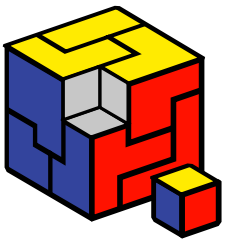
Prologin 2015



Noosphère

Vous êtes connecté.

Sujet de la finale du Concours National d'Informatique
Vendredi 8 mai 2015

Table des matières

1 Introduction **3**

2 Votre mission **4**

3 Prologin 2015 : Briefing détaillé **4**
3.1 Synthèse exécutive 4
3.2 État de la simulation 5
3.2.1 Carte 5
3.2.2 Portails, liens et champs 5
3.2.3 Agent 6
3.3 Déroulement d'un tour 6
3.3.1 Déplacement 6
3.3.2 Actions sur des portails 7
3.4 Score 8
3.4.1 Format de la carte 9

4 Conclusion **10**

5 API **11**

6 Notes sur l'utilisation de l'API **23**
6.1 C 23
6.2 C++ 23
6.3 C# 23
6.4 Haskell 23
6.5 Java 24
6.6 OCaml 24
6.7 PHP 24
6.8 Python 25
6.8.1 Python 2 25


DE QUEL CÔTÉ ÊTES-VOUS ?

1 Introduction

Power is in tearing human minds to pieces and putting them together again in new shapes of your own choosing. – 1984, Georges Orwell

La salle était pleine. Le Candidat, jetant des regards perdus à ses voisins impassibles, essayait tant bien que mal de se rappeler comment il s'était retrouvé dans cette situation.

Écoutant d'une oreille hagarde les instructions de l'Agent en fixant le dossier confidentiel qui venait d'être déposé sur son bureau, le Candidat se remémorait toutes les épreuves qui l'avaient conduites jusqu'ici. Qui aurait pu croire que remplir un simple questionnaire d'un prétendu concours d'informatique…

« Vous avez été choisis. »

L'Agent avait attiré l'attention du Candidat. Beaucoup de questions lui rongeaient l'esprit. Pour quelle raison avait-on convoqué à une heure aussi matinale dans un lieu secret une centaine de jeunes étudiants, ayant vraisemblablement des compétences en informatique hors du commun ? Pourquoi prendre autant de précautions pour une simple présentation ? Mais surtout, que pouvait bien renfermer ce mystérieux dossier ?

D'une main hésitante, le Candidat ouvrit la première page…

2 Votre mission

Le monde qui vous entoure n'est pas comme vous le pensez [1]

Il y a plusieurs années, en analysant le champ électromagnétique de la planète, une agence gouvernementale a découvert que certains points de la planète, dits **Portails** se comportaient de façon anormale. Des études supplémentaires ont permis de déterminer que ces portails pouvaient servir de supports à la création d'un gigantesque réseau de champs de manipulation mentale, ou *champs de contrôle*.

Afin d'empêcher une attaque terroriste sur le réseau de portails, l'Agence cherche à garder un contrôle absolu sur celui-ci. L'étendue immense du réseau de portails ne permettant bien évidemment pas aux opérations de contrôle d'être planifiées par des humains, il est nécessaire de recourir à des Intelligences Artificielles dans ce but.

C'est là que vous intervenez.

L'Agence a corrompu et pris le contrôle du Concours National d'Informatique pour recruter 100 jeunes [2] génies en informatique. Vous aurez pour mission [3] de créer une intelligence artificielle capable de planifier les opérations de manipulation de champs de contrôle sur terre.

À l'issu d'un tournoi visant à départager vos intelligences artificielles, l'algorithme vainqueur sera utilisé pour coordonner les agents sur le terrain.

3 Prologin 2015 : Briefing détaillé

3.1 Synthèse exécutive

Votre objectif est de créer une intelligence artificielle pour une simulation d'une guerre de contrôle mental d'un territoire par l'utilisation de portails.

Cette simulation prend la forme d'un jeu de stratégie en tour par tour qui se déroule sur une carte sur laquelle sont présents des portails.

1. Contrairement à une croyance populaire, la terre est en forme de langouste et vibre.
2. Et donc manipulables.
3. Que vous accepterez, sinon pas de dessert. [4]
4. Quand on vous disait manipulables.

Deux agents interagissent lors d'une simulation, l'agent *bleu* et l'agent *vert*. Les portails, les liens et les champs apparaissent sur la carte de la même couleur que l'agent qui les possède. Les agents jouent chacun leur tour pendant toute la partie. Le but est de lier les portails entre eux afin de créer des champs de contrôle, et ainsi contrôler la plus grande aire possible.

3.2 État de la simulation

3.2.1 Carte

La carte consiste en une grille carrée, de `TAILLE_TERRAIN` cases de côté. (La valeur de `TAILLE_TERRAIN` est la même pour toutes les cartes.) Des portails y sont disposés ; leur nombre et leur position varient selon la carte et restent constants durant toute une partie. Il en va de même pour les positions de départ des agents.

3.2.2 Portails, liens et champs

Chaque portail peut être soit neutre, soit *contrôlé* (on dira aussi *possédé*) par l'un des deux agents.

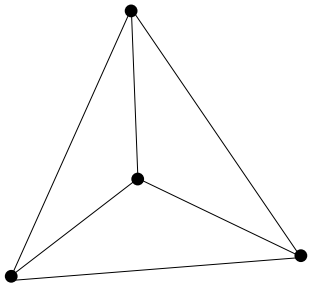
Un agent contrôlant deux portails peut les joindre par un *lien*, qui se matérialise sur la carte par un segment entre les deux portails. L'agent qui contrôle les deux portails aux extrémités est considéré comme possesseur du lien.

Lorsque trois portails sont tous reliés entre eux, l'intérieur du triangle constitué par ces liens forme un *champ*. Les trois portails du champ sont alors forcément contrôlés par le même agent, dont on dira qu'il possède le champ.

Une *interférence* entre deux liens, c'est-à-dire une intersection ou une superposition entre les deux segments, ne pourra *jamais* se produire et *toute action devant mener à une telle situation sera interdite par les règles*. Notez que cette condition est indépendante des possesseurs des liens.

De plus, la seule façon pour un lien de recontrer l'intérieur d'un champ sera d'être entièrement inclus dans ce champ, qui aura été construit après le lien.

Tous les triangles de la carte sont considérés comme des champs de contrôle, et pas simplement les faces triangulaires. Ainsi, sur la figure suivante, il faut compter 4 triangles : les trois faces et le triangle extérieur.



3.2.3 Agent

Chaque IA contrôle un agent positionné sur la carte. La position de l'agent n'est soumise à aucune restriction, si ce n'est de rester à l'intérieur de la carte : un agent pourra tout autant se trouver sur un portail, sur une case non-portail, sur une case traversée par un lien, à l'intérieur d'un champ ou à l'extérieur de tout champ, etc.

3.3 Déroulement d'un tour

Au début de chacun de vos tours, vous vous verrez octroyer `NB_POINTS_DEPLACEMENT` *points de déplacement*, et `NB_POINTS_ACTION` *points d'action*. Les compteurs des points sont remis à cette valeur à chaque début de tour (il n'y a pas de cumul de points sur plusieurs tours), et ne peuvent jamais atteindre une valeur négative.

Pour consommer ces points, vous pouvez effectuer les actions énumérées ci-dessous. Vous pouvez jouer autant d'actions que vous voulez durant un tour, dans n'importe quel ordre, tant que vous avez suffisamment de points.

3.3.1 Déplacement

Pour déplacer la position de votre agent, vous devez dépenser un nombre de points de déplacement égal à la distance de Manhattan (calculée par la fonction `distance`) entre votre position actuelle et votre destination. Cela revient à considérer qu'on autorise des déplacements horizontaux et verticaux d'une case, chacun coûtant un point de déplacement.

Turbo Le turbo vous permet de dépenser `COUT_TURBO` points d'action pour gagner un point de déplacement. La réciproque, à savoir convertir des points de déplacement en points d'action, est impossible.

3.3.2 Actions sur des portails

En règle générale, *pour toute action liée à un portail, votre agent doit se trouver sur la case du portail*. La création de lien est un cas particulier puisqu'elle fait intervenir deux portails.

Capturer un portail Vous pouvez dépenser `COUT_CAPTURE` points d'action pour capturer un portail *neutre*.

Relier deux portails Vous pouvez dépenser `COUT_LIEN` points d'action pour relier le portail où votre agent se trouve à un autre quelconque si :

— vous possédez les deux portails ;
— le lien n'interférerait avec aucun autre lien existant ;
— aucun des deux portails ne se trouve à l'intérieur d'un champ existant.

Neutraliser un portail Vous pouvez dépenser `COUT_NEUTRALISATION` points d'action pour neutraliser un portail qui appartient à l'adversaire, celui-ci devient alors neutre. Si des boucliers (cf. « Ajouter un bouclier ») sont présents sur le portail, un coût supplémentaire de `nombre de boucliers` × `COUT_NEUTRALISATION_BOUCLIER` vient s'ajouter au coût de base.

La neutralisation d'un portail lui fait perdre tous ses boucliers et détruit tous les liens reliés à ce portail, et donc également tous les triangles incidents à ce portail. Il *n'est pas possible* de payer partiellement pour détruire des boucliers sans entièrement neutraliser le portail.

Ajouter un bouclier Vous pouvez rajouter un bouclier sur un portail que vous contrôlez ; cette action coûte `COUT_BOUCLIER` pour un portail sans bouclier, et chaque bouclier présent fait monter le coût de 1 point d'action.

Les boucliers sur un portail sont conservés d'un tour à l'autre. Les boucliers disparaissent quand le portail sur lequel ils sont posés est détruit.

Le nombre de boucliers que l'on peut mettre sur un même portail est borné par `MAX_BOUCLIERS`. Ceci garantit qu'un portail restera toujours neutralisable avec les points d'action dont on dispose dans un tour.

3.4 Score

Le score des deux agents est initialisé à zéro en début de partie et ne peut qu'augmenter au cours de la partie. Pour cela, vous pouvez :

— capturer des portails, ce qui incrémente de `POINTS_CREATION_PORTAIL` votre score ;
— posséder des champs : à la fin de chacun de vos tours, chaque champ que vous contrôlez à ce moment vous rapporte un nombre de points proportionnel à l'aire qu'il recouvre (`POINTS_CHAMP` points par unité d'aire).

3.4.1 Format de la carte

La carte est représentée sous la forme d'un fichier texte où `X` désigne l'emplacement d'un portail et `.` une case sans portail.
Le fichier doit suivre le format suivant : :

```
depart_joueur1.x depart_joueur1.y
depart_joueur2.x depart_joueur2.y
ASCII map representation
```

Voici un exemple : :

```
14 14
15 15
..............................
..........X........X..........
....X..................X.......
.X.................X...........
.........X.................X...
.............X..........X.....
...X........................X..
........X............X.........
............X.....X............
.....X..................X......
..X...................X........
......X......................X.
............X...X.............
.......X..................X....
..............................
..............................
....X..................X.......
.............X...X............
.X......................X......
........X...................X..
......X..................X.....
............X.....X...........
.........X............X........
..X........................X...
.....X..........X.............
...X..................X.........
............X..................X.
.......X..................X....
..........X........X..........
..............................
```

4 Conclusion

Le Candidat referma son dossier. Il vit que l'Agent ne souriait plus. Ses voisins semblaient dans le même désarroi. Il lui fallut quelques secondes pour se rendre compte de la situation dans laquelle il se trouvait, et les responsabilités qui lui incombaient désormais.

Après avoir lu ces informations, il n'était évidemment plus question de partir ou de refuser la mission. Sans poser plus de questions, il suivit les autres vers le département 423. Savait-il vraiment ce qui l'attendait ?

5 API

Constante : TAILLE_TERRAIN
Valeur : 30
Description : Taille du terrain (longueur et largeur).

Constante : NB_TOURS
Valeur : 100
Description : Nombre de tours à jouer avant la fin de la partie.

Constante : NB_POINTS_DEPLACEMENT
Valeur : 6
Description : Nombre de points de déplacement par tour (avant utilisation du turbo).

Constante : NB_POINTS_ACTION
Valeur : 42
Description : Nombre de points d'action par tour.

Constante : COUT_CAPTURE
Valeur : 3
Description : Nombre de points d'action que coûte la capture d'un portail.

Constante : COUT_LIEN
Valeur : 2
Description : Nombre de points d'action que coûte la création d'un lien.

Constante : COUT_NEUTRALISATION
Valeur : 10
Description : Nombre de points d'action que coûte la neutralisation d'un portail de base.

Constante : COUT_NEUTRALISATION_BOUCLIER
Valeur : 5
Description : Nombre de points d'action supplémentaires que coûte la neutralisation pour chaque bouclier présent.

Constante : COUT_BOUCLIER
Valeur : 3
Description : Nombre de points d'action que coûte la mise en place d'un bouclier sur un portail sans bouclier.

Constante : COUT_TURBO
Valeur : 6
Description : Nombre de points d'action que coûte l'utilisation d'un turbo.

Constante : MAX_BOUCLIERS
Valeur : 6
Description : Nombre maximum de boucliers sur un même portail.

Constante : POINTS_CAPTURE
Valeur : 10
Description : Nombre de points que rapporte la création d'un portail.

Constante : POINTS_CHAMP
Valeur : 2
Description : Constante de proportionnalité reliant l'aire d'un champ au nombre de points qu'il rapporte par tour.

- **erreur**

Description : Erreurs possibles
Valeurs :

OK :	L'action a été exécutée avec succès
PA_INSUFFISANTS :	Vous ne possédez pas assez de points d'action pour cette action.
AUCUN_PORTAIL :	La position spécifiée n'est pas un portail.
POSITION_INVALIDE :	La position spécifiée est hors de la carte.
POSITION_ELOIGNEE :	La destination est trop éloignée.
PORTAIL_AMI :	Le portail vous appartient.
PORTAIL_NEUTRE :	Le portail est neutre.
PORTAIL_ENNEMI :	Le portail appartient à votre adversaire.
LIEN_INTERSECTION :	Le lien croise un lien existant.
LIEN_CHAMP :	Le lien se trouve dans un champ existant.
LIEN_DEGENERE :	Les deux extrémités du lien coïncident.
LIMITE_BOUCLIERS :	Ce portail est équipé du nombre maximal de boucliers.

- **position**

```
struct position {
    int x;
    int y;
};
```

Description : Position sur la carte, donnée par deux coordonnées.
Champs :

x :	Coordonnée en X
y :	Coordonnée en Y

- **lien**

```
struct lien {
    position extr1;
    position extr2;
    int joueur_l;
};
```

Description : Représente un lien existant.
Champs :

extr1 :	Première extrémité du lien.
extr2 :	Seconde extrémité du lien.
joueur_l :	Joueur possédant ce lien.

- **champ**

```
struct champ {
```

```
    position som1;
    position som2;
    position som3;
    int joueur_c;
};
```

Description : Représente un champ de contrôle existant.
Champs :
- *som1* : Premier sommet du champ.
- *som2* : Deuxième sommet du champ.
- *som3* : Troisième sommet du champ.
- *joueur_c* : Joueur possédant ce champ.

• **deplacer**

```
erreur deplacer(position dest)
```

Description : Déplace votre agent sur la case passée en argument.
Parametres : *dest* : Case de destination.

• **utiliser_turbo**

```
erreur utiliser_turbo()
```

Description : Utilise un turbo.

• **capturer**

```
erreur capturer()
```

Description : Capture le portail où est positionné votre agent.

• **lier**

```
erreur lier(position portail)
```

Description : Crée un lien entre le portail où se trouve votre agent et le portail de destination donné en argument.
Parametres : *portail* : Portail de destination du lien.

• **neutraliser**

`erreur neutraliser()`

Description : Neutralise le portail où se trouve votre agent.

• **ajouter_bouclier**

`erreur ajouter_bouclier()`

Description : Ajoute un bouclier au portail sur lequel se trouve votre agent.

• **liste_liens**

`lien array liste_liens()`

Description : Renvoie la liste de tous les liens présents.

• **liste_champs**

`champ array liste_champs()`

Description : Renvoie la liste de tous les champs de contrôle.

• **liste_portails**

`position array liste_portails()`

Description : Renvoie la liste de tous les portails de la carte.

• **liens_bloquants**

```
lien array liens_bloquants(position ext1, position ext2)
```

Description : Renvoie la liste de tous les liens existants qui croisent un segment, entravant la création d'un lien.

Parametres : *ext1* : Première extrémité du segment.
ext2 : Seconde extrémité du segment.

• **lien_existe**

```
bool lien_existe(position ext1, position ext2)
```

Description : Prend les positions de deux portails, et renvoie un booléen indiquant s'ils sont reliés. Le résultat est 'false' lorsque l'une des deux positions ne repère pas un portail.

Parametres : *ext1* : Premier portail.
ext2 : Second portail.

• **champ_existe**

```
bool champ_existe(position som1, position som2, position som3)
```

Description : Renvoie un booléen indiquant si les 3 positions repèrent bien 3 portails tous reliés entre eux.

Parametres : *som1* : Premier portail.
som2 : Deuxième portail.
som3 : Troisième portail.

• **case_dans_champ**

```
bool case_dans_champ(position pos)
```

Description : Renvoie un booléen indiquant si la case "pos" se trouve dans un champ.

Parametres : *pos* : Position de la case.

• **case_champs**

```
champ array case_champs(position pos)
```

Description : Renvoie la liste des champs à l'intérieur desquels "pos" se trouve. Si la case est un portail, le résultat de "case_champs" sera disjoint de celui de "champs_incidents_portail".

Parametres : *pos* : Position de la case.

• **portail_joueur**

```
int portail_joueur(position portail)
```

Description : Renvoie le numéro du joueur correspondant au portail donné, -1 si le portail est neutre, -2 si la case n'est pas un portail. Vous pouvez utiliser cette fonction pour vérifier qu'une case donnée est bien un portail.

Parametres : *portail* : Position du portail.

• **portail_boucliers**

```
int portail_boucliers(position portail)
```

Description : Renvoie le nombre de boucliers présents sur un portail (-2 si la case n'est pas un portail).

Parametres : *portail* : Position du portail.

• **liens_incidents_portail**

```
lien array liens_incidents_portail(position portail)
```

Description : Renvoie la liste de tous les liens dont le portail donné est une extrémité.

Parametres : *portail* : Position du portail.

• **champs_incidents_portail**

```
champ array champs_incidents_portail(position portail)
```

Description : Renvoie la liste de tous les champs dont le portail donné est un sommet.
Parametres : *portail* : Position du portail.

• **champs_incidents_segment**

```
champ array champs_incidents_segment(position ext1, position ext2)
```

Description : Renvoie la liste de tous les champs dont le lien donné est un côté. Si le segment n'est pas un lien présent, renvoie la liste de tous les champs que la création du lien ferait apparaître.
Parametres : *ext1* : Première extrémité du segment.
ext2 : Seconde extrémité du segment.

• **hist_portails_captures**

```
position array hist_portails_captures()
```

Description : Renvoie la liste des portails capturés par votre adversaire au dernier tour.

• **hist_portails_neutralises**

```
position array hist_portails_neutralises()
```

Description : Renvoie la liste des portails neutralisés par votre adversaire au dernier tour. Cela inclut toutes les utilisations de virus.

• **hist_liens_crees**

```
lien array hist_liens_crees()
```

Description : Renvoie la liste des liens créés par votre adversaire au dernier tour.

• **hist_champs_crees**

```
champ array hist_champs_crees()
```

Description : Renvoie la liste des champs créés par votre adversaire au dernier tour.

• **hist_boucliers_ajoutes**

```
position array hist_boucliers_ajoutes()
```

Description : Renvoie la liste des positions où votre adversaire a ajouté des boucliers au dernier tour.

• **distance**

```
int distance(position pos1, position pos2)
```

Description : Renvoie la distance de Manhattan entre deux positions.

Parametres : *pos1* : Première position
pos2 : Seconde position

• **score_triangle**

```
int score_triangle(position som1, position som2, position som3)
```

Description : Renvoie le nombre de points que rapporte(rait) chaque tour un champ existant ou hypothétique.

Parametres : *som1* : Premier sommet du triangle.
som2 : Deuxième sommet du triangle.
som3 : Troisième sommet du triangle.

• **intersection_segments**

```
bool intersection_segments(position a1, position a2,
                           position b1, position b2)
```

Description : Indique si deux segments se croisent. Cette fonction correspond exactement à la condition d'interférence entre liens, c'est-à-dire qu'elle renvoie "false" si l'intersection est une extrémité des deux segments.

Parametres :
- *a1* : Première extrémité du premier segment.
- *a2* : Seconde extrémité du premier segment.
- *b1* : Première extrémité du second segment.
- *b2* : Seconde extrémité du second segment.

• **point_dans_triangle**

```
bool point_dans_triangle(position p, position som1,
                         position som2, position som3)
```

Description : Indique si un point se trouve à l'intérieur d'un triangle. Le critère coïncide avec celui de "case_champs".

Parametres :
- *p* : Point à tester.
- *som1* : Premier sommet du triangle.
- *som2* : Deuxième sommet du triangle.
- *som3* : Troisième sommet du triangle.

• **moi**

```
int moi()
```

Description : Renvoie votre numéro de joueur.

• **adversaire**

```
int adversaire()
```

Description : Renvoie le numéro de votre adversaire.

• **position_agent**

```
position position_agent(int id_joueur)
```

Description : Indique la position de l'agent du joueur désigné par le numéro "id_joueur".
Parametres : *id_joueur* : Numéro du joueur.

• **points_action**

```
int points_action()
```

Description : Indique votre nombre de points d'actions restants pour ce tour-ci.

• **points_deplacement**

```
int points_deplacement()
```

Description : Indique votre nombre de points de déplacement restants pour ce tour-ci.

• **score**

```
int score(int id_joueur)
```

Description : Renvoie le score du joueur désigné par le numéro "id_joueur".
Parametres : *id_joueur* : Identifiant du joueur

• **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.

6 Notes sur l'utilisation de l'API

6.1 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.

6.2 C++

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

6.3 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#.

6.4 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)
```

```haskell
-- La pragma NOINLINE est importante !
-- MonType ne doit pas être polymorphe !
{-# NOINLINE maVariable #-}
maVariable :: IORef MonType
maVariable = unsafePerformIO (newIORef maValeurInitiale)

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

6.5 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.

6.6 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.

6.7 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)`.

6.8 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).

6.8.1 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.