

# 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.

```
1  Nombre d'allumettes restantes : 13
   Au tour de Xavier.
   Combien prenez-vous d'allumettes ? 2
   Xavier prend 2 allumettes.

6  Nombre d'allumettes restantes : 11
   Au tour de Ordinateur.
   Ordinateur prend 1 allumette.

   Nombre d'allumettes restantes : 10
11  Au tour de Xavier.
   Combien prenez-vous d'allumettes ? X
   Vous devez donner un entier.
   Combien prenez-vous d'allumettes ? 5
   Xavier prend 5 allumettes.
16  Erreur ! Prise invalide : 5 (> 3)
   Recommencez !

   Nombre d'allumettes restantes : 10
   Au tour de Xavier.
21  Combien prenez-vous d'allumettes ? 0
   Xavier prend 5 allumettes.
   Erreur ! Prise invalide : 0 (< 1)
   Recommencez !

26  Nombre d'allumettes restantes : 10
```

```
Au tour de Xavier.  
Combien prenez-vous d'allumettes ? 3  
Xavier prend 3 allumettes.  
  
31 Nombre d'allumettes restantes : 7  
   Au tour de Ordinateur.  
   Ordinateur prend 1 allumette.  
  
   Nombre d'allumettes restantes : 6  
36 Au tour de Xavier.  
   Combien prenez-vous d'allumettes ? 2  
   Xavier prend 2 allumettes.  
  
   Nombre d'allumettes restantes : 4  
41 Au tour de Ordinateur.  
   Ordinateur prend 2 allumettes.  
  
   Nombre d'allumettes restantes : 2  
   Au tour de Xavier.  
46 Combien prenez-vous d'allumettes ? 1  
   Xavier prend 1 allumette.  
  
   Nombre d'allumettes restantes : 1  
   Au tour de Ordinateur.  
51 Ordinateur prend 2 allumettes.  
   Erreur ! Prise invalide : 2 (> 1)  
   Recommencez !  
  
   Nombre d'allumettes restantes : 1  
56 Au tour de Ordinateur.  
   Ordinateur prend 1 allumette.  
   Ordinateur a perdu !  
   Xavier a gagné !
```

## 2 Architecture

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

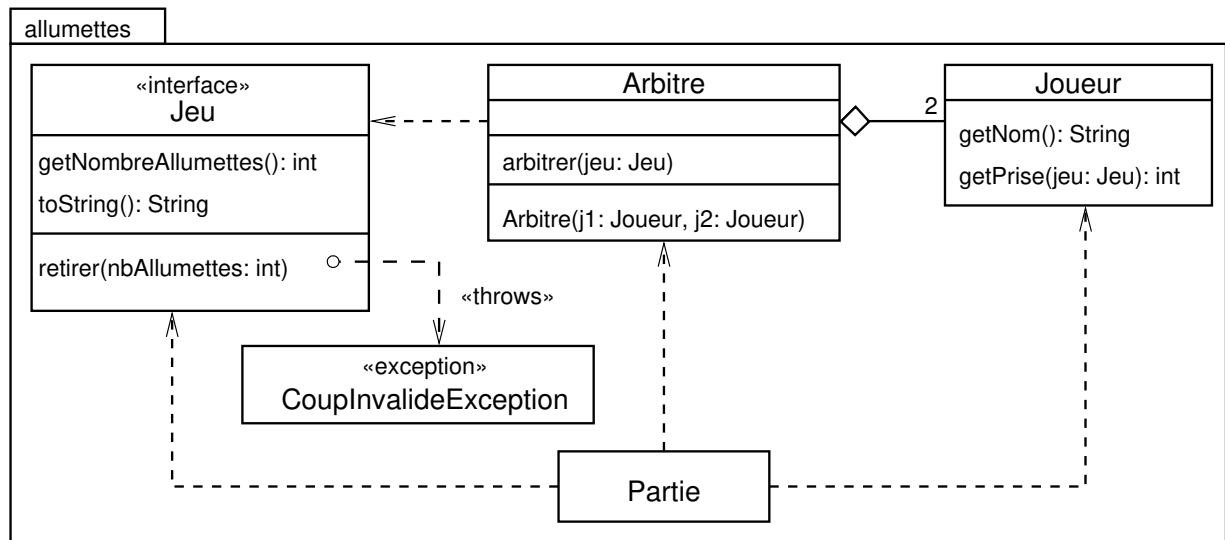


FIGURE 1 – Diagramme de classe d'analyse pour le jeu des 13 allumettes

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 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 *CoupeInvalideException*), 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 triche. Dans ce cas la partie devrait être immédiatement arrêtée et le joueur désigné comme tricheur !

On définira une stratégie de jeu qui correspond à un joueur qui triche. Voici un exemple de déroulement de partie entre un joueur *Ordinateur* qui applique une stratégie rapide et un *Tricheur* qui applique la stratégie tricheur.

```

Nombre d'allumettes restantes : 13
Au tour de Ordinateur.
Ordinateur prend 3 allumettes.

```

```

5 Nombre d'allumettes restantes : 10
   Au tour de Tricheur.
   Partie abandonnée car Tricheur a triché !

```

En l'état actuel de la modélisation, l'arbitre ne peut pas détecter<sup>1</sup> ce type de triche. 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<sup>2</sup> 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 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`, `expert` et `tricheur`. L'exemple d'exécution donné en début de sujet correspond au lancement suivant :

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

L'exemple de triche correspond à :

```
java allumettes.Partie Ordinateur@rapide Tricheur@tricheur
```

Un argument optionnel, forcément en première position, permet de rendre l'arbitre confiant. L'arbitre transmet alors directement le jeu au joueur sans utiliser le proxy. Cet argument optionnel est "-confiant".

Voici le résultat de l'exécution d'une partie avec un arbitre confiant et un joueur qui triche :

```
> java allumettes.Partie -confiant Ordinateur@rapide Tricheur@tricheur
```

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. Cette solution **ne doit pas** être mise en œuvre.

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

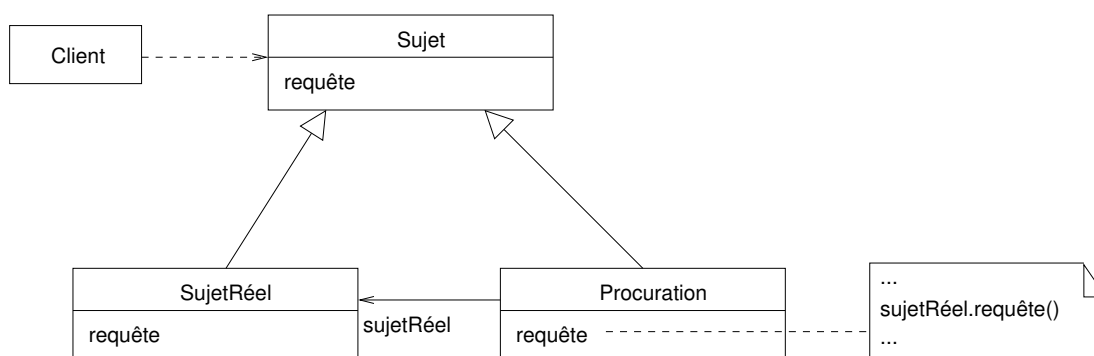


FIGURE 2 – Diagramme de classe du patron de conception Procuration

Nombre d'allumettes restantes : 13  
Au tour de Ordinateur.  
Ordinateur prend 3 allumettes.

Nombre d'allumettes restantes : 10  
Au tour de Tricheur.  
Tricheur prend 1 allumette.

Nombre d'allumettes restantes : 1  
Au tour de Ordinateur.  
Ordinateur prend 1 allumette.  
Ordinateur a perdu !  
Tricheur a gagné !

Que s'est-il passé ?

L'arbitre demande au joueur Ordinateur combien il prend d'allumettes. Ordinateur répond 3 et l'arbitre les retire du jeu. Il en reste 10.

L'arbitre demande au joueur Tricheur combien il prend d'allumettes. Tricheur commence par retirer des allumettes du jeu pour n'en laisser que deux. Il répond ensuite à l'arbitre qu'il en prend une. L'arbitre retire une allumette du jeu. Il en reste une seule.

Ordinateur est obligé de la prendre : Ordinateur perd et Tricheur gagne !

Si on exécute sans l'option "-confiant", l'arbitre détecte la triche et sanctionne le tricheur.

```
> java allumettes.Partie Ordinateur@rapide Tricheur@tricheur
```

```
Nombre d'allumettes restantes : 13  
Au tour de Ordinateur.  
Ordinateur prend 3 allumettes.
```

```
Nombre d'allumettes restantes : 10  
Au tour de Tricheur.  
Partie abandonnée car Tricheur a triché !
```

### 3 Contraintes de réalisation

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

- C<sub>1</sub> Ce projet est un travail individuel. *Toute triche sera sanctionnée par la note minimale.*
- C<sub>2</sub> Le projet doit fonctionner sans aucune modification sur les machines Unix des salles de TP du bâtiment C de l'ENSEEIH.
- C<sub>3</sub> Les principes énoncés en cours et étudiés en TD et TP doivent être respectés lors de la réalisation de ce projet.

- C<sub>4</sub> La documentation des classes n'est pas explicitement demandée. Elle peut cependant être utile à la compréhension du travail fait.
- C<sub>5</sub> Les lettres accentuées ne seront pas utilisées dans les identifiants.
- C<sub>6</sub> Toutes les classes de l'application seront placées dans un unique paquetage appelé `allumettes`.
- C<sub>7</sub> 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<sub>8</sub> Pour demander à l'utilisateur le nombre d'allumettes qu'il souhaite prendre, on utilisera la classe `java.util.Scanner`.
- C<sub>9</sub> Même si ici nous utiliserons toujours 13 allumettes, le jeu doit permettre d'avoir un nombre initial d'allumettes quelconque.
- C<sub>10</sub> Pour écrire la classe `Partie`, on utilisera la méthode `split` de `String` pour récupérer le nom et la stratégie de chacun des joueurs.
- C<sub>11</sub> 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<sub>12</sub> Pour le niveau *naïf*, on utilisera la classe `java.util.Random`.
- C<sub>13</sub> On doit pouvoir ajouter de nouvelles stratégies pour un joueur sans modifier aucune des classes de l'application à l'exception de la classe `Partie`. Par exemple, on pourrait ajouter une stratégie *lente* qui consiste à toujours prendre une seule allumette.
- C<sub>14</sub> La solution retenue doit permettre de changer en cours de partie la stratégie suivie par un joueur. Il n'est cependant pas demandé d'implanter ce changement de stratégie.
- C<sub>15</sub> Les classes de tests unitaires utiliseront le suffixe `Test` et doivent s'appuyer sur `JUnit`.
- C<sub>16</sub> Les tests unitaires pour la stratégie *rapide* doivent être complets.
- C<sub>17</sub> 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<sub>18</sub> L'IHM (interface homme machine) doit respecter les exemples donnés dans ce sujet. Ceci est impératif car votre programme sera testé en boîte noire, donc en s'appuyant seulement sur les saisies qu'il fait et les affichages qu'il produit.  
En particulier, les exemples présentés dans ce sujet doivent pouvoir être reproduits.
- C<sub>19</sub> Les classes ou interfaces fournies ne doivent pas être modifiées à l'exception de la classe `Partie` qui doit bien sûr être complétée.
- C<sub>20</sub> On veillera à n'avoir que des méthodes courtes.
- C<sub>21</sub> 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<sub>1</sub> Le code source de vos programmes (et seulement le code source, ni les .class, ni la documentation au format html).
- L<sub>2</sub> Le code source des programmes de test réalisés.
- L<sub>3</sub> Le fichier texte LISEZ-MOI.txt qui contient des informations sur les choix réalisés et les informations jugées utiles pour comprendre le travail fait.

## 5 Principales dates

Les principales dates concernant ce projet sont :

- 24 février 2020 : mise en ligne (et mise à disposition) du sujet. Le descriptif en ligne du module explique comment récupérer les fichiers fournis.
- vendredi 20 mars 2020 : remise des livrables (version complète pour première évaluation);  
**Attention :** Cette première version n'est pas notée. 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.
- avant le jeudi 2 avril 2020 : retour de la première évaluation ;
- vendredi 24 avril 2020 : remise de la deuxième version.