (int i = 0; i < numMoves; ++i)
    {
        newMoves[i] = moves[i];
    }
    newMoves[numMoves] = move;

    delete[] moves; // deallocate memory
    moves = newMoves;

    numMoves++; // Increment number of moves
    return move; // Return the move made
}

void printMoves() // Method to print all moves
{
    cout << "Player " << ID << ": ";
    for (int i = 0; i < numMoves; ++i)
    {
        cout << moves[i] << " "; // Print each move
    }
    cout << endl;
}

int setLastMove(char lastMove) // Set the last move made
{
    if (numMoves > 0)
    {
        moves[numMoves - 1] = lastMove;
        return 0;
    }
    return -1; // Failure, no moves made yet
}

int updateStrategy(int newStrategy) // Updates player's strategy
{
    if (strategy.setStrategy(newStrategy) == newStrategy)
    {
        return 0;
    }
    return -1;
    }

    // Setters and Getters
    int getPlayerID()
    {
        return ID; // Return player ID
    }

    int getScore()
    {
        return score; // Return player score
    }

    int getNumMoves()
    {
        return numMoves; // Return number of moves made
    }

    char* getMoves()
    {
        return moves; // Return moves array
    }

    void updateScore(int points)
    {
        score += points; // Add points to score
    }
};

// Game class: manages the gameplay.
class Game
{
private:
    Player* players[2]; // Array to create and limit two players
    int MaxNumMoves; //Maximum number of moves in the game

public:
    // Constructors
    Game(int maxnum)  //Initialize the game with a maximum number of moves
    {
        players[0] = nullptr;
        players[1] = nullptr;
        MaxNumMoves = maxnum;
    }
```

```
~Game() //Destructor
{
   players[0] = nullptr;
   players[1] = nullptr;
}
// Methods for game
int addPlayer(Player* newPlayer)  // Add a new player to the game
{
   if (players[0] == nullptr)
   {
      players[0] = newPlayer; // Add to first slot
      return 0;
   }
   else if (players[1] == nullptr)
   {
      players[1] = newPlayer; // Add to second slot
      return 1;
   }
   else
   {
      return -1; // if both slots are full
   }
}
int dropPlayer(int playerIndex)
{
   if (playerIndex < 0 || playerIndex >= 2) {
      return -1; // Index out of bounds
   }

   if (players[playerIndex] != nullptr) {
      delete players[playerIndex]; // Deallocate memory
      players[playerIndex] = nullptr; // Remove the player
      return 0; // Successful removal
   }
   else {
      return -1; // No player in the given slot
   }
}
int play()
{
   if (players[0] == nullptr || players[1] == nullptr)
   {
      cout << "Game cannot start, one or more players are missing." << endl;
      return -1; // if players are missing
```

```
    }
    for (int i = 0; i < MaxNumMoves; ++i) // Loop for the maximum number of moves.
    {
      // Get the move for player 1
      char player1Move;

      // Check if Player 2 has made any moves to determine Player 1's response
      if (players[1]->getNumMoves() > 0)
      {
        // Get the last move of Player 2 to inform Player 1's strategy
        char lastMove = players[1]->getMoves()[players[1]->getNumMoves() - 1];
        player1Move = players[0]->makeMove(lastMove);
      }
      else // Case where Player 2 has not made any moves yet.
      {
        player1Move = players[0]->makeMove(' ');
      }

      // Get the move for player 2
      char player2Move;

      // Check if Player 1 has made any moves to determine Player 2's response
      if (players[0]->getNumMoves() > 0)
      {
        // Get the last move of Player 1 to inform Player 2's strategy
        char lastMove = players[0]->getMoves()[players[0]->getNumMoves() - 1];
        player2Move = players[1]->makeMove(lastMove); // Determine Player 2's move based
on Player 1's last move.
      }
      else // Case where Player 1 has not made any moves yet.
      {
        player2Move = players[1]->makeMove(' ');
      }

      // Update the scores based on the moves
      if (player1Move == 'C' && player2Move == 'C') // Both players cooperate.
      {
        players[0]->updateScore(3);
        players[1]->updateScore(3);
      }
      else if (player1Move == 'C' && player2Move == 'D') // Player 1 cooperates, Player 2
defects.
      {
        players[0]->updateScore(0);
```

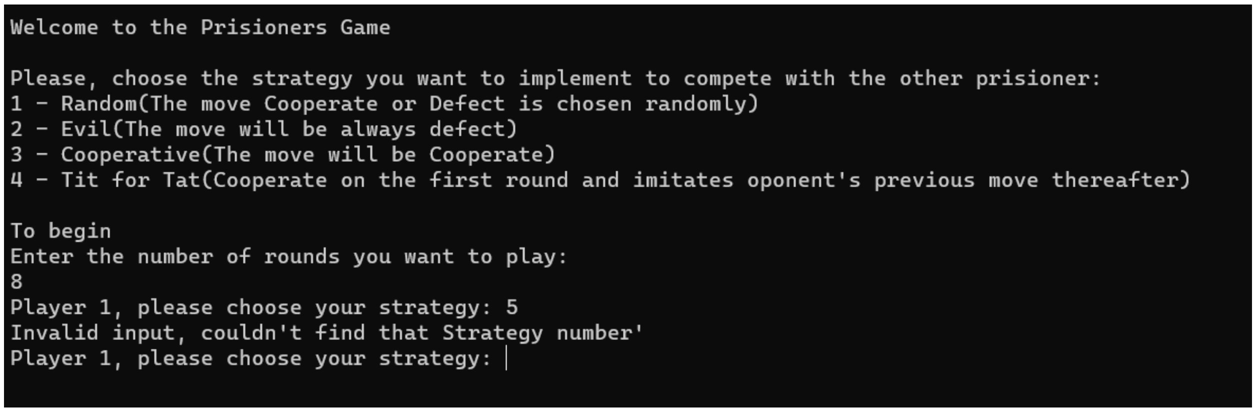
```
            players[1]->updateScore(5);
         }
         else if (player1Move == 'D' && player2Move == 'C') // Player 1 defects, Player 2
cooperates.
         {
            players[0]->updateScore(5);
            players[1]->updateScore(0);
         }
         else if (player1Move == 'D' && player2Move == 'D') // Both players defect.
         {
            players[0]->updateScore(1);
            players[1]->updateScore(1);
         }
      }
      return 0;

   }
   // Report game results
   void printResults()
   {
      cout << "Game Scores:" << endl;
      for (int i = 0; i < 2; ++i)
      {
         cout << "Player " << players[i]->getPlayerID() << " - Score: " << players[i]->getScore() <<
endl;
         players[i]->printMoves(); // Print moves of each player
      }
   }
};
int Player::idcounter = 0; // Initialize the static counter for Player IDs
//Main function
int main()
{
   Player p1, p2;
   int strategy1 = 0, strategy2 = 0;
   int numMoves = 0;
   cout << "Welcome to the Prisioners Game" << endl;
   cout << endl;
   cout << "Please, choose the strategy you want to implement to compete with the other
prisioner:" << endl;
   cout << "1 - Random(The move Cooperate or Defect is chosen randomly)" << endl;
   cout << "2 - Evil(The move will be always defect)" << endl;
   cout << "3 - Cooperative(The move will be Cooperate)" << endl;
```

```
    cout << "4 - Tit for Tat(Cooperate on the first round and imitates oponent's previous move
thereafter)" << endl;
    cout << endl;
    cout << "To begin" << endl;
    cout << "Enter the number of rounds you want to play: " << endl;
    cin >> numMoves;
    while (numMoves <= 0) {
        cout << "Please enter a positive number for the number of rounds:" << endl;
        cin >> numMoves;
    }

    while (strategy1 < 1 || strategy1 > 4)
    {
        cout << "Player " << p1.getPlayerID() << ", please choose your strategy: ";
        cin >> strategy1;
        if (strategy1 < 1 || strategy1 > 4)
        {
            cout << "Invalid input, couldn't find that Strategy number'" << endl;
        }
    }
    cout << endl;
    while (strategy2 < 1 || strategy2 > 4)
    {
        cout << "Player " << p2.getPlayerID() << ", please choose your strategy: ";
        cin >> strategy2;
        if (strategy2 < 1 || strategy2 > 4)
        {
            cout << "Invalid input, couldn't find that Strategy number" << endl;
        }
    }
    cout << endl;

    p1.updateStrategy(strategy1);
    p2.updateStrategy(strategy2);
    Game game(numMoves);
    game.addPlayer(&p1);
    game.addPlayer(&p2);
    game.play();
    game.printResults();
    cout << "Thank you for playing Prisioner's game!  I hope you get back to playing again!" <<
endl;


    return (0);
```

}

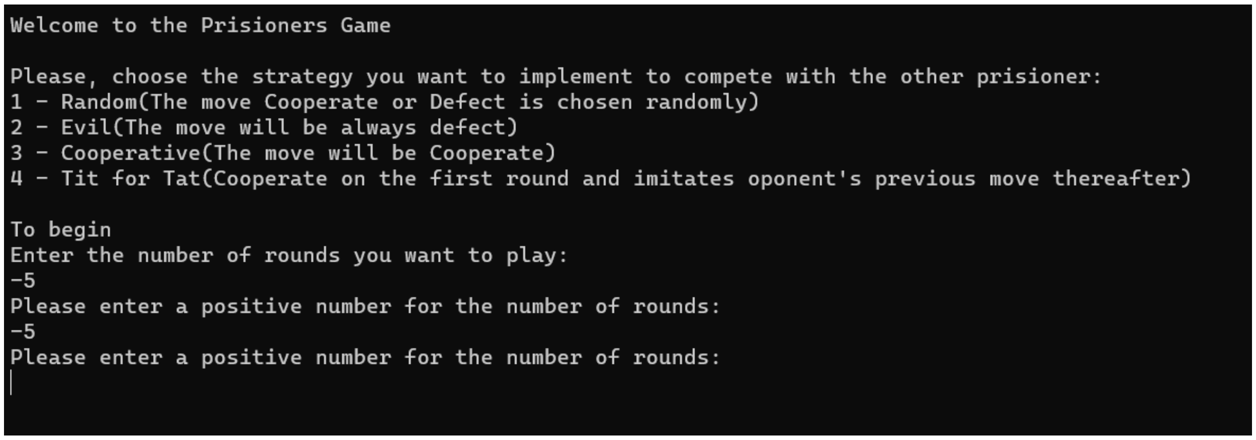
Step 5: Software Testing and Verification

Test Case 1: Invalid Strategy.



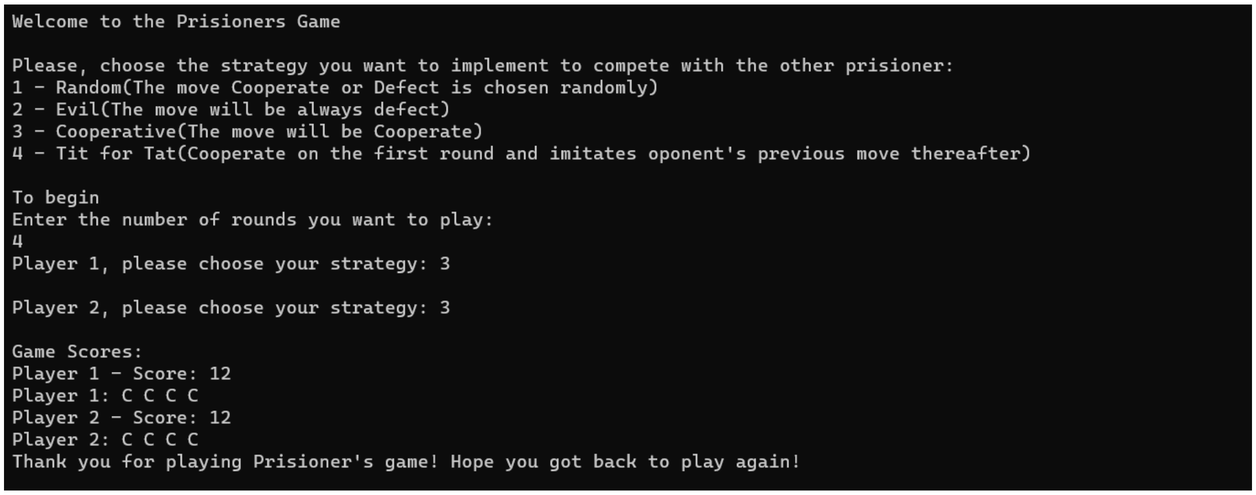
```
Welcome to the Prisioners Game

Please, choose the strategy you want to implement to compete with the other prisioner:
1 - Random(The move Cooperate or Defect is chosen randomly)
2 - Evil(The move will be always defect)
3 - Cooperative(The move will be Cooperate)
4 - Tit for Tat(Cooperate on the first round and imitates oponent's previous move thereafter)

To begin
Enter the number of rounds you want to play:
8
Player 1, please choose your strategy: 5
Invalid input, couldn't find that Strategy number'
Player 1, please choose your strategy:
```

Test Case 2: Negative Number of Rounds

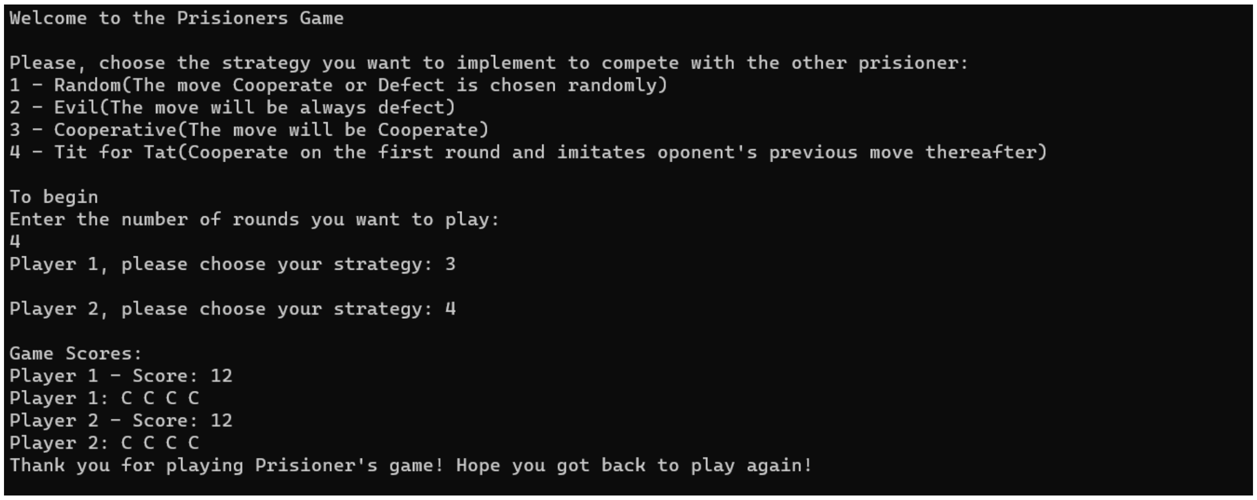


```
Welcome to the Prisioners Game

Please, choose the strategy you want to implement to compete with the other prisioner:
1 - Random(The move Cooperate or Defect is chosen randomly)
2 - Evil(The move will be always defect)
3 - Cooperative(The move will be Cooperate)
4 - Tit for Tat(Cooperate on the first round and imitates oponent's previous move thereafter)

To begin
Enter the number of rounds you want to play:
-5
Please enter a positive number for the number of rounds:
-5
Please enter a positive number for the number of rounds:
```

Test Case 3: Both strategies are Cooperative


```
Welcome to the Prisioners Game

Please, choose the strategy you want to implement to compete with the other prisioner:
1 - Random(The move Cooperate or Defect is chosen randomly)
2 - Evil(The move will be always defect)
3 - Cooperative(The move will be Cooperate)
4 - Tit for Tat(Cooperate on the first round and imitates oponent's previous move thereafter)

To begin
Enter the number of rounds you want to play:
4
Player 1, please choose your strategy: 3

Player 2, please choose your strategy: 3

Game Scores:
Player 1 - Score: 12
Player 1: C C C C
Player 2 - Score: 12
Player 2: C C C C
Thank you for playing Prisioner's game! Hope you got back to play again!
```

Test Case 4: Both Strategies are tit-for-tat


```
Welcome to the Prisioners Game

Please, choose the strategy you want to implement to compete with the other prisioner:
1 - Random(The move Cooperate or Defect is chosen randomly)
2 - Evil(The move will be always defect)
3 - Cooperative(The move will be Cooperate)
4 - Tit for Tat(Cooperate on the first round and imitates oponent's previous move thereafter)

To begin
Enter the number of rounds you want to play:
4
Player 1, please choose your strategy: 3

Player 2, please choose your strategy: 4

Game Scores:
Player 1 - Score: 12
Player 1: C C C C
Player 2 - Score: 12
Player 2: C C C C
Thank you for playing Prisioner's game! Hope you got back to play again!
```

User Guide:

This code consists of a simulation of a game. To play, you just need to select the number of rounds you want to simulate and choose 1 of the strategies shown on the menu per player. You can only have two players in the game. After all the information above, the code will print the moves (Cooperative or Defect) and give your score.