Jeu des 13 allumettes

1 Description fonctionnelle

Le jeu des 13 allumettes se joue à deux joueurs qui, à tour de rôle, prennent 1, 2 ou 3 allumettes d'un tas qui en contient initialement 13. Le joueur qui prend la dernière allumette perd.

L'application à développer doit permettre à deux joueurs de s'affronter. C'est l'ordinateur qui est l'arbitre de la partie. Il affiche le nombre d'allumettes restant en jeu, donne la main à tour de rôle à chaque joueur et vérifie que les joueurs respectent les règles du jeu. Ainsi, si un joueur veut prendre trop ou trop peu d'allumettes, le coup est refusé et le joueur doit rejouer. Un joueur doit retirer entre 1 et 3 allumettes. En fin de partie, l'arbitre affiche le résultat.

Chaque joueur choisit le coup à jouer en fonction d'une stratégie. Une stratégie *humain* consiste à demander à l'utilisateur ce qu'il veut jouer. Ceci permet d'avoir un joueur humain. Les autres stratégies consistent à faire jouer l'ordinateur et définissent son niveau de jeu. Une liste non exhaustive des niveaux de jeu, et donc des stratégies correspondantes, inclut :

- niveau *naïf* : l'ordinateur choisit aléatoirement le nombre d'allumettes à jouer,
- niveau *rapide* : l'ordinateur prend le maximum d'allumettes possible (de manière à ce que la partie se termine le plus rapidement possible),
- niveau *expert* : l'ordinateur joue du mieux qu'il peut (s'il peut gagner, il gagnera).

Le listing suivant constitue une copie des informations affichées à l'écran lors du déroulement d'une partie opposant l'utilisateur (Xavier) à l'ordinateur (Ordinateur) en niveau naïf.

```
Nb d'allumettes restantes : 13
Au tour de Xavier.
Combien prenez-vous d'allumettes ? 2
Xavier prend 2 allumette(s).

Nb d'allumettes restantes : 11
Au tour de Ordinateur.
Ordinateur prend 1 allumette(s).

Nb d'allumettes restantes : 10
Au tour de Xavier.
Combien prenez-vous d'allumettes ? X
Vous devez donner un entier compris entre 1 et 3.
Combien prenez-vous d'allumettes ? 5
Xavier prend 5 allumette(s).
Erreur ! Prise invalide : 5
Recommencez !

Nb d'allumettes restantes : 10
Au tour de Xavier.
Combien prenez-vous d'allumettes ? 3
Xavier prend 3 allumette(s).
```

```
    Nb d'allumettes restantes : 7
    Au tour de Ordinateur.
26  Ordinateur prend 1 allumette(s).

    Nb d'allumettes restantes : 6
    Au tour de Xavier.
    Combien prenez-vous d'allumettes ? 2
31  Xavier prend 2 allumette(s).

    Nb d'allumettes restantes : 4
    Au tour de Ordinateur.
    Ordinateur prend 2 allumette(s).
36
    Nb d'allumettes restantes : 2
    Au tour de Xavier.
    Combien prenez-vous d'allumettes ? 1
    Xavier prend 1 allumette(s).
41
    Nb d'allumettes restantes : 1
    Au tour de Ordinateur.
    Ordinateur prend 2 allumette(s).
    Erreur ! Prise invalide : 2
46  Recommencez !

    Nb d'allumettes restantes : 1
    Au tour de Ordinateur.
    Ordinateur prend 1 allumette(s).
51  Ordinateur a perdu !
```

2 Architecture

L'analyse du problème a été commencée et un diagramme de classe partiel est proposé sur la figure 1. La classe `Jeu` modélise le jeu des 13 allumettes, y compris les règles sur la prise des allumettes. Le jeu est caractérisé par le nombre d'allumettes, initialement 13.

Il est possible de retirer du jeu un nombre d'allumettes compris entre 1 et 3. 3 est le nombre maximal d'allumettes que peut prendre un joueur par tour. Bien entendu, il ne peut pas prendre plus d'allumettes qu'il n'en reste en jeu. Le non respect de ces règles sera signalé par une exception, contrôlée par le compilateur, appelée `CoupInvalideException`.

Le nombre maximal d'allumettes que peut prendre un joueur sera représenté par une constante de la classe Jeu. Cette constante est la même pour tous les jeux.

La classe `Joueur` modélise un joueur. Un joueur a un nom. On peut demander à un joueur le nombre d'allumettes qu'il veut prendre pour un jeu donné (`getPrise`). Un joueur détermine le nombre d'allumettes à prendre en fonction de sa stratégie : naïf, rapide, expert ou humain. La stratégie humain signifie que c'est l'utilisateur de l'application qui décide du nombre d'allumettes à prendre. Il pourrait y avoir d'autres stratégies.

La classe `Arbitre` fait respecter les règles du jeu aux deux joueurs. Cette classe possède un constructeur qui prend en paramètre les deux joueurs `j1` et `j2` qui vont s'affronter. Une partie peut alors être arbitrée entre ces deux joueurs. L'arbitre fait jouer, à tour de rôle, chaque joueur

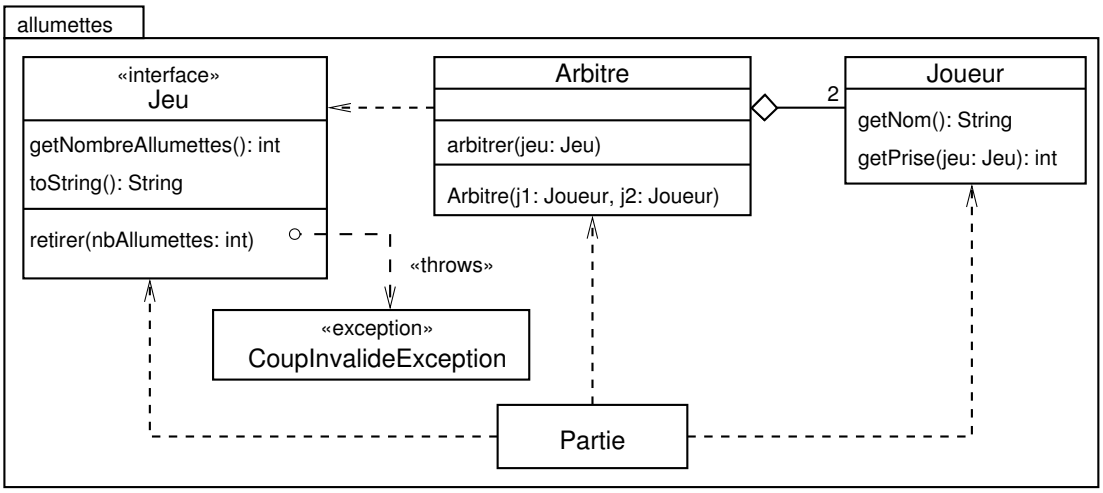

FIGURE 1 – Diagramme de classe pour le jeu des 13 allumettes

en commençant par le joueur `j1`. Celui qui prend la dernière allumette a perdu. Faire jouer un joueur consiste à lui demander le nombre d'allumettes qu'il souhaite prendre, puis à les retirer du jeu. Si ceci provoque une violation des règles du jeu (exception `CoupInvalideException`), le même joueur devra rejouer.

Dans la modélisation proposée, on constate que l'arbitre communique le jeu aux joueurs. Un joueur a ainsi accès au jeu. Il serait possible de programmer une stratégie consistant à retirer toutes les allumettes du jeu sauf 2. En proposant alors de ne prendre qu'une seule allumette, le joueur indélicat est sûr de gagner. Bien sûr, il y a tricherie. Dans ce cas la partie devrait être immédiatement arrêtée et le joueur désigné comme tricheur !

En l'état actuel de la modélisation, l'arbitre ne peut pas détecter [1] ce type de tricherie. Une solution consiste à s'appuyer sur le patron de conception Procuration (aussi appelé Mandataire, Proxy, etc.) dont l'architecture est donnée à la figure 2. Au lieu d'accéder au sujet réel, le client accède à l'objet procuration qui relaie l'opération vers le sujet réel. Ici, le client est le joueur et le sujet réel est le jeu. Nous allons nous servir de la procuration pour interdire au joueur [2] de modifier le jeu. Si le joueur appelle la méthode `retirer` de la procuration, la procuration lèvera une exception `OperationInterditeException`. Pour les autres méthodes, la procuration appelle simplement l'opération correspondante du sujet réel. On note que l'utilisation de la procuration se fait sans aucune modification du client.

Enfin, la classe `Partie` permet de lancer un jeu des 13 allumettes entre deux joueurs. C'est la classe principale de l'application. Elle exploite les arguments de la ligne de commande pour configurer la partie à lancer. Les deux arguments désignent respectivement le premier et le second

1. En fait, avec un jeu aussi simple que les 13 allumettes, il suffirait de 1) mémoriser le nombre d'allumettes encore en jeu avant de communiquer le jeu au joueur et 2) vérifier que le nombre d'allumettes en jeu n'a pas changé après que le joueur a indiqué le nombre d'allumettes choisi. Pour un jeu plus compliqué, il peut être difficile de vérifier qu'il n'y a pas eu de changement apporté au jeu.

2. En fait, un tricheur connaissant bien Java pourrait quand même arriver à ses fins, même avec cette solution.

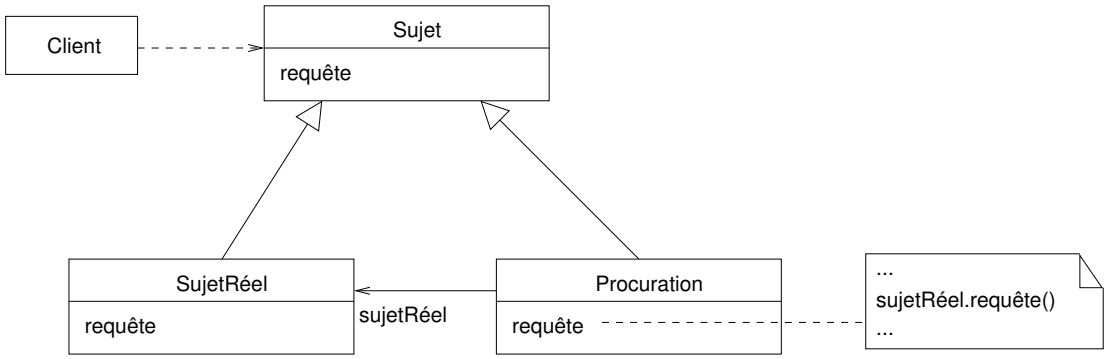

FIGURE 2 – Diagramme de classe du patron de conception Procuration

joueur et respectent le même format : `nom@stratégie`, où `nom` est le nom du joueur et `stratégie` est le nom de la stratégie à utiliser pour ce joueur. Les valeurs possibles pour la stratégie sont `humain`, `naif`, `rapide` ou `expert`. L'exécution donnée correspond au lancement suivant.

```
java allumettes.Partie Xavier@humain Ordinateur@naif
```

3 Contraintes de réalisation

Les contraintes de réalisation à respecter *impérativement* sont les suivantes :

C_1 Ce projet est un travail individuel. *Toute tricherie sera sanctionnée par la note minimale.*

C_2 Le projet doit fonctionner sans aucune modification sur les machines Unix des salles de TP du bâtiment C de l'ENSEEIHT.

C_3 Les principes énoncés en cours et étudiés en TD et TP doivent être respectés lors de la réalisation de ce projet.

C_4 La documentation des classes n'est pas explicitement demandée. Elle peut cependant être utile à la compréhension du travail fait.

C_5 Les lettres accentuées ne seront pas utilisées dans les identifiants.

C_6 Toutes les classes de l'application seront placées dans un unique paquetage appelé `allumettes`.

C_7 La solution proposée doit respecter le diagramme de classe de la figure 1. Bien entendu, toutes les classes de l'application ne sont pas représentées sur ce diagramme.

C_8 Pour demander à l'utilisateur le nombre d'allumettes qu'il souhaite prendre, on utilisera la classe `java.util.Scanner`.

C_9 Même si ici nous utiliserons toujours 13 allumettes, le jeu doit permettre d'avoir un nombre initial d'allumettes quelconque.

C_{10} Pour écrire la classe Partie, on utilisera la méthode `split` de `String` pour récupérer le nom et la stratégie de chacun des joueurs.

C_{11} Pour traiter la robustesse lors de l'interprétation des arguments de la ligne de commande, il est conseillé d'utiliser le mécanisme d'exception.

C_{12} Pour le niveau *naïf*, on utilisera la classe `java.util.Random`.

C_{13} On doit pouvoir ajouter de nouvelles stratégies pour le joueur sans modifier aucune des classes du diagramme de la figure 1, à l'exception bien sûr de la classe `Partie`. Par exemple, on pourrait ajouter une stratégie *lente* qui consiste à toujours prendre une seule allumette.

C_{14} La solution retenue doit permettre de changer en cours de partie la stratégie suivie par un joueur. Il n'est cependant pas demander d'implanter ce changement de stratégie.

C_{15} Les classes de tests unitaires utiliseront le suffixe Test et doivent s'appuyer sur JUnit.

C_{16} Les tests unitaires pour la stratégie *rapide* doivent être complets.

C_{17} Les tests unitaires pour les autres éléments de l'application ne sont pas demandés (mais sont certainement utiles pour détecter et éliminer les erreurs que vous auriez pu commettre).

C_{18} Écrire une classe ExempleTriche qui montre que l'arbitre sait détecter les tricheurs. Cette classe ne doit pas être interactive et n'a pas à utiliser JUnit.

C_{19} Les classes ou interfaces fournies ne doivent pas être modifiées à l'exception de la classe Partie qui doit bien sûr être complétée.

C_{20} On veillera à n'avoir que des méthodes courtes.

C_{21} Il est interdit d'avoir des répétitions imbriquées dans une même méthode.

4 Livrables

Vous devez rendre sur le SVN (voir descriptif en ligne du module) :

L_1 Le code source de vos programmes (et seulement le code source, ni les .class, ni la documentation au format html).

L_2 Le code source des programmes de test réalisés.

L_3 Le fichier texte `LISEZ-MOI.txt` qui contient des informations sur les choix réalisés et les informations jugées utiles pour comprendre le travail fait.

Les documents listés ci-dessus doivent être rendus en utilisant SVN.

5 Principales dates

Les principales dates concernant ce projet sont :

- 23 avril 2012 : distribution (et mise en ligne) du sujet. Le descriptif en ligne du module explique comment récupérer les fichiers fournis.
  Attention : Cette première version n'est pas noté. Cependant, vous pouvez avoir des points de pénalité (qui ne pourront donc pas être rattrapés avec la deuxième version) si le travail n'est pas fait sérieusement.
- 11 mai 2012 : remise des livrables (première version) ;
- semaine du 29 mai 2012 : retour de la première évaluation ;
- 8 juin 2012 : remise de la deuxième version.