Jeu de Memory

1 Description fonctionnelle

Le Memory se joue à plusieurs joueurs avec un jeu de cartes composé de plusieurs familles comportant toutes le même nombre de cartes. On appelle *taille* de la famille le nombre de cartes qui la compose.

Initialement, les cartes sont battues puis distribuées sur un tapis, face cachée. Les joueurs, à tour de rôle, retournent successivement un nombre de cartes égal à la taille d'une famille. Tous les joueurs voient les cartes retournées. Si toutes les cartes retournées appartiennent à la même famille, le joueur prend les cartes et rejoue. Dans le cas contraire, les cartes sont de nouveau retournées pour que leur face ne soit plus visible.

La partie se termine quand toutes les cartes ont été prises. Le gagnant est le joueur qui a pris le plus de cartes (et donc constitué le plus grand nombre de familles).

Chaque joueur choisit les cartes à retourner en fonction d'une stratégie. Une stratégie *humain* consiste à demander à l'utilisateur du programme la carte à retourner. Il peut abandonner la partie en tapant ABANDON. La partie se continue alors sans lui. 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 :

— *naïf :* l'ordinateur n'a aucune mémoire et choisit toujours au hasard les cartes.
— *expert :* l'ordinateur se souvient de toutes les cartes retournées. Quand il peut constituer une famille, il le fait.
— *tricheur :* mode où l'ordinateur essaie de tricher. Tricher consiste à retourner discrètement les cartes. En connaissant la position des cartes, le joueur indélicat est sûr de gagner !

D'autres niveaux de jeu seraient possibles. Par exemple, on pourrait imaginer un niveau où le joueur ne mémorise que les cartes qu'il a lui-même retournées. Un autre où il mémorise une carte sur deux, etc. On pourrait aussi définir d'autres niveaux pour rendre le jeu de l'ordinateur plus réaliste ou plus efficace.

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 `Tapis` modélise un tapis de jeu. Le tapis gère des emplacements qui accueillent des cartes. Initialement toutes les cartes sont masquées (face cachée). Il est ensuite possible de montrer une carte (la rendre visible), la masquer (fache cachée) ou la prendre (elle ne pourra plus être ni montrée, ni masquée). Si la carte est visible, il est possible de la récupérer (get). Pour simplifier, on supposera que chaque emplacement est repéré par une position qui est un entier (0 pour le premier emplacement). Le nombre d'emplacements est défini lors de la création du tapis en fonction du jeu de cartes utilisé.

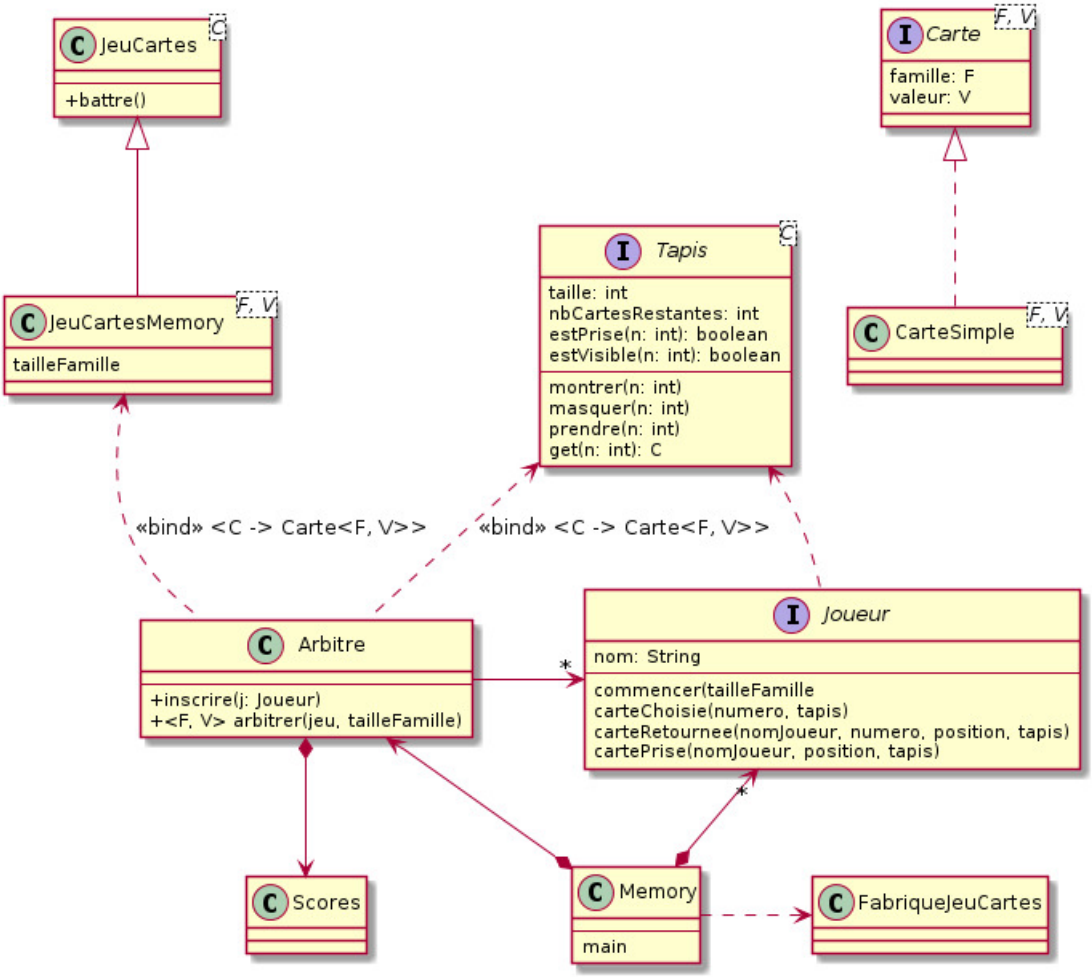

FIGURE 1 – Diagramme de classe d'analyse pour le jeu de Memory

Un jeu de cartes est une séquence (une liste) de cartes que l'on peut battre. Il n'y a pas de méthode pour accéder à une carte en particulier. En revanche, un jeu de cartes est « itérable ».

Pour jouer au Memory, on utilisera des cartes qui sont définies par une famille et une valeur dans la famille. Le type de la famille et de la valeur pouvant être quelconques, on utilise la généricité. Par exemple, on peut considérer que la famille est une lettre (A, B, C, etc.) et la valeur un entier (0, 1, 2, etc.). On peut aussi jouer au Memory avec jeu de 52 cartes. Dans ce cas la famille est la valeur de la carte (de l'as, 1, au roi, 13) et une couleur (rouge ou noir). Pour chaque famille, il y a deux cartes, par exemple le 8 de Pique et le 8 de Trèfle pour la famille 8 Noir.

L'arbitre est la classe qui gère une partie du jeu de Memory. C'est auprès de lui que chaque joueur s'inscrit. Un joueur est caractérisé par un nom. Il possède plusieurs méthodes qui permettront le dialogue avec l'arbitre au cours d'une partie. Le déroulement d'une partie est modélisé par la méthode « arbitrer » de l'arbitre (voir le diagramme de séquence de la figure 2). La méthode « arbitrer » prend en paramètre un jeu de cartes et la taille d'une famille. Elle commence par battre le jeu de carte puis crée le tapis. Elle informe tous les joueurs [1] que la partie commence en indiquant la taille d'une famille. Ensuite, tant qu'il y a encore des cartes sur le tapis, l'arbitre fait jouer le joueur courant. Il lui demande de retourner un nombre de cartes égal à la taille d'une famille. Pour chaque carte, il demande au joueur courant la carte qu'il souhaite retourner (le joueur choisira cette carte en fonction de sa stratégie), la rend visible sur le tapis et informe tous les joueurs que la carte a été retournée. Si toutes les cartes retournées sont de la même famille, elles sont retirées du jeu et le même joueur continue à jouer. Dans le cas contraire, les cartes sont masquées et l'arbitre passe au joueur suivant.

Dans le dialogue entre l'arbitre et le joueur, l'arbitre précise le numéro de la carte à retourner (`carteChoisie`) ou retournée (`carteRetournée`) (1, 2 puis 3 pour des familles de taille 3). Ainsi le joueur peut savoir si la carte qu'il avait proposée était valide ou non. Si un nouvel appel à `carteChoisie` est fait avec le même numéro, c'est que la carte proposée n'était pas valide (déjà prise, déjà retournée, etc.).

Tous les détails ne sont pas présentés sur le diagramme de séquence. En particulier, la gestion des scores n'apparait pas, l'abandon d'un joueur humain n'est pas décrit, ni une tentative de triche, etc.

Enfin, la classe Memory est la classe principale : elle crée, en fonction des arguments de la ligne de commande, les joueurs un jeu de carte, un arbitre et lui demande d'arbitrer une partie.

Dans la modélisation proposée, on constate que l'arbitre communique le tapis aux joueurs. Un joueur a ainsi accès au tapis. Il est donc facile de programmer une stratégie qui permet au joueur de tricher : il retourne une carte (montrer), récupère sa valeur (get) et la cache (masquer). Dans ce cas, le joueur devrait être identifié comme tricheur et être exclu de la partie.

En l'état actuel de la modélisation, l'arbitre ne peut pas détecter 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 3. 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 tapis. Nous allons nous servir de la procuration pour interdire au joueur [2]

1. L'étoile indique que la méthode « commencer » est appelée plusieurs fois, ici pour chaque joueur.
2. En fait, un tricheur connaissant bien Java pourrait quand même arriver à ses fins, même avec cette solution.

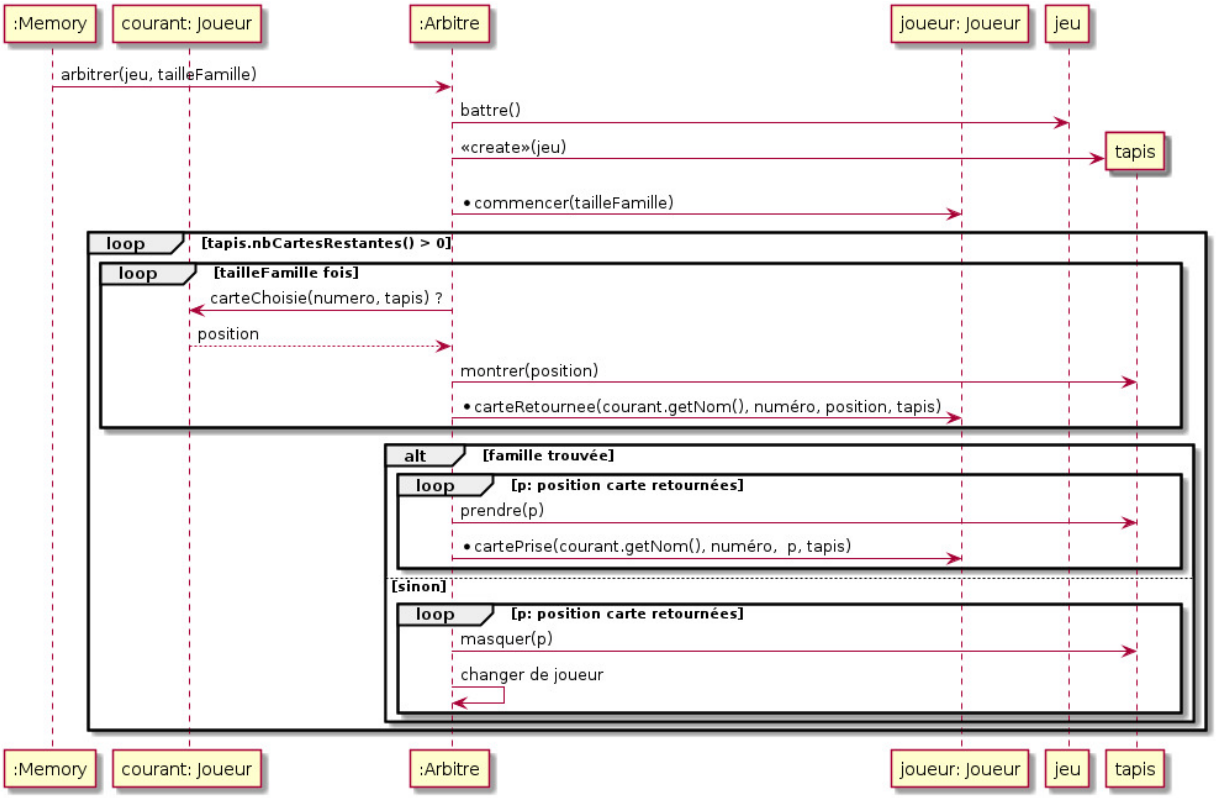

FIGURE 2 – Diagramme de séquence de Arbitre.arbitrer

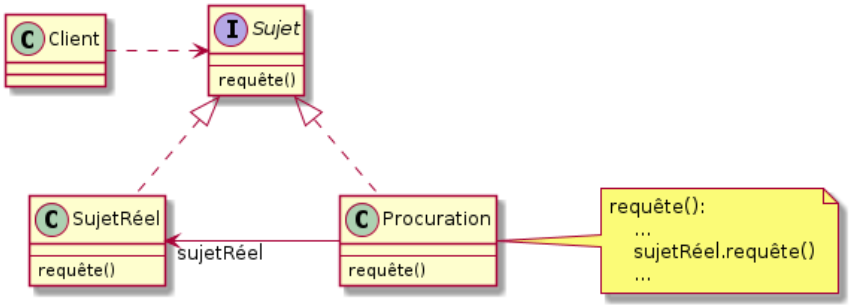

FIGURE 3 – Diagramme de classe du patron de conception Procuration

de modifier le tapis. Si le joueur appelle les méthodes `montrer`, `masquer` ou `prendre` 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.

3 Interface avec l'utilisateur

L'application à développer doit permettre à plusieurs joueurs de s'affronter au Memory. Lors de son lancement, l'utilisateur indiquera sur la ligne de commande, dans l'ordre :

1. la largeur de l'écran (un entier),
2. le nombre de famille (un entier),
3. la taille d'une famille (un entier),
4. les joueurs en lice. Chaque joueur est un argument de la ligne de commande qui doit respecter le 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`, `expert` et `tricheur`.

Une option `-confiant` ou `-mefiant` (toujours placée avant la description des joueurs) permettra de définir comment l'arbitre se comporte vis à vis des joueurs : s'il est confiant, il ne vérifie par si les joueurs trichent, s'il est méfiant il détecte les tricheurs, les signale et les exclut du jeu ! Par défaut, l'arbitre est considéré comme méfiant.

Le listing ci-après donne un extrait d'un exemple d'exécution correspondant à :

```
java Memory 100 6 3 Xavier@humain Simplet@naif Prof@expert
```

Pour suivre le déroulement de la partie, l'arbitre affiche ce qu'il fait. Tous les affichages sont donc réalisés par l'arbitre à l'exception de la demande au joueur humain "Position de la carte ..." (lignes 8, 12, 14, etc.) et de l'affichage "Appuyer sur Entrée..." (lignes 24, 33, etc.) qui sont gérés par le joueur humain. Les "Appuyer sur Entrée..." permettent à un joueur humain de voir les cartes retournées par les autres joueurs. Quand le joueur humain abandonne (ligne 68), il n'y a plus d'arrêt pour laisser voir le déroulement de la partie.

Les scores sont affichés dans l'ordre d'apparition des joueurs sur la ligne de commande.

java Memory 100 6 3 Xavier@humain Simplet@naif Prof@expert

```
1  Je mélange les cartes...
2  Je distribue les cartes...
3  La partie commence avec des familles de taille 3
4
5  C'est au tour de Xavier.
6   0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
7  10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
8  Position de la carte #1 à retourner : 0
9  Xavier retourne la carte #1 à la position 0 : B 0
10  0- B 0 |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
11 10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
12 Position de la carte #2 à retourner : 0
13 La carte 0 est déjà visible.
14 Position de la carte #2 à retourner : 20
15 La position 20 est invalide.
```

```
Position de la carte #2 à retourner : 11
Xavier retourne la carte #2 à la position 11 : D 2
 0- B 0 |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
10- xxx | 11- D 2 | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
Position de la carte #3 à retourner : 7
Xavier retourne la carte #3 à la position 7 : F 0
 0- B 0 |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- F 0 |  8- xxx |  9- xxx
10- xxx | 11- D 2 | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
Appuyer sur Entrée...
Xavier n'a pas trouvé de famille.

C'est au tour de Simplet.
 0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
Simplet retourne la carte #1 à la position 16 : C 1
 0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- C 1 | 17- xxx |
Appuyer sur Entrée...
Simplet retourne la carte #2 à la position 7 : F 0
 0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- F 0 |  8- xxx |  9- xxx
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- C 1 | 17- xxx |
Appuyer sur Entrée...
Simplet retourne la carte #3 à la position 5 : F 1
 0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- F 1 |  6- xxx |  7- F 0 |  8- xxx |  9- xxx
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- C 1 | 17- xxx |
Appuyer sur Entrée...
Simplet n'a pas trouvé de famille.

C'est au tour de Prof.
 0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
Prof retourne la carte #1 à la position 2 : D 1
 0- xxx |  1- xxx |  2- D 1 |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
Appuyer sur Entrée...
Prof retourne la carte #2 à la position 13 : A 0
 0- xxx |  1- xxx |  2- D 1 |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
10- xxx | 11- xxx | 12- xxx | 13- A 0 | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
Appuyer sur Entrée...
Prof retourne la carte #3 à la position 10 : C 2
 0- xxx |  1- xxx |  2- D 1 |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
10- C 2 | 11- xxx | 12- xxx | 13- A 0 | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
Appuyer sur Entrée...
Prof n'a pas trouvé de famille.

C'est au tour de Xavier.
 0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
Position de la carte #1 à retourner : 4
Xavier retourne la carte #1 à la position 4 : B 2
 0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- B 2 |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
Position de la carte #2 à retourner : ABANDON
Xavier abandonne !

C'est au tour de Simplet.
 0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
Simplet retourne la carte #1 à la position 9 : B 1
 0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- B 1
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
Simplet retourne la carte #2 à la position 15 : C 0
 0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- B 1
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- C 0 | 16- xxx | 17- xxx |
Simplet retourne la carte #3 à la position 8 : F 2
```

```
 0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- F 2 |  9- B 1
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- C 0 | 16- xxx | 17- xxx |
Simplet n'a pas trouvé de famille.

C'est au tour de Prof.
 0- xxx |  1- xxx |  2- xxx |  3- xxx |  4- xxx |  5- xxx |  6- xxx |  7- xxx |  8- xxx |  9- xxx
10- xxx | 11- xxx | 12- xxx | 13- xxx | 14- xxx | 15- xxx | 16- xxx | 17- xxx |
Prof retourne la carte #1 à la position 0 : B 0

...

Simplet retourne la carte #3 à la position 14 : E 2
 0- ~~~ |  1- E 1 |  2- ~~~ |  3- ~~~ |  4- ~~~ |  5- ~~~ |  6- ~~~ |  7- ~~~ |  8- ~~~ |  9- ~~~
10- ~~~ | 11- ~~~ | 12- E 0 | 13- ~~~ | 14- E 2 | 15- ~~~ | 16- ~~~ | 17- ~~~ |
Simplet a trouvé une famille, marque 3 points et rejoue.

La partie est finie. Voici les scores :
 - Xavier :  0
 - Simplet :  6
 - Prof : 12
```

D'autres exemples seront donnés sur la page du module.

4 Contraintes de réalisation

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

C_1 Ce projet est un travail individuel. *Toute triche 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 en utilisant la version 1.7 de Java (opton -source 1.7).

C_3 Les principes énoncés en cours et étudiés en TD et TP doivent être respectés.

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 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_7 Les classes ou interfaces fournies ne doivent pas être modifiées à l'exception des classes `JeuCartes` et `Arbitre` qui doivent être complétées.

C_8 Pour gérer les interactions avec l'utilisateur (saisies), on utilisera la classe `java.util.Scanner`.

C_9 Pour la stratégie *humain*, on indiquera un abandon en levant l'exception `AbandonException`.

C_{10} Lorsqu'un joueur humain abandonne ou qu'un joueur triche, il faut retourner les cartes qu'il avait choisies.

C_{11} Pour écrire la classe Memory, on utilisera la méthode `split` de `String` pour récupérer le nom et la stratégie de chacun des joueurs.

C_{12} 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_{13} Deux joueurs différents doivent avoir des noms différents.

C_{14} Pour obtenir un nombre aléatoire, on utilisera la classe `java.util.Random`.

C_{15} On doit pouvoir ajouter de nouvelles stratégies pour un joueur sans modifier aucune des classes du diagramme de la figure 1, à l'exception bien sûr de la classe `Memory`.

C_{16} 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_{17} Il est impératif d'utiliser l'API des collections (et de ne pas refaire ou réinventer ce qui existe).

C_{18} Il n'est pas demandé d'écrire des classes de tests. Elles peuvent toutefois être utiles pour détecter des erreurs dans votre programme sans attendre d'avoir l'application complète. Elles sont aussi utiles pour détecter qu'une modification n'introduit pas d'erreur.

C_{19} On veillera à n'avoir que des méthodes courtes.

C_{20} Il est interdit d'avoir des répétitions (for, do, while) imbriquées dans une même méthode.

5 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` avec les réponses aux questions posées et les choix réalisés ou toute information jugée utile pour comprendre le travail fait.

6 Principales dates

Les principales dates concernant ce projet sont :

— 1er avril 2016 : distribution (et mise en ligne) du sujet. Le descriptif en ligne du module explique comment récupérer les fichiers fournis.

— vendredi 15 avril avril 2016 : remise des livrables (première version) ;
Attention : Vous devez rendre une version complète. Le terme première version signifie que vous aurez un retour de votre enseignant de TP qui vous permettra d'améliorer cette première version.
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.

— semaine du 25 avril 2016 : retour de la première évaluation ;

— 6 mai 2016 : remise de la deuxième version.