Prisonniers “Java Projet Documentation”

Présentation:

💡 **Le dilemme du prisonnier** est un jeu théorique en théorie des jeux qui met en lumière le conflit entre la poursuite de l'intérêt individuel et celui du bien commun. Voici le principe du jeu :

Deux participants, A et B, sont impliqués dans un crime et sont interrogés séparément par la police. Chacun a deux choix : coopérer avec l'autre en gardant le silence (**coopération**) ou trahir l'autre en témoignant contre lui (**dénonciation**). Voici les récompenses associées à chaque combinaison de choix :

- Si les deux coopèrent (coopération mutuelle), ils reçoivent une peine légère pour complicité mineure.
- Si l'un coopère et l'autre trahit, celui qui trahit reçoit une peine très légère (il devient un témoin collaborateur) tandis que l'autre reçoit une peine très sévère.
- Si les deux trahissent (dénonciation mutuelle), ils reçoivent tous les deux une peine sévère.

💡 Le dilemme réside dans le fait que, du point de vue individuel, il est avantageux de trahir car cela minimise la peine individuelle, quelle que soit la décision de l'autre. Cependant, si les deux trahissent, ils obtiennent un résultat global moins favorable que s'ils avaient tous deux coopéré, met en évidence la tension entre **l'intérêt individuel** et **l'intérêt collectif**, soulignant comment des incitations individuelles peuvent conduire à des résultats suboptimaux pour l'ensemble du groupe

Description pour chaque classe:

1. **Class Player**:
    - **Fonctionnalité** : Classe abstraite qui définit le comportement de base d'un joueur.
    - **Méthodes** :
        - `getName()`: Retourne le nom de la stratégie.
        - `play(String opponentName)`: Méthode abstraite qui permet à chaque stratégie de prendre une décision en fonction du nom de l'adversaire.
2. **Stratégies** (Collaborator, Denouncer, Grudge, Random, Suspicious, TitForTat, Prober, Alternate, Gradual) :
    - **Fonctionnalité** : Chaque classe représente une stratégie de jeu différente avec son propre comportement de jeu.
    - **Méthode play(String opponentName)** : Implémentation spécifique à chaque stratégie pour prendre des décisions en fonction du comportement de l'adversaire.
3. **Class Game** :
    - **Fonctionnalité** : Gère les règles du jeu du dilemme du prisonnier.
    - **Méthodes** :
        - `play(Player player1, Player player2)`: Simule un tour entre deux joueurs et retourne les résultats.
        - `setLastOutcome(String player1Name, String player2Name, int player1Choice, int player2Choice)`: Enregistre les résultats du dernier tour.
        - `getLastOutcome(String player1Name, String player2Name)`: Récupère les résultats du dernier tour.
4. **Classe Tournament** (Interface) :
    - **Fonctionnalité** : Définit les méthodes pour organiser et évaluer les tournois entre différentes stratégies.
    - **Méthodes** :

- `play()` : Méthode à implémenter pour exécuter le tournoi.
- `printRanking(Player[] players, int[] ranking)` : Affiche le classement en fonction des victoires.
- `printRanking(Player[] players, double[] ranking)` : Affiche le classement en fonction des points.
- `competeEvolutionary(Player[] players, int numGenerations)` : Simule une compétition évolutionnaire.

5. **Classe GameExample** :
   - **Fonctionnalité** : Permet de simuler un jeu entre deux joueurs et d'afficher les résultats.
   - **Méthodes** :
     - `play()` : Simule un jeu entre les joueurs et affiche les résultats.
6. **Classe Arena** :
   - **Fonctionnalité** : Organise des tournois entre différentes stratégies et évalue leurs performances selon diverses mesures.
   - **Méthodes** :
     - `play()` : Organise des tournois et affiche les classements basés sur les victoires et les points.
     - `competeEvolutionary(Player[] players, int numGenerations)` : Simule une compétition évolutionnaire.

Scénario dans Game :

- Un tour de jeu est simulé entre deux joueurs en utilisant la méthode `play(Player player1, Player player2)` en passant les deux instances de joueurs comme arguments. Le résultat est un tableau de doubles qui représente les gains des deux joueurs.

Scénario dans Arena :

- La classe Arena organise un tournoi entre différentes stratégies en utilisant la méthode `play()`. Elle utilise les méthodes `calculateVictories` et `calculatePoints` pour évaluer les performances de chaque stratégie en fonction du nombre de victoires et des points obtenus. Ensuite, les classements sont affichés pour les mesures M0 et M1.

L'intérêt de chaque classe et méthode réside dans l'organisation et l'évaluation de différentes **stratégies** de jeu, permettant ainsi de comparer leurs performances dans le dilemme du prisonnier. Cela permet d'explorer diverses approches de jeu et de comprendre quelles stratégies sont les plus efficaces dans différentes situations.