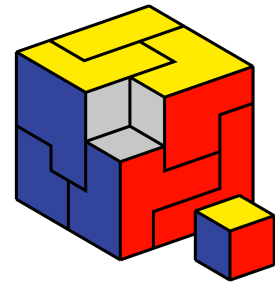


Prologuin

2022



LA BAGUETTE LÉGENDAIRE

La marche des canards

Sujet de la finale du Concours National d'Informatique
Vendredi 27 Mai 2022

Table des matières

1	Contexte	3
2	Objectif	4
3	Joueurs	4
3.1	Troupes	4
4	Parc	6
4.1	Niveau Principal	7
4.2	Niveau Souterrain	8
4.3	Format de la carte	9
5	Règles	9
5.1	Début d'une partie	9
5.2	Apparition et dispersion	9
5.3	Déroulement d'un tour	10
5.4	Comptabilisation du score	10
6	Tournois	11
6.1	Tournois intermédiaires	11
6.2	Rendu final	11
7	Considérations techniques	12
8	API	13
9	Notes sur l'utilisation de l'API	24
9.1	C	24
9.2	C++	24
9.3	C#	24
9.4	Haskell	24
9.5	Java	25
9.6	OCaml	25
9.7	PHP	25
9.8	Python	25
9.9	Rust	26



1 Contexte

La baguette

Inscrite profondément dans la culture française, la baguette prend place dans de nombreux foyers chaque jour. Tartinée de confiture le matin, surplombée de fromage le midi et baignant dans la soupe le soir, ce pain a su conquérir et ravir les Humains.

3 légendes entourent sa création. Certains affirment que c'est sous les boulangers de Napoléon, au début du XIX^{ème} siècle qu'elle naquit, plus légère et moins volumineuse que la miche traditionnelle, la rendant ainsi plus facile à transporter dans les poches des soldats.

Une seconde source avance que c'est l'autrichien August Zang qui, ouvrant une boulangerie à Paris en 1839, proposa ce nouveau format de pain.

Enfin, la dernière légende disparaît sous terre, arguant que dans un chantier du métro parisien dans les années 1900, pour permettre aux ouvriers bretons et auvergnats de se quereller sans risquer la présence d'arme blanche, les maîtres d'œuvre auraient demandé aux boulangers de concevoir un pain se rompant sans couteau.

Ces trois histoires cachent en fait une vérité toute autre. Une vérité que peu d'entre nous connaissent. Car en effet, l'origine de la baguette, ce sont les canards.

Le scientifique

Il y a bien longtemps, l'évolution offrit au monde les canards. Se dressant fièrement sur deux pattes, ces animaux au bec jaune se sont mis en quête de nourriture.

Au fur et à mesure de ses expérimentations, Quack Sparrow, un illustre scientifique canadien, invente le pain. Nous sommes alors en -3000. Si le projet semble intéressant au premier abord, Quack se rend rapidement compte que sa miche n'est pas la solution. Gonflant dans l'estomac de ses congénères, faible en nutriments essentiels, cet aliment représentait un danger pour son espèce.

Il en fera donc don à un autre animal, nettement moins bien développé, qui s'amusait alors à entasser des blocs de pierre dans le désert ¹.

L'entrepreneur

Ce n'est que des milliers d'années plus tard que Chad Firequacker, un jeune entrepreneur, senti un filon. Les Humains avaient depuis longtemps augmenté leur production de pain et, pour une raison obscure, s'étaient mis à le lancer par petits morceaux dans les plans d'eau et les espaces verts.

1. Si la technique employée pour la construction reste encore floue, les experts ²tendent à penser que les chats, véritables maîtres de ce monde, y ont quelque chose à voir

2. Ils ne sont pas forcément experts dans ce domaine

Si ce comportement restait inexplicable, Firequacker se dit qu'il pouvait en tirer profit en rassemblant les morceaux et en vendant le pain alors reconstitué. Muni d'un tuyau à large diamètre pour son stockage, il entreprit de récolter tous les morceaux laissés à l'abandon. Se rendant compte de la complexité temporelle de cette tâche, il tenta une approche auprès des membres de sa troupe³. « Deux sachets de graines à celui qui reconstituera la plus grande miche-allongée-de-pain-reconstituée ! », leur dit-il. L'offre fut si convaincante que bientôt d'autres troupes de canards se prirent au jeu⁴. Tant et si bien que quelques années plus tard, c'est devenu une tradition ancrée dans les mœurs que de créer la plus grosse miche-allongée-de-pain-reconstituée avec toute la famille.

2 Objectif

Dans l'objectif de remporter les deux sachets de graines attendues, vous allez devoir amasser un score total plus élevé que celui de votre adversaire à la fin des NB_TOURS tours. L'obtention de score ne peut se faire qu'en rapportant des miches de pain sur un nid vous appartenant, grâce à l'une de vos troupes. Une troupe ne peut transporter qu'un certain nombre de miches de pain en fonction de sa taille. Une troupe rentrant en collision avec un obstacle, une autre troupe ou elle-même, se disperse, en laissant ses miches de pain au sol.

3 Joueurs

3.1 Troupes

Chaque joueur possède NB_TROUPES troupes chacun. À chaque tour, un joueur doit faire avancer et/ou agrandir ses troupes. Par tour, chaque troupe doit effectuer un certain nombre de ces mouvements. Une troupe est un ensemble de canards contigus, formant une file. On appelle « taille » de la troupe le nombre de canards présents dans la troupe. Le premier canard de la file est appelé la « maman canard » et le dernier canard de la file est appelé le « vilain petit canard ». On appelle « canard antérieur d'un canard » le canard directement adjacent dans la file à celui-ci, en direction de la maman canard.

Similairement, le « canard postérieur » d'un canard est le canard directement adjacent dans la file à celui-ci, en direction du vilain petit canard. Lors d'un déplacement, la maman canard avance d'une case dans une certaine direction, et chaque autre canard prend la place de son canard antérieur. Il est

3. Groupe de canard

4. Oui, vous avez perdu

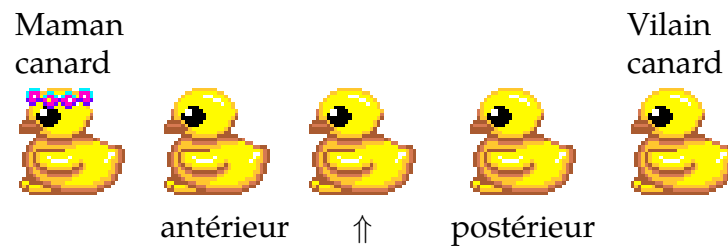


FIGURE 1 – Une troupe de cinq canards

possible d’agrandir la troupe, auquel cas un nouveau canard s’insérera derrière le vilain petit canard au prochain déplacement de la troupe.

Lorsque la maman canard entre en collision avec une case non traversable, la troupe entière se disperse, et la troupe réapparaît, sans son inventaire, et d’une taille de `TAILLE_DEPART` à un point d’apparition défini.

Une troupe dispose d’un inventaire d’une taille qui est déterminée par la taille de la troupe. Il faut 3 canards pour porter une miche de pain. Une troupe peut ainsi transporter un nombre de pains qui correspond à un tiers de sa taille. Cette valeur est également accessible via la fonction `inventaire`.

Une troupe peut collecter des miches de pain situées au sol. Pour ce faire, il suffit de faire avancer la maman canard sur une case contenant une ou plusieurs miches de pain. Ce faisant, la troupe collecte automatiquement la totalité des miches de pain présentes sur la case, tant que la taille de l’inventaire le permet.

Si une troupe se disperse en ayant des pains dans son inventaire, chaque miche de pain est laissée au sol de la manière suivante : la première miche est déposée à la position du canard se trouvant 2 rangs derrière la maman, la suivante par le canard se trouvant 3 rangs derrière celui-ci et ainsi de suite. En mourant, une troupe possédant 4 pains déposera ainsi ses pains aux positions des canards 2, 5, 8 et 11. La même règle s’applique lorsqu’une troupe est divisée par une barrière⁵ de sorte à ce qu’une miche soit toujours portée par trois canards.

3.1.1 Pigeons

Un joueur peut, s’il le souhaite, et à volonté, poser des pigeons sur la carte. Ces pigeons peuvent être de trois couleurs différentes : **bleu**, **jaune** ou **rouge**. Les pigeons n’ont aucune incidence sur la partie, et sont destinés à vous aider pour déboguer votre programme⁶.

5. Voir la section 4.1.3

6. Les pigeons sont à consommer sans modération



FIGURE 2 – Un pigeon

4 Parc

La carte représente un parc constitué de $LARGEUR \times HAUTEUR$ cases, et se décompose en deux niveaux : le niveau principal ainsi que le souterrain. Les troupes se déplacent dans la même carte, et chaque canard composant les troupes se situe sur une case. L'origine, c'est à dire la position d'abscisse, d'ordonnée et de niveau 0, apparaît en bas à gauche du niveau principal.

Le type d'une case indique le type de l'élément présent dessus, notamment les obstacles. Ce type n'est modifiable que par les joueurs, selon des contraintes précises. L'état d'une case ajoute des précisions supplémentaires sur le contenu de la case. Une case peut être traversable ou non par une troupe selon son type et son état.

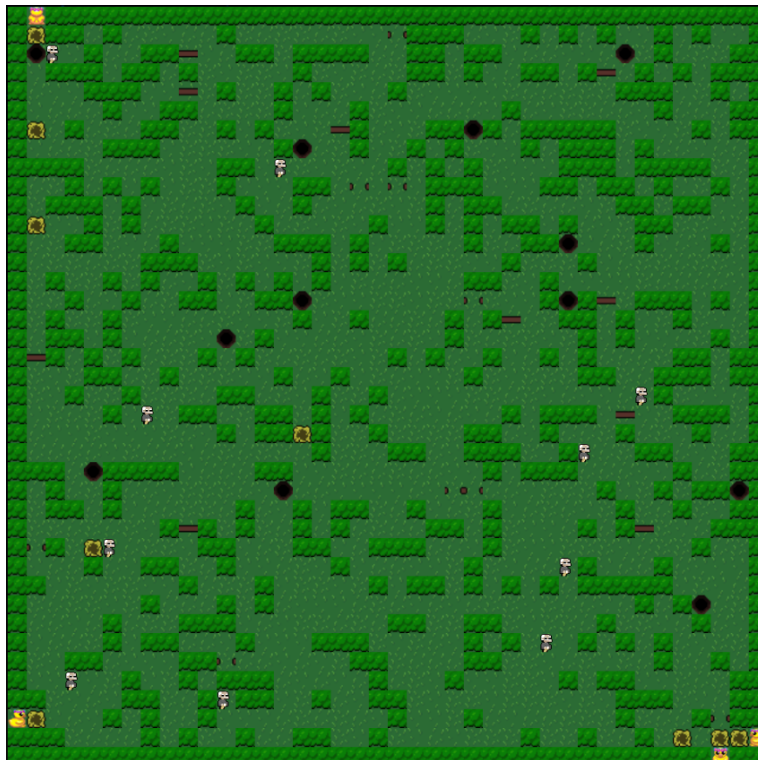


FIGURE 3 – Un parc

4.1 Niveau Principal

Le niveau principal correspond au niveau 0. Il constitue le terrain essentiel du jeu. Il est constitué dès l'initialisation de divers obstacles, et c'est sur ce terrain uniquement qu'il est possible de gagner du score. C'est sur ce niveau que sont situés les 4 points d'apparition. Voici la liste exhaustive des différents types de cases qu'il est possible de rencontrer au niveau principal :

4.1.1 Gazon

Une case de type GAZON est représentée par du gazon. Le gazon peut être constructible ou non, et cela est immuable tant que personne ne construit sur la case. Si une case de gazon est constructible, alors un joueur a la possibilité de poser un buisson sur la case, ce qui change le type de la case en BUISSON, la rendant non traversable. Le gazon est toujours traversable par une troupe.

4.1.2 Buisson

Une case de type BUISSON est représentée par un buisson. Un buisson n'est pas traversable ni constructible. Un buisson est totalement immuable.

4.1.3 Barrière

Une case de type BARRIERE est représentée par une barrière en bois, ouverte ou fermée. Une barrière ouverte est traversable, une barrière fermée n'est pas traversable. Une barrière n'est pas constructible. Au tour TOUR_FERMETURE et uniquement à celui-ci, toute barrière fermée s'ouvre, et toute barrière ouverte se ferme. Une barrière se fermant au milieu d'une troupe entraîne la dispersion de la partie arrière de la troupe, du canard qui était présent sur le portique au vilain petit canard. La partie avant de la troupe se disperse également si sa taille se retrouve inférieure à TAILLE_MIN.

4.1.4 Nid

Une case de type NID est représentée par un nid. Un nid est toujours traversable et non constructible. Chaque nid peut appartenir à un unique joueur ou à aucun des deux joueurs. Un joueur peut prendre possession d'un nid n'appartenant encore à aucun joueur en passant simplement l'une de ses mamans canards sur la case. La prise d'un nid est automatique et ne nécessite aucune autre action. Si une maman canard passe sur un nid lui appartenant, l'inventaire de la troupe est vidé et le score est comptabilisé. Rien ne se passe si une maman canard passe sur un nid adverse.

4.1.5 Papy

Une case de type PAPY est représentée par un papy⁷. Un papy est toujours traversable et non constructible. De manière périodique, tous les INTERVALLE_DISTRIB tours, chaque papy dépose une miche de pain à ses pieds. Cependant, tous les papys ne déposent pas une miche de pain en même temps. Chaque papy possède une phase, c'est-à-dire le numéro du premier tour auquel le papy va poser une miche de pain. La phase peut être différente selon les papys. Un papy peut commencer à distribuer une miche au 2^{ème} tour puis tous les INTERVALLE_DISTRIB tours, tandis qu'un autre papy peut commencer à distribuer une miche au 3^{ème} tour puis tous les INTERVALLE_DISTRIB tours.

4.1.6 Trou

Une case de type TROU est représentée par une fissure dans le sol. Un trou est toujours traversable et non constructible. Si une maman canard se situe sur un trou, elle a la possibilité d'avancer vers le bas afin d'accéder au niveau souterrain. De la même manière, si une maman canard se situe sous un trou, elle a la possibilité d'avancer vers le haut afin d'accéder au niveau principal.

4.2 Niveau Souterrain

Le niveau souterrain correspond au niveau -1. Le niveau souterrain est un niveau alternatif, initialement entièrement rempli de terre. Contrairement au niveau principal, les joueurs peuvent « creuser » ce niveau afin de créer des raccourcis entre différents points du niveau principal.

4.2.1 Terre

Une case de type TERRE est représentée par un bloc de terre. Un bloc de terre n'est ni traversable ni constructible. En revanche, il est possible de creuser un bloc de terre FREQ_TUNNEL fois par tour, ce qui entraîne la transformation de ce bloc en un bloc de type TUNNEL.

4.2.2 Tunnel










Une case de type TUNNEL est représentée par l'absence d'un bloc de terre⁸. Un tunnel est traversable mais pas constructible. Rien ne peut remplacer un tunnel. Il est possible de remonter à la surface depuis un tunnel situé sous un trou en avançant vers le haut.

7. Il n'existe aucune interaction entre les cases de type PAPY et les pigeons contrairement à certains préjugés

8. Vous ne pouvez creuser trop profond, pour ne pas déranger les nains standard !

4.3 Format de la carte

La carte du parc est représentée dans un fichier texte qui contient HAUTEUR lignes et LARGEUR caractères. Pour chaque caractères la représentation ASCII est :

Type	Échantillon	Représentation
Gazon non constructible		' ' (un espace)
Gazon		'.'
Point d'apparition		'S'
Buisson		'#'
Barrière ouverte		'B'
Barrière fermée		'b'
Nid		'N'
Papy		entre '0' et '9'
Trou		'X'

5 Règles

5.1 Début d'une partie

Aucune miche de pain n'est présente sur la carte lors de l'initialisation de la partie. Chaque joueur commence avec NB_TROUPES troupes d'une taille de TAILLE_DEPART, situées sur des points d'apparition distincts.

5.2 Apparition et dispersion

Chaque carte dispose de 4 points d'apparition. Un point d'apparition est situé sur chacune des bordures (Nord, Sud, Est, Ouest) de la carte.

Si un déplacement aurait impliqué que la maman canard se retrouve sur une case non traversable ou sur la même case qu'un autre canard (d'une de vos troupes ou non), cela entraîne une dispersion de l'entière de la troupe. La troupe qui a effectué le mouvement, **et uniquement celle-ci**, se retire de la carte jusqu'à la fin du tour. Au début du prochain tour, la troupe réapparaît, si possible, au point d'apparition associé à la direction correspondant au mouvement tenté.⁹ Ainsi, les collisions entraînées par un mouvement vers le haut

9. Si une troupe tente d'aller vers le Nord alors qu'un mur se situe juste au Nord de sa maman canard, alors la troupe se disperse, et tente de revenir au tour suivant depuis le point d'apparition situé au Nord de la carte.

sont associées au point d'apparition Nord, et les collisions entraînées par un mouvement vers le bas sont associées au point d'apparition Sud.

Si le point d'apparition est déjà occupé par une troupe, alors le point d'apparition choisi sera celui situé juste après celui prévu dans le sens horaire.

Lorsqu'une troupe apparaît, tous les canards n'apparaissent pas en même temps. Ils apparaissent en effet, au fur et à mesure que la troupe avance. Ainsi, la taille de la troupe augmente, commençant à 1 au moment de son apparition jusqu'à atteindre sa taille maximale lorsque tous les canards sont présents.

5.3 Déroulement d'un tour

Chaque troupe possède PTS_ACTIONS points d'action au début de son tour. Tant qu'une troupe est sur le terrain et possède suffisamment de points d'action, elle peut, durant le même tour :

- Avancer dans une direction, contre 1 de ses points d'action
- Grandir, contre COUT_CROISSANCE de ses points d'action

De plus un joueur peut, FREQ_TUNNEL fois durant un tour, et cela gratuitement, creuser un bloc de terre du niveau souterrain.

Finalement, un joueur peut, à volonté, placer un buisson sur une case constructible, en échange de COUT_BUISSON points de score. Attention, ce coût est directement prélevé du score, et il est nécessaire d'avoir un score suffisant pour placer le buisson.

Un tour ne peut se terminer que lorsqu'un joueur n'a plus de point d'action sur aucune de ses troupes ou plus de troupe disponible. Si un joueur termine son tour alors qu'une de ses troupes est encore sur le terrain et possède toujours des points d'action, cette troupe avancera automatiquement dans sa direction actuelle jusqu'à épuisement de ses points d'action.

À la fin de chaque tour, les papys qui doivent déposer une miche de pain le font.

5.4 Comptabilisation du score

Lorsqu'une maman canard arrive sur un nid lui appartenant, l'inventaire de la troupe est vidé sur le nid. Le score du joueur auquel la troupe appartient est augmenté en fonction du nombre de miches de pain déposé. L'incrément de score est calculé par la fonction gain. Globalement, sachez que, pour un nombre de miches de pain donné, **il est bien plus intéressant tout déposer d'un seul coup plutôt qu'en plusieurs fois.**

6 Tournois

6.1 Tournois intermédiaires

Afin de vous aider à perfectionner vos algorithmes, des tournois intermédiaires vous seront proposés. Ces matchs n'ont absolument aucune influence sur le classement final, mais sont néanmoins à prendre au sérieux, car ils vous permettront de vous situer par rapport aux autres joueurs, de connaître vos ennemis, vos points forts et vos faiblesses, et vous donneront des pistes pour vous améliorer pendant la finale. L'annonce des tournois intermédiaires se fera dans l'heure qui précède.

Pour chaque tournoi, nous prendrons le dernier champion que chaque candidat aura envoyé sur le site de soumission pour le faire participer au tournoi, et nous vous donnerons les résultats ainsi que votre progression dès que les tournois se seront terminés, avec un récapitulatif de votre progression globale. Les tournois seront exécutés sur des cartes officielles de notre choix, qui seront potentiellement amenées à changer au fur et à mesure.

Des tournois intermédiaires auront lieu¹⁰ :

- Vendredi 15 h 42 (tournoi test)
- Vendredi 17 h 42
- Vendredi 23 h 42
- Samedi 5 h 42
- Samedi 11 h 42
- Samedi 17 h 42

6.2 Rendu final

Le tournoi final aura lieu samedi à 23 h 42.

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

Lors du tournoi final, plusieurs cartes seront ajoutées. Celles-ci resteront inconnues de tous les joueurs jusqu'à la fin du concours, afin de mesurer l'adaptabilité de vos algorithmes à des situations inconnues.

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

10. Nous nous réservons le droit de changer les horaires et le nombre de tournois intermédiaires

7 Considérations techniques

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

D'autres limitations sont appliquées :

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

8 API

Constante : HAUTEUR
Valeur : 40
Description : Nombre de lignes dans la carte

Constante : LARGEUR
Valeur : 40
Description : Nombre de colonnes dans la carte

Constante : NB_TOURS
Valeur : 400
Description : Nombre de tours à jouer avant la fin de la partie

Constante : TAILLE_DEPART
Valeur : 5
Description : Taille de départ d'une troupe

Constante : TAILLE_MIN
Valeur : 3
Description : Taille minimale qu'une troupe peut avoir avant de se disperser

Constante : NB_TROUPES
Valeur : 2
Description : Nombre de troupes que chaque joueur controle

Constante : INTERVALLE_DISTRIB
Valeur : 5
Description : Intervalle de distribution de pains par les papys

Constante : FREQ_TUNNEL
Valeur : 1
Description : Nombre de tunnels qu'un joueur peut creuser par tour

Constante : PTS_ACTION
Valeur : 5
Description : Nombre de déplacements que peut faire une troupe en un tour

Constante : COUT_CROISSANCE
Valeur : 3
Description : Nombre de points de mouvement requis pour incrémenter la taille

Constante : COUT_BUISSON
Valeur : 3
Description : Coût en score de la pose de buisson

Constante : ROUND_FERMETURE
Valeur : 99
Description : Round à la fin duquel les barrières s'ouvrent ou se ferment

• **erreur**

Description : Erreurs possibles après avoir effectué une action

Valeurs :

OK :	L'action a été effectuée avec succès
TROUPE_INVALIDE :	Mauvais identifiant de troupe
HORS_TOUR :	Aucune action n'est possible hors de joueur_tour
MOUVEMENTS_INSUFFISANTS :	Il ne reste plus assez de points de mouvements pour effectuer l'action demandée
TROP_GRANDI :	La troupe a déjà trop grandi pendant le tour
TROP_CREUSE :	Trop de trous ont déjà été creusés pendant le tour
NON_CREUSABLE :	Il n'est pas possible de creuser à la position demandée
NON_CONSTRUCTIBLE :	La zone demandée n'est pas constructible
SCORE_INSUFFISANT :	Le joueur n'a pas assez de points pour construire un buisson
POSITION_INVALIDE :	La position demandée est hors du parc
DIRECTION_INVALIDE :	La direction spécifiée n'existe pas.
PIGEON_INVALIDE :	Le pigeon spécifié n'existe pas.

• **direction**

Description : Directions possibles

Valeurs :

- NORD* : Sens positif pour les lignes
- SUD* : Sens négatif pour les lignes
- EST* : Sens positif pour les colonnes
- OUEST* : Sens négatif pour les colonnes
- HAUT* : Sens positif pour le niveau
- BAS* : Sens négatif pour le niveau

• type_case

Description : Type de l'élément présent sur une case

Valeurs :

- GAZON* : Absence d'élément
- BUISSON* : Obstacle impossible à traverser
- BARRIERE* : Élément pouvant être ouvert ou fermé.
Une barrière fermée est infranchissable alors qu'une barrière ouverte est analogue à une case vide
- NID* : Élément traversable permettant à la troupe de déposer son inventaire en échange de points
- PAPY* : Élément traversable générant de manière périodique des miches de pain
- TROU* : Interface entre le niveau principal est le niveau souterrain
- TUNNEL* : Bloc du souterrain ayant été creusé
- TERRE* : Bloc du souterrain n'ayant pas encore été creusé

• etat_barriere

Description : État d'une barrière, soit ouvert, soit fermé, soit non-applicable

Valeurs :

- OUVERTE* : La barrière est ouverte
- FERMEE* : La barrière est fermée
- PAS_DE_BARRIERE* : L'élément dont on requiert l'état n'est pas une barrière

• etat_nid

Description : Joueur auquel appartient un nid

Valeurs :

- LIBRE* : Le nid n'a pas été attribué
- JOUEUR_0* : Joueur 0
- JOUEUR_1* : Joueur 1
- PAS_DE_NID* : L'élément dont on requiert l'état n'est pas un nid

• pigeon_debug

Description : Type de pigeon de debug

Valeurs :

<i>PAS_DE_PIGEON</i> :	Aucun pigeon, enlève le pigeon présent
<i>PIGEON_BLEU</i> :	Pigeon bleu
<i>PIGEON_JAUNE</i> :	Pigeon jaune
<i>PIGEON_ROUGE</i> :	Pigeon rouge

• type_action

Description : Types d'actions

Valeurs :

<i>ACTION_AVANCER</i> :	Action "avancer"
<i>ACTION_GRANDIR</i> :	Action "grandir"
<i>ACTION_CONSTRUIRE</i> :	Action "construire buisson"
<i>ACTION_CREUSER</i> :	Action "creuser tunnel"

• position

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

Description : Position dans la carte, donnée par trois coordonnées

Champs :

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

• troupe

```
struct troupe {
    position maman;
    position array canards;
    int taille;
    direction dir;
    int inventaire;
    int pts\_action;
    int id;
};
```

Description : Une troupe, composée de la maman canard et de ses canetons

Champs :

<i>maman</i> :	Position de la maman canard
<i>canards</i> :	Position des différents canards de la troupe, incluant la maman en première position
<i>taille</i> :	Taille de la troupe
<i>dir</i> :	Direction de la troupe
<i>inventaire</i> :	Nombre de pains de la troupe
<i>pts_action</i> :	Nombre de points d'action de la troupe
<i>id</i> :	Identifiant de la troupe

• etat_case

```
struct etat\_case {
    position pos;
    type\_case contenu;
    bool est\_constructible;
    int nb\_pains;
};
```

Description : Élément constituant le parc

Champs :

<i>pos</i> :	Position de la case. Le niveau vaut nécessairement 0
<i>contenu</i> :	Type de la case
<i>est_constructible</i> :	La case est constructible
<i>nb_pains</i> :	Nombre de pains contenus sur la case

• action_hist

```
struct action\_hist {
    type\_action action\_type;
    int troupe\_id;
    direction action\_dir;
    position action\_pos;
};
```

Description : Action représentée dans l'historique

Champs :

<i>action_type</i> :	Type de l'action
<i>troupe_id</i> :	Identifiant de la troupe
<i>action_dir</i> :	Direction de l'action
<i>action_pos</i> :	Position de l'action

- avancer

erreur avancer(int id, direction dir)

Description : La troupe avance d'une case vers une direction donnée

Paramètres : *id* : Identifiant de la troupe à avancer
dir : Direction vers laquelle avancer

- grandir

erreur grandir(int id)

Description : La troupe grandit

Paramètres : *id* : Identifiant de la troupe à faire grandir

- construire_buisson

erreur construire_buisson(position pos)

Description : Construit un buisson à la position donnée

Paramètres : *pos* : Position où construire le buisson

- creuser_tunnel

erreur creuser_tunnel(position pos)

Description : Creuse un tunnel à la position donnée

Paramètres : *pos* : Position de la case à creuser

- info_case

etat_case info_case(position pos)

Description : Renvoie les informations concernant une case

Paramètres : *pos* : Position de la case

- info_barriere

etat_barriere info_barriere(position pos)

Description : Renvoie les informations d'état d'une barrière

Paramètres : *pos* : Position de la barrière

- info_nid

etat_nid info_nid(position pos)

Description : Renvoie les informations d'état d'un nid

Paramètres : *pos* : Position du nid

- papy_tours_restants

int papy_tours_restants(position pos)

Description : Renvoie le nombre de tours restants avant qu'un papy dépose une miche de pain. Retourne -1 si aucun papy ne se trouve à la position demandée

Paramètres : *pos* : Position du papy

- troupes_joueur

troupe array troupes_joueur(int id_joueur)

Description : Renvoie les troupes d'un joueur. Si le joueur est invalide, tous les champs valent -1.

Paramètres : *id_joueur* : Numéro du joueur concerné

- pains

position array pains()

Description : Renvoie la position des pains récupérables

- debug_poser_pigeon

erreur debug_poser_pigeon(position pos, pigeon_debug pigeon)

Description : Pose un pigeon de debug sur la case indiquée

Paramètres : *pos* : Case où poser le pigeon

pigeon : Pigeon à afficher sur la case

- 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 debug n'apparaissent pas dans cette liste.

- gain

```
int gain(int nb\_pains)
```

Description : Renvoie le gain en score que le nombre de pains passé en entrée rapporterait s'ils étaient tous déposés d'un coup dans un nid

Paramètres : *nb_pains* : Nombre de miches de pain déposées

- inventaire

```
int inventaire(int taille)
```

Description : Renvoie la taille de l'inventaire d'une troupe de taille donnée

Paramètres : *taille* : Taille de la troupe

- trouver_chemin

```
direction array trouver\_chemin(position depart, position arrivee)
```

Description : Trouve un plus court chemin ouvert entre deux positions. Renvoie une liste vide si les deux positions sont égales ou si aucun chemin n'existe.

Paramètres : *depart* : Position de départ
arrivee : Position d'arrivée

- moi

```
int moi()
```

Description : Renvoie votre numéro de joueur.

- adversaire

```
int adversaire()
```

Description : Renvoie le numéro du joueur adverse.

- score

int score(int id_joueur)

Description : Renvoie le score du joueur 'id_joueur'. Renvoie -1 si le joueur est invalide.

Paramètres : *id_joueur* : Numéro du joueur concerné

- 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_direction

void afficher_direction(direction v)

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

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_etat_barriere

void afficher_etat_barriere(etat_barriere v)

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

Paramètres : *v* : The value to display

- afficher_etat_nid

void afficher_etat_nid(etat_nid v)

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

Paramètres : *v* : The value to display

- afficher_pigeon_debug

void afficher_pigeon_debug(pigeon_debug v)

Description : Affiche le contenu d'une valeur de type pigeon_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_position

void afficher_position(position v)

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

Paramètres : *v* : The value to display

- afficher_troupe

void afficher_troupe(troupe v)

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

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 être 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.

Que la baguette légendaire soit avec toi !