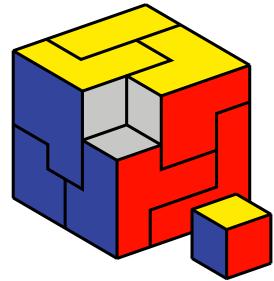


Prologin

2024



DES ÎLES ET DES AIGLES

Exploration d'archipels et pouvoirs divins

Sujet de la finale du Concours National d'Informatique
Vendredi 31 Mai 2024

Table des matières

1	Introduction	3
2	Objectif	3
3	Carte	3
3.1	Cases	4
3.2	Village	5
3.3	Emplacements	5
3.4	Îles	6
3.5	Rotation	6
4	Territoire	7
5	Points d'action	8
6	Aigles	8
6.1	Pouvoirs	8
7	Règles	11
8	API	12
9	Notes sur l'utilisation de l'API	21
9.1	C	21
9.2	C++	21
9.3	C#	21
9.4	Haskell	21
9.5	Java	22
9.6	OCaml	22
9.7	PHP	22
9.8	Python	23
9.9	Rust	23
9.10	JavaScript	23



1 Introduction

Tout d'abord, nos plus grandes félicitations pour votre impressionnante performance tout au long des qualifications et des épreuves régionales de Pro-login. 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 !

2 Objectif

Lors d'une partie, deux joueurs s'affrontent en un contre un sur une carte rectangulaire. Les deux joueurs cherchent à maintenir le plus grand territoire possible¹. Pour aider les deux joueurs dans leur mission, ils peuvent capturer des aigles capables de façonner la terre².

3 Carte

La carte est une grille rectangulaire de largeur \times hauteur cases. La largeur est comprise entre LARGEUR_MIN et LARGEUR_MAX (inclus), et la hauteur est comprise entre HAUTEUR_MIN et HAUTEUR_MAX (inclus).

Les dimensions exactes de la carte ne sont pas fixées, et peuvent dépendre de la carte.

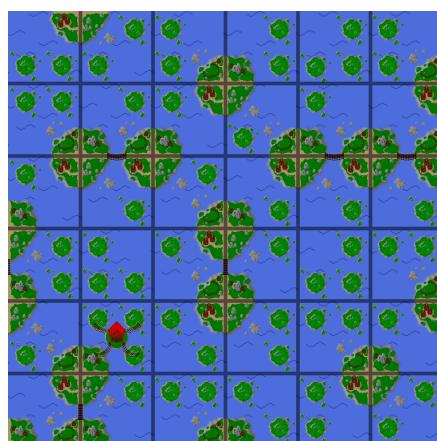


FIGURE 1 – Exemple de carte de 6×6 cases

-
1. Sauf quand ils ne veulent pas.
 2. Sauf quand ils ne veulent pas.

3.1 Cases

Les cases de la grille sont identifiées par leur position, un couple (colonne, ligne), où la case dans le coin nord-ouest possède la position (0, 0), et la case dans le coin sud-est possède la position (largeur – 1, hauteur – 1).

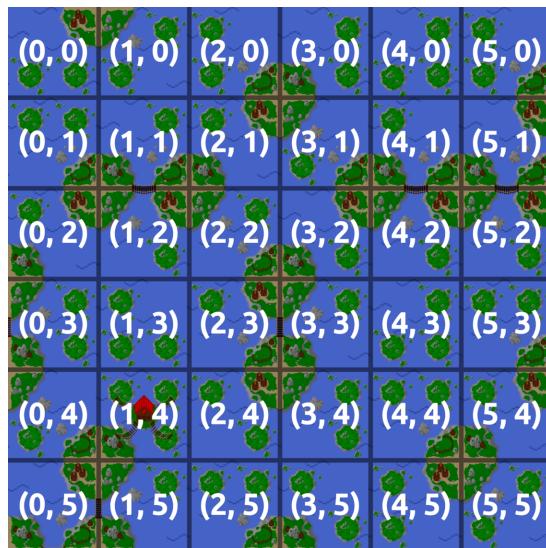


FIGURE 2 – Coordonnées des cases de la carte

Une case peut contenir soit un village, soit trois îlots dans trois des quatre coins de la case. Une case contenant trois îlots est identifiée par le point cardinal du coin ne possédant **pas** d'îlot.

La plupart des cases peuvent être tournées (Section 3.5), à moins qu'elles ne soient gelées (Section 6.1).



Un village

Dans l'interface, le nord est toujours vers le haut de l'écran, l'est vers la droite, et ainsi de suite.

Dans l'encodage d'une carte :

- l'orientation NORD_EST est encodé par un 1
- l'orientation NORD_OUEST est encodé par un 2
- l'orientation SUD_OUEST est encodé par un 3
- l'orientation SUD_EST est encodé par un 4
- un village est encodé par un X

3.2 Village

Un village est une case spéciale dont les 4 coins sont des îlots. Un village peut être possédé par un joueur ou être neutre³. Les joueurs commencent la partie en ne possédant qu'un seul village⁴.

3.3 Emplacements

Un emplacement représente l'intersection de quatre cases adjacentes. On dit que ces quatre cases *bordent* l'emplacement. Il y a donc $(\text{largeur} - 1) \times (\text{hauteur} - 1)$ emplacements.

Chaque emplacement peut être identifié par sa position, une paire (colonne, ligne), où l'emplacement le plus au nord-ouest possède la position (0, 0), et l'emplacement le plus au sud-est possède la position (largeur - 2, hauteur - 2). Les coordonnées d'un emplacement correspondent donc aux coordonnées de la case directement au nord-ouest de cet emplacement.

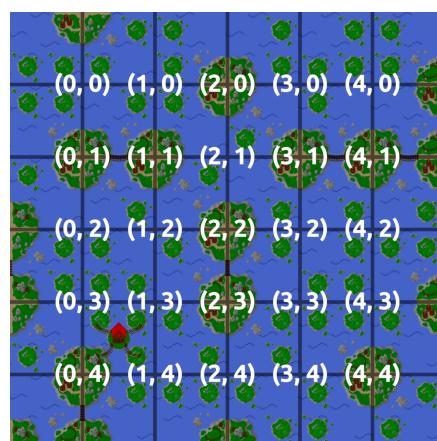


FIGURE 3 – Coordonnées des emplacements de la carte

-
- 3. Un village neutre n'est possédé par aucun joueur.
 - 4. Selon la configuration de la carte, il est possible de capturer un village avant le début de la partie.

Notez qu'il n'y a pas d'emplacement sur les bords de la carte.

Chaque emplacement se voit attribuer un gain potentiel, qui permet d'obtenir du score. Ce gain est un entier, pouvant être positif, négatif, ou nul.

Cet encadré vous est offert pour comptabiliser le nombre de fois où vous avez confondu case et emplacement dans votre code :

3.4 Îles

Lorsque quatre îlots bordent un emplacement, une île se forme à cet emplacement. L'île persiste tant que quatre îlots bordent l'emplacement. L'île disparaît dès qu'une des quatre cases bordant l'emplacement ne fournit plus d'îlot pour l'emplacement.

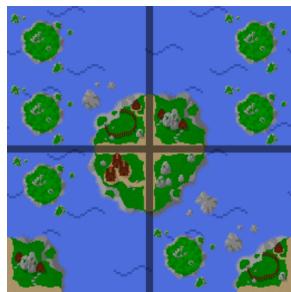


FIGURE 4 – Île formée par quatre îlots



FIGURE 5 – Deux îles connectées

Deux îles sont connectées si elles sont adjacentes, c'est-à-dire, si elles se situent sur deux emplacements côté-à-côte sur la même ligne ou la même colonne. Deux îles en diagonale ne sont pas considérées comme directement connectées.

3.5 Rotation

Les joueurs peuvent tourner à un certain coût les cases d'un quart de tour dans le sens trigonométrique (anti-horaire).

Il est impossible pour un joueur de faire tourner un village, ou de faire tourner une case gelée.

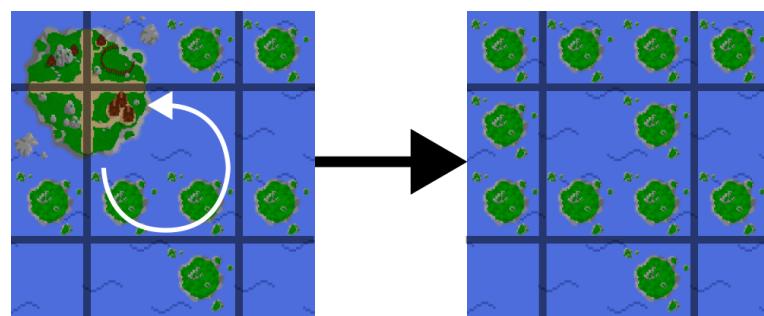


FIGURE 6 – Rotation d'une case

4 Territoire

Le territoire d'un joueur représente le réseau des îles connectées, reliées à au moins un village contrôlé par le joueur.



FIGURE 7 – Un territoire de quatre îles

Si une chaîne d'îles relie un village de chaque joueur, alors ces îles appartiennent au territoire des deux joueurs.

Si, à la fin d'un tour, un village neutre appartient au territoire d'un unique joueur, alors le joueur capture le village de manière définitive. Si un village neutre appartient au territoire des deux joueurs, il reste neutre.

Le territoire n'est pas forcément connexe. Si vous disposez de plusieurs villages, chaque village peut posséder son propre réseau d'îles qui contribuera à votre territoire.

5 Points d'action

Chaque tour, un joueur commence avec TOUR_POINTS_ACTION points d'action. Les points d'action ne servent qu'à tourner des cases.

La rotation d'une case qui borde au moins un emplacement du territoire adverse demande COUT_ROTATION_ENNEMI points d'action. Sinon, la rotation demande COUT_ROTATION_STANDARD point d'action.

Toutes les autres actions ne consomment aucun point d'action, et peuvent être répétées sans contraintes.

6 Aigles

Un aigle est une créature divine qui peut être présent sur un emplacement dès le début de la partie, sous la forme d'un œuf.

Au début de la partie, plusieurs œufs peuvent être disposés sur la carte. Chaque œuf possède un certain tour d'éclosion, avant lequel l'aigle n'est pas capturable. Une fois son tour d'éclosion atteint, un aigle sauvage sort de l'œuf et reste sur son emplacement jusqu'à sa capture.

À la fin d'un tour, si l'emplacement d'un aigle sauvage appartient au territoire d'un unique joueur, le joueur capture l'aigle. Si l'emplacement d'un aigle sauvage appartient au territoire des deux joueurs, l'aigle reste sauvage. Plusieurs aigles et œufs peuvent se trouver sur un même emplacement.

Un aigle capturé peut être déplacé autant fois que souhaité lors d'un même tour et n'importe où sur la carte. Plusieurs aigles peuvent se situer sur le même emplacement, indépendamment de s'ils sont sauvage ou non, de leur propriétaire, ni de leur effet.

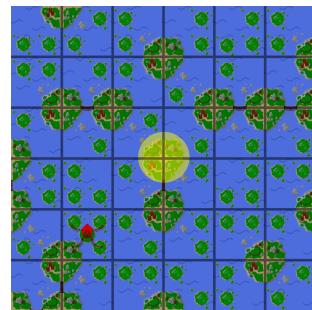
6.1 Pouvoirs

Les aigles sont tous dotés d'un pouvoir et d'une puissance. La puissance est un entier dont la signification est spécifique au pouvoir. On détaillera ce que représente cette donnée plus tard dans le sujet.

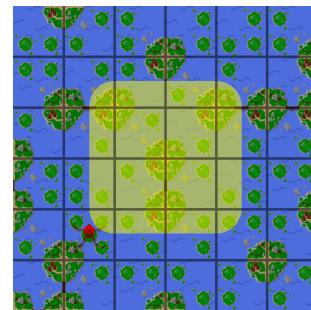
Il existe deux types d'aigles : les aigles éphémères et les aigles persistants. Les aigles éphémères ont un pouvoir qui ne s'active qu'une fois avant de disparaître à jamais. Les aigles persistants ont un pouvoir qui perdure tant qu'ils restent sur la carte.

Rayon d'action Certains aigles ont un pouvoir dont l'effet s'applique sur des cases ou des emplacements autour de l'aigle. La portée de l'effet formera alors un carré, dont la taille dépend de la puissance de l'aigle.

Par exemple, si l'effet d'un aigle s'applique sur des emplacements, une puissance de 0 appliquera l'effet uniquement sur l'emplacement où se situe l'aigle. Une puissance de 1 appliquera l'effet sur les neuf emplacements autour de l'aigle.

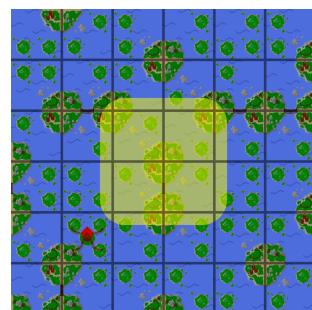


La portée d'un aigle de puissance 0

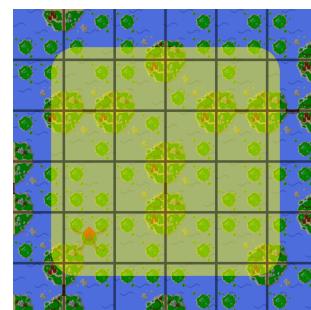


La portée d'un aigle de puissance 1

Si l'effet d'un aigle s'applique sur des cases, alors l'effet s'applique sur toutes les cases bordant les emplacements impliqués. Une puissance de 0 appliquera donc l'effet sur les 4 cases bordant l'emplacement de l'aigle.



Cases affectées par un aigle de puissance 0



Cases affectées par un aigle de puissance 1

Aigle de vie L'aigle de vie est un aigle éphémère qui, une fois activé, attribue au joueur sa puissance en points d'actions pour le tour actuel.

La puissance d'un aigle de vie peut avoir n'importe quelle valeur, positive ou négative.



Aigle météore L'aigle météore est un aigle éphémère qui, une fois activé, tourne deux fois toutes les cases non gelées dans son rayon d'action. Sa puissance est une portée, et ne peut donc pas être négative.



Aigle de feu L'aigle de feu est un aigle persistant qui multiplie le gain de l'emplacement sur lequel il se trouve. Sa puissance indique le multiplicateur qui s'applique sur le gain de l'emplacement. Ce multiplicateur peut être négatif.

L'effet des aigles de feu est cumulable de manière multiplicative. Par exemple, s'il y a deux aigles de feu de puissance 2 et -5 sur un même emplacement, alors le gain de l'île est multiplié par -10 . Si cette île était censé rapporter 3 points, alors le gain sera de $3 \times (-10) = -30$.



Aigle de gel L'aigle de gel est un aigle persistant qui empêche la rotation de toutes les cases dans son rayon d'action. Il empêche aussi bien les rotations des deux joueurs que les rotations des aigles météores.



Aigle de mort L'aigle de mort est un aigle éphémère qui, une fois activé, fait disparaître tous les aigles dans son rayon d'action^a. Il peut donc faire disparaître les aigles adverses, mais aussi vos propres aigles et les aigles sauvages. Un aigle de mort ne peut cependant pas faire fuir un aigle qui n'a pas encore éclos.



^a. Aucun aigle n'est maltraité pendant la durée de la partie. Contrairement à ce que son nom pourrait faire penser, l'aigle de mort fait simplement **fuir** les autres aigles en dehors de la carte.

Aigle	Longévité	Effet
Vie	Éphémère	Rajoute des points d'actions pour le tour actuel
Météore	Éphémère	Tourne d'un demi-tour les cases dans son rayon d'action
Feu	Persistant	Multiplie le gain d'un emplacement
Gel	Persistant	Empêche les rotations dans son rayon d'action
Mort	Éphémère	Fait fuir les aigles dans son rayon d'action

7 Règles

Début d'une partie Chaque joueur commence la partie avec en sa possession un unique village, sans aucun aigle. Une partie est composée de $\frac{\text{NB_TOURS}}{2}$ rounds, chaque round étant divisé en deux tours, un tour par joueur.

Avant le commencement du premier tour, la fonction `partie_init` de votre champion est appelée.

Déroulement d'un tour Au commencement de chaque tour, s'il s'agit de votre tour, la fonction `jouer_tour` de votre champion est appelée. Vous disposerez alors de `TOUR_POINTS_ACTION` points d'actions pour effectuer vos actions. Vous pouvez alors, autant de fois que vous le souhaitez :

- Tourner une case arbitraire d'un quart de tour dans le sens trigonométrique, tant que vous disposez de suffisamment de points d'actions. Si la case tournée borde un emplacement du territoire adverse, alors cette action demande `COUT_ROTATION_ENNEMI` points d'action, sinon l'action demande `COUT_ROTATION_STANDARD` points d'action. Vous ne pouvez pas tourner un village, ni tourner une case étant gelée par un aigle de gel.
- Déplacer un aigle vous appartenant sur un emplacement arbitraire. Cela ne coûte aucun point d'action et peut être réalisé librement autant de fois que souhaité. Plusieurs aigles peuvent se situer sur le même emplacement.
- Activer l'effet d'un aigle éphémère qui vous appartient. L'aigle ainsi utilisé disparaît alors. Cela ne demande aucun point d'action, vous pouvez activer autant d'aigles que vous souhaitez pendant un même tour. L'effet d'un aigle permanent est toujours actif.

Score À la fin de chaque tour, la somme des gains de chaque île dans votre territoire est ajouté à votre score. Un aigle de feu (vous appartenant ou non) permet de multiplier le gain d'un emplacement. Au dernier tour, cette somme est multipliée par `MULTIPLICATEUR_DERNIER_TOUR`.

Fin de jeu À la fin du dernier tour, la partie prend fin et la fonction `partie_fin` de votre champion est appelée. Le joueur ayant le plus grand score est déclaré vainqueur de la partie.

8 API

Constante : LARGEUR_MAX

Valeur : 100

Description : Largeur maximale de la carte

Constante : HAUTEUR_MAX

Valeur : 100

Description : Hauteur maximale de la carte

Constante : LARGEUR_MIN

Valeur : 10

Description : Largeur minimale de la carte

Constante : HAUTEUR_MIN

Valeur : 10

Description : Hauteur minimale de la carte

Constante : NB TOURS

Valeur : 400

Description : Nombre de tours à jouer avant la fin de la partie

Constante : GAINS_MAX

Valeur : 100

Description : Gains maximum qu'apporte une île

Constante : GAINS_MIN

Valeur : -100

Description : Gains minimum qu'apporte une île

Constante : TOUR_POINTS_ACTION

Valeur : 2

Description : Points d'action au début d'un tour

Constante : COUT_ROTATION_ENNEMI

Valeur : 2

Description : Coût de rotation d'une case en lien avec une île ennemi

Constante : COUT_ROTATION_STANDARD
Valeur : 1
Description : Coût de rotation d'une case qui n'est pas en lien avec une île ennemi

Constante : MULTIPLICATEUR_DERNIER_TOUR
Valeur : 42
Description : Multiplicateur de score du dernier tour

Constante : LA_CONSTANTE_K
Valeur : -41
Description : k est une constante (relou)

• erreur

Description : Erreurs possibles après avoir effectué une action
Valeurs :

OK :	L'action a été effectuée avec succès
HORS_TOUR :	Vous ne pouvez pas faire d'action en dehors de votre tour
CASE_BLOQUEE :	La case est bloquée par un aigle
POSITION_INVALIDE :	La position fournie est invalide
DESTINATION_INVALIDE :	La position d'arrivée est invalide
PLUS_DE_PA :	Vous n'avez plus de points d'action
AIGLE_INVALIDE :	L'identifiant de l'aigle est invalide
ROTATION_VILLAGE :	Vous essayez de tourner un village

• type_case

Description : Contenu topographique d'une case
Valeurs :

VILLAGE :	Village
NORD_EST :	Case dont le coin manquant est au nord est
NORD_OUEST :	Case dont le coin manquant est au nord ouest
SUD_OUEST :	Case dont le coin manquant est au sud ouest
SUD_EST :	Case dont le coin manquant est au sud est
CASE_INVALIDE :	Case invalide

• drakkar_debug

Description : Type de drakkar de debug

Valeurs :

<i>PAS_DE_DRAKKAR</i> :	Aucun drakkar, enlève le drakkar présent
-------------------------	--

<i>DRAKKAR_BLEU</i> :	Drakkar bleu
-----------------------	--------------

<i>DRAKKAR_JAUNE</i> :	Drakkar jaune
------------------------	---------------

<i>DRAKKAR_ROUGE</i> :	Drakkar rouge
------------------------	---------------

• type_action

Description : Types d'actions

Valeurs :

<i>ACTION_TOURNER_CASE</i> :	Tourne une case, action “tourner_case”
<i>ACTION_ACTIVER_AIGLE</i> :	Active l'effet d'un aigle, action “activer_aigle”
<i>ACTION_DEPLACER_AIGLE</i> :	Déplace un aigle appartenant à l'utilisateur, action “deplacer_aigle”

• effet_aigle

Description : Effet de l'aigle

Valeurs :

<i>EFFET_METEORE</i> :	Fait tomber un météore qui tourne les cases
<i>EFFET_VIE</i> :	Donne des points actions
<i>EFFET_MORT</i> :	Effraye les aigles d'un emplacement
<i>EFFET_FEU</i> :	Multiplie les gains d'une île
<i>EFFET_GEL</i> :	Bloque les rotations de cases

• position

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

Description : Position dans la carte, donnée par deux coordonnées

Champs :

<i>colonne</i> :	Abscisse
<i>ligne</i> :	Ordonnée

• dimension

```
struct dimension {
    int largeur;
    int hauteur;
};
```

Description : Dimensions de la carte

Champs :

<i>largeur</i> :	Largeur de la carte
<i>hauteur</i> :	Hauteur de la carte

• aigle

```
struct aigle {
    int identifiant;
    int joueur;
    position pos;
    effet_aigle effet;
    int puissance;
    int tour_eclosion;
};
```

Description : Aigle

Champs :

<i>identifiant</i> :	Identifiant de l'aigle
<i>joueur</i> :	Identifiant du joueur auquel appartient l'aigle, -1 si n'appartient à aucun des joueurs
<i>pos</i> :	Position de l'aigle
<i>effet</i> :	Effet de l'aigle
<i>puissance</i> :	Valeur de la puissance de l'aigle
<i>tour_eclosion</i> :	Tour d'éclosion de l'oeuf

• etat_case

```
struct etat_case {
    type_case contenu;
    int gains;
    position pos_case;
};
```

Description : Description complète d'une case

Champs :

<i>contenu</i> :	Contenu topographique de la case
<i>gains</i> :	Gains dans le coin sud-est de la case
<i>pos_case</i> :	Position de la case

• action_hist

```
struct action_hist {
    type_action action_type;
    position debut;
    position fin;
    int identifiant_aigle;
};
```

Description : Action représentée dans l'historique

Champs :

<i>action_type</i> :	Type de l'action
<i>debut</i> :	Position de début de déplacement ou position de la case tournée
<i>fin</i> :	Position de fin
<i>identifiant_aigle</i> :	Identifiant de l'aigle utilisé

- **tourner_case**

erreur tourner_case(position pos)

Description : Rotation d'un quart de tour d'une case dans le sens trigonométrique (anti-horaire)

Paramètres : *pos* : Position de la case

- **activer_aigle**

erreur activer_aigle(int id)

Description : Activer l'effet d'un aigle

Paramètres : *id* : Identifiant de l'aigle

- **deplacer_aigle**

erreur deplacer_aigle(int id, position destination)

Description : Déplace un aigle

Paramètres : *id* : Identifiant de l'aigle

destination : Position où l'aigle sera déplacé

- **dimensions_carte**

dimension dimensions_carte()

Description : Renvoie les dimensions hauteur largeur de la carte

- **info_case**

etat_case info_case(position pos)

Description : Renvoie les informations concernant une case

Paramètres : *pos* : Position de la case

- **info_aigles**

aigle array info_aigles()

Description : Renvoie la liste d'aigles

- **liste_villages**

```
position array liste_villages(int joueur)
```

Description : Renvoie la liste des villages. Identifiant -1 pour les villages libres.

Paramètres : *joueur* : Identifiant du joueur

- **points_action**

```
int points_action(int joueur)
```

Description : Renvoie le nombre de points d'action restant. Renvoie -1 si le joueur est invalide.

Paramètres : *joueur* : Identifiant du joueur

- **score**

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

Description : Renvoie le score d'un joueur. Renvoie -1 si le joueur est invalide.

Paramètres : *joueur* : Identifiant du joueur

- **debug_poser_drakkar**

```
erreur debug_poser_drakkar(position pos, drakkar_debug drakkar)
```

Description : Pose un drakkar de debug sur la case indiquée

Paramètres : *pos* : Case où poser le drakkar
drakkar : Type du drakkar

- **historique**

```
action_hist array historique()
```

Description : Renvoie la liste des actions effectuées par l'adversaire durant son tour, dans l'ordre chronologique. Les actions de débug n'apparaissent pas dans cette liste.

- **recuperer_territoire**

```
position array recuperer_territoire(int joueur)
```

Description : Renvoie une liste des positions du territoire d'un joueur.

Paramètres : *joueur* : Identifiant du joueur

- **case_dans_rayon**

```
bool case_dans_rayon(int id, position pos)
```

Description : Renvoie vrai si la case est dans le rayon de l'aigle. Si l'aigle est invalide, renvoie faux.

Paramètres : *id* : Identifiant de l'aigle

pos : Position de la case

- **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.

- afficher_erreur

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

Description : Affiche le contenu d'une valeur de type erreur

Paramètres : *v* : The value to display

- afficher_type_case

```
void afficher_type_case(type_case v)
```

Description : Affiche le contenu d'une valeur de type type_case

Paramètres : *v* : The value to display

- afficher_drakkar_debug

```
void afficher_drakkar_debug(drakkar_debug v)
```

Description : Affiche le contenu d'une valeur de type drakkar_debug

Paramètres : *v* : The value to display

- afficher_type_action

```
void afficher_type_action(type_action v)
```

Description : Affiche le contenu d'une valeur de type type_action

Paramètres : *v* : The value to display

- afficher_effet_aigle

```
void afficher_effet_aigle(effet_aigle v)
```

Description : Affiche le contenu d'une valeur de type effet_aigle

Paramètres : *effe 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_dimension**

```
void afficher_dimension(dimension v)
```

Description : Affiche le contenu d'une valeur de type dimension

Paramètres : v : The value to display

- **afficher_aigle**

```
void afficher_aigle(aigle v)
```

Description : Affiche le contenu d'une valeur de type aigle

Paramètres : v : The value to display

- **afficher_etat_case**

```
void afficher_etat_case(etat_case v)
```

Description : Affiche le contenu d'une valeur de type etat_case

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

9 Notes sur l'utilisation de l'API

9.1 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.

9.2 C++

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

9.3 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#.

9.4 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'
```

9.5 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.

9.6 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.

9.7 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)`.

9.8 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)`.

9.9 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.

9.10 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.

Vous n'êtes plus des petits vers de terre.
Vous êtes l'aigle!