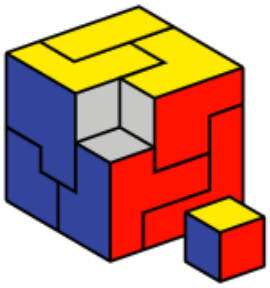
Prologin 2023



L'Archéologie en 5 dimensions

Avec des chiens qui voyagent dans le multivers !!

Sujet de la finale du Concours National d'Informatique
Vendredi 19 Mai 2023

Table des matières

1 Introduction **3**

2 Objectif **4**

3 Joueurs **4**
3.1 Chiens . . . 5
3.2 Maîtres . . . 7

4 Carte **8**
4.1 Flaque . . . 8
4.2 Vide . . . 8
4.3 Mur . . . 9
4.4 Arbre . . . 9
4.5 Squelette . . . 10
4.6 Niche . . . 11

5 Les Portails **12**

6 Règles **14**
6.1 Début d'une partie . . . 14
6.2 Déroulement d'un tour . . . 15
6.3 Comptabilisation du score . . . 16

7 Tournois **16**
7.1 Tournois intermédiaires . . . 16
7.2 Rendu final . . . 17

8 Considérations techniques **17**

9 API **19**
9.1 Constantes . . . 19
9.2 Énumérations . . . 21
9.3 Structures de données . . . 22
9.4 Fonctions . . . 24

10 Notes sur l'utilisation de l'API **33**
10.1 C . . . 33
10.2 C++ . . . 33
10.3 C# . . . 33
10.4 Haskell . . . 33
10.5 Java . . . 34
10.6 OCaml . . . 34
10.7 PHP . . . 34
10.8 Python . . . 34
10.9 Rust . . . 35


PROLOGIN
2023

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 !

Dans les qualifications et les demi-finales, vous avez suivi les aventures de six jeunes à la recherche d'un moyen de voyager dans l'univers de leurs films préférés. Malgré des difficultés pour trouver le bon film et les obstacles technologiques rencontrés, ils ont réussi grâce à vous à construire une machine à voyager dans le temps, qui les a transportés dans différentes époques, de l'Antiquité au Futur.

Aujourd'hui, dans cette finale, vous allez devoir aider des chiens archéologues du futur, capables de voyager dans le temps, à récupérer un maximum de ressources dans trois époques différentes. Nous sommes convaincus que vous saurez relever ce défi avec brio, grâce à vos compétences en algorithmique et à votre créativité.

Que vous soyez au collège ou en études supérieures, peu importe votre âge, sachez que vous êtes ici parce que vous avez un niveau remarquable. Ne vous découragez pas, quel que soit le résultat final de cette compétition, vous avez déjà accompli quelque chose d'extraordinaire en étant sélectionné pour la finale.

Nous tenons également à vous féliciter pour vos talents en construction de tour de boîtes à pizza. Sérieusement, c'était très fort et très marrant, bien joué à vous.

Nous vous souhaitons bonne chance et espérons que vous apprécierez cette aventure unique dans le temps avec vos nouveaux compagnons, les chiens pentadimensionnels !

Une Pizza Six Papattes Sous Terre

Il sera une fois, dans un futur lointain, où les chiens seront les meilleurs amis des archéologues. Une grande découverte sera faite : les archéologues retrouveront des indices prouvant l'existence d'une mystérieuse boîte à pizza enterrée sous une couche de terre de plusieurs milliers d'années. Cette boîte est censée contenir les restes de la civilisation la plus avancée de l'histoire de l'humanité.

Cependant, les informations concernant la localisation de cette boîte ne sont pas encore assez précises, et deux équipes d'archéologues se sont lancées

sur la recherche d'indices. Mais chaque équipe pense que la boîte contient des ressources inestimables pour leur propre époque, et que leur succès dépend de la quantité de ressources qu'ils pourront récupérer !

Les chiens pentadimensionnels sont certes capables de voyager à travers le temps et l'espace pour récupérer les ressources qui se trouvent dans différentes époques. Cependant, ils ont besoin d'aide pour décider où aller et quoi chercher.

C'est là que vous entrez en jeu ! Votre don en logique et en algorithmique est précieux pour aider votre équipe de chiens à décider où chercher et comment le faire de manière efficace. Chaque équipe d'archéologie sera dirigée par les instructions des candidats pour récupérer les ressources à travers les différentes époques.

2 Objectif

Deux équipes d'archéologie vont s'affronter lors de chaque partie. Chaque équipe est composée de 4 chiens et 6 maîtres. Votre objectif va être de diriger votre équipe afin d'obtenir un score plus élevé que l'équipe adverse à la fin de la partie.

Chaque équipe peut également se déplacer dans trois époques. Vous commencez dans la première époque, et vous pouvez quand vous le souhaitez passer à l'époque suivante. Attention cependant, vous ne pouvez pas revenir en arrière dans les époques !

Il existe trois types de ressources que vos chiens peuvent récupérer pour obtenir des points, chaque ressource correspondant à une époque. Il n'est possible de récupérer une ressource que pendant son époque. Il faut donc passer à l'époque d'une ressource pour pouvoir la récupérer, et une fois que vous quittez l'époque d'une ressource, il ne sera plus jamais possible de la récupérer.

Afin de récolter une ressource, il faut qu'un de vos chiens la récupère sur la carte en se positionnant exactement sur la case où la ressource est contenue, puis la ramène à votre unique niche.

Chaque combinaison d'une ressource provenant de chaque époque permet de faire avancer la recherche, c'est pourquoi votre score correspond au produit du nombre de ressources en votre possession de chaque époque. [1]

3 Joueurs

Chaque joueur contrôle au total 10 personnages : 4 chiens et 6 maîtres. À chaque tour, tous les personnages commencent avec un certain nombre de

1. Voir le calcul exact dans la section 6.3

points de mouvement, lui permettant d'effectuer diverses actions. Contre un point de mouvement, un personnage peut se déplacer d'une case vers le haut, le bas, la gauche ou la droite, si la carte le lui permet. En particulier, un personnage ne peut jamais se déplacer sur la même case qu'un autre personnage, ni se déplacer dans certaines cases tel que les murs. Les détails exacts concernant quelles cases il est possible de traverser sont présents dans la section 4.

Chaque joueur peut également voyager dans trois différentes époques :
— L'époque 0, appelée « âge interstellaire » ;
— L'époque 1, appelée « âge cybernétique » ;
— L'époque 2, appelée « âge pizzalogique ».
Les deux joueurs commencent à l'âge 0. Lorsqu'un joueur le souhaite, il peut passer à l'âge suivant grâce à l'action `age_suivant`. **Cette action est irréversible**. Il n'est pas possible de retourner à une époque antérieure. **L'âge n'est pas commun aux deux joueurs**. Les deux joueurs peuvent se situer dans deux époques différentes.

Cependant, même si les deux joueurs se situent dans deux époques différentes, il n'est toujours pas possible pour deux personnages d'une époque différente de se positionner sur la même case. [2]

3.1 Chiens



FIGURE 1 – Un chien



FIGURE 2 – Une croquette



FIGURE 3 – Un os

Les chiens commencent le tour avec 4 points de mouvement. La Figure 1 représente un chien dans l'interface graphique.

Croquettes Les croquettes sont des objets utiles pour améliorer les déplacements de vos chiens. La Figure 2 représente une croquette dans l'interface graphique. Le processus d'apparition des croquettes est expliqué plus en détail dans la section 4.4.

Si un chien se déplace sur une case contenant une ou plusieurs croquettes, il récupère la totalité des croquettes dans son inventaire.

Si une croquette apparaît sur le chien, le chien ne la récupère pas. Il faut que le chien se déplace sur la case lorsque la croquette est déjà présente pour la récupérer.

2. Cette régulation a été mise en vigueur le 4 février 2690 afin d'éviter les accidents temporels. En effet, si deux personnages se positionnaient sur la même case et qu'un personnage venait à voyager dans l'époque de l'autre personnage, cela entraînerait un malencontreux accident !

Un chien ne peut pas récupérer une croquette suspendue à un arbre [3]. C'est un maître qui doit récupérer la croquette pour la donner au chien.

Un chien peut à tout moment consommer une croquette de son inventaire pour gagner deux points de mouvement, grâce à l'action `manger`.

Si un chien se situe à proximité d'un portail, il peut également consommer une croquette pour traverser le portail, grâce à l'action `prendre_portail`. Ceci est expliqué plus en détail dans la section 5.

À la fin du tour, les chiens perdent la totalité des croquettes de leur inventaire [4].

Os Les os sont les objets permettant aux joueurs de gagner des points. La Figure 3 représente un os dans l'interface graphique.

Il existe trois types d'os correspondants aux trois époques du jeu :
— Les os interstellaires, correspondants à l'époque 0 ;
— Les os cybernétiques, correspondants à l'époque 1 ;
— Les os pizzalogiques, correspondants à l'époque 2.

Pour récupérer un os, un chien doit également se déplacer sur la même case que l'os. **Si un os apparaît sur le chien, le chien ne le récupère pas**. Il faut que le chien se déplace sur la case lorsque l'os est déjà présent.

Un chien ne peut récupérer ou transporter un os que si le joueur se situe à la même époque que l'os. Si un chien se positionne sur une case contenant un os qui ne vient pas de son époque, l'os disparaît mais n'est pas récupéré par le chien. [5] Le processus d'apparition des os est expliqué plus en détail dans la section 4.5.

Un chien peut se permettre de porter jusqu'à 10 os du même type dans son inventaire. Lorsqu'un chien se positionne sur une case où se trouve un os du même type que l'époque du joueur, si son inventaire n'est pas complet, il récupère l'os dans son inventaire. Au contraire, si son inventaire est complet, l'os disparaît, mais n'est pas récupéré par le chien. [6]

Lorsqu'un chien se positionne sur sa niche (voir section 4.6), la totalité des os de son inventaire sont récoltés par le joueur. **À la fin de la partie, les os présents dans l'inventaire des chiens ne sont pas comptabilisés**. Il faut impéra-

3. Les chiens archéologues ne montent pas sur les arbres. Ramasser les croquettes est la spécialité des pandas-roux d'Anjou.

4. Les chiens archéologues ne sont pas assez disciplinés pour résister à l'envie de manger les croquettes.

5. En effet, les chiens pentadimensionnels ont un odorat si puissant qu'ils peuvent flairer des ossements dans le futur et dans le passé. Cependant, s'ils ne peuvent pas le récupérer, ils peuvent tout de même l'enterrer plus profondément afin que l'équipe adverse ne puisse pas le retrouver.

6. Le chien essaye tout de même de porter l'os, mais étant surchargé, il finit malheureusement par le faire tomber, l'endommageant trop fortement pour qu'il garde une quelconque valeur.

tivement repasser par la niche pour déposer les os, ce qui permet de récupérer les points. Les chiens conservent les os dans leur inventaire à travers les tours.

3.2 Maîtres



FIGURE 4 – Un maître

Les maîtres commencent le tour avec 8 points de mouvement. La Figure 4 représente un maître dans l'interface graphique.

Un maître peut conserver jusqu'à trois croquettes dans son inventaire. Lorsqu'un maître se déplace sur une case contenant une croquette, si son inventaire n'est pas plein, il récupère la croquette dans son inventaire. Les maîtres conservent les croquettes dans leur inventaire à travers les tours. Au contraire, si son inventaire est plein, la croquette n'est pas altérée.

Les maîtres peuvent également récupérer une croquette suspendue à un arbre de la même manière.

Si une croquette apparaît sur le maître, le maître ne la récupère pas. Il faut que le maître se déplace sur la case lorsque la croquette est déjà présente pour la récupérer.

Grâce à l'action `donner_croquette`, un maître peut directement transférer une croquette de son inventaire à l'inventaire d'un chien adjacent. [7]

Grâce à l'action `poser_croquette`, un maître peut également déposer une croquette de son inventaire au sol, sur sa case ou sur une case adjacente. Il peut y avoir plusieurs croquettes au sol sur la même case, cependant il ne peut y avoir qu'une croquette suspendue par case.

Contre quatre points de mouvement, un maître peut effectuer l'action `construire` afin de construire un mur sur une case adjacente.

Contre quatre points de mouvement également, un maître peut effectuer l'action `detruire` afin de détruire un mur situé sur une case adjacente.

Un maître possède également la capacité de déplacer un portail sur la carte. Cette mécanique de jeu est décrite plus en détail dans la section 5.

7. Ici et pour tout le reste du sujet, une case adjacente est une case accessible en un déplacement, c'est-à-dire juste à gauche, juste à droite, juste en haut ou juste en bas. Deux cases en diagonale ne sont pas adjacentes.

4 Carte

La carte est une carte carrée composée de 40×40 cases. Les différents types de cases sont expliqués dans cette section.

4.1 Flaque



FIGURE 5 – Une flaque

La Figure 5 représente une case de type `FLAQUE` dans l'interface graphique.

Une flaque ne peut pas être traversée ni altérée d'une quelconque manière. Il est impossible de déplacer une quelconque entité sur une flaque, que ce soit un chien, un maître ou un portail. Il est également impossible de placer une croquette ou de construire un mur sur une flaque.

4.2 Vide



FIGURE 6 – Un terrain vide

Les cases de type `VIDE` représentent les terrains vagues. La Figure 6 représente une case vide dans l'interface graphique.

Les cases vides sont possibles à traverser par n'importe quelle entité si aucune autre entité n'est déjà présente sur la case, que ce soit un chien, un maître ou un portail. [8] Il est possible de placer une croquette sur un terrain vide. Si aucun personnage ni portail n'est présent sur la case, il est possible de construire un mur sur un terrain vide, le transformant en `MUR` (voir section 4.3). Si une croquette se trouve sur la case lorsqu'un mur est construit, la croquette est détruite. Si un portail est poussé sur une case vide où se trouve une croquette, la croquette est également détruite.

8. Lorsque l'on mentionne « une case vide », cela fait référence au type de la case et non son contenu. Une case de type `VIDE` peut contenir un chien, un maître, un portail ou une croquette, et sera tout de même référée en tant que « case vide ».

4.3 Mur



FIGURE 7 – Un mur

Les cases de type `MUR` représentent un terrain vague sur lequel un mur a été construit. Il n'est possible de construire un mur que sur une case vide. La Figure 7 représente un mur dans l'interface graphique.

Les cases `MUR` sont impossibles à traverser par n'importe quelle entité, que ce soit un chien, un maître ou un portail. Il est impossible de placer une croquette sur un mur.

Il est cependant possible pour un maître de détruire un mur s'il se trouve sur une case adjacente, ce qui transforme la case en case `VIDE` (voir section 4.2).

4.4 Arbre



FIGURE 8 – Un arbre

Les cases de type `ARBRE` représentent un arbre à croquettes. La Figure 8 représente un arbre dans l'interface graphique.

Un arbre à croquettes fait apparaître périodiquement une croquette suspendue sur sa case. Chaque arbre possède une période, qu'il est possible d'obtenir grâce à la fonction `info_arbre`.

Si une croquette est déjà suspendue sur l'arbre au moment où l'arbre est censé faire apparaître une croquette, aucune croquette supplémentaire n'apparaît. Il ne peut donc y avoir qu'une seule croquette suspendue à chaque arbre. Cependant, il est possible d'avoir des croquettes au sol sur la même case qu'un arbre et une croquette suspendue à l'arbre simultanément.

Par exemple, si la période d'un arbre est 5 et que la première apparition d'une croquette est prévue au tour 3, alors aux tours 3, 8, 13, 18 et cætera, si aucune croquette n'est déjà présente suspendue dans l'arbre, alors une croquette apparaît.

Il est possible pour un chien ou un maître de se déplacer sur un arbre, cependant, seul le maître peut récupérer les croquettes suspendues dans les arbres. Il est possible de poser une croquette au sol sous un arbre, qu'un chien

peut alors récupérer. Il n'est pas possible de construire un mur sur un arbre [9]. Il n'est pas possible de pousser un portail sur un arbre.

4.5 Squelette



FIGURE 9 – Un squelette



FIGURE 10 – Un squelette avec un os interstellaire



FIGURE 11 – Un squelette avec un os cybernétique



FIGURE 12 – Un squelette avec un os pizzalogique

Les cases de type `SQUELETTE` représentent les gisements d'os. La Figure 9 représente un squelette dans l'interface graphique.

Un squelette fait apparaître périodiquement un os sur sa case de la même manière qu'un arbre à croquettes fait apparaître ses croquettes. Chaque squelette possède un type et une période, qu'il est possible d'obtenir grâce à la fonction `info_squelette`.

Le type du squelette indique le type d'os qui apparaîtra sur la case. Le type d'un squelette est constant tout le long de la partie.

Si un os est déjà présent sur la case au moment où le squelette est censé faire apparaître un os, aucun os supplémentaire n'apparaît.

Par exemple, si la période d'un squelette est 6 et que la première apparition d'un os est prévue au tour 5, alors aux tours 5, 11, 17, 23 et cætera, si aucun os n'est présent sur la case du squelette, alors un os apparaît. Il est possible pour un chien ou un maître de se déplacer sur un squelette. Un chien qui se déplace sur un os le ramasse ou le détruit. Un os se fait détruire si l'âge du joueur est différent de celui de l'os ou que le chien a déjà le nombre maximum d'os. Il est possible de poser une croquette sur un squelette. Il n'est pas possible de construire un mur sur un squelette [10]. Il n'est pas possible de pousser un portail sur un squelette.



FIGURE 13 – Une niche

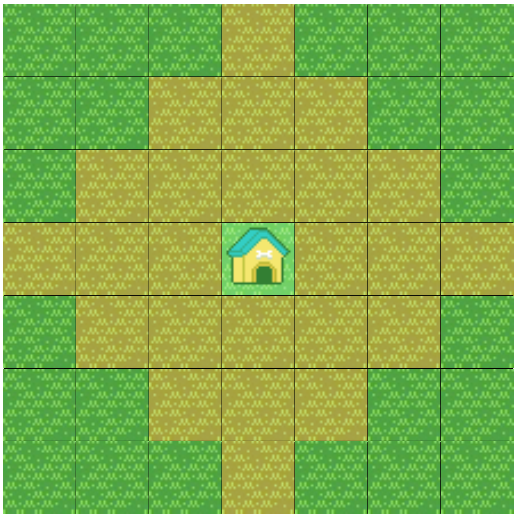


FIGURE 14 – La zone de protection de la niche

4.6 Niche

Les cases de type `NICHE` représentent la niche d'un joueur. La Figure 13 représente une niche dans l'interface graphique.

Chaque joueur possède une unique niche dans la carte. Il est possible de retrouver la position de la niche d'un joueur grâce à la fonction `niche_joueur`.

25 cases en forme de diamant sont protégées autour des niches des joueurs. La Figure 14 montre la zone de protection autour de chaque niche. Il est impossible pour un joueur de déplacer un de ses maîtres ou un de ses chiens dans la zone de protection adverse. Il est impossible de pousser un portail dans une zone de protection. Il est impossible de construire un mur dans une zone de protection.

Un chien passant sur sa niche y dépose la totalité des os de son inventaire. Les os sont alors récoltés et entraîneront un gain de points pour le joueur.

Il est possible pour un chien ou un maître de se déplacer sur sa niche. Il est possible de poser une croquette sur une niche.



FIGURE 15 – Un portail

5 Les Portails

Au début de chaque tour, des portails peuvent apparaître à divers endroits de la carte, indiquant qu'un chien du futur[11] a voyagé dans le présent ! La Figure 15 représente un portail dans l'interface graphique.

La fonction `apparition_portails` permet de prédire à quels tours et où les portails vont apparaître. Les portails apparaissent au début du tour indiqué dans cette fonction, si la case est vide. Si un personnage ou un mur se trouve sur la case au moment où le portail est censé apparaître, alors le portail n'apparaît pas.

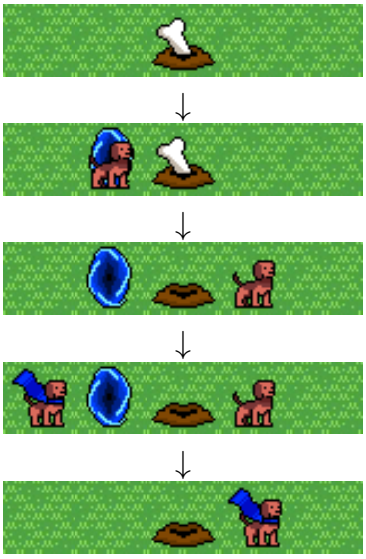


FIGURE 16 – Voyage temporel d'un chien à travers un portail

Voyages temporels Lorsqu'un portail apparaît, un chien du futur sort du portail. Ce chien appartient sûrement à l'un des deux joueurs, mais cela n'est pas déterminé au moment où le portail apparaît. Tant que l'appartenance du chien n'est pas déterminée, aucun joueur ne contrôle le chien. Le chien est joué automatiquement en suivant cet algorithme :

9. La loi 117 de l'*Organisation des Systèmes Stellaires*, interdit de couper, déplacer ou altérer un arbre pour tout aménagement sans autorisation des autorités locales.
10. Les squelettes sont de très mauvaises fondations, particulièrement pour les murs.
11. Jusqu'à ce qu'on sache à qui appartient réellement ce chien, nous dirons que se sont des vagabonds

— Si une croquette ou un os est accessible, il se dirige vers la ressource la plus proche pour tenter de la récupérer. Si plusieurs croquettes ou os sont à la même distance, alors il fera le premier déplacement dans l'ordre (est, ouest, sud, nord) qui diminue strictement sa distance minimum aux ressources.
— Sinon, il ne bouge pas.

Il est possible de simuler exactement les déplacements prévus du chien grâce à la fonction `deplacements_chien`. Cependant, notez que les déplacements indiqués dépendent fortement de la position des autres entités sur la carte. Toute modification de la carte (construction ou destruction d'un mur, déplacement d'une entité ou apparition d'un portail) peut changer le comportement futur du chien, et alors, rappeler la fonction `deplacements_chien` donnera son trajet mis à jour.

Les chiens dont l'appartenance n'est pas encore déterminée effectuent leur « tour » juste après le tour de chaque joueur. Cependant, ils possèdent deux fois moins de points de déplacements par tour.

Il est possible d'obtenir la liste des portails en présence [12] ainsi que la liste des chiens sortis de chaque portail grâce à la fonction `liens_temporels`.

Contre une croquette, un chien peut traverser un portail qui lui est adjacent. Cela ne lui coûte aucun point de mouvement !

Lorsqu'un chien d'un joueur traverse un portail par lequel un chien d'origine inconnue est sorti, alors le mystère est résolu, c'est ce chien-là qui a voyagé dans le passé ! Le chien voyage alors dans le passé et le portail disparaît avec lui. À ce moment-là, l'appartenance du chien vagabond qui était sorti de ce portail est déterminée, et le joueur prend dorénavant le contrôle de ce chien à la place.

L'inventaire du chien dont l'appartenance vient d'être déterminée correspond alors à la somme de l'inventaire du chien qui vient de passer dans le portail, plus la totalité des objets que le chien a récupérés en étant vagabond. Si cette somme dépasse la capacité d'inventaire du chien, alors le surplus est détruit.

Appeler la fonction `inventaire_chien` sur un chien vagabond vous indiquera le nombre d'os correspondant à l'age demandé que le chien a pu récupérer en étant vagabond. [13]

Après avoir traversé un portail, un chien conserve la totalité de ses points de mouvement.

La Figure 16 montre un exemple d'interaction avec un portail.

12. c'est-à-dire : qui sont apparus et pas disparus.
13. En réalité, le chien ne récupère que les os de l'époque à laquelle il appartient, et les autres os sont détruits. Cependant, tant que personne ne sait quel chien a traversé le portail, personne ne sait de quelle époque appartient le chien. En attendant, vous pouvez grâce à cette fonction déterminer le nombre d'os que votre chien récupérerait en fonction de l'époque à laquelle vous appartiendrez lorsque vous prendrez le portail.

1. La première image montre la situation initiale ;
2. Dans la deuxième image, un portail temporel apparaît et un chien d'origine inconnue en sort ;
3. La troisième image montre le chien vagabond récoltant la ressource la plus proche : un os ;
4. Dans la quatrième image, un chien appartenant à l'équipe bleue se met dans une case adjacente au portail afin d'être en mesure de l'emprunter ;
5. Dans la cinquième image, contre une croquette, le chien de l'équipe bleue traverse le portail pour voyager dans le passé. On connaît désormais l'identité du chien vagabond. Le chien se retrouve alors deux cases plus loin, avec la totalité de ses points de mouvement, et un os de plus, qu'il a récupéré en étant vagabond.

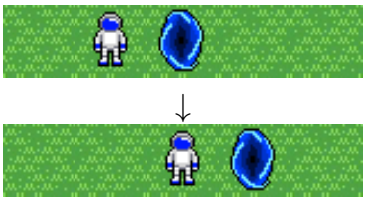


FIGURE 17 – Déplacement d'un portail par un maître

Déplacer un portail De plus, un maître a la capacité de déplacer un portail sur la carte en le poussant. Lorsqu'un maître tente de se déplacer vers un portail, si la case derrière le portail est libre, alors le portail sera déplacé sur cette case par le maître. Sinon, le déplacement est refusé.

La Figure 17 montre un exemple de maître poussant un portail vers la droite.

Il n'est possible de déplacer un portail que sur une case de type `VIDE` ne contenant aucun maître, chien ou autre portail. Il n'est pas non plus possible de déplacer un portail dans une zone protégée par une niche.

Déplacer un portail sur une case contenant une ou plusieurs croquettes au sol détruit les croquettes.

6 Règles

6.1 Début d'une partie

Aucune croquette n'est présente sur la carte lors de l'initialisation de la partie. Chaque joueur commence avec 4 chiens et 6 maîtres répartis dans la zone de protection de leur niche respective.

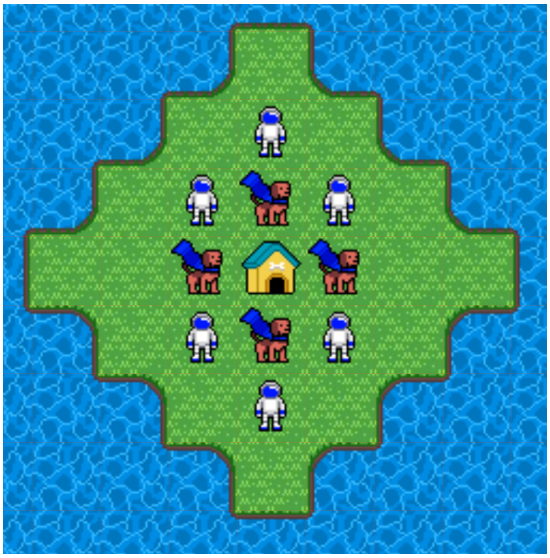


FIGURE 18 – Position initiale

La Figure 18 représente un exemple de position initiale d'une équipe dans la partie.

Une partie comprend 100 rounds. À la fin du centième round, la partie prend fin et le vainqueur est déterminé.

Un round se déroule comme suit :

1. Le joueur 0 effectue son tour ;
2. Les chiens vagabonds effectuent leurs déplacements ;
3. Le joueur 1 effectue son tour ;
4. Les chiens vagabonds effectuent leurs déplacements.

Un round équivaut donc à deux tours !

6.2 Déroulement d'un tour

Chaque entité contrôlée par un joueur possède un certain nombre de points de mouvement au début de son tour, dépendant du type de l'entité.

— Un chien contrôlé par un joueur commence son tour avec 4 points de mouvement ;
— Un maître commence son tour avec 8 points de mouvement ;
— Un chien d'origine inconnue commence son tour avec 2 points de mouvement.

Durant le tour d'un joueur, un joueur peut, tant qu'il le souhaite :

— Déplacer une entité grâce à la fonction `se_deplacer`, contre un point de mouvement de l'entité déplacée ;
— Faire construire un mur par un maître sur une case adjacente grâce à la fonction `construire`, contre 4 points de mouvement du maître concerné ;

— Faire détruire un mur par un maître sur une case adjacente grâce à la fonction `detruire`, contre 4 points de mouvement du maître concerné ;
— Déplacer un chien dans un portail adjacent grâce à la fonction `prendre_portail`, contre une croquette de l'inventaire du chien concerné ;
— Faire manger une croquette à un chien pour lui faire gagner deux points de mouvement grâce à la fonction `manger`, contre une croquette de l'inventaire du chien concerné ;
— Faire donner par un maître une croquette à un chien adjacent, grâce à la fonction `donner_croquette`;
— Faire poser par un maître une croquette par terre, grâce à la fonction `poser_croquette`;
— Passer à l'Âge suivant, grâce à la fonction `age_suivant`.

Les apparitions des os et des croquettes sur les squelettes et les arbres se font en début de tour. Les apparitions de portails se font également en début de tour.

6.3 Comptabilisation du score

Lorsqu'un chien passe sur sa niche, il y dépose la totalité des os présents dans son inventaire. Le score du joueur dépend exclusivement du nombre d'os de chaque type déposés dans sa niche.

Le score final du joueur correspond à

$$(1 + I) \times (1 + C) \times (1 + P),$$

où I correspond au nombre d'os interstellaires récoltés, C correspond au nombre d'os cybernétiques récoltés, et P correspond au nombre d'os pizza-logiques récoltés.

7 Tournois

7.1 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 lieu [14] :
— Vendredi 15 h 42 (tournoi test)
— Vendredi 17 h 42
— Vendredi 23 h 42
— Samedi 7 h 42
— Samedi 11 h 42
— Samedi 17 h 42

7.2 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.

8 Considérations techniques

Vous disposez d'une seconde (temps réel !) à chaque fois qu'une de vos fonctions est appelée pour prendre 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 ;

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

— 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.

9 API

9.1 Constantes

Constante : LARGEUR
Valeur : 40
Description : Largeur de la carte

Constante : HAUTEUR
Valeur : 40
Description : Hauteur de la carte

Constante : NB_TOURS
Valeur : 200
Description : Nombre de tours à jouer avant la fin de la partie

Constante : NB_CHIENS
Valeur : 4
Description : Nombre de chiens contrôlés par joueur

Constante : NB_MAITRES
Valeur : 6
Description : Nombre de maîtres contrôlés par joueur

Constante : DEPLACEMENT_MAITRE
Valeur : 8
Description : Nombre de points de mouvement par tour des maîtres

Constante : DEPLACEMENT_CHIEN
Valeur : 4
Description : Nombre de points de mouvement par tour des chiens

Constante : COUT_PORTAIL
Valeur : 1
Description : Coût en croquettes pour traverser un portail

Constante : COUT_CONSTRUCTION
Valeur : 4
Description : Coût en points de mouvement de la construction d'un mur

Constante : COUT_DESTRUCTION
Valeur : 4
Description : Coût en points de mouvement de la destruction d'un mur

Constante : PORTEE_MAITRE
Valeur : 1
Description : Portée des maîtres pour pouvoir construire, détruire ou donner des croquettes à un chien

Constante : NB_AGES
Valeur : 3
Description : Nombre d'Âges

Constante : NUTRITION
Valeur : 2
Description : Nombre de points de mouvement gagnés en mangeant une croquette

Constante : TAILLE_INVENTAIRE_MAITRE
Valeur : 3
Description : Taille de l'inventaire d'un maître en croquettes

Constante : TAILLE_INVENTAIRE_CHIEN
Valeur : 10
Description : Taille de l'inventaire d'un chien en os

Constante : PROTECTION_NICHE
Valeur : 3
Description : Distance de Manhattan de protection d'une niche

9.2 Énumérations

- **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
POSITION_INVALIDE :	La position fournie est invalide
CIBLE_INVALIDE :	L'entité demandée est invalide
DERNIER_AGE :	Vous êtes déjà dans le dernier Âge
HORS_PORTEE :	Vous êtes trop loin de la cible
DEPLACEMENT_INVALIDE :	Vous ne pouvez pas vous déplacer sur la case
POSE_INVALIDE :	Vous ne pouvez pas poser une croquette sur la case
PLUS_DE_CROQUETTES :	Cette entité n'a plus de croquettes
PLUS_DE_PM :	Vous n'avez plus de points de mouvement
CONSTRUCTION_IMPOSSIBLE :	Vous ne pouvez pas construire sur la case
DESTRUCTION_IMPOSSIBLE :	Vous ne pouvez rien détruire sur la case

- **type_case**

Description : Contenu topographique d'une case

Valeurs :

VIDE :	Terrain vide
NICHE :	Niche
MUR :	Mur
FLAQUE :	Flaque
ARBRE :	Arbre à Croquettes
SQUELETTE :	Squelette

- **papatte_debug**

Description : Type de papatte de debug

Valeurs :

PAS_DE_PAPATTE :	Aucune papatte, enlève la papatte présente
PAPATTE_BLEU :	Papatte bleue
PAPATTE_JAUNE :	Papatte jaune
PAPATTE_ROUGE :	Papatte rouge

- **type_action**

Description : Types d'actions
Valeurs :

ACTION_SE_DEPLACER :	Action "se_deplacer"
ACTION_PRENDRE_PORTAIL :	Action "prendre_portail"
ACTION_MANGER :	Action "manger"
ACTION_CONSTRUIRE :	Action "construire"
ACTION_DETRUIRE :	Action "detruire"
ACTION_DONNER_CROQUETTE :	Action "donner_croquette"
ACTION_POSER_CROQUETTE :	Action "poser_croquette"
ACTION_AGE_SUIVANT :	Action "age_suivant"

9.3 Structures de données

- **position**

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

Description : Position dans la carte, donnée par deux coordonnées
Champs :

colonne :	Abscisse
ligne :	Ordonnée

- **lien_temporel**

```
struct lien_temporel {
    position chien;
    position portail;
};
```

Description : Couple de positions d'un portail et son chien associé
Champs :

chien :	Position du chien
portail :	Position du portail

- **apparition_portail**

```
struct apparition_portail {
    int tour;
    position pos_portail;
};
```

Description : Tour et position d'appartition d'un portail
Champs :

tour :	Tour d'appartition du portail
pos_portail :	Position du portail

- **etat_case**

```
struct etat_case {
    type_case contenu;
    position pos_case;
    bool est_constructible;
    int nb_croquette;
};
```

Description : Description complète d'une case
Champs :

contenu :	Contenu topographique de la case
pos_case :	Position de la case
est_constructible :	Est-il possible de construire sur la case
nb_croquette :	Nombre de croquettes sur la case

- **etat_squelette**

```
struct etat_squelette {
    int prochain_squelette;
    int type_squelette;
    int periode_squelette;
    bool a_squelette;
};
```

Description : Description complète d'un squelette
Champs :

prochain_squelette :	Tours restants avant la prochaine apparition
type_squelette :	Âge de récolte de la ressource pouvant apparaître
periode_squelette :	Période d'apparition des os
a_squelette :	Existe-t-il un os récupérable sur le squelette

- **etat_arbre**

```
struct etat_arbre {
    int prochain_croquette;
    int periode_croquette;
    bool a_croquette;
};
```

Description : Description complète d'un arbre à croquettes
Champs :

prochain_croquette :	Tours restants avant la prochaine apparition
periode_croquette :	Période d'apparition des croquettes
a_croquette :	Existe-t-il une croquette sur l'arbre

• **action_hist**

```
struct action_hist {
    type_action action_type;
    position src;
    position dest;
};
```

Description : Action représentée dans l'historique
Champs : *action_type* : Type de l'action
src : Première position
dest : Deuxième position

9.4 Fonctions

• **se_deplacer**

```
erreur se_deplacer(position entite, position arrive)
```

Description : Permet de déplacer une entité
Paramètres : *entite* : Position de l'entité
arrive : Position d'arrivée

• **prendre_portail**

```
erreur prendre_portail(position chien, position portail)
```

Description : Permet d'emprunter un portail avec un chien
Paramètres : *chien* : Position du chien
portail : Position du portail

• **manger**

```
erreur manger(position chien)
```

Description : Permet à un chien de manger une croquette et d'augmenter son nombre de déplacement
Paramètres : *chien* : Position du chien

• **construire**

```
erreur construire(position maitre, position mur)
```

Description : Permet de construire un mur avec un maître sur une case qui lui est adjacente
Paramètres : *maitre* : Position du maître
mur : Position du mur à construire

• **detruire**

```
erreur detruire(position maitre, position mur)
```

Description : Permet de détruire un mur avec un maître sur une case qui lui est adjacente
Paramètres : *maitre* : Position du maître
mur : Position du mur à détruire

• **donner_croquette**

```
erreur donner_croquette(position maitre, position chien)
```

Description : Permet à un maître de donner une croquette à un chien
Paramètres : *maitre* : Position du maître
chien : Position du chien

• **poser_croquette**

```
erreur poser_croquette(position maitre, position croquette)
```

Description : Permet à un maître de poser une croquette sur une case adjacente
Paramètres : *maitre* : Position du maître
croquette : Position de la croquette

• **age_suivant**

```
erreur age_suivant()
```

Description : Permet de passer à l'Âge suivant

• **info_case**

```
etat_case info_case(position pos)
```

Description : Renvoie les informations concernant une case.
Paramètres : *pos* : Position de la case

• **info_squelette**

```
etat_squelette info_squelette(position pos)
```

Description : Renvoie les informations concernant un squelette. Tous les champs seront à -1 si la position ne contient pas de squelette.
Paramètres : *pos* : Position du squelette

• **info_arbre**

```
etat_arbre info_arbre(position pos)
```

Description : Renvoie les informations concernant un arbre à croquettes. Tous les champs seront à -1 si la position ne contient pas d'arbre.
Paramètres : *pos* : Position de l'arbre

• **apparition_portails**

```
apparition_portail array apparition_portails()
```

Description : Renvoie la liste des positions et des tours auquels les portails vont apparaître.

• **positions_portails**

```
position array positions_portails()
```

Description : Renvoie la liste des positions des portails.

• **liens_temporels**

```
lien_temporel array liens_temporels()
```

Description : Renvoie la liste des liens temporels entre un portail et un chien.

• **points_mouvement**

```
int points_mouvement(position pos)
```

Description : Renvoie le nombre de points de mouvement restants d'une entité.
Paramètres : *pos* : Position de l'entité

• **age_joueur**

```
int age_joueur(int joueur)
```

Description : Renvoie l'Âge actuel d'un joueur. Renvoie -1 si le joueur est invalide.
Paramètres : *joueur* : Joueur inspecté

• **niche_joueur**

```
position niche_joueur(int joueur)
```

Description : Renvoie la position de la niche d'un joueur. Renvoie la position (-1, -1) si le joueur est invalide.
Paramètres : *joueur* : Joueur inspecté

• **chiens_joueur**

```
position array chiens_joueur(int joueur)
```

Description : Renvoie la liste des positions des chiens d'un joueur. Renvoie une liste vide si le joueur est invalide.
Paramètres : *joueur* : Joueur inspecté

• **maitres_joueur**

```
position array maitres_joueur(int joueur)
```

Description : Renvoie la liste des positions des maîtres d'un joueur. Renvoie une liste vide si le joueur est invalide.
Paramètres : *joueur* : Joueur inspecté

• **score**

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

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

• **score_age**

```
int score_age(int joueur, int age)
```

Description : Renvoie le score d'un Âge d'un joueur. Renvoie -1 si le joueur ou l'Âge est invalide.
Paramètres : *joueur* : Joueur inspecté
age : Âge inspecté

• **inventaire_chien**

```
int inventaire_chien(position pos, int age)
```

Description : Renvoie la quantité d'os que possède le chien à un âge donné. Renvoie -1 si la position ne contient pas de chien.
Paramètres : *pos* : Position du chien
age : Âge des os

• **nb_croquettes**

```
int nb_croquettes(position pos)
```

Description : Renvoie la quantité de croquettes de l'entité Renvoie -1 si la position ne contient pas d'entité.
Paramètres : *pos* : Position du chien

• **debug_poser_papatte**

```
erreur debug_poser_papatte(position pos, papatte_debug papatte)
```

Description : Pose une papatte de debug sur la case indiquée.
Paramètres : *pos* : Case où poser la papatte
papatte : Papatte 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 débug n'apparaissent pas dans cette liste.

• **deplacements_chien**

```
position array deplacements_chien(position pos, int k)
```

Description : Renvoie les k prochains déplacements d'un chien d'origine inconnue à une position donnée.

Paramètres : *pos* : Position du chien
k : Nombre des prochains déplacements

• **trouver_chemin**

```
position array trouver_chemin(position depart, position arrivee, int
    ↪ joueur)
```

Description : Renvoie une liste de positions reliant le départ à l'arrivée.

Paramètres : *depart* : Position de départ
arrivee : Position d'arrivée
joueur : Joueur à simuler (0 ou 1), 2 pour un chien vagabond

• **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 : La valeur à afficher

• **afficher_type_case**

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

Description : Affiche le contenu d'une valeur de type type_case
Paramètres : v : La valeur à afficher

• **afficher_papatte_debug**

```
void afficher_papatte_debug(papatte_debug v)
```

Description : Affiche le contenu d'une valeur de type papatte_debug
Paramètres : v : La valeur à afficher

• **afficher_type_action**

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

Description : Affiche le contenu d'une valeur de type type_action
Paramètres : v : La valeur à afficher

• **afficher_position**

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

Description : Affiche le contenu d'une valeur de type position
Paramètres : v : La valeur à afficher

• **afficher_lien_temporel**

```
void afficher_lien_temporel(lien_temporel v)
```

Description : Affiche le contenu d'une valeur de type lien_temporel
Paramètres : v : La valeur à afficher

• **afficher_apparition_portail**

```
void afficher_apparition_portail(apparition_portail v)
```

Description : Affiche le contenu d'une valeur de type apparition_portail
Paramètres : v : La valeur à afficher

• **afficher_etat_case**

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

Description : Affiche le contenu d'une valeur de type etat_case
Paramètres : v : La valeur à afficher

• **afficher_etat_squelette**

```
void afficher_etat_squelette(etat_squelette v)
```

Description : Affiche le contenu d'une valeur de type etat_squelette
Paramètres : v : La valeur à afficher

• **afficher_etat_arbre**

```
void afficher_etat_arbre(etat_arbre v)
```

Description : Affiche le contenu d'une valeur de type etat_arbre
Paramètres : v : La valeur à afficher

• **afficher_action_hist**

```
void afficher_action_hist(action_hist v)
```

Description : Affiche le contenu d'une valeur de type action_hist
Paramètres : v : La valeur à afficher

10 Notes sur l'utilisation de l'API

10.1 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.

10.2 C++

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

10.3 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#.

10.4 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'
```

10.5 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.

10.6 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.

10.7 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)`.

10.8 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)`.

10.9 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.

Vous avez déployé les chiens.
(Les chiens absorbent l'artefact.)