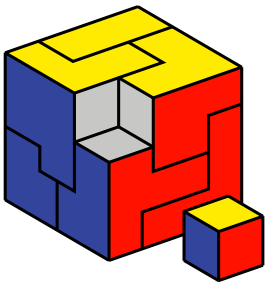
Prologin 2025



Barbotage Glacé

Assemblage macro de molécules d'eau par des mustélidés en pleine étendue d'eau salée

Sujet de la finale du Concours National d'Informatique
Vendredi 30 Mai 2025

Table des matières

1 Introduction **3**
1.1 Contexte 3

2 Objectif **4**

3 Carte **4**
3.1 Nœuds 4
3.2 Arêtes 5

4 Capture **6**
4.1 Pénalités 9
4.2 Piège 9

5 Contraction **10**

6 API **12**

7 Notes sur l'utilisation de l'API **22**
7.1 C 22
7.2 C++ 22
7.3 C# 22
7.4 Haskell 22
7.5 Java 23
7.6 OCaml 23
7.7 PHP 23
7.8 Python 24
7.9 Rust 24
7.10 JavaScript 24

1 Introduction

Tout d'abord, nos plus grandes félicitations pour votre impressionnante performance tout au long des qualifications et des épreuves régionales de Prologin. Vous avez surmonté de nombreux défis, des qualifications en ligne aux demi-finales régionales, comprenant de nombreuses épreuves algorithmiques et divers autres challenges. Votre réussite témoigne de votre talent et de votre persévérance, et nous vous souhaitons désormais la bienvenue à la toute dernière étape de cette aventure : la finale !

1.1 Contexte

En avril 2017, plus de 450 icebergs ont été signalés depuis le début de l'année sur les routes maritimes en Atlantique Nord. [1] Ces grands blocs de glaces sans arômes sont des dangers significatifs qui obligent les navires à faire de longs détours coûteux en temps et en carburant. [2] Certains capitaines téméraires passent tout de même dans des zones denses en gros glaçons, menant parfois à la mort des artistes vagabonds nés en Angleterre et 1500 autres personnes. [3] Ces icebergs ne sont pas seulement responsables de crises économiques, de catastrophes, du dérèglement climatique, ou de l'allongement du temps de livraisons de colis, ils sont aussi les coupables de la perturbation d'habitats d'animaux sauvages via la favorisation de la croissance du phytoplancton. [4]

Ainsi, les loutres [5] se sont décidées à assembler les icebergs entre eux afin de créer des dangers mortels bien plus faciles à repérer et à éviter.

Ces animaux carnivores et compétitifs ont donc choisi de faire s'affronter deux équipes l'une contre l'autre dans la capture et l'assemblage des icebergs. Pour ce faire, elles vont s'aider de grosses cordes reliant les icebergs afin de les fusionner.

Les loutres utilisent des cailloux comme outil. [6]

1. Source : latribune.fr
2. Source : C'est pas sorcier
3. Source : Titanic
4. Source : AFP
5. Animaux soucieux de l'environnement (que ça soit les gaz à effet de serre ou de la perturbation d'espaces naturels), de l'activité économique mondiale ainsi que du temps de livraison de nos colis.
6. Cette information n'est pas pertinente pour le jeu de la finale. Mais l'anecdote est sympa, donc on a choisi de la laisser là. D'après Gurvan, c'est faux puisqu'il y a des cailloux de débogage.

2 Objectif

Lors d'une partie, deux joueurs s'affrontent sur une carte formée par un graphe non orienté. Les deux opposants vont, tour à tour, relier des icebergs entre eux, jusqu'à éventuellement les fusionner [7]. Chaque contraction va impacter le score des joueurs.

La partie se termine dès que :

— plus aucune liaison n'est possible ; ou
— un joueur ne peut plus jouer **et** n'a pas l'avantage ; ou
— un joueur atteint `NBR_MAX_SAUTS` pénalités.

Le joueur ayant le plus de points à la fin de la partie remporte le jeu. [8]

3 Carte

La carte est un graphe non orienté, représenté de manière planaire. [9] Chaque arête et chaque nœud de la carte sont colorés avec une couleur parmi les `NBR_MAX_COULEURS` couleurs existantes.

La couleur de chaque nœud est représentée par un entier entre `0` et `NBR_MAX_COULEURS - 1`.

3.1 Nœuds

Les nœuds sont identifiés par un identifiant `id_noeud`, un nombre positif unique.

En plus de l'identifiant, chaque nœud possède :

— Des coordonnées qui permettent de le placer sur le plan ;
— Une couleur ;
— L'identifiant de son représentant ;
— Le gain associé à ce nœud.

On utilisera uniquement le terme *nœud* pour faire référence aux points (statiques) de la carte. On utilisera le terme *iceberg* pour désigner un ensemble non vide de nœuds. Initialement, chaque nœud de la carte forme son propre *iceberg*.

Les joueurs auront l'occasion, de par leurs actions, de faire se fusionner des icebergs, formant des icebergs de plus en plus gros.

7. On utilise fusion et contraction de manière interchangeable
8. Cependant, peu importe votre score, vous finirez toujours par perdre au Jeu.
9. C'est-à-dire que le graphe est toujours représenté de sorte qu'il n'y ait jamais deux arêtes qui se croisent.

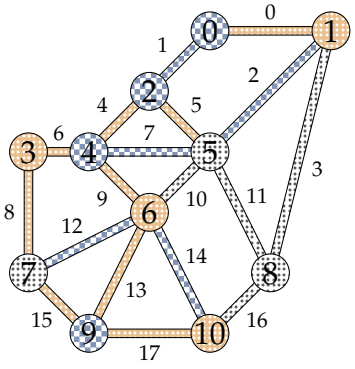
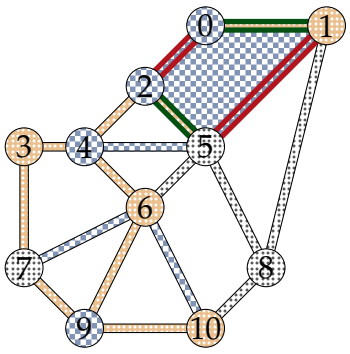


FIGURE 1 – Exemple de carte. Les numéros correspondent aux identifiants des nœuds et des arêtes.

Dans un iceberg, le nœud avec l'identifiant le plus petit est choisi comme représentant. Chaque nœud d'un même iceberg possède le même représentant. Dans un iceberg constitué d'un seul nœud, ce nœud est son propre représentant. La couleur (resp. le gain) d'un iceberg est définie comme étant la couleur (resp. le gain) de son représentant.



Dans cet exemple, les nœuds 0, 1, 5 et 2 sont reliés et forment alors un même iceberg. Au niveau de la carte, cela se traduit par les quatre nœuds ayant le même représentant, 0. La couleur de l'iceberg est définie par la couleur de son représentant.
Les autres nœuds décrivent chacun leur propre iceberg, et sont ainsi leur propre représentant.

3.2 Arêtes

Les arêtes de la carte décrivent les possibilités pour relier deux icebergs. Elles sont identifiées par un identifiant `id_arete`, un nombre positif unique. [10]

En plus de l'identifiant, chaque arête possède :

— Un bout 0 qui est l'identifiant d'un nœud ;
— Un bout 1 qui est l'identifiant d'un autre nœud ;
— Une couleur ;
— Un propriétaire, qui est soit un joueur, soit personne, soit inexistant.

Une arête inexistante est une arête faisant partie d'un iceberg. Elle ne peut donc pas être capturée.

10. Une arête peut avoir le même identifiant qu'un nœud, mais pas deux arêtes entre elles ou deux nœuds entre eux.

4 Capture

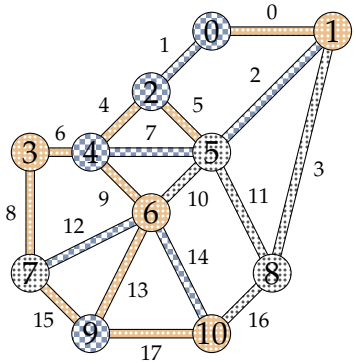
Chacun son tour, un joueur **doit** relier deux icebergs suivant une arête du graphe. Cette action est appelée *capture* d'une arête. Ne pas capturer une arête pendant son tour entraîne immédiatement une pénalité.

La capture d'une arête est régie par plusieurs règles fondamentales décrites dans la Procédure Régissant les Obligations sur les Liaisons Océaniques, qui forment l'essentiel des règles du jeu.

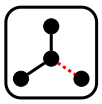
Article 1 – Liaisons respectueuses

La PROLO a préalablement effectué des études sur la géographie maritime des icebergs présents dans le terrain de jeu, et a déterminé quelles liaisons sont respectueuses de l'environnement. Ces liaisons respectueuses sont indiquées dans la carte.
Il est strictement interdit d'effectuer une liaison entre deux icebergs qui ne sont pas reliés dans la carte. En clair, la liaison effectuée doit obligatoirement être choisie parmi les arêtes décrites dans la carte.

Dans cet exemple, le joueur 1 peut effectuer une liaison entre les nœuds 6 et 10, mais pas entre les nœuds 3 et 6. Chaque arête de la carte possède un identifiant. La capture d'une arête se fait d'ailleurs par la fonction `capturer_arete`, qui prend en paramètre l'*identifiant* d'une arête existante. Ici, capturer l'arête qui relie les nœuds 6 et 10 se fait en appelant `capturer_arete(14)`.

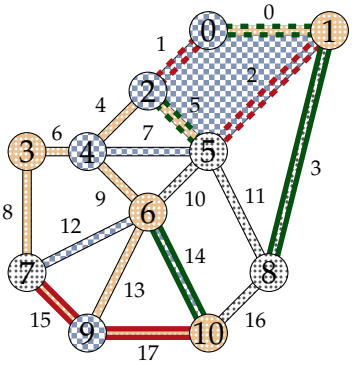


Article 2 – Surcharge des icebergs



Comme placer trop de liaisons sur un iceberg risquerait de l'endommager, la PROLO interdit de lier un iceberg à plus de deux autres icebergs. En clair, il est interdit de placer une nouvelle liaison sur un iceberg si celui-ci est déjà relié à deux autres icebergs.

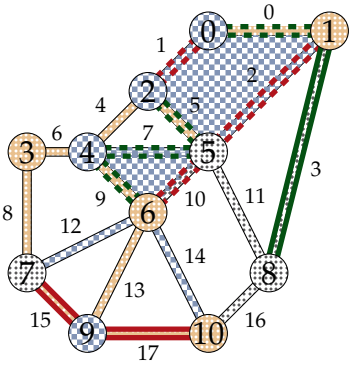
Dans cet exemple, les arêtes 0, 1, 2, 3, 5, 14, 15 et 17 ont déjà été capturées. C'est au joueur 1 d'effectuer une capture.
Dans cet exemple, il est interdit pour le joueur 1 de capturer l'arête 13, car cela surchargerait l'iceberg 9, qui serait ainsi relié à trois icebergs différents (6, 7 et 10). De la même manière, l'iceberg 10 est déjà relié à deux autres icebergs (6 et 9). Il est donc interdit de rajouter une liaison entre l'iceberg 10 et l'iceberg 8 en capturant l'arête 16.
En revanche, l'iceberg 0 (formé des nœuds 0, 1, 2 et 5) n'est relié qu'à un seul autre iceberg (l'iceberg 8). Il est donc tout à fait autorisé de capturer l'arête 4 ou même 10.



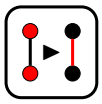
Article 3 – Liaisons inutiles

Pour éviter de polluer l'environnement, la PROLO interdit de placer une liaison entre deux icebergs qui sont déjà reliés, ni de placer une liaison reliant un iceberg à lui-même. Cela implique qu'il est impossible de capturer une arête qui a déjà été capturée, mais également qu'il est interdit de capturer deux arêtes distinctes qui relient deux mêmes icebergs.

Dans cet exemple, l'iceberg 0 est composé des nœuds 0, 1, 2, 4, 5 et 6. Il est donc interdit de capturer l'arête 4, qui relie deux fois l'iceberg 0 via les nœuds 2 et 4.
Aussi, il est interdit de capturer l'arête 11, qui relie les icebergs 0 et 8, car ces deux icebergs sont déjà directement reliés par l'arête 3.



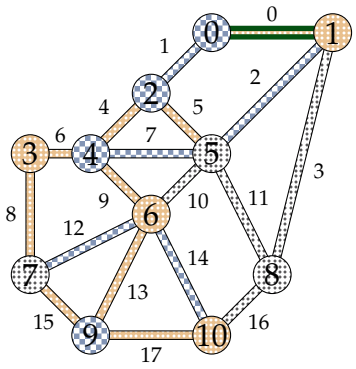
Article 4 – Conditionnement adverse



La couleur de l'arête que vous devez capturer est déterminée par l'adversaire. [11]

- Si, à son coup précédent, l'adversaire a capturé une arête reliant deux sommets de couleur a et b, alors vous **devez** capturer une arête de couleur a ou de couleur b.
- Si l'adversaire n'a pas joué au coup précédent, ou que vous êtes le premier joueur à capturer une arête, alors vous pouvez capturer n'importe quelle arête.

Le joueur 1 vient de jouer et a capturé l'arête 0. Cette arête reliait les icebergs 0 (de couleur *bleue*) et 1 (de couleur *beige*).
À son tour, le joueur 2 va donc devoir capturer une arête *bleue*, ou une arête *beige*.
Il est donc ici possible pour le joueur 2 de capturer, par exemple, les arêtes 1 ou 4, mais pas l'arête 3 (puisqu'elle est de couleur *grise*).



11. La justification de cette règle est laissée en exercice au lecteur.

4.1 Pénalités

Le fait de terminer son tour sans capturer d'arête entraîne une pénalité. Au bout de `NBR_MAX_SAUTS` pénalités, le joueur est directement éliminé, et la victoire revient à son adversaire.

En cas de perte par pénalités, les scores sont modifiés de manière à donner l'avantage à l'autre joueur :

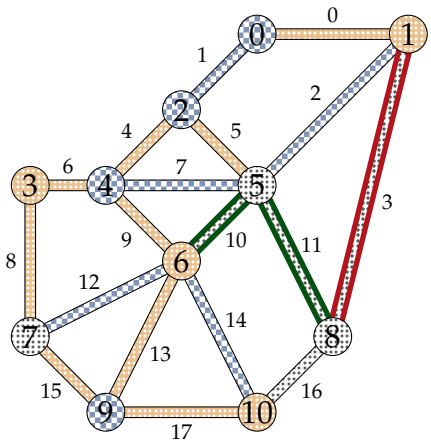
— Si le joueur perdant possédait un score strictement supérieur au gagnant, alors les scores sont intervertis ;

— Si les deux scores étaient égaux, alors on rajoute 1 point au score du gagnant.

4.2 Piège

On appelle *piège* le fait de choisir une arête de telle sorte que l'adversaire n'ait aucune capture possible au tour suivant. Piéger l'adversaire termine immédiatement la partie si vous possédez une majorité stricte de points.

Dans le cas contraire, le joueur adverse ne peut pas jouer, et son tour est directement sauté. L'adversaire reçoit une pénalité et vous pouvez rejouer immédiatement. [12]



Exemple : Le joueur 1 vient de capturer l'arête 11, qui relie deux sommets de couleur grise. Selon l'article 4, le joueur 2 doit obligatoirement capturer une arête de couleur grise à son tour. Malheureusement, la seule arête grise restante est l'arête 16, qui est également interdite car cela surchargerait l'iceberg 8. Le joueur 2 n'a donc aucune possibilité de coup à jouer : il s'est fait piéger.

— Si le joueur 1 possède plus de points que le joueur 2, le joueur 1 remporte immédiatement la partie.

— Si le joueur 1 possède autant ou moins de point que le joueur 2, le joueur 2 se prend une pénalité, et le joueur 1 peut rejouer.

12. Votre fonction `jouer_tour` est tout de même appelée une seconde fois.

5 Contraction

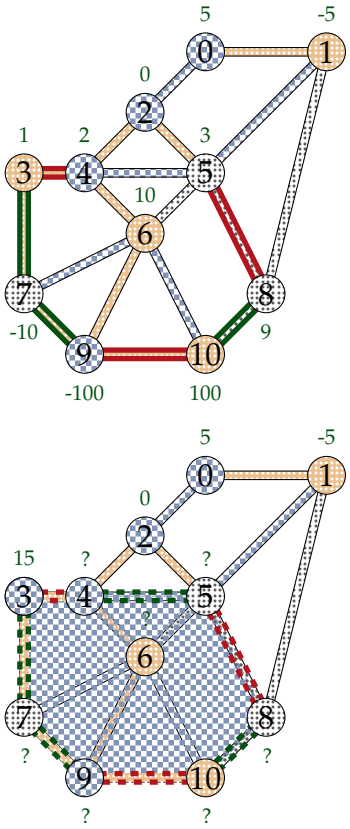
Si, à la suite d'une capture, un cycle est formé dans le graphe, alors tous les icebergs impliqués dans le cycle sont fusionnés en un seul iceberg géant. Cela inclut les icebergs formant le cycle, ainsi que tous les icebergs à l'intérieur de celui-ci dans le plan. On appelle alors les icebergs formant le cycle les *icebergs externes*, et les icebergs strictement à l'intérieur les *icebergs internes*.

Pour faciliter le traitement, la contraction est modélisée uniquement par un changement des propriétés des nœuds. Lors de la formation d'un cycle, tous les icebergs impliqués (internes et externes) fusionnent en un seul iceberg. Au niveau du graphe, cela se traduit par tous les nœuds impliqués choisissant le plus petit nœud d'entre eux comme représentant.

— La couleur du nouvel iceberg formé est définie comme étant la couleur de la dernière arête ayant été capturée pour former cet iceberg. Au niveau du graphe, cela se traduit par le nouveau représentant changeant sa couleur pour prendre la couleur de la dernière arête capturée.
— Le gain du nouvel iceberg formé est défini comme étant la somme des gains des icebergs (internes ou externes) impliqués dans la contraction. Au niveau du graphe, cela se traduit par le nouveau représentant changeant son gain pour la somme des gains des icebergs impliqués.
— Le gain de tous les icebergs *externes* de la contraction est alors additionné au score du joueur ayant effectué la contraction.

Attention

La couleur et la valeur du gain d'un nœud n'étant pas son propre représentant n'est pas déterminée, alors faites bien attention de toujours vérifier que vous cherchez les informations d'un iceberg en appelant `info_noeud` chez son représentant !



Dans cet exemple, le joueur 1 capture l'arête bleue entre les nœuds 4 et 5. Ce faisant, il forme un cycle, contenant 7 icebergs *externes* (3, 4, 5, 7, 8, 9 et 10) et 1 iceberg interne (6).
Le représentant du nouvel iceberg est le nœud de plus petit identifiant de l'iceberg, c'est-à-dire le 3. Ce nœud (et donc, l'iceberg) prend alors la couleur de la dernière arête capturée, et devient donc bleue.
La somme des gains des icebergs *externes* est de $1 + 2 + 3 - 10 + 9 - 100 + 100 = 5$, donc le joueur 1 remporte 5 points grâce à la formation de ce cycle. Le gain du nouvel iceberg ainsi formé est la somme des gains des icebergs *internes* **et** *externes* de la contraction, soit 15 dans cet exemple.

6 API

Constante : NBR_TOUR_MAX
Valeur : 306
Description : Nombre maximal de tours

Constante : NBR_MAX_NOEUDS
Valeur : 150
Description : Nombre maximal de nœuds

Constante : NBR_MIN_NOEUDS
Valeur : 3
Description : Nombre minimal de nœuds

Constante : Y_MAX
Valeur : 100
Description : Ordonnée maximale d'un nœud

Constante : Y_MIN
Valeur : -100
Description : Ordonnée minimale d'un nœud

Constante : X_MAX
Valeur : 100
Description : Abscisse maximale d'un nœud

Constante : X_MIN
Valeur : -100
Description : Abscisse minimale d'un nœud

Constante : GAIN_MAX
Valeur : 100
Description : Gain maximal d'un nœud

Constante : GAIN_MIN
Valeur : -100
Description : Gain minimal d'un nœud

Constante : NBR_MAX_COULEURS
Valeur : 10
Description : Nombre maximal de couleurs

Constante : NBR_MIN_COULEURS
Valeur : 1
Description : Nombre minimal de couleurs

Constante : NBR_MAX_SAUTS
Valeur : 3
Description : Nombre maximal de tours sautés

- **erreur**

Description : Erreurs possibles après avoir effectué une action
Valeurs :

OK :	L'action a été effectuée avec succès.
DEJA_CAPTURE :	Cet arête a déjà été capturée.
A_DEJA_JOUE :	Vous avez déjà capturé pendant ce tour.
COULEUR_INVALIDE :	La couleur de l'arête ne correspond pas à une des couleurs à jouer.
ARETE_INVALIDE :	L'arête est invalide : elle ne peut pas être prise ou n'existe pas.
POSITION_INVALIDE :	La position est à l'extérieur de la carte.

- **appartenance**

Description : Appartenance d'une arrête
Valeurs :

PERSONNE :	N'appartient à aucun joueur
INEXISTANT :	N'existe plus
JOUEUR_1 :	Appartient au joueur 1
JOUEUR_2 :	Appartient au joueur 2

- **caillou_debug**

Description : Type de caillou de debug
Valeurs :

PAS_DE_CAILLOU :	Aucun caillou, enlève le caillou présent
CAILLOU_BLEU :	Caillou bleu
CAILLOU_JAUNE :	Caillou jaune
CAILLOU_ROUGE :	Caillou rouge
CAILLOU_VERT :	Caillou vert

- **position**

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

Description : Position dans la carte, donnée par deux coordonnées
Champs :
x : Abscisse
y : Ordonnée

- **noeud**

```
struct noeud {
    int id_noeud;
    position pos;
    int couleur_noeud;
    int id_repr;
    int gain;
};
```

Description : Information d'un nœud
Champs :

id_noeud :	Identifiant du nœud, -1 si le nœud n'existe pas
pos :	Position du nœud
couleur_noeud :	Couleur de l'iceberg
id_repr :	Identifiant du représentant
gain :	Gain de l'iceberg

- **arete**

```
struct arete {
    int id_arete;
    int bout_0;
    int bout_1;
    int couleur_arete;
    appartenance proprietaire;
};
```

Description : Information d'une arête
Champs :

id_arete :	Identifiant de l'arête, -1 si l'arête n'existe pas
bout_0 :	Identifiant du nœud du bout 0 de l'arête
bout_1 :	Identifiant du nœud du bout 1 de l'arête
couleur_arete :	Couleur de l'arête
proprietaire :	Propriétaire de l'arête

- **carte**

```
struct carte {
    noeud array noeuds;
    arete array aretes;
};
```

Description : Information de la carte
Champs : *noeuds* : Liste des nœuds
aretes : Liste des arêtes

- **paire_couleurs**

```
struct paire_couleurs {
    int couleur_0;
    int couleur_1;
};
```

Description : Paire de couleurs
Champs : *couleur_0* : Couleur 0
couleur_1 : Couleur 1

- **action_hist**

```
struct action_hist {
    int joueur;
    int arete_id;
};
```

Description : Historique d'un tour
Champs : *joueur* : Joueur ayant effectué le coup
arete_id : Identifiant de l'arête ayant été capturée

• **fusion**

```
struct fusion {
    int tour;
    int joueur_fusion;
    int repr;
    int array noeuds_fusion;
    int array aretes_fusion;
};
```

Description : Description d'une fusion
Champs :

tour :	Tour de la fusion, -1 s'il n'y a jamais eu de fusion
joueur_fusion :	Joueur ayant provoqué la fusion, -1 s'il n'y a jamais eu de fusion
repr :	Représentant de l'iceberg, -1 s'il n'y a jamais eu de fusion
noeuds_fusion :	Identifiants des nœuds ayant été fusionnés, vide s'il n'y a jamais eu de fusion
aretes_fusion :	Identifiants des arêtes ayant été fusionnées, vide s'il n'y a jamais eu de fusion

• **capturer_arete**

```
erreur capturer_arete(int id)
```

Description : Capture d'une arête si possible, renvoie une erreur correspondant à l'état de la capture.
Paramètres : *id* : Identifiant de l'arête à capturer

• **info_noeud**

```
noeud info_noeud(int id)
```

Description : Renvoie les informations concernant un nœud.
Paramètres : *id* : Identifiant du nœud

• **info_arete**

```
arete info_arete(int id)
```

Description : Renvoie les informations concernant une arête.
Paramètres : *id* : Identifiant de l'arête

• **info_carte**

```
carte info_carte()
```

Description : Renvoie la liste des arêtes et nœud.

• **info_couleurs_a_jouer**

```
paire_couleurs info_couleurs_a_jouer()
```

Description : Renvoie les deux couleurs à jouer. Renvoie (-1, -1) si toutes les couleurs sont jouables.

• **score**

```
int score(int joueur)
```

Description : Renvoie le score d'un joueur. Renvoie -1 si le joueur est invalide.
Paramètres : *joueur* : Identifiant du joueur

• **info_fusion**

```
fusion info_fusion()
```

Description : Renvoie la fusion précédente.

• **debug_position**

```
erreur debug_position(position pos, caillou_debug caillou)
```

Description : Pose un caillou de debug sur la position indiquée.
Paramètres : *pos* : Position du caillou
caillou : Type de caillou

• **debug_arete**

```
erreur debug_arete(int id, caillou_debug caillou)
```

Description : Pose un caillou de debug sur l'arête indiquée.
Paramètres : *id* : Identifiant de l'arête
caillou : Type de caillou

• **historique**

```
action_hist array historique()
```

Description : Renvoie la liste des coups joués depuis le début de la partie.

• **constante_k**

```
int constante_k()
```

Description : Renvoie un nombre pseudo aléatoire entre 0 et 42 inclus de manière déterministe.

• **est_un_cycle**

```
bool est_un_cycle(int array cycle)
```

Description : Renvoie si une liste de nœuds forme un cycle. Il faut que les nœuds soient voisins. Il n'y a pas besoin de répéter le premier nœud à la fin de la liste.
Paramètres : *cycle* : Liste des identifiants des nœuds du cycle

• **dans_cycle**

```
bool dans_cycle(int id, int array cycle)
```

Description : Renvoie si un nœud est à l'intérieur ou fait partie d'un cycle.
Paramètres : *id* : Identifiant du nœud
cycle : Liste des identifiants des nœuds du cycle

• **dehors_cycle**

```
bool dehors_cycle(int id, int array cycle)
```

Description : Renvoie si un nœud est à l'extérieur d'un cycle et ne fait partie du cycle.
Paramètres : *id* : Identifiant du nœud
cycle : Liste des identifiants des nœuds du cycle

• **aretes_libres**

```
arete array aretes_libres()
```

Description : Renvoie la liste des arêtes libres.

• **aretes_a_jouer**

```
arete array aretes_a_jouer(paire_couleurs couleurs)
```

Description : Renvoie la liste des arêtes libres d'une couleur donnée (-1 pour toutes les couleurs).
Paramètres : *couleurs* : Paire de couleurs cherchées

• **aretes_du_noeud**

```
arete array aretes_du_noeud(int id)
```

Description : Renvoie la liste des arêtes autour d'un nœud.
Paramètres : *id* : Identifiant du nœud

• **moi**

```
int moi()
```

Description : Renvoie votre numéro de joueur.

• **adversaire**

```
int adversaire()
```

Description : Renvoie le numéro du joueur adverse.

• **annuler**

```
bool annuler()
```

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

• **tour_actuel**

```
int tour_actuel()
```

Description : Retourne le numéro du tour actuel.

• **dump_tour_actuel**

```
string dump_tour_actuel()
```

Description : Retourne le dump du tour actuel.

• **afficher_erreur**

```
void afficher_erreur(erreur v)
```

Description : Affiche le contenu d'une valeur de type erreur
Paramètres : v : The value to display

• **afficher_appartenance**

```
void afficher_appartenance(appartenance v)
```

Description : Affiche le contenu d'une valeur de type appartenance
Paramètres : v : The value to display

• **afficher_caillou_debug**

```
void afficher_caillou_debug(caillou_debug v)
```

Description : Affiche le contenu d'une valeur de type caillou_debug
Paramètres : v : The value to display

• **afficher_position**

```
void afficher_position(position v)
```

Description : Affiche le contenu d'une valeur de type position
Paramètres : v : The value to display

• **afficher_noeud**

`void afficher_noeud(noeud v)`

Description : Affiche le contenu d'une valeur de type noeud
Paramètres : v : The value to display

• **afficher_arete**

`void afficher_arete(arete v)`

Description : Affiche le contenu d'une valeur de type arete
Paramètres : v : The value to display

• **afficher_carte**

`void afficher_carte(carte v)`

Description : Affiche le contenu d'une valeur de type carte
Paramètres : v : The value to display

• **afficher_paire_couleurs**

`void afficher_paire_couleurs(paire_couleurs v)`

Description : Affiche le contenu d'une valeur de type paire_couleurs
Paramètres : v : The value to display

• **afficher_action_hist**

`void afficher_action_hist(action_hist v)`

Description : Affiche le contenu d'une valeur de type action_hist
Paramètres : v : The value to display

• **afficher_fusion**

`void afficher_fusion(fusion v)`

Description : Affiche le contenu d'une valeur de type fusion
Paramètres : v : The value to display

7 Notes sur l'utilisation de l'API

7.1 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* items`, et la taille, `size_t length`. Dans tous les cas, la libération des données est laissée au soin du candidat;
— Tout le reste est comme indiqué dans le sujet.

7.2 C++

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

7.3 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#.

7.4 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'
```

7.5 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.

7.6 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.

7.7 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)`.

7.8 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 utilisées comme ceci : `nom_enum.CHAMP.` ;
— Les structures sont représentées 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)`.

7.9 Rust

— L'API est fournie par le module `api`, dont tout le contenu est importé par défaut par le code à compléter. ;
— Les noms des structures et énumérations sont en `CamelCase`. Ainsi, une structure nommée `foo_bar` dans le sujet s'appellera `FooBar` en Rust.
— Les tableaux sont représentés par des `Vec<T>` et les strings par des `String`. Les fonctions prennent leurs primitives empruntées `&[T]` et `&str` en entrée.

7.10 JavaScript

— L'API est définie sur l'objet global et documentée dans le fichier `api.d.ts`.
— Les structures sont représentées par des objets, les énumérations par des constantes sur des variables globales, les tuples par des tableaux.
— L'écriture sur la sortie standard et l'erreur standard s'effectue avec `console.log` et `console.error` respectivement. Les autres méthodes standard de `console` ne sont pas définies.

Étymologie d'*iceberg* :

— De l'anglais *iceberg*
  — Du néerlandais *ijsberg*
    — Du néerlandais *ijs*
      — Du moyen néerlandais *ijs*
        — Du vieux néerlandais *īs*
          — Du proto-germanique occidental *īs*
            — Du proto-germanique *īsą*
              — Du proto-indo-européen *$h_1$eyH-so-*
                — Du proto-indo-européen *$h_1$eyH-*
    — Du néerlandais *berg*
      — Du moyen néerlandais *berch*
        — Du vieux néerlandais *berg*
          — Du proto-germanique occidental *berg*
            — Du proto-germanique *bergaz*
              — Du proto-indo-européen *b^{h}érǵ^{h}os*
                — Du proto-indo-européen *b^{h}erǵ^{h}-*