Game Theory

1. Main Class (Main.java)

- **Purpose**: This is the entry point of the application, responsible for setting up and running the game.
    - **Game Setup**: The user selects strategies for two players and specifies the number of rounds to simulate.
    - **Available Strategies**: The program provides a list of predefined strategies (e.g., Always Cooperate, Always Defect, Tit for Tat).
        - The user chooses 3 strategies for **Player 1** and 3 strategies for **Player 2**.
- **Game Simulation**:
    - After selecting strategies, the simulation enters a loop, where the user selects strategies for each round.
    - The **Game** class handles the simulation of both players' choices and calculates the resulting scores based on the rules of the Prisoner's Dilemma.
    - **Score Calculation**:
        - Both cooperate: Each player gets **3 points**.
        - One cooperates, one defects: The cooperator gets **0 points**, the defector gets **5 points**.
        - Both defect: Each player gets **1 point**.

---

2. Strategy Class (Strategy.java)

- **Purpose**: This is the abstract base class for all strategies, providing common functionality and structure.
    - **Key Methods**:
        - `makeMove()`: Abstract method that defines the decision-making process for each strategy (whether to cooperate or defect).
        - `getStrategyName()`: Returns the name of the strategy (based on the class name).
        - `addOpponentMove(boolean opponentMove)`: Records the opponent's move for future reference.
        - `getPoints()`: Calculates the total score by summing the results of all rounds.
        - `clearStrategy()`: Resets move history and scores for a new game session.
        - `addOutcome(int outcome)`: Adds a round's point outcome.

---

3. Game Class (Game.java)

- **Purpose**: Simulates the interaction between two players over a series of rounds.
    - **Key Methods**:

- `executeGame(int rounds)`: Runs the game for a specified number of rounds, updating the players' scores based on their decisions (cooperate or defect).
- **Scoring**: The game follows the Prisoner's Dilemma scoring system:
  - Both cooperate: **3 points each**.
  - One defects: The defector gets **5 points**, the cooperator gets **0 points**.
  - Both defect: **1 point each**.

---

4. Tournament Class (Tournament.java)

- **Purpose**: Manages a tournament where multiple strategies compete against each other.
  - **Key Methods**:
    - `executeTournamentRounds(int numRounds)`: Executes the tournament for a set number of rounds, matching up all strategies and calculating total scores.
    - `tournamentRound(int n)`: Handles one round of the tournament, where each strategy plays against every other strategy for `n` rounds.
    - `addNewPoints(HashMap<Strategy, Integer> newPoints)`: Adds the points accumulated during a round to each strategy's total.
    - `sortEntries(Set<Map.Entry<Strategy, Integer>> entrySet)`: Sorts strategies by their total score in descending order.

---

5. Strategy Implementations

Each concrete strategy defines how it behaves (cooperates or defects) based on different rules or mechanisms:

a. AlwaysCooperate (AlwaysCooperate.java)

- **Behavior**: Always cooperates, regardless of the opponent's actions.
- **Implementation**:
  - `makeMove()`: Returns `true` (cooperate) every time.
  - **Outcome**: Scores **3 points** if the opponent cooperates, **0 points** if the opponent defects.

b. AlwaysDefect (AlwaysDefect.java)

- **Behavior**: Always defects, regardless of the opponent's actions.
- **Implementation**:
  - `makeMove()`: Returns `false` (defect) every time.
  - **Outcome**: Scores **1 point** if the opponent defects, **5 points** if the opponent cooperates.

c. GeneticMemory (GeneticMemory.java)

- **Behavior**: Adapts to the opponent's behavior based on memory.
- **Implementation**:
    - Tracks the opponent's previous moves.
    - `makeMove()`: Decides whether to cooperate or defect based on the frequency of the opponent's defections. If the opponent defects frequently, it becomes more likely to defect.
    - **Outcome**: Cooperation is possible, but defection increases if the opponent defects often.

d. GeneticOneMove (GeneticOneMove.java)

- **Behavior**: Uses a dynamic, probabilistic approach to decide moves based on a payoff matrix.
- **Implementation**:
    - The strategy evaluates the payoff matrix to calculate a "weight" that influences the decision.
    - **Payoff Matrix**:
        - The matrix defines possible outcomes for Player 1 (Man) and Player 2 (Woman) based on their choices (boxing or ballet).
    - **Weight Calculation**:
        - The weight is determined by normalizing the payoff differences between boxing and ballet.
        - If the weight exceeds **0.5**, Player 1 chooses boxing; otherwise, Player 1 chooses ballet.

e. GeneticStrategy (GeneticStrategy.java)

- **Behavior**: Uses a probabilistic approach, with a "weight" influencing the likelihood of cooperation or defection.
- **Implementation**:
    - The strategy uses a weight between **0 and 1** that determines the probability of cooperation.
    - Over time, the weight can mutate slightly, evolving the strategy.
    - If the weight is high, cooperation is more likely; if the weight is low, defection is more probable.

f. Grudger (Grudger.java)

- **Behavior**: Starts by cooperating but defects permanently if the opponent defects.
- **Implementation**:
    - Initially cooperates, but if the opponent defects at any point, the Grudger will defect for the rest of the game.
    - **Outcome**: It punishes the opponent for defection and never returns to cooperation.

g. Simpleton (Simpleton.java)

- **Behavior**: Chooses moves randomly, without any strategy or memory.
- **Implementation**:

- `makeMove()`: Randomly returns `true` (cooperate) or `false` (defect) with equal probability.
- **Outcome**: The performance of this strategy is unpredictable, as it depends entirely on random choices.

h. TitForTat (TitForTat.java)

- **Behavior**: Starts by cooperating and then mirrors the opponent's previous move.
- **Implementation**:
  - Initially cooperates.
  - In subsequent rounds, the strategy mirrors the opponent's previous move.
    - If the opponent cooperates, TitForTat cooperates.
    - If the opponent defects, TitForTat defects.
  - **Outcome**: Encourages cooperation but punishes defection by retaliating.

---

6. How the Game Works

- **Rounds**: The game is played over multiple rounds (as specified by the user).
  - In each round, both players decide whether to cooperate or defect.
  - The strategies interact, and points are awarded based on their decisions.
- **Tournament Mode**: Multiple strategies can compete against each other. The strategy with the highest score at the end of the tournament wins.

---

7. User Interaction

- **Console Interaction**: The user selects strategies for both players and sets the number of rounds.
  - After each round, the game displays the results (scores for both players).
- **Tournament Mode**: Strategies compete against each other in a tournament, and the total score across all rounds determines the winner.

Summary of Strategies:

1. **AlwaysCooperate**: Always cooperates, simple and trusting.
2. **AlwaysDefect**: Always defects, exploiting the opponent's cooperation.
3. **GeneticMemory**: Adapts based on the opponent's past defections.
4. **GeneticOneMove**: Uses a probabilistic approach based on a payoff matrix and evolving behavior.
5. **GeneticStrategy**: Evolves by adjusting a cooperation weight, simulating mutation over time.
6. **Grudger**: Cooperates initially but defects permanently if the opponent defects.
7. **Simpleton**: Makes random choices with no strategy or memory.
8. **TitForTat**: Starts by cooperating and then mirrors the opponent's previous move.

These strategies showcase a range of approaches to the **Prisoner's Dilemma**: from simple, deterministic strategies (e.g., AlwaysCooperate, AlwaysDefect) to adaptive strategies (e.g., TitForTat, GeneticStrategy), and even random strategies (e.g., Simpleton).

Conclusion

This code provides a simulation of the **Prisoner's Dilemma**, allowing different strategies to compete and interact. The simulation illustrates the complexities of decision-making in game theory, where the choices of cooperation and defection evolve over time, depending on each strategy's behavior.