

Concours de programmation 2005
I.U.T. De Lens

3 Mars 2005

Table des matières

1	Les circonstances	4
2	Plus sérieusement	4
3	Éléments du jeu	4
3.1	Les surfaces	4
3.1.1	L'usine	4
3.1.2	Surface de jeu globale	4
3.1.3	Surface de jeu locale	4
3.2	Les zones	4
3.2.1	Zone neutre	4
3.2.2	Zone neutre d'origine	5
3.2.3	Zone neutre de destination	5
3.2.4	Zone hors jeu	5
3.3	Les tapis roulants	5
3.3.1	Tapis roulant uni-directionnel	5
3.3.2	Tapis roulant bi-directionnel	5
3.3.3	Tapis bi-directionnels liés	5
3.4	Les paquets	5
3.4.1	Perdre un paquet	6
3.4.2	Déplacer les paquets	6
3.4.3	Collision de paquets	6
3.4.4	Redirection d'un paquet	6
3.4.5	Bug de collision	6
3.5	Le Joueur	6
3.5.1	Position d'un joueur	6
3.6	Les coordonnées	6
3.6.1	Coordonnées globales	6
3.6.2	Coordonnées locales	7
3.6.3	Changement de position	7
3.6.4	Inversion de tapis	7
4	Les règles du jeu	7
4.1	But du jeu	7
4.2	Unité atomique	7
4.3	Rounds	7
4.4	Tour	7
4.5	Action	8
4.6	Nombre d'actions par tour	8
4.7	Passer son tour	8
4.8	Durée du jeu	8
4.9	Gagner des points	8
4.10	Augmentation du score d'un paquet	8
4.11	Débit de paquets	8
4.12	Dysfonctionnements	8
4.13	Fin du jeu	9
4.14	Vainqueur	9

5	Programmer les IA	9
6	Utilisation de Stallmanite	9
6.0.1	Compilation des IA	9
6.0.2	Exécution du programme	10
7	Programmer	10
7.1	Le Type vector	10
7.2	Le Type packet	10
7.3	Constantes fondamentales	11
7.4	Autres constantes	11
7.5	Manipulation des vecteurs	12
7.6	Manipulation des paquets	12
7.7	Actions	13
7.8	Enregistrement des variables	13
7.9	Utilisation des coordonnées globales	13
A	Exemple d'IA	15
A.1	L'idée	15
A.2	Quelques idées de stratégies	16
A.2.1	Tout sauver!	16
A.2.2	Visez les gros!	16
A.2.3	Saisir sa chance	17
A.2.4	Perfectionnisme	17
A.3	Note sur les stratégies	17

1 Les circonstances

Vous êtes ingénieur en robotique dans une entreprise multinationale chargée du traitement de la **stallmanite**, matériel destiné à la fabrication des femto-processeurs supercordoïdaux.

Mais la société est en péril : suite aux conséquences désastreuses du bug de l'an 65536, les machines sont devenues complètement imprévisibles et menacent de faire chuter les fragiles paquets remplis à ras-bord de stallmanite de première qualité!

Complètement affolé, votre patron en personne vous appelle au beau milieu de votre réveillon, en proposant à qui sauvegardera les précieux paquets une prime colossale, jamais vue depuis les pots de vins de Microsoft (empire féodal tyrannique)

Par conscience professionnelle (et aussi un peu pour la prime), vous vous armez de courage face aux robots devenus fou.

2 Plus sérieusement

Votre but est de programmer un plugin pour un jeu dont le but est de déplacer des paquets dans des champs de tapis roulants. Ce document présente toutes les informations nécessaires à la bonne compréhension des règles du jeu, ainsi que pour programmer les plugins.

3 Éléments du jeu

3.1 Les surfaces

3.1.1 L'usine

L'usine est le lieu d'où arrivent les paquets. Elle est située exactement au centre de la surface de jeu globale.

3.1.2 Surface de jeu globale

On entend par ce terme la surface de jeu totale, c'est à dire qui s'étend d'en haut à gauche jusqu'en bas à droite de la partie de l'écran utilisée par le jeu.

3.1.3 Surface de jeu locale

Chaque joueur possède une surface de jeu locale qui lui est propre, caractérisée par une colorisation rouge, verte, bleue ou jaune.

3.2 Les zones

3.2.1 Zone neutre

Les zones neutres sont les tapis roulants qui n'appartiennent à aucun des joueurs. C'est les seuls tapis qui ne sont pas colorés. Chaque surface de jeu locale est limitée par deux zones neutres.

3.2.2 Zone neutre d'origine

La zone neutre d'origine est la zone neutre d'où proviennent les paquets qui rentrent dans une surface de jeu locale. Ex : la zone neutre d'origine du joueur rouge est composée des quatre cases grises directement sous la surface de jeu locale rouge.

3.2.3 Zone neutre de destination

La zone neutre de destination est la neutre à l'extrémité d'une surface de jeu locale où les paquets une fois engagés ne peuvent plus revenir sur la surface de jeu locale. La zone neutre de destination du joueur rouge est composée des quatre cases grises directement à droite de la surface de jeu locale rouge.

3.2.4 Zone hors jeu

Les zones aux extrémités de la surface de jeu globale, représentées en noir, sont dites zones hors jeu.

3.3 Les tapis roulants

La plus grande partie de la surface de jeu est remplie de tapis roulants. Certains de ces tapis roulants sont uni-directionnels, d'autres sont bi-directionnels.

3.3.1 Tapis roulant uni-directionnel

Un tapis roulant uni-directionnel roulera toujours vers la même direction, quoi qu'il arrive.

3.3.2 Tapis roulant bi-directionnel

Les tapis roulants bi-directionnels, sous l'effet d'un joueur ou d'un paquet, sont susceptibles de changer de sens. Si l'un d'eux se déplaçait de haut en bas, il se déplacerait à présent de bas en haut.

3.3.3 Tapis bi-directionnels liés

La zone neutre d'origine contient des tapis bi-directionnels liés entre eux. Pour le joueur rouge, les trois tapis les plus à gauche de l'écran dans la zone neutre d'origine sont liés. L'un d'entre eux, le tapis actif, va vers la zone de jeu locale du joueur. Les trois autres, les tapis passifs, vont vers une zone hors jeu. Lorsqu'un paquet est redirigé par le tapis actif, un nouveau tapis actif sera sélectionné aléatoirement parmi les 4 tapis actifs disponibles, et les autres tapis deviendront passifs. Notez bien que sur les cinq tapis constituant la zone neutre d'origine, celui le plus proche de l'usine n'est pas lié aux autres mais unidirectionnel.

3.4 Les paquets

Des paquets se déplacent sur les tapis roulants. Un paquet arrivant sur un tapis roulant sera redirigé vers la direction indiquée par ce tapis. Par réaction, si ce tapis est bi-directionnel, le sens de ce tapis sera inversé.

3.4.1 Perdre un paquet

Si un paquet sort de la surface de jeu globale pour “tomber” dans une zone hors jeu, celui-ci est perdu et disparaît.

3.4.2 Déplacer les paquets

En inversant judicieusement le sens de certains tapis roulants, les joueurs doivent être capables de superviser le déplacement des paquets.

3.4.3 Collision de paquets

Il y a collision de paquets lorsque le déplacement de deux paquets les conduits au même emplacement. Lors d’une collision, les paquets rebondissent l’un sur l’autre et reviennent à leur emplacement initial.

3.4.4 Redirection d’un paquet

Un paquet apporté par un joueur dans sa zone neutre de destination se verra redirigé vers la zone neutre d’origine du joueur adjacent.

3.4.5 Bug de collision

Malgré tout le soin apporté au programme, un bug de collision persiste. Lors d’une collision, les paquets reviennent chacun à leur case initiale. Si ce retour provoque à nouveau une collision avec un autre paquet, la collision n’a pas lieu et les paquets deviennent confondus.

Afin de palier à ce bug, l’un des paquets est immédiatement détruit, et son total de point est ajouté au paquet confondu. De cette façon, le joueur concerné ne perdra aucun point suite aux conséquences de ce bug.

3.5 Le Joueur

Le joueur est représenté par les faisceaux de lumières plus intense que l’on trouve sur la surface de jeu locale d’un joueur.

3.5.1 Position d’un joueur

La position d’un joueur est la position de l’intersection des deux faisceaux de lumières.

3.6 Les coordonnées

3.6.1 Coordonnées globales

Les coordonnées globales sont les coordonnées génériques, utilisées pour représenter la totalité de la zone de jeu. L’origine du repère est le point en haut à gauche de l’écran. Les coordonnées progressent d’elles même, vers la droite pour les abscisses et vers le bas pour les ordonnées.

3.6.2 Coordonnées locales

Les coordonnées locales sont les coordonnées relatives à votre surface de jeu locale. L'origine est l'angle de votre surface de jeu locale située vers l'angle de la surface de jeu globale le plus proche de vous. Le repère est orienté en fonction de votre couleur.

Les coordonnées locales considèrent en réalité que vous êtes toujours le joueur rouge. Le repère sera rapporté au repère rouge, quel que soit votre position sur la zone de jeu.

Les coordonnées locales ne prenant en compte qu'une seule zone, il est impossible d'accéder hors de la surface de jeu locale par les coordonnées locales. Les coordonnées locales couvrent toute la surface de jeu locale, plus les tapis roulants en zone neutre directement adjacents à votre surface de jeu locale.

3.6.3 Changement de position

La première actions consiste à déplacer le faisceau de lumière dans l'une des quatre directions standards (droite, haut, gauche, bas).

3.6.4 Inversion de tapis

La seconde action disponible consiste à inverser le tapis roulant situé sur la position du joueur.

4 Les règles du jeu

4.1 But du jeu

Le but du jeu est d'apporter un maximum de paquet de la zone neutre d'origine jusqu'à la zone neutre de destination.

4.2 Unité atomique

L'unité de base utilisée est la case. Il n'est pas possible (et il est totalement inutile) de mesurer une distance plus petite qu'une case.

4.3 Rounds

Une partie contient un nombre de round pré-défini. Chaque round dure un certain nombre de secondes. On peut observer le round actuel par le nombre en blanc situé en haut au milieu de l'écran.

4.4 Tour

Un tour de jeu représente le temps qu'un paquet met pour aller d'un tapis roulant à l'autre. Pendant un tour de jeu, chaque joueurs est capable d'effectuer certaines actions.

4.5 Action

Le programmeur peut, à l'aide des API présentées ci-dessous, faire effectuer à son joueurs certaines actions.

4.6 Nombre d'actions par tour

Chaque tour de jeu, le joueur commander une et une seule action.

4.7 Passer son tour

Cette action consiste à passer son tour sans réaliser aucune autre action. Notez que si aucune action n'a été donnée à réaliser à la fin du script, le passage de tour sera choisit par défaut.

4.8 Durée du jeu

Une partie durera 3 minutes. Durant ces 3 minutes se dérouleront 6 rounds, de 30 secondes chacun. Un jeu complet contient 3 parties, ce qui fait un total de 9minutes.

4.9 Gagner des points

Chaque paquet rapporté en zone neutre finale permet de gagner un nombre de point variable.

4.10 Augmentation du score d'un paquet

Un paquet sortant de l'usine vaut 10 points. A chaque fois que ce paquet est apporté dans la zone neutre finale d'un joueur, sa valeur augmente de 10 points. Plus un paquet est vieux sur le terrains de jeu, plus sa valeur est grande.

4.11 Débit de paquets

Le débit des paquets est déterminé en fonction du round actuel. Selon le round courant, les paquets sortiront de l'Usine à un interval de tours donné. Durant les premiers round, le débit de paquets est faible, mais accélère au fil des rounds qui passent.

- Round 1 : 1 paquets tous les 12 tours
- Round 2 : 1 paquets tous les 8 tours
- Round 3 : 1 paquets tous les 4 tours

4.12 Dysfonctionnements

Si votre script contient des erreurs qui entraîne son arrêt (erreur de segmentation, erreur mathématique, etc...) ou des boucles/fonctions récursives infinies, le programme mettra immédiatement fin à son exécution. Si une action a été déclenchée avant le plantage, celle-ci sera annulée et le tour sera perdu. Si une variable a été enregistrée grâce aux fonctions SaveInt/SaveTab (voir plus loin), cette variable sera conservée.

4.13 Fin du jeu

Le jeu se termine au bout d'un temps donné. On peut observer un compte à rebours en blanc en bas au milieu de l'écran indiquant le temps restant avant la fin du jeu.

4.14 Vainqueur

Le vainqueur est celui qui possède le plus de points à la fin du temps requis.

En cas d'ex-aequo, une partie sera relancée avec les joueurs à départager. Le premier joueur qui apportera un paquet à destination sera déclaré vainqueur. Si les deux joueurs apportent un paquet en même temps, on attendra le paquet suivant, jusqu'à ce qu'un joueur apporte un paquet avant l'autre.

5 Programmer les IA

Afin de sauver un maximum de paquets, vous aurez besoin de programmer les actions des faisceaux de lumière.

Vous disposez pour cela de certains outils pour observer le déroulement du jeu, et d'autres pour assigner des actions.

Les programmes doivent être réalisés en langage C++. Toutes les extensions du langage peuvent être utilisés. En revanche, aucune bibliothèque autre que celle présentée ci-dessous n'est autorisée. On tolérera cependant l'utilisation de fonction d'affichage sur la sortie standard, telles que `cout` ou `printf`, pour faciliter le débogage des IA des joueurs. En revanche, aucun de ces fonction ne devra plus être présente lors de la remise des IA à la fin du concours.

Le nom de la bibliothèque utilisée est "ia_user.h". Toute IA créé devra donc commencer par la ligne : `#include "ia_user.h"`

Puisque vous réalisez une extension d'un programme déjà existant, et non pas un programme indépendant, vous ne devez pas réaliser une fonction `main`, mais une fonction appelée `IaMain` que vous devrez écrire de la façon suivante :

```
extern 'C' void IaMain()
{
}
```

En annexe vous trouverez un exemple très simple d'IA programmée.

6 Utilisation de Stallmanite

6.0.1 Compilation des IA

Toute IA devra préalablement être compilée avant d'être utilisé. Étant donné les options fastidieuses à fournir au compilateur `g++` pour permettre cette compilation, un script nommé "cia" a été créé pour automatiser la tâche. Vous pourrez l'appeler simplement grâce à la ligne de commande :

```
cia fichier_source
```

fichier_source devra être remplacé par le chemin du fichier à compiler. Seuls les extensions de fichiers .cc et .cpp sont supportées par le script. votre script devra impérativement avoir le nom de votre login. l'utilisateur de nom foo devra donc avoir un fichier source nommé foo.cpp (ou foo.cc)

6.0.2 Exécution du programme

Pour exécuter le programme, vous devrez utiliser la commande “stallmanite” utilisé de la façon suivante :

```
stallmanite login [login] [login] [login]
```

login devra être votre nom d'utilisateur, qui est aussi le nom de votre ia. Le premier login entré sera le joueur rouge. Le second sera le joueur bleu. Le troisième sera le joueur vert et le quatrième sera le joueur jaune. Un seul joueur est obligatoire. Si la partie n'est pas complète, les tapis des joueurs manquants seront réorienté pour passer automatiquement tout les paquets à sa charge au joueur suivante. Les points de ce paquet ne seront pas augmenté après passage dans la zone de jeu du joueur absent.

7 Programmer

7.1 Le Type vector

la structure vector permet de représenter une position sur la zone de jeu à l'aide de ses coordonnées (abscisses et ordonnées) :

```
typedef struct
{
    int x;
    int y;
} vector;
```

7.2 Le Type packet

```
typedef struct
{
    vector pos;
    int score;
    int direction;
} packet;
```

- **pos** représente la position sur le terrain de jeu du paquet.
- **score** représente le nombre de point qui sera rapporté si le paquet est amené à destination
- **direction** est la direction actuelle du paquet (qui n'est pas forcément - et même rarement - celle du tapis roulant sous lequel il se trouve)

7.3 Constantes fondamentales

```
int RED
int GREEN
int BLUE
int YELLOW
```

Représentent les quatre joueurs différents.

```
int UNCOLOR
```

Ne représente aucune couleur. (généralement utilisé pour signaler un territoire neutre).

```
int RIGHT
int TOP
int LEFT
int BOTTOM
```

Représentent les quatre directions utilisées.

```
int NOTHING
```

Aucune direction. Généralement utilisé pour représenter un tapis roulant statique.

```
int MAX_ROUND
```

Nombre de round avant la fin du jeu.

```
int MAX_TIME
```

Temps total au début de la partie, en seconde.

```
int LOCAL_SIZE
```

Nombre de cases de côté comprises dans une surface de jeu locale.

7.4 Autres constantes

```
int BORDER_SIZE
```

Représente la taille des bords, soit les parties de la surface de jeu qui n'ont pas de tapis roulants et où les paquets sont susceptibles d'être perdus. (en général égale à 1)

```
int MIDDLE_SIZE
```

Représente la largeur des zones neutre. (en général égale à 2)

```
int TERRAIN_SIZE
```

Représente la taille totale du terrain. (égal à **BORDER_SIZE * 2 + LOCAL_SIZE * 2 + MIDDLE_SIZE**)

7.5 Manipulation des vecteurs

`vector Vector(int x, int y);`

Renvoie un vecteur d'abscisse **x** et d'ordonnée **y**.

7.6 Manipulation des paquets

`int NumPacket();`

Cette fonction renvoie le nombre de paquets accessibles via vos coordonnées locales. Les paquets des autres joueurs seront donc ignorés.

`packet GetPacket(int pos);`

Cette fonction renvoie le paquet de position **pos** accessibles via vos coordonnées locales. Les paquets des autres joueurs seront donc ignorés. Le premier paquet est le paquet de position 0, et chaque position suivante contient un paquet jusqu'au dernier paquet de la liste.

ATTENTION : si la position est supérieure au nombre de paquet - 1, le résultat sera indéfini. Dans le pire des cas, vous pouvez y perdre votre tour.

`vector PlayerPos();`

Renvoie la position locale de votre joueur.

`int Turn();`

Renvoie le nombre de tours écoulés depuis le début de la partie.

`int Round();`

Renvoie le round actuel.

`int TimeLeft();`

Renvoie le temps restant avant la fin de la partie.

`int NewThrow();`

Renvoie le nombre de tours restant avant la prochaine distribution de paquets.

`int Random(int max);`

Renvoie un nombre aléatoire compris entre 0 et **max-1**.

`int Score(int player=SELF);`

Renvoie le score actuel du joueur passé en paramètre (**RED**, **GREEN**, **BLUE** ou **YELLOW**). Votre score sera renvoyée si vous laissez le paramètre par défaut (**SELF**).

`int RollingFloor(vector pos, bool active=true);`

Cette fonction renvoie la direction actuelle du tapis roulant situé aux coordonnées locales passées en argument. Si la variable **active** est positionnée à **true**, c'est la direction active du tapis roulant qui sera renvoyée. Sinon, c'est la position inactive qui sera renvoyée.

Renvoie **NOTHING** si la zone indiquée n'est pas un tapis roulant, ou si la zone inactive d'un tapis roulant unidirectionnel est demandée.

7.7 Actions

```
void ActionMove(int direction);
```

Permet le déplacement du joueur dans une direction locale passée en paramètre (**RIGHT**, **TOP**, **LEFT** ou **BOTTOM**).

ATTENTION : si vous demandez le déplacement vers une direction hors de la zone propriétaire du joueur, l'action ne sera pas effectuée mais elle sera perdue.

```
void ActionReverse();
```

Inverse la direction du tapis roulant situé à la position du joueur.

```
void ActionPass();
```

Indique le passage du tour du joueur sans aucune action.

7.8 Enregistrement des variables

```
void SaveInt(string name, int value);
```

Enregistre pour des utilisation ultérieures la variable de type int sous un nom indiqué par **name**.

ATTENTION : **name** doit pas contenir plus de 8 caractères.

```
void SaveTab(string name, int *tab, int length);
```

Enregistre pour des utilisation ultérieures un tableau d'entier de taille indiqué par length, sous un nom indiqué par **name**.

ATTENTION : **name** doit pas contenir plus de 8 caractères.

```
int* RecoverTab(string name);
```

Renvoie le tableau sauvegardé de nom **name**.

```
int RecoverInt(string name);
```

Renvoie l'entier sauvegardé sous le nom **name**.

7.9 Utilisation des coordonnées globales

Ces fonctions, homologues aux précédentes, utilisent les coordonnées globales plutôt que les coordonnées locales. Dans la plupart des cas, il est fortement recommander de préférer les coordonnées locales aux coordonnées globales. Elle sont beaucoup plus faciles à gérer puisque vous n'avez pas à vous soucier de votre position sur la surface de jeu.

Les seules cas où les coordonnées globales doivent être utilisées est lorsque vous souhaitez surveiller des paquets qui ne se trouve pas dans votre surface de jeu locale, par exemple pour anticiper leur arrivée dans votre surface de jeu locale.

```
int Global_NumPacket();
```

Renvoie le nombre de paquets présents sur la surface de jeu.

```
packet Global_GetPacket(int num);
```

Renvoie le paquet de numéro indiqué en paramètre.

ATTENTION : voir les avertissements de la fonction GetPacket().

```
vector Global_PlayerPos(int player=SELF);
```

Renvoie les coordonnées globale du joueur dont la couleur est passée en argument (**RED**, **GREEN**, **BLUE** ou **YELLOW**). Si la valeur par défaut (*SELF*) est utilisée, les coordonnées globale du joueur lui même seront renvoyées à la place.

```
int GlobalRollingFloor(vector pos, bool active=true);
```

Renvoie la direction globale du tapis roulant de coordonnée globale indiquée en paramètre. Si la variable active est positionnée à true, c'est la direction active du tapis roulant qui sera renvoyée. Sinon, c'est la position inactive qui sera renvoyée. Renvoie **NOTHING** si la zone indiquée n'est pas un tapis roulant, ou si la zone inactive d'un tapis roulant unidirectionnel est demandée.

```
void ActionGlobalMove(int direction);
```

Permet le déplacement du joueur dans une direction globale passée en paramètre (**RIGHT**, **TOP**, **LEFT** ou **BOTTOM**).

ATTENTION : si vous demandez le déplacement vers une direction hors de la zone propriétaire du joueur, l'action ne sera pas effectuée mais elle sera perdue.

```
vector Glob2Loc(vector glob);
```

Renvoie les coordonnées locales de coordonnées globales passées en paramètre. Si les coordonnées globale sont situées en dehors de la portée des coordonnées locale, les coordonnées (-1,-1) seront renvoyée à la place.

```
vector Loc2Glob(vector loc);
```

Renvoie les coordonnées globales d'une coordonnées locales passées en paramètre.

```
int Glob2Loc(int direction);
```

Renvoie la direction locale relative à la direction globale passée en paramètre.

```
int Loc2Glob(int direction);
```

Renvoie la direction globale relative à la direction locale passée en paramètre.

```
int Owner(vector pos);
```

Renvoie la couleur du joueur propriétaire de la zone de coordonnées globales indiquées en paramètre. Renvoie **UNCOLOR** si la position est en zone neutre.

```
int Color();
```

Renvoie votre couleur.

A Exemple d'IA

A.1 L'idée

Cette IA repose sur un principe très simple. On part du principe que le seul moyen de perdre un paquet est de voir un tapis roulant orienté en direction du vide. Cette IA primitive promène donc le faisceau aléatoirement, et inverse la direction d'un tapis roulant lorsque celle-ci met en danger un paquet.

```
#include "ia_user.h"

extern "C" void IaMain()
{
    // La position du joueur est récupérée pour être analysée ultérieurement.
    vector pos = PlayerPos();

    // Si le faisceau ne se trouve pas contre un bord...
    if(pos.x != 0 && pos.y != 0)
    {
        // ... on l'y déplace le plus vite possible.
        ActionMove(LEFT);
    }
    else
    {
        // Sinon on récupère la variable sauvegardée précédemment indiquant la
        // direction actuelle.
        int direction = RecoverInt("direction");

        // Si le faisceau pointe vers un tapis dirigé vers le vide, on inverse
        // ce tapis.
        if(pos.x == 0 && RollingFloor(pos, true) == LEFT ||
           pos.y == 0 && RollingFloor(pos, true) == TOP)
        {
            ActionReverse();
        }
        // sinon on cherche à déplacer le faisceau.
        else
        {
            // si cette direction n'existe pas, on va vers le bas.
            if(direction == -1)
            {
                direction = BOTTOM;
            }
            // sinon, on vérifie la position
            // si le faisceau est tout en bas, on le déplace vers le haut
            else if(pos.y == LOCAL_SIZE-1)
            {
                direction = TOP;
            }
            // si le faisceau est dans le coin haut/gauche, la direction suivante
            // dépendra de la direction actuelle
        }
    }
}
```

```

else if(pos.x == 0 && pos.y == 0)
{
    if(direction == TOP)
    {
        direction = RIGHT;
    }
    else
    {
        direction = BOTTOM;
    }
}
// si on est complètement à droite, on repart vers la gauche
else if(pos.x == LOCAL_SIZE-1)
{
    direction = LEFT;
}

// On se déplace vers la direction déterminée.
ActionMove(direction);

// Puis on sauvegarde la direction courante pour la réutiliser
// le tour suivant.
SaveInt("direction", direction);
}
}
}

```

De nombreuses améliorations peuvent bien entendu être apportée à cette IA. Au lieu de faire des déplacement aléatoire, l'IA pourrait par exemple déterminer le tapis roulant de direction compromettente le plus proche du faisceau, pour ainsi dépenser moins d'actions à se déplacer.

A.2 Quelques idées de stratégies

A.2.1 Tout sauver !

Cette stratégie consiste à empêcher les paquets d'arriver dans des situations critiques qui risqueraient de les faire tomber. Il s'agit donc d'inverser prioritairement tout les tapis roulants qui sont dirigés vers le vide, ou qui menacent un paquet de s'approcher du vide. C'est la stratégie utilisée par l'exemple présenté ci-dessous.

A.2.2 Visez les gros !

Les paquets apporter a bon port par vos adversaire valent généralement beaucoup plus de points que les autres. Rapporter prioritairement de tels paquets vous permettent de faire des points plus efficacement que par des paquets standards. Une stratégie pourrait consister à anticiper le trajet de tels paquets pour leur assurer une arrivée à bon port.

A.2.3 Saisir sa chance

Les paquets peuvent entrer dans votre surface de jeu locale par quatre tapis roulants différents. On constate qu'un paquet qui rentre dans votre surface de jeu locale très près de l'usine se trouve beaucoup plus proche de sa destination. Cette stratégie consiste à ne s'occuper que des paquets qui entre tôt en jeu, et de leurs assurer un chemin tout tracé à destination.

A.2.4 Perfectionnisme

Ce qui pourrait être sans doute la plus puissante des stratégies serait d'anticiper les trajectoires de tout les paquets présents, ainsi que les paquets futurs qui vous arrivent de chez votre adversaire. Vous n'avez plus qu'à vous servir de ces aptitudes visionnaire pour déterminer les paquets n'ayant pas besoin de votre aide pour arriver a bon port, des paquets destinés a être perdus quoi qu'il arrive, ou des paquets ayant besoin d'un petit (ou gros) coup de pouce de votre part.

A.3 Note sur les stratégies

La clé de la victoire ne se trouvent certainement pas dans l'implémentation directe des stratégies présentées ici. À vous de les adapter, de les combiner, de trouver des solutions à leurs défauts, et bien sur de trouver vos propres stratégies pour mettre en place l'IA la plus fiable qui vous assurera la victoire.

Cependant, n'ayez pas trop d'ambition : il vaut mieux une petite IA terminée et débuggée qu'une usine a gaz prévoyant toute les situations sur 60 coups à l'avance mais que vous ne pourrez jamais finir dans les temps.

Pensez également que les ressources restent limitées : vous avez en principe suffisamment de ressources système pour faire a peu près ce que vous voulez, mais si vous en abusez, vous risquez de perdre des tours bêtement.

De même, évitez les boucles infinies et les erreurs de segmentations : elles vous feront perdre votre tour bêtement.

Bonne chance et que le meilleur gagne !