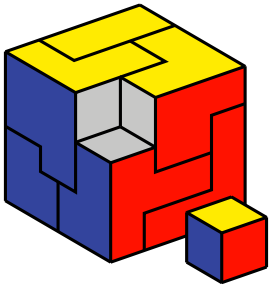
Prologin 2017



Tabula Prologina

Felix qui potuit cognoscere algorithmum

Sujet de la finale du Concours National d'Informatique
Samedi 29 avril 2017

Table des matières

Nouvelles découvertes en Alchimie & principes de manipulation de la table alchimique double par Joseph Eligius Rémi Le Marchand.

Très grand & très excellent Philosophe & célèbre Mathématicien, Prince des sectateurs Hermétiques & Algorithmiques.

Textes tirés de nombreux ouvrages du grand père de Joseph E. R. Le Marchand. Augmenté des notes manuscrites trouvées dans ceux-ci. Extraits de lettres reçues et envoyées par celui-là même, collectées par son aïeul. Nombre de textes originaux traduits du latin.

Auxquels on a ajouté en Appendice les Opinions certaines à l'attention de l'apprenti alchimiste et de son compilateur, personnes très-doctes.

À Paris - 29 avril 1771

C'est un point assuré plein d'admiration,
Que le haut & et le bas n'est qu'une même chose :
Pour faire d'une seule en tout le monde enclose,
Des effets merveilleux par adaptation.

D'un seul en a tout fait la méditation,
Et pour parents, matrice, & nourrice, on lui pose,
Phœbus, Diane, l'air, & la terre, où repose,
Cette chose en qui gît toute perfection.

Si on la mue en terre elle a sa force entière :
Séparant par grand art, mais facile manière,
Le subtil de l'épais, & la terre du feu.

De la terre elle monte au ciel, & puis en terre,
Du Ciel elle descend, recevant peu à peu,
Les vertus de tous deux qu'en son ventre elle enserre.

– – Table d'émeraude - Hermes Trismegiste

1 Préambule

La sagesse que j'ai reccueilli ici pour vous, mes disciples, ne doit jamais, *jamais*, sortir du cadre des leçons que je vous propose. Contrairement aux habitudes de mes pairs, j'ai décidé d'expliciter ici mes connaissances sans abstrus distraits et surtout, sans embrumer mon propos. Cela fait de cet ouvrage le recueil d'Alchimie le plus dangereux qui ai jamais été rédigé. Imaginez seulement si le profane mettait la main sur ces lignes !

J'ai élucidé pour vous de nombreux mystères de ce monde. Je suis parvenu à des miracles devant lesquels tant de mes confrères ont abandonné. J'ai réussi là où tant d'autres ont échoué. Ce n'est pas sans apréhension que je vous transmet aujourd'hui mes déductions, mais je ne puis plus continuer sans l'assistance précieuse d'un apprenti fiable.

J'ai l'honneur de pouvoir clamer que j'ai véritablement percé le mystère du *Magnum Opus* [1]. Je détient le secret de la panacée, et sa simplicité, son élégance, m'ont ébloui. Cette parcelle de divinité, je ne puis la partager avec quiconque. Lorsque l'Homme sera éclairé et la civilisation dépouillée de son impureté, je m'élèverai pour enfin lui confier *l'immortalité*. Je serai le Prométhée de l'ère moderne !

C'est pour cela que ces travaux sont si importants, et pour cela que je ne laisserai personne s'y opposer. Si le secret est jamais éventré, par n'importe lequel d'entre vous, soyez sûrs que ce ne sera pas oublié ni impuni. C'est pour cela que seul le meilleur d'entre vous se verra présenter le secret de la panacée. Pour juger des plus méritants, j'ai mit un point un petit exercice, et je consens pour cela de partager avec vous un autre secret : celui de la transmutation.

2 Transmutation

Nul n'ignore qu'il faut agir sur la *Materia Prima*, alors un métal vil, pour le transformer en or, car c'est là notre premier but. C'est cette opération que nous appelons la **transmutation**. De nombreux alchimistes renommés, et de nombreux charlatans également, se sont penchés sur ce sujet avec l'espoir vain d'y trouver la richesse.

Mais L'Alchimie est un art qui se veut pur, et quiconque tente de produire des métaux nobles pour son seul profit ne pourra qu'échouer. C'est pour cela, uniquement, que j'ai réussi là où de nombreux amateurs se sont vu vaincus.

1. Avec encore plus de caramel.

2.1 Méthode

Votre Maître préconise une transmutation rapide par *voie sèche* sur les trois métaux vils suivants : le **cuivre**, le **fer**, ou le **plomb**. L'ajout de l'alkahest au moment de l'œuvre au rouge et l'isolation selon le principe des feuillets paracelsiens entraînera la réaction métamorphique appropriée [2]. Agissez avec la prudence nécessaire — la difficulté de cette méthode réside dans la minutie dont vous devrez faire preuve.

Il vous faudra une quantité généreuse de *Materia Prima* pour un résultat satisfaisant. De plus, n'essayez pas d'user de plusieurs métaux ensemble. Leurs corps étant radicalement différents, leurs esprits ne peuvent se mêler et vous n'obtiendrez que rien d'autre que des cendres [3].

2.2 De la transmutation en métaux vils

La transmutation de *Materia Prima* en autres métaux vils est plus utile qu'on ne pourrait le croire, notamment lorsque la transmutation en or n'est pas directement possible. J'appelle cette action la *catalysation*, pour des raisons bien sûr évidentes [4]. À l'aide de **soufre** ou de **mercure** — éléments qui se doivent d'être familiers à tout alchimiste qui se respecte — il est possible de transmuter n'importe quel métal vil en un autre métal vil grâce à la sublimation des éléments sus-cités [5]. Usez de ce savoir avec parcimonie et sagesse.

3 Pour de bonnes pratiques de l'alchimie

Qui dit Alchimie parle d'une Science dangereuse et complexe. Les accidents sont nombreux pour ceux qui s'y essayent sans prudence et mesure. Voici quelques conseils que je puis vous prodiguer pour réaliser votre tâche le mieux qu'il soit :

- Les gants protègeront le cuir du maladroit.
- Le sceau de Salomon n'est pas un pentacle et ne vous permettra pas d'invoquer Satan.
- Celui qui manque de sommeil, commetra sottise sans pareil.
- La belle Œuvre naît de l'harmonie entre un travail conscencieux et un repos légitime.

2. *"Mais ça ne veut rien dire !"* s'exclamme votre voisin, perdu.
3. Et le goût amer de l'échec.
4. *"Pourquoi évidentes ?"* s'écrie votre voisin, sanglotant.
5. À ce stade, votre voisin émotif pleure franchement sur son livret. Vous vous sentez un peu mal pour lui, surtout parce que franchement, vous n'avez pas comprit grand chose non plus.

4 Manuel

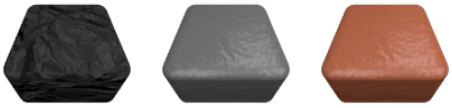
4.1 Établi

Chaque établi peut être représenté par une grille carrée de 6 cases de côté. Chaque apprenti a son propre établi.

4.1.1 Élément

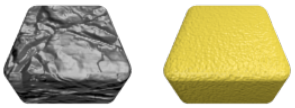
Les éléments sont les matières premières mises à votre disposition. Il y en a cinq :

— Le *plomb*, le *fer* et le *cuivre* sont les trois métaux que vous pouvez transmuter en or. Les trois ont le même rendement, mais ne peuvent pas être mélangés.



Plomb Fer Cuivre

— Le *mercure* et le *soufre* sont les deux éléments volatiles qui permettront de catalyser un élément vers un autre élément.



Mercure Soufre

4.1.2 Échantillon

Un échantillon est une paire d'éléments. Il peut être composé de deux éléments identiques ou distincts.

4.2 Déroulement d'une partie

Une partie dure 150 tours. Chaque apprenti jouera donc 75 tours chacun.

Initialement les deux établis sont vides. Le premier apprenti, choisi au hasard, disposera d'un échantillon par défaut composé de plomb et de fer.

Le vainqueur sera désigné comme étant l'apprenti ayant amassé le plus d'or pendant la partie.

4.3 Déroulement d'un tour

Au début de votre tour, vous recevrez l'échantillon créé par votre adversaire pour vous au tour précédent. Vous pouvez réaliser les actions ci-dessous dans n'importe quel ordre.

4.3.1 Actions

Donner un échantillon Pour inculper la solidarité et exacerber l'entraide entre les apprentis, il leur est demandé de se rendre service et de ne jamais se servir soi-même dans la réserve.

À chaque tour N, un apprenti crée un échantillon pour son adversaire, échantillon qui sera posé au tour N+1. Si vous oubliez de créer cet échantillon, votre adversaire recevra un échantillon identique à celui qu'il a lui-même créé au tour précédent. Par souci d'égalité, chaque échantillon doit avoir au minimum *un élément en commun* avec l'échantillon reçu au tour précédent. Par exemple, si l'apprenti A confie un échantillon de plomb et de mercure à l'apprenti B, l'apprenti B devra lui rendre un échantillon contenant un élément de son choix, et soit du mercure, soit du plomb.

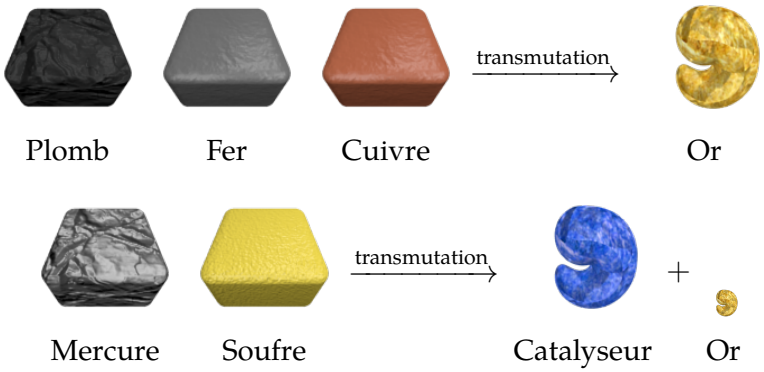
Poser un échantillon Une fois l'échantillon en main, vous devez délicatement le déposer sur votre établi.

— Les deux éléments composant l'échantillon doivent êtres placés sur des emplacements adjacents.
— Un élément au moins de l'échantillon doit être posé à un emplacement adjacent à un élément de même type.
— Si il n'y a aucun élément identique entre l'échantillon et ce qui est déjà posé sur l'établi, l'échantillon peut être posé n'importe où.
— Si il n'y a pas d'emplacement valide, vous serez obligés de transmuter une zone pour libérer de la place sur l'établi, et il ne vous reste plus qu'à assassiner discrètement votre compère qui - n'en doutez pas ! - savait pertinemment quel tour il était en train de vous jouer.

Si vous ne posez pas d'échantillon durant le tour, l'intégralité de votre plateau sera vidé à la fin de votre tour, alors n'oubliez pas !

Transmuter Lorsque vous possédez une zone d'un certain élément, vous pouvez lancer une *transmutation*.

Les métaux (cuivre, fer ou plomb) sont sujets à une transmutation *forte* qui produit beaucoup d'or. Le mercure et le soufre sont quand eux sujets à une transmutation *faible* qui produit principalement des catalyseurs, ainsi qu'une quantité faible d'or.



Une zone est une région de cases adjacentes contenant le même élément. Plus cette zone est grande, plus vous obtiendrez d'or (ou de catalyseurs). Une zone peut être définie uniquement par une case en faisant partie, puisque toutes les cases adjacentes du même type (et récursivement) en feront alors partie.

Vous pouvez transmuter un unique bloc de métal. Néanmoins, cela revient à détruire ce bout d'élément et donc à perdre de l'or - stratégie aussi audacieuse que dangereuse.

Une transformation forte d'une zone de taille t fait perdre 3 d'or si $t = 1$, sinon elle en fait gagner $\lfloor \frac{t^2}{4} \rfloor - 1$.

Une transformation faible d'une zone de taille t fait perdre 3 d'or si $t = 1$, sinon elle en fait gagner $t - 2$, et fait gagner $\lfloor \frac{t-1}{2} \rfloor$ catalyseurs.

Taille	1	2	3	4	5	6	7	8	9	10	..
Or (forte)	-3	0	1	3	5	8	11	15	19	24	
Or (faible)	-3	0	1	2	3	4	5	6	7	8	
Catalyseurs	0	0	1	1	2	2	3	3	4	4	

Catalyser Lorsque vous possédez une zone suffisamment large de catalyseur (mercure ou soufre), vous pouvez sublimer ces éléments, les faisant ainsi disparaître de l'établi. Le catalyseur obtenu vous permettra de transformer un élément quelconque de l'établi en un autre élément de votre choix. Plus la zone sublimée est grande, plus vous aurez de catalyseur et plus vous pourrez transformer de cases. Transmuter une zone de catalyseurs vous donnera également un peu d'or. Rien ne vous empêche de catalyser un élément de l'établi d'un autre apprenti, tant que celui-ci a le dos tourné…

Si vous n'utilisez pas vos catalyseurs ce tour-ci, en entier, ils se volatiseront et il n'en restera rien au tour suivant.

4.3.2 Score

Votre score est déterminé par la quantité d'or que vous serez parvenu à transmuter en `NB_TOURS`.

5 Tournois

5.1 Tournois intermédiaires

Afin de vous aider à perfectionner vos algorithmes, des tournois intermédiaires vous seront proposés toutes les six heures environ. Ces matchs n'ont absolument aucune influence sur le classement final, mais sont néanmoins à prendre au sérieux car ils vous permettront de vous situer par rapport aux autres joueurs, de connaître vos ennemis, vos points forts et vos faiblesses, et vous donneront des pistes pour vous améliorer pendant la finale.

Les tournois se dérouleront aux horaires suivants :

— Samedi 15 h 42 (tournoi de test)
— Samedi 17 h 42
— Samedi 23 h 42
— Dimanche 5 h 42
— Dimanche 11 h 42
— Dimanche 17 h 42
— **Lundi 00 h 42 (rendu final)**

À chacun des horaires indiqués ci-dessous, nous prendrons le dernier champion que chaque candidat aura envoyé sur le site de soumission pour le faire participer au tournoi, et nous vous donnerons les résultats ainsi que votre progression dès que les tournois se seront terminés, avec un récapitulatif de votre progression globale.

Les tournois seront exécutés sur des cartes officielles de notre choix, qui seront potentiellement amenées à changer au fur et à mesure.

5.2 Rendu final

Le rendu final est le seul rendu qui comptera pour le classement. Les mêmes règles s'appliquent : le dernier champion soumis à l'heure du début du tournoi sera le champion utilisé pour le tournoi final.

Lors du tournoi final, plusieurs cartes seront ajoutées, qui resteront inconnues de tous les joueurs à l'avance, afin de mesurer l'adaptabilité de vos algorithmes à des situations inconnues.

Pour le rendu final, nous vous demandons de rajouter des commentaires qui résument le fonctionnement des différents blocs logiques de votre code, ainsi qu'un **commentaire global en haut de votre fichier principal** qui détaille votre stratégie ainsi que les différents algorithmes que vous avez employés pour l'implémenter.

6 Considérations techniques

Vous disposez d'une seconde (temps réel !) à chaque fois qu'une de vos fonctions est appelée pour rendre la main. Passé ce délai, votre programme est tué, le match continue sans vous et vos fonctions ne sont plus appelées. Il n'est pas possible de revenir en jeu tout simplement parce qu'il n'y a aucun moyen de rétablir l'état des environnements des langages après une interruption. Les limites de mémoire sont faites avec des `cgroups`, ce qui fait que l'allocation échouera si vous essayez de dépasser la limite qui vous est accordée. Cette limite compte aussi la taille de la pile.

D'autres limitations sont appliquées :

— le système de fichiers est entièrement en lecture seule ;
— seuls `/usr`, `/var` et `/tmp` sont montés ;
— vous n'avez pas le droit d'utiliser des processus en parallèle ;
— la mémoire est limitée à 500 Mio ;
— la taille totale de votre output ne doit pas dépasser 256 Kio (elle sera tronquée à partir de cette limite) ;
— le temps d'exécution total du processus est limité à 300 secondes de temps réel ;
— chaque appel de fonction est limité à une seconde de temps réel plus 500 millisecondes de marge pour prendre en compte le surcoût de sérialisation/désérialisation des valeurs depuis et vers les langages cibles.

7 API

Constante : TAILLE_ETABLI
Valeur : 6
Description : Taille de l'établi de travail (longueur et largeur)

Constante : NB_TOURS
Valeur : 150
Description : Nombre de tours à jouer avant la fin de l'affrontement

Constante : NB_TYPE_CASES
Valeur : 6
Description : Taille de l'énumération "case_type"

- **case_type**

Description : Types de cases
Valeurs :

VIDE :	Case vide
PLOMB :	Plomb ; transmutable en or
FER :	Fer ; transmutable en or
CUIVRE :	Cuivre ; transmutable en or
SOUFRE :	Soufre ; transmutable en catalyseur
MERCURE :	Mercure ; transmutable en catalyseur

- **element_propriete**

Description : Types de propriétés des éléments
Valeurs :

AUCUNE :	Les cases vides ne contiennent pas d'élément, et n'ont donc aucune propriété
TRANSMUTABLE_OR :	Élement transmutable en or
TRANSMUTABLE_CATALYSEUR :	Élément transmutable en catalyseur

- **erreur**

Description : Erreurs possibles

Valeurs :

OK :	L'action a été exécutée avec succès
POSITION_INVALIDE :	La position spécifiée n'est pas sur l'établi
PLACEMENT_INVALIDE :	Les deux positions ne correspondent pas à des cases adjacentes
PLACEMENT_IMPOSSIBLE :	Les cases ciblées ne sont pas vides
PLACEMENT_INCORRECT :	Un des deux éléments de l'échantillon doit être placé adjacent à un élément du même type déjà présent sur l'établi
CASE_VIDE :	La case ciblée est vide
ECHANTILLON_INCOMPLET :	L'échantillon doit contenir deux éléments.
ECHANTILLON_INVALIDE :	L'échantillon doit contenir au moins un des éléments de l'échantillon reçu auparavant
AUCUN_CATALYSEUR :	Aucun catalyseur disponible
CATALYSE_INVALIDE :	L'élément de destination ne peut pas être vide.
DEJA_POSE :	L'échantillon a déjà été posé ce tour-ci
DEJA_DONNE :	L'échantillon a déjà été donné ce tour-ci

- **action_type**

Description : Types d'actions

Valeurs :

ACTION_PLACER :	Action "placer_echantillon"
ACTION_TRANSMUTER :	Action "transmuter"
ACTION_CATALYSER :	Action "catalyser"
ACTION_DONNER_ECHANTILLON :	Action "donner_echantillon"

- **position**

```
struct position {
    int ligne;
    int colonne;
};
```

Description : Position sur la carte, donnée par deux coordonnées

Champs :

ligne :	Coordonnée : ligne
colonne :	Coordonnée : colonne

- **echantillon**

```
struct echantillon {
    case_type element1;
    case_type element2;
```

```
};
```

Description : Échantillon, défini par deux types d'éléments
Champs : *element1* : Élément 1
element2 : Élément 2

- **position_echantillon**

```
struct position_echantillon {
    position pos1;
    position pos2;
};
```

Description : Position d'un échantillon, donnée par deux positions adjacentes
Champs : *pos1* : Position de l'élément 1 de l'échantillon
pos2 : Position de l'élément 2 de l'échantillon

- **action_hist**

```
struct action_hist {
    action_type atype;
    position pos1;
    position pos2;
    int id_apprenti;
    case_type nouvelle_case;
};
```

Description : Action représentée dans l'historique. L'action "placer_echantillon" utilise "pos1" et "pos2". L'action "transmuter" utilise "pos1". L'action "catalyser" utilise "pos1", "id_apprenti" et "nouvelle_case". L'action "donner_echantillon" n'est pas représentée dans l'historique, car "echantillon_tour" donne l'information.
Champs : *atype* : Type de l'action
pos1 : Position, pour les actions placer (1er élément), transmuter et catalyser
pos2 : Position, pour l'action placer (2e élément)
id_apprenti : ID de l'apprenti, pour l'action catalyser
nouvelle_case : Élément pour l'action catalyser

• **placer_echantillon**

```
erreur placer_echantillon(position pos1, position pos2)
```

Description : Place l'échantillon du tour sur l'établi, avec les coordonnées de deux cases adjacentes.

Parametres :
pos1 : Case du terrain où doit être posé le premier élément de l'échantillon
pos2 : Case du terrain où doit être posé le second élément de l'échantillon

• **transmuter**

```
erreur transmuter(position pos)
```

Description : Provoque la transformation chimique de l'élément à la case ciblée, ainsi que tous les éléments adjacents du même type, ceux du même type adjacents à ces derniers, etc. Ils disparaissent alors tous dans leur transmutation en or ou en catalyseur.

Parametres :
pos : Case de l'établi dont la région doit être activée

• **catalyser**

```
erreur catalyser(position pos, int id_apprenti, case_type terrain)
```

Description : Utilise un catalyseur sur la case ciblée de l'apprenti indiqué. Transforme l'ancien élément en l'élément indiqué.

Parametres :
pos : Case de l'élément qui doit être transmuté
id_apprenti : Identifiant de l'apprenti dont l'élément est ciblé
terrain : Type d'élément qui doit remplacer l'ancien

• **donner_echantillon**

```
erreur donner_echantillon(echantillon echantillon_donne)
```

Description : Définit l'échantillon que l'adversaire recevra à son prochain tour.

Parametres :
echantillon_donne : Échantillon que l'adversaire recevra à son prochain tour

• **type_case**

```
case_type type_case(position pos, int id_apprenti)
```

Description : Renvoie le type d'une case donnée, ou 0 si la case est invaide.
Parametres : *pos* : Case choisie
id_apprenti : Apprenti choisi

• **est_vide**

```
bool est_vide(position pos, int id_apprenti)
```

Description : Indique si une case donnée est vide ou contient un élément. Renvoie faux en cas d'erreur.
Parametres : *pos* : Case choisie
id_apprenti : Apprenti choisi

• **propriete_case**

```
element_propriete propriete_case(position pos, int id_apprenti)
```

Description : Renvoie la propriété de l'élément sur une case donnée. Un élément invalide n'a pas de propriété.
Parametres : *pos* : Case choisie
id_apprenti : Apprenti choisi

• **propriete_case_type**

```
element_propriete propriete_case_type(case_type ctype)
```

Description : Renvoie la propriété d'un type de case donné.
Parametres : *ctype* : Type de case

• **taille_region**

```
int taille_region(position pos, int id_apprenti)
```

Description : Renvoie la taille de la région à laquelle appartient un élément. Renvoie -1 si la position est invalide.
Parametres : *pos* : Case choisie
id_apprenti : Apprenti choisi

• **positions_region**

```
position array positions_region(position pos, int id_apprenti)
```

Description : Renvoie la liste des positions des cases composant la région à laquelle appartient un élément donné. Renvoie une liste vide en cas d'erreur.

Parametres :
pos : Case choisie
id_apprenti : Apprenti choisi

• **placement_possible_echantillon**

```
bool placement_possible_echantillon(echantillon echantillon_a_placer, position pos1, posit
```

Description : Détermine si le placement d'un échantillon est valide.

Parametres :
echantillon_a_placer : Échantillon à placer
pos1 : Case du terrain où doit être posé le premier élément de l'échant
pos2 : Case du terrain où doit être posé le second élément de l'échantil
id_apprenti : Apprenti possédant l'établi où poser l'échantillon

• **placements_possible_echantillon**

```
position_echantillon array placements_possible_echantillon(echantillon echantillon_a_place
```

Description : Renvoie la liste des placements possibles pour un échantillon donné sur l'établi d'un apprenti donné. Renvoie une liste vide en cas d'erreur.

Parametres :
echantillon_a_placer : Échantillon à placer
id_apprenti : Apprenti possédant l'établi où poser l'échantillon

• **historique**

```
action_hist array historique()
```

Description : Renvoie la liste des actions jouées par l'adversaire pendant son tour, dans l'ordre chronologique.

• **moi**

```
int moi()
```

Description : Renvoie votre numéro d'apprenti.

• **adversaire**

```
int adversaire()
```

Description : Renvoie le numéro d'apprenti de votre adversaire.

• **score**

```
int score(int id_apprenti)
```

Description : Renvoie la quantité d'or amassée par l'apprenti désigné par le numéro "id_apprenti". Renvoie 0 si "id_apprenti" est invalide (attention, le score d'un apprenti valide peut aussi être 0).

Parametres : *id_apprenti* : Identifiant de l'apprenti

• **tour_actuel**

```
int tour_actuel()
```

Description : Renvoie le numéro du tour actuel.

• **annuler**

```
bool annuler()
```

Description : Annule la dernière action. Renvoie "false" quand il n'y a pas d'action à annuler ce tour-ci.

• **nombre_catalyseurs**

```
int nombre_catalyseurs()
```

Description : Indique le nombre de catalyseurs en votre possession.

• **echantillon_tour**

```
echantillon echantillon_tour()
```

Description : Indique l'échantillon reçu pour ce tour.

• **a_pose_echantillon**

```
bool a_pose_echantillon()
```

Description : Indique si l'échantillon reçu pour ce tour a déjà été posé.

• **a_donne_echantillon**

```
bool a_donne_echantillon()
```

Description : Indique si un échantillon a déjà été donné ce tour.

• **quantite_transmutation_or**

```
int quantite_transmutation_or(int taille_region)
```

Description : Renvoie la quantité d'or (et donc le score) obtenue par la transmutation de "taille_region" éléments transmutables en or.

Parametres : *taille_region* : Nombre d'éléments d'une région à transmuter

• **quantite_transmutation_catalyseur**

```
int quantite_transmutation_catalyseur(int taille_region)
```

Description : Renvoie la quantité de catalyseurs obtenue par la transmutation de "taille_region" éléments transmutables en catalyseur.

Parametres : *taille_region* : Nombre d'éléments d'une région à transmuter

• **quantite_transmutation_catalyseur_or**

```
int quantite_transmutation_catalyseur_or(int taille_region)
```

Description : Renvoie la quantité d'or obtenue par la transmutation de "taille_region" éléments transmutables en catalyseur.

Parametres : *taille_region* : Nombre d'éléments d'une région à transmuter

• **echantillon_defaut_premier_tour**

```
echantillon echantillon_defaut_premier_tour()
```

Description : Indique l'échantillon par défaut lors du premier tour

• **afficher_etablis**

```
void afficher_etablis()
```

Description : Affiche l'état actuel des deux établis dans la console.

8 Notes sur l'utilisation de l'API

8.1 C

— Les booléens sont représentés par le type `bool`, défini par le standard du C99, et que l'on retrouve dans le header `stdbool.h`;
— Les fonctions prenant des tableaux en paramètres et retournant des tableaux utilisent à la place de ces tableaux une structure `type_array`, où `type` est le type des données dans le tableau. Ces structures contiennent deux éléments : les données, `type* datas`, et la taille, `size_t size`. Dans tous les cas, la libération des données est laissée au soin du candidat;
— Tout le reste est comme indiqué dans le sujet.

8.2 C++

— Les tableaux sont représentés par des `std::vector<type>`;
— Le reste est identique au sujet.

8.3 C#

— Les fonctions à utiliser sont des méthodes statiques de la classe `Api`. Ainsi, pour utiliser la fonction `Foo`, il faut faire `Api.Foo`;
— Les noms des fonctions, structures et énumérations sont en `CamelCase`. Ainsi, une fonction nommée `foo_bar` dans le sujet s'appellera `FooBar` en C#.

8.4 Haskell

— L'API est fournie par le module `Api`.
— Les énumérations sont représentées par des types sommes, les structures par des records. Seule la première lettre des noms de types et de constructeurs est en majuscule. Le nom du constructeur d'une structure est son nom de type.
— La commande `make doc` permet de générer la documentation dans le fichier `doc/index.html` pour votre code ainsi que pour l'API.
— Pour pouvoir conserver des valeurs entre différents appels à vos fonctions à compléter, il faut utiliser des variables mutables :

```
import Data.IORef
import System.IO.Unsafe (unsafePerformIO)

-- La pragma NOINLINE est importante !
-- MonType ne doit pas etre polymorphe !
{-# NOINLINE maVariable #-}
```

```
maVariable :: IORef MonType
maVariable = unsafePerformIO (newIORef maValeurInitiale)

fonctionACompleter :: IO ()
fonctionACompleter = do
  maValeur <- readIORef maVariable
  ...
  writeIORef maVariable maValeur'
```

8.5 Java

— Les fonctions à utiliser sont des méthodes statiques de la classe `Interface`. Ainsi, pour utiliser la fonction `foo`, il faut faire `Interface.foo;`
— Les structures sont représentées par des classes dont tous les attributs sont publics.

8.6 OCaml

— L'API est fournie par le fichier `api.ml`, qui est `open` par défaut par le fichier à compléter;
— Les énumérations sont représentées par des types sommes avec des constructeurs sans paramètres. Seule la première lettre des noms des constructeurs est en majuscule;
— Les structures sont représentées par des records, sauf pour la structure `position` qui est représentée par un couple `int * int;`
— Les tableaux sont représentés par des `array` Caml classiques.

8.7 PHP

— Les constantes sont définies via des `define` et doivent donc être utilisées sans les précéder d'un signe dollar;
— Les énumérations sont définies comme des séries de constantes. Se référer à la puce au-dessus;
— Les structures sont gérées sous forme de tableaux associatifs. Ainsi, une structure contenant un champ `x` et un champ `y` sera créée comme ceci : `array('x' => 42, 'y' => 1337)`.

8.8 Python

— L'API est fournie par le module `api`, dont tout le contenu est importé par défaut par le code à compléter;

— Les énumérations sont représentées par des `IntEnum` Python, qui peuvent-être utilisés comme ceci : `nom_enum.CHAMP.`;
— Les structures sont représentés par des `namedtuple` Python, dont on peut accéder aux champs via la notation pointée habituelle, et qui peuvent être créés comme ceci : `foo(bar=42, x=3)`, sauf pour la structure `position` qui est représentée par un couple (x, y).

8.8.1 Python 2

La version de Python par défaut est Python 3. Cependant, si vous souhaitez toujours coder en Python 2, vous pouvez utiliser le dossier *python2* à la place du dossier *python*.

Les différences par rapport à Python 3 sont les suivantes :

— les constantes des énumérations sont représentées par de simples entiers ;
— les gens dans la rue vous jettent des tomates et crient au scandale.