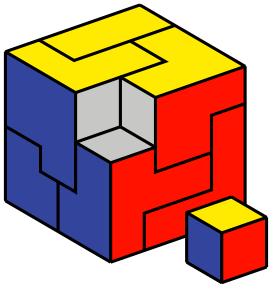
Prologin 2020



Des baffes fleuries

Tout est dans l'élégance

Sujet de la finale en ligne du Concours National d'Informatique
Vendredi 28 Août 2020

Table des matières


PROLOGIN
2020

1 Contexte

1.1 Au commencement

Il y a fort, fort longtemps [1], dans un monde pas si lointain, vivait un peuple du nom de Prolo.

Ses éminents représentants étaient réputés pour leur beauté à nulle autre pareille et leur élégance époustouflante. Personne n'avait jamais vu d'aussi jolis spécimens : ils étaient la fine fleur de leur espèce.

Hélas, ce peuple ne parvenait pas à prospérer. Leurs terres étaient devenues arides, les pluies se faisaient rares, et même le soleil semblait les bouder. Ayant toujours été sédentaires, ils dépérissaient, leurs pétales fanaient, et leurs feuilles flétrissaient bien avant l'automne.

Pas question de prendre racine dans cet endroit maudit ! Il en allait de leur survie. La décision fut prise : pour la première fois de leur histoire, les Prolotistes allaient prendre leur courage à deux mains, sortir leurs pots [2] et se lever de leur terre natale pour sautiller vers l'avenir.

1.2 Un nouveau monde

Après moultes péripéties qu'il serait trop long de raconter ici [3], nos chers Prolotistes parviennent à un jardin de forme singulière : un grand potager en carré, muni de ressources en abondance !

Chacune des cases est riche en une ressource en particulier : une substance chimique très stable, du rayonnement électromagnétique émis par un astre céleste, et une énergie thermique assez élevée. Un paradis très propice à la création de magnifiques champs de fleurs !

1. Exactement 6 jours, 4 heures et 57 minutes
2. Moyen de transport agréé SNCF
3. Impliquant un arrosoir, une rousse, des costumes étranges et une pièce de théâtre improvisée

1.3 Oups, des ennuis

Bien sûr, tout cela était bien trop beau, et alors qu'ils prennent leurs quartiers, on vient marcher sur leurs plates-bandes. Les intrus, répondant au nom peu élégant de peuple Tafili [4], arrivent en sautillant de façon très disgracieuse au jardin où les Prolotistes commençent à s'installer. Loin de vouloir partager les lieux, ils semblent vouloir couper l'herbe sous le pied des précédents occupants pour profiter seuls de cet havre de paradis. Qui plus est, ils ont osé les traiter de mauvaises herbes !

Verts de rage, les Prolotistes ne sont absolument pas prêts à céder leur terrain si durement gagné cinq minutes plus tôt. Les plantes, ce n'est pas belliqueux, c'est même plutôt fleur bleue... Mais il en va de leur survie ! Il est temps de montrer à ces Tafili mal rempotés que toute rose a aussi des épines.

1.4 Votre mission

Heureusement, vous veillez au grain ! Désigné volontaire par une assemblée exceptionnelle rassemblée pour assurer la victoire du peuple Prolo, vous voilà général des armées [5]. Votre objectif est de faire manger les pissenlits par la racine à vos adversaires lors de la bataille des Baffles Fleuries, qui débutera Vendredi à 13h42. Pour ce faire, vous devrez gérer les effectifs de vos armées. Mais surtout, au milieu de toute cette violence, n'oubliez pas que vous restez des plantes belles et délicates... Le vainqueur sera déterminé par un score d'élégance !

2 Déroulement de l'affrontement

2.1 Champ de bataille

Le jardin, fruit de toutes les convoitises, servira ici de champ de bataille. Afin de faciliter le déroulement de l'escarmouche, il sera divisé en une grille qui sera toujours de même taille : 20 x 20.

2.2 Ressources

Vos fleurs ont besoin pour se reproduire de certaines richesses. Elles vont collecter à chaque tour de l'eau, de la lumière et de la chaleur. Ces trois ressources sont disponibles en abondance dans votre petit jardin. Chaque case

4. En langue commune, "parasite"
5. Assorti du titre honorifique de Jardinier En Chef

contient une ressource, qui sera répartie équitablement à toutes les plantes qui y ont accès.

Attention :
Le quantité de ressources collectées par case sera divisée par la constante COUT_PAR_CASE_COLLECTE arrondie au supérieur.

2.2.1 Les biomes

	Nom	Ressources
	Gravier	Aucune
	Prairie	Lumière
	Océan	Eau
	Désert	Chaleur
	Oasis	Eau et Chaleur
	Lave	Lumière et Chaleur
	Glace	Eau et Lumière
	Tropical	Eau et Chaleur et Lumière

2.3 Reproduction

Si une case libre est entourée de deux ou plus plantes ayant assez de ressources à leur disposition, une plante poussera sur cette case au prochain tour. Ses statistiques seront la moyenne des statistiques des plantes adjacentes, arrondi au supérieur.

2.4 Plantes

Vos plantes sont vos unités de guerre. Que vous choisissiez de les envoyer au combat ou de farouchement les protéger, elles seront à votre service dès leur naissance. Chaque plante dispose de 4 caractéristiques importantes :

— Sa force physique, qui se traduit par la taille de ses feuilles.
— Sa vie, qui comme vous l'aurez deviné, conduit à sa mort si elle est nulle.
— Son élégance, qui est bien visible à la longueur et la beauté de ses pétales.
— Le rayon de dépotage, communément appelé le rayon de déplacement.

Lorsqu'une plante naît, elle est encore immature et ne peut pas se reproduire. Elle peut par contre récolter des ressources autour d'elle dans un certain rayon de collecte. Il faudra attendre 3 tours pour qu'elle parvienne à l'âge adulte et puisse engendrer de nouvelles plantes. Hélas, toute chose a une fin, et sa mort inévitable se produira au 10e tour. C'est l'histoire de la vie...

Vos fleurs savent parfaitement bien se débrouiller seules : pour avoir une armée toujours plus grande, leur reproduction est donc automatique. Bien entendu, certaines conditions sont nécessaires : la nouvelle jeune pousse doit être adjacente à ses parents adultes qui doivent disposer de suffisamment de ressources pour lui permettre d'avoir une bonne éducation et grandir tranquillement.

2.5 Actions

2.5.1 Arroser

Pour en faire de vraies machines de guerre, vous pouvez choisir d'offrir une petite amélioration à vos plantes à l'aide d'un petit peu d'amour et d'eau fraîche. Arroser une plante permet d'augmenter de 10 points une caractéristique choisie. Ainsi, vous pouvez augmenter sa force pour lui permettre d'envoyer ses adversaires vers d'autres cieux, sa vie pour la rendre plus résistante, son élégance pour booster votre score... Cependant, prenez garde à ne pas en abuser : vous n'avez l'opportunité de le faire qu'une seule fois par plante, et uniquement si elle est déjà assez mature et rend adulte la plante. Un petit bonus de temps en temps, ça ne fait de mal à personne, sauf aux Tafili.

2.5.2 Se déplacer

Afin d'écraser votre adversaire, il vous sera utile de savoir dépoter vos plantes. En effet, celles-ci naissent dans un petit pot, proche de leurs parents. Mais, une fois mises en terre, elles seront immobilisées pour le reste de leur vie. Choisissez bien vos déplacements pour conquérir l'intégralité du jardin !

Attention :
La distance de déplacement maximale sera le rayon de déplacement de la plante divisée par la constante COUT_PAR_CASE_DEPOTAGE arrondie au supérieur.

2.5.3 Baffer

Cet affrontement ne s'appellerait pas la bataille des Baffles Fleuries sans un peu de violence. Pour envoyer sur les roses certains de vos adversaires, et donc réduire leurs points de vie, vous aurez l'opportunité de les baffer. Attention pourtant, la portée maximale d'une attaque n'est que de 6 cases. Vous me direz, c'est bien suffisant pour les mettre en déroute...

2.6 Jeu de chien !

Pour t'aider dans cette lutte et t'aider à débuger, des chiens vont te rejoindre. Ils n'ont aucune incidence sur le jeu, mais tu peux leur dire de se poster sur une parcelle du jardin et ils y resteront jusqu'à leur prochaine instruction. Tu peux appeler autant de chiens de débugs que tu le souhaites et de 3 races différentes : les fleurs bleues, ceux qui ont la main verte et les autres qui voient rouge.

2.7 À la guerre comme à la guerre

Vous êtes tout de même des plantes civilisées qui n'aiment pas le désordre et le chahut, c'est pourquoi la bataille se déroule au tour par tour. Néanmoins, puisqu'il faut bien y avoir un vainqueur, il a été décidé entre les deux camps que les hostilités prendraient fin au 100e tour. À ce moment, le jardinier avec le plus haut score d'élégance l'emporte.

3 API

Constante : NB_JARDINIERS
Valeur : 2
Description : Nombre de jardiniers

Constante : TAILLE_GRILLE
Valeur : 20
Description : Largeur et hauteur de la grille

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

Constante : AGE_MAX
Valeur : 10
Description : Durée de vie maximale d'une plante

Constante : AGE_DE_POUSSE
Valeur : 3
Description : Âge auquel la plante atteint la maturité et peut donc être arrosée

Constante : PORTEE_BAFFE
Valeur : 6
Description : Portée maximale d'une baffe

Constante : NB_TYPES_RESSOURCES
Valeur : 3
Description : Nombre de types de ressources existantes

Constante : APPORT_CARACTERISTIQUE
Valeur : 10
Description : Apport pour une caractéristique lors de l'arrosage

Constante : COUT_PAR_CASE_COLLECTE
Valeur : 15
Description : Nombre de points de caractéristiques nécessaires pour pouvoir collecter une case plus loin

Constante : COUT_PAR_CASE_DEPOTAGE
Valeur : 15
Description : Nombre de points de caractéristiques nécessaires pour pouvoir dépoter une case plus loin

- **erreur**

Description : Erreurs possibles
Valeurs :

OK :	L'action s'est effectuée avec succès
HORS_TOUR :	Il est interdit de faire des actions hors de jouer_tour
HORS_POTAGER :	La case désignée n'est pas dans le potager
CASE_OCCUPEE :	Il y a déjà une plante sur la case ciblée
PAS_DE_PLANTE :	Il n'y a pas de plante sur la case ciblée
MAUVAIS_JARDINIER :	La plante n'appartient pas au bon jardinier
SANS_POT :	La plante est déjà dépotée
DEJA_ARROSEE :	La plante a déjà été arrosée
DEJA_BAFFEE :	La plante a déjà baffé ce tour ci
PAS_ENCORE_ARROSEE :	La plante n'a pas encore été arrosée
PAS_ENCORE_ADULTE :	La plante ne peut pas encore être arrosée
PLANTE_INVALIDE :	Les caractéristiques de la plante sont invalides
TROP_LOIN :	La plante n'a pas un assez grand rayon de dépotage
CARACTERISTIQUE_INVALIDE :	Valeur de 'Caracteristique' inconnue
CHIEN_INVALIDE :	Valeur de 'Chien' inconnue

- **action_type**

Description : Types d'actions
Valeurs :

ACTION_DEPOTER :	Action "depoter"
ACTION_BAFFER :	Action "baffer"
ACTION_ARROSER :	Action "arroser"

- **caracteristique**

Description : Caractéristiques améliorables d'une plante
Valeurs :

CARACTERISTIQUE_FORCE :	Force
CARACTERISTIQUE_VIE :	Vie
CARACTERISTIQUE_ELEGANCE :	Élégance
CARACTERISTIQUE_RAYON_DEPOTAGE :	Portée de dépotage

- **debug_chien**

Description : Types de chien de débug
Valeurs :

AUCUN_CHIEN :	Aucun chien, enlève le chien présent
CHIEN_BLEU :	Chien bleu
CHIEN_VERT :	Chien vert
CHIEN_ROUGE :	Chien rouge

- **position**

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

Description : Position dans le jardin, donnée par deux coordonnées.
Champs :

x :	Coordonnée : x
y :	Coordonnée : y

- **plante**

```
struct plante {
    position pos;
    int jardinier;
    bool adulte;
    bool enracinee;
    int vie;
    int vie_max;
    int force;
    int elegance;
    int rayon_deplacement;
    int rayon_collecte;
    int array consommation;
    int age;
};
```

Description : Une plante
Champs :

pos :	Position de la plante
jardinier :	Jardinier ayant planté la plante
adulte :	La plante est adulte
enracinee :	La plante a déjà été dépotée
vie :	Point(s) de vie restant(s) de la plante
vie_max :	Point(s) de vie maximum de la plante
force :	Force de la baffe de la plante
elegance :	Élégance de la plante
rayon_deplacement :	Distance maximale parcourable par la plante en creusant
rayon_collecte :	Rayon de collecte des ressources pour la plante
consommation :	Quantité de ressources consommées par la plante
age :	Âge de la plante

- **action_hist**

```
struct action_hist {
    action_type atype;
    position position_baffante;
    position position_baffee;
    position position_depart;
    position position_arrivee;
    position position_plante;
    caracteristique amelioration;
};
```

Description : Représentation d'une action dans l'historique
Champs :

atype :	Type de l'action
position_baffante :	Position de la plante baffante (si type d'action "action_baffer")
position_baffee :	Position de la plante baffée (si type d'action "action_baffer")
position_depart :	Position de la plante à déplacer (si type d'action "action_depoter")
position_arrivee :	Position où déplacer la plante (si type d'action "action_depoter")
position_plante :	Position de la plante (si type d'action "action_arroser")
amelioration :	Caractéristique à améliorer (si type d'action "action_arroser")

• **depoter**

```
erreur depoter(position position_depart, position position_arrivee)
```

Description : La plante creuse vers une destination donnée
Paramètres : *position_depart* : Position de la plante à déplacer
position_arrivee : Position où déplacer la plante

• **arroser**

```
erreur arroser(position position_plante, caracteristique
    ↪ amelioration)
```

Description : Arrose une plante
Paramètres : *position_plante* : Position de la plante
amelioration : Caractéristique à améliorer

• **baffer**

```
erreur baffer(position position_baffante, position position_baffee)
```

Description : Une plante en gifle une autre
Paramètres : *position_baffante* : Position de la plante baffante
position_baffee : Position de la plante baffée

• **debug_afficher_chien**

```
erreur debug_afficher_chien(position pos, debug_chien chien)
```

Description : Affiche le chien spécifié sur la case indiquée
Paramètres : *pos* : Case ciblée
chien : Chien à afficher sur la case

• **plantes_jardinier**

```
plante array plantes_jardinier(int jardinier)
```

Description : Renvoie la liste des plantes du jardinier
Paramètres : *jardinier* : ID du jardinier concerné

• **plante_sur_case**

```
plante plante_sur_case(position pos)
```

Description : Renvoie la plante sur la position donnée, s'il n'y en a pas tous les champs sont initialisés à -1
Paramètres : *pos* : Case ciblée

• **plantes_arrosables**

```
plante array plantes_arrosables(int jardinier)
```

Description : Renvoie la liste des plantes du jardinier qui peuvent être arrosées
Paramètres : *jardinier* : ID du jardinier concerné

• **plantes_adultes**

```
plante array plantes_adultes(int jardinier)
```

Description : Renvoie la liste des plantes du jardinier qui sont adultes
Paramètres : *jardinier* : ID du jardinier concerné

• **plantes_depotables**

```
plante array plantes_depotables(int jardinier)
```

Description : Renvoie la liste des plantes du jardinier qui peuvent être dépotées
Paramètres : *jardinier* : ID du jardinier concerné

• **ressources_sur_case**

```
int array ressources_sur_case(position pos)
```

Description : Renvoie les ressources disponibles sur une case donnée
Paramètres : *pos* : Case ciblée

• **reproduction_possible**

```
bool reproduction_possible(position pos, int rayon_collecte, int
    ↪ array consommation)
```

Description : Vérifie si une plante à la position donnée aura suffisamment de ressources pour se reproduire. S'il y a déjà une plante à cette position, le calcul suposera qu'elle a été remplacée

Paramètres :
- *pos* : Case ciblée
- *rayon_collecte* : Rayon de collecte des ressources pour la plante
- *consommation* : Quantité de ressources consommées par la plante

• **plante_reproductible**

```
bool plante_reproductible(position pos)
```

Description : Vérifie si une plante à la position donnée peut se reproduire, retourne faux s'il n'y pas de plante à la position donnée

Paramètres :
- *pos* : Case ciblée

• **croisement**

```
plante croisement(plante array parents)
```

Description : Caractéristiques d'une plante résultant du croisement de plusieurs parents donnés. Les champs sont initialisés à -1 si aucune plante n'est donnée en paramètre

Paramètres :
- *parents* : Les plantes qui sont croisées

• **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.

• **score**

```
int score(int id_jardinier)
```

Description : Renvoie le score du jardinier "id_jardinier". Renvoie -1 si le jardinier est invalide.

Paramètres :
- *id_jardinier* : Numéro du jardinier

• **moi**

```
int moi()
```

Description : Renvoie votre numéro de jardinier.

• **adversaire**

```
int adversaire()
```

Description : Renvoie le numéro du jardinier 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_action_type**

```
void afficher_action_type(action_type v)
```

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

• **afficher_caracteristique**

```
void afficher_caracteristique(caracteristique v)
```

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

• **afficher_debug_chien**

```
void afficher_debug_chien(debug_chien v)
```

Description : Affiche le contenu d'une valeur de type debug_chien
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_plante**

```
void afficher_plante(plante v)
```

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

4 Notes sur l'utilisation de l'API

4.1 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.

4.2 C++

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

4.3 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#.

4.4 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'
```

4.5 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.

4.6 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.

4.7 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)`.

4.8 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)`.

4.9 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.