TD Pierre-Feuille-Ciseaux

Le jeu est probablement bien connu. Il se joue à deux joueurs. Lors d'un tour de jeu, les joueurs jouent simultanément[1] et disposent de 3 coups possibles : *pierre*, *feuille*, *ciseaux*.
La résolution d'un tour de jeu est la suivante :

1. si les deux joueurs jouent la même chose, le coup est nul,
2. la *pierre* bat (casse) les *ciseaux*,
3. le *feuille* bat (enferme) la *pierre*,
4. les *ciseaux* battent (coupent) le *feuille*.

Un coup nul rapporte 1 point, un coup victorieux 2 points et un coup perdant 0.
Une partie se joue en une suite de tours de jeu en nombre fixé au départ. Le vainqueur est celui (quand il y en a un) qui a le plus de points à l'issue de tous les tours, sinon la partie est nulle.

Nous allons procéder à l'analyse d'un programme objet permettant de jouer à ce jeu. Il est important de traiter les questions dans l'ordre. Exceptionnellement, il est peut-être même préférable de ne pas lire une question avant d'avoir répondu aux précédentes.

Q 1 . Donnez l'algorithme d'un traitement qui permet de faire jouer un tour de jeu (méthode `playOneRound` d'une classe `Game`).

Q 2 . En partant de la réponse à la question précédente, déterminez et définissez les types nécessaires à la modélisation de ce jeu.

Ecrivez le code java de la méthode `playOneRound` de `Game`.

NB : si cela vous aide, pour cette question vous pouvez considérer que les joueurs jouent en choisissant aléatoirement l'un des trois coups possibles.

Q 3 . On souhaite maintenant que l'un des joueurs (ou les deux) ne joue plus aléatoirement mais applique une autre stratégie de jeu, comme celle de jouer toujours le coup *feuille* par exemple.

Que proposez-vous de modifier à ce qui a été défini précèdemment pour prendre en compte cette modification ?

Q 4 . Pour traiter la question précedente, êtes-vous en mesure de faire une proposition qui permette de ne pas modifier ce qui est écrit lors de la prise en compte de la seconde stratégie ?

Si oui, laquelle ?

Q 5 . On souhaite en fait pouvoir varier autant que possible les stratégies appliquées par chacun des deux joueurs. En plus des deux stratégies déjà évoquées on peut ainsi en imaginer d'autres telles que :

- ▷ toujours jouer *pierre*,
- ▷ toujours jouer *feuille*,
- ▷ jouer en boucle *pierre* suivi de *feuille*, ou toute autre séquence,
- ▷ etc.

A nouveau :

Que faut-il modifier à ce qui a été défini précèdemment pour prendre en compte toutes ces différentes stratégies ?

Faites une proposition. Est-il nécessaire de modifier les classes écrites ?

Q 6 . Enfin, comme dernière stratégie de jeu on ajoute la possibilité pour un joueur humain de choisir le coup à jouer par une saisie au clavier.

Que faut-il faire pour intégrer cette stratégie de jeu ?

Faites une proposition.

[1] D'un point de vue du programme cela signifie que lorsqu'un joueur choisit son coup de jeu, il n'a aucune information sur ce que choisit l'autre joueur au même tour.

TP Pierre-Feuille-Ciseaux

Vous devez réaliser un programme qui permette de jouer au jeu *Pierre-Feuille-Ciseaux*. Pour cela vous implémenterez donc le travail fait en TD.

Voici quelques consignes pour ce TP :

- Placez votre travail dans un dossier nommé `pfc` de votre dépôt git.
- Créez un paquetage `pfc` pour l'application et un paquetage `pfc.strategy` pour les stratégies de jeu.

  Vous placerez au moins quatre stratégies différentes dans ce "sous-paquetage".
- La classe `Game` doit comporter une méthode :

  ```
  public Player play(int nbRounds)
  ```

  qui fait jouer `nbRounds` tours du jeu à deux joueurs.

  Le résultat de cette méthode est le joueur vainqueur.

  Comment proposez-vous de gérer le résultat en cas d'égalité ?
- Faites le nécessaire dans votre code pour que lors de l'exécution de cette méthode `play()`, on ait une trace de ce qu'il se passe lors d'une partie, par exemple avec un affichage de la forme :

  > xxx *a joué PIERRE.* yyy *a joué FEUILLE*
  > xxx *l'emporte et marque 2 points*
  > *Le score est maintenant :* xxx = X *points* - yyy = Y *points*

  De plus à la fin d'une partie le vainqueur est annoncé ou match nul si aucun.
- Pour la saisie au clavier, vous pouvez réutiliser la classe `io.Input` fournie pour le TP *bataille navale*.
- Pour la génération de valeurs aléatoires, consultez la classe `java.util.Random`.
- Vous créerez une classe contenant une méthode `main()` dont l'exécution qui permettra de faire jouer une partie de ce jeu entre un joueur qui applique la stratégie aléatoire et un joueur humain (donc avec saisie au clavier des coups joués).

  Le nombre de tours de la partie sera passé en argument de la ligne de commande.
- En plus de toutes les informations habituelles (génération de documentation, compilation des tests et du code, etc.), indiquez dans le fichier `readme.md` de votre dépôt comment créer un jar exécutable qui exécutera cette méthode `main()`.
- Dans le fichier `readme.md` de votre dépôt, rédigez une section dans laquelle vous expliquez clairement et précisément ce qu'il faut faire pour créer une nouvelle stratégie pour ce jeu.