

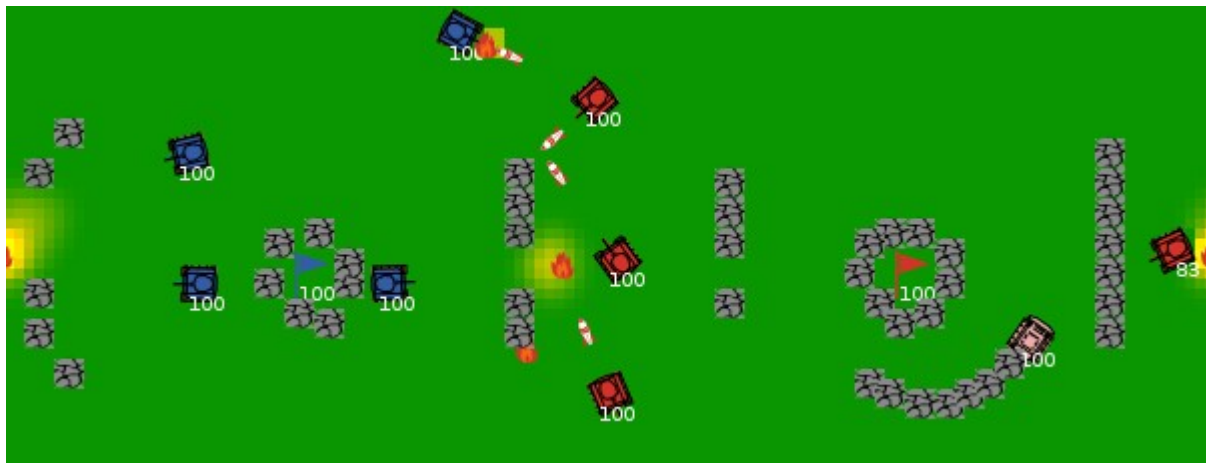
# WinGineer3

## Table of Contents

Préambule.....	1
Installation du langage Netlogo et du jeu.....	2
Introduction au langage Netlogo.....	4
Piloter les unités.....	5
Piloter les tanks.....	5
Bibliothèque de codes.....	7
Attaque.....	7
Défense.....	8
Programmation.....	9
Bouclier.....	13
Soyons plus « constructif »...(piloter les bulldozers).....	13
« Inspection des troupes ».....	16
Bibliothèque de fonctions utiles.....	16
Conclusion.....	19

## Préambule

Ceci est une première version, à la fois du jeu et de ce document destiné à la prise en main du langage de programmation. Vous trouverez une courte vidéo d'explication de l'interface à cette adresse : <http://www.youtube.com/watch?v=03FFtxrAdJs>



L'objectif du jeu est simple : détruire le drapeau de l'équipe adverse, et protéger son drapeau.

Pour cela vous disposez de 5 unités mobiles que vous pouvez choisir parmi deux types différents :

- les tanks : ils peuvent tirer mais se déplacent lentement.
- Les bulldozers : ils ne peuvent pas tirer mais peuvent construire des murs de protection (appelés « bunker »), se déplacent plus vite que les tanks, et ne sont pas bloqué par les bunkers.

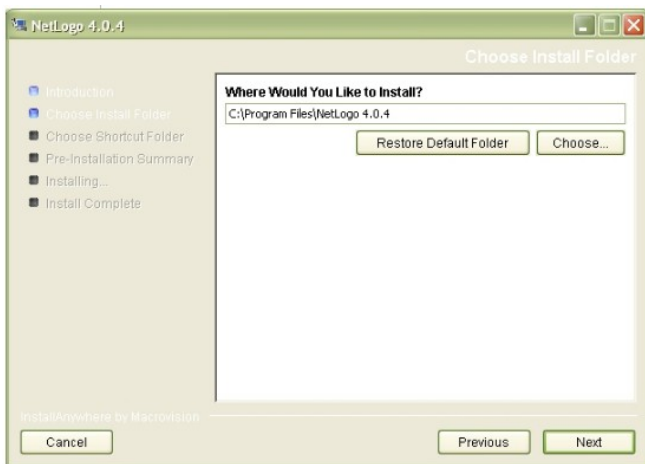
Vous pouvez aussi disposer d'une petite quantité de murs que vous pouvez disposer comme

protection initiale. Le pilotage des unités mobiles se fait par l'intermédiaire de 5 fonctions, c'est à dire 5 blocs de textes écrits dans le langage Netlogo. De nombreuses fonctions et codes d'exemples vous son fournis pour vous simplifier la tâche. Il n'est absolument pas nécessaire de savoir programmer pour jouer à ce jeu. Toutes les bases dont vous aurez besoin sont détaillées dans ce document.

## Installation du langage Netlogo et du jeu

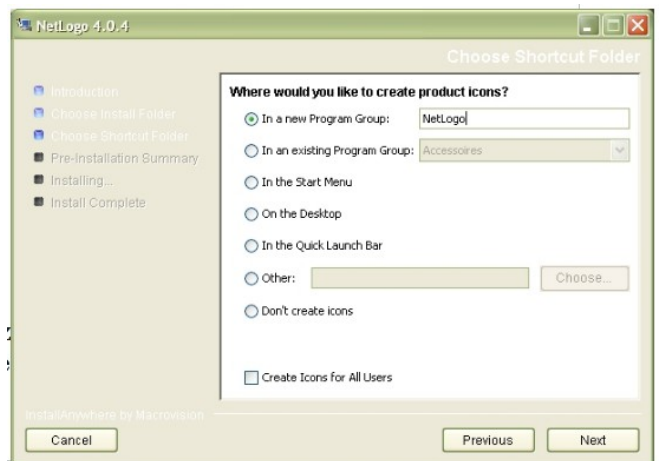
. Vous trouverez dans ce documents une suite d'exemples de code qui vous permettront de comprendre la logique très simple de ce langage. Nous avons crée une « *arène d'entrainement* », pour pouvoir l'utiliser, il vous faut d'abord installer le langage Netlogo sur votre ordinateur...

**Téléchargez l'archive du langage Netlogo (la plateforme de développement) : [Netlogo 4.1](#)**



Lancez l'installation et suivez les instructions...  
(cliquez sur « next »)

puis ....



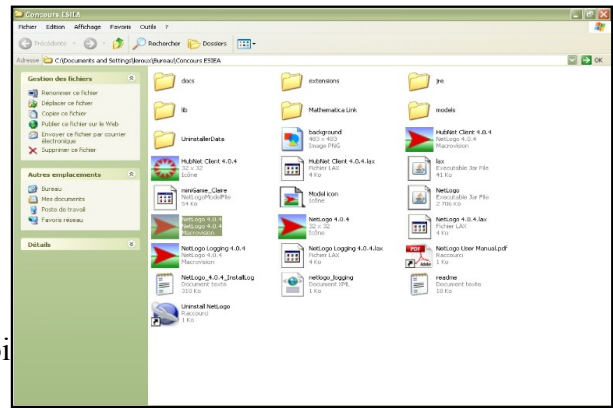
Netlogo est maintenant installé...Pour lancer l'application Netlogo, vous devez aller rechercher le fichier « .exe » dans le répertoire où vous avez installé le logiciel, et simplement double cliquer dessus.

Vous avez installé la plateforme de développement Netlogo, il vous faut maintenant charger le jeu en lui-même...

Téléchargez le fichier du jeu ici : **TODO**

Vous devez maintenant "ouvrir" le fichier du jeu dans la plateforme Netlogo : comme dans tout autre logiciel, il y a une barre de menus. Cliquez sur « file », ensuite « open », ce qui ouvrira un sélecteur de fichiers. Choisissez le fichier du jeu que vous avez téléchargé précédemment (fichier dont l'extension est ".nlogo").

Vous devez alors voir apparaître le jeu, voici à quoi servent les différents boutons de l'interface.



Ça y est vous êtes prêt à programmer en Netlogo...

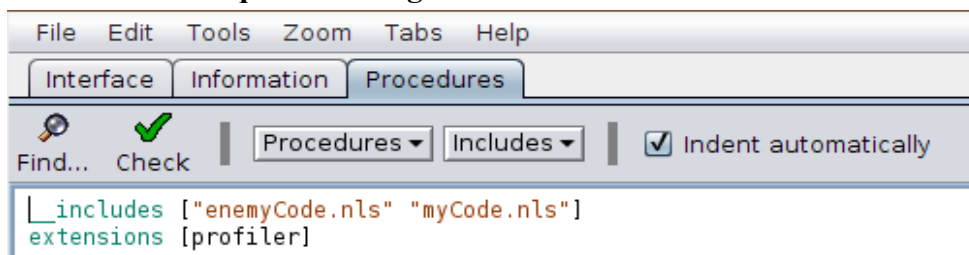
Cette arène utilise les mêmes règles de fonctionnement que l'arène utilisée pour le concours, mais est faite uniquement pour pouvoir tester des robots...

Comme sur le site web du concours, cette arène met en compétition 2 équipes : « A » et « B ». L'équipe « B » possède déjà un comportement basique, cela vous permettra de vous entraîner.

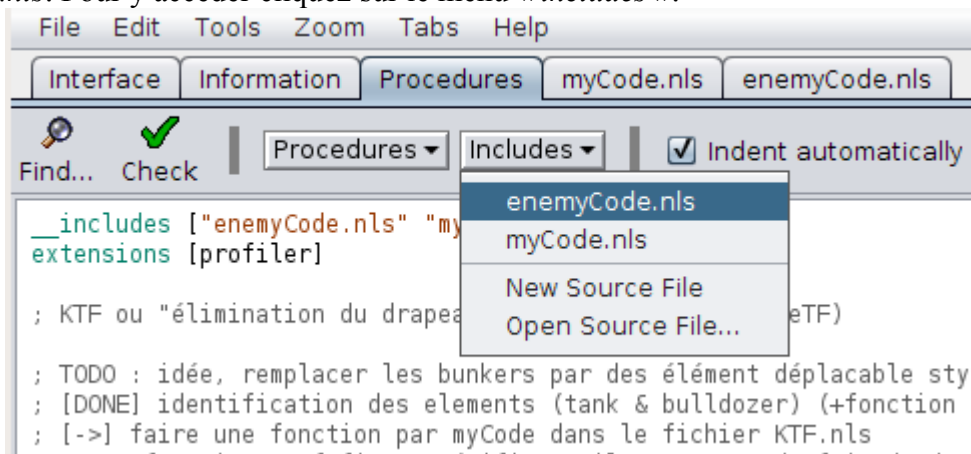
Vous codez directement des fonctions de la simulation (le comportement de vos robots). Cela vous permettra de profiter de tous les outils de développement du langage Netlogo.

Les exemples donnés dans ce document doivent être insérés dans n'importe laquelle des 5 fonctions de chaque candidat. MyCode[1...5] (pour l'équipe « A ») et enemyCode[1...5] (pour l'équipe « B »)

**Pour accéder au code cliquez sur l'onglet « Procédure ».**



Le texte que vous voyez alors est le programme qui gère l'ensemble du jeu. Il ne nous est pas utile. Le code concernant le comportement des robots est écrit dans 2 fichiers séparés : *myCode.nls*, *enemyCode.nls*. Pour y accéder cliquez sur le menu « includes ».



Ces 2 fichiers sont alors ouverts dans deux onglets supplémentaires...

**Remarque importante** : il n'est pas nécessaire de regarder l'ensemble du code du jeu, vous n'avez besoin de programmer que le comportement de vos robots, tout le reste est déjà fait.

- Un fois votre code écrit (vous trouverez des exemple dans les pages suivantes), cliquez sur le bouton *check*, la syntaxe de votre programme sera alors vérifiée, en cas d'erreur un message tentera de vous donner des indices pour le corriger.
- Si le bouton *check* devient grisé, c'est que la syntaxe de votre fonction est correcte. Vous pouvez alors démarrer le programme, en allant sur l'onglet interface. Il faut alors cliquez sur le bouton *setup*, puis *go* pour voir le jeu s'animer. Vous pouvez re-cliquer sur ce bouton pour arrêter son déroulement.
- Le bouton *oneStep* déroule le jeu « pas à pas », cela vous laisse bien le temps de voir ce qu'il se passe. Attention, le bouton *go* ne doit pas être enfoncé pour pouvoir utiliser cette fonctionnalité.
- Lors de vos test vous pourrez vouloir recommencer une partie sans attendre la fin de celle qui est en cours. Pour cela, arrêtez la simulation en re-cliquant sur le bouton *go* (cela arrête la partie en cours), puis cliquez sur le bouton *setup* et enfin *go* pour redémarrer. Dans la plupart des cas un simple clic sur le bouton *setup* suffira (même en cours d'exécution).

## Introduction au langage Netlogo

Netlogo est un langage « à agents » cela veut dire qu'il est prévu pour gérer les actions et interactions entre des éléments qu'on appelle des agents. Dans notre jeu, tout ce que vous voyez est un agent : les tanks, les bulldozers, mais aussi les bunkers, etc... même les éléments qui constituent le sol sont des agents (qui ont la particularité de ne pas pouvoir bouger).

On peut donner des ordres aux agents :

- tourner : *right*, *left*, suivi d'un angle
- avancer : *move*
- s'orienter : *face* (faire face à un autre agent)
- tirer n rockets : *fire n* (pour les tanks)
- construire : *build* (pour les bulldozers)

et obtenir des informations sur les agents :

- leur position
- leur orientation
- l'état de leur bouclier
- etc...

Dans le jeu vous êtes autorisé à récupérer toute information de la simulation, à agir sur la simulation mais uniquement avec les instructions : *move*, *fire*, *build*, *right*, *left*, *face*. Lors de la soumission de votre code un programme vérifiera que vous n'essayez pas d'utiliser d'autres instructions permettant une action directe sur les éléments de la simulation. Ceci serait considéré comme une triche, le code correspondant ne sera donc pas accepté et votre score mis à 0 , jusqu'à soumission d'un nouveau code...

Netlogo dispose de toutes les bases d'un langage de programmation classique :

- les variables
- les structures de contrôle
- les opérations logiques
- les opérations mathématiques de base : +, -, /, \*

Si ces termes ne vous parlent pas, ce n'est pas grave, vous allez très vite comprendre ce que l'on peut faire. Voyons cela sur des exemples...

# Piloter les unités

## Piloter les tanks

Nous allons commencer par un programme extrêmement simple : avancer.

```
; ceci est un programme super simple  
move
```

Voilà comment vous devez modifier le fichier « myCode.nls » pour que votre programme soit intégré au reste du jeu.

Choisissez n'importe laquelle des 5 fonctions. Dans cet exemple nous avons choisit d'associer ce premier exemple de code sur l'unité No 1. C'est donc le tank 1 qui va avancer...

Notez l'utilisation du caractère « ; » permet d'écrire des commentaires : des lignes qui ne seront pas interprétées par Netlogo, mais qui permettent de faire des remarques pour vous faciliter la compréhension du code.

Allez dans l'onglet « interface » et cliquez sur le bouton « go », pour voir votre programme s'exécuter...

Résultat : on voit que l'unité No 1 avance, elle reste vite bloquée par des obstacles, c'est normal, notre programme est vraiment trop simple, nous allons le faire évoluer...

Cliquez à nouveau sur « go » pour arrêter le programme, puis sur l'onglet procédure pour retourner au code pour aller le modifier.

```
move ; avancer  
right 2 ; tourner à droite de 2°
```

Ce nouveau code demande au robot de tourner légèrement sur sa droite après avoir avancé. Résultat le robot tourne en rond. Changez l'argument (=l'angle) de l'instruction *right* pour voir ce que cela donne...

Vous constatez que quand on regarde la trajectoire de l'unité, il semble que l'angle de rotation soit bien plus que de 2 degrés. En fait il faut voir le code des unités est exécuté environ 25 fois par seconde. Ce qui veut dire que 25 fois par secondes, le tank avancera et tournera d'un angle de 2 degrés, ainsi dans une seconde il aura tourné de 50 degrés.

Note1 : Il est inutile de mettre plusieurs fois la commande *move*, seul la première exécution aura un effet, toutes les autres seront ignorées. Donc mettre plusieurs fois de suite l'instruction *move* ne vous fera pas avancer plus vite.

Note 2 : Il en va de même pour l'instruction *fire* (qui permet de tirer), si il y a deux appels successifs dans le code seul le premier sera pris en compte. De plus il y a un temps minimum entre deux tir (un



peu moins d'une demi seconde), il faut bien le temps de recharger... Il y a aussi un paramètre de surchauffe qui vous empêchera de tirer si vous tirer avec une trop grande cadence trop longtemps. Vous verrez cela en détail plus loin dans ce tutoriel...

Note : si vous désirez que deux unités (ou plus) aient le même comportement, il suffit d'écrire le programme une seule fois pour une unité. Pour toutes les autres unités devant réaliser la même tâche, il suffit d'utiliser la commande *same-as* suivit du numéro de l'unité qui contient le code à suivre.

Dans l'exemple juste à coté, les unités 3, 4, 5 font la même chose que l'unité 1.

**ATTENTION** : on pourrait être tenté de simplement appeler la fonction *myCode* suivit d'un numéro, cela marche en local sur votre ordinateur... Cependant sur le serveur cela ne marchera pas. **Il faut impérativement utiliser la fonction *same-as*.**

Maintenant que vous avez compris les bases de « commandement » des unités, nous pouvons aller un peu plus loin dans la programmation...

Vous avez maintenant le choix pour aborder ce concours :

- **Soit vous n'avez jamais programmé**, dans ce cas continuez la lecture de document nous allons vous guider pas à pas vers la programmation en commençant par la « bibliothèque de code » où vous pourrez piocher des comportements d'unité « standard ». Vous n'aurez donc qu'à composer votre équipe en piochant dans cette bibliothèque de code, et bien choisir les positions initiales ainsi que les bunkers de base. Une fois que vous serez familiarisé avec ces quelques codes, et le comportement en résultant, vous identifierez vous même les modification à faire pour les adapter à votre stratégie. Vous pourrez alors lire la partie « programmation » de ce tutoriel, qui vous aidera à créer vous même vos codes ou simplement modifier ceux de la « bibliothèque de code ».
- **Soit vous avez déjà des notions de programmation** (quelque soit le langage, java, javascript, php, C, etc...). Dans ce cas vous pouvez directement passer à la section « Programmation » et les suivantes. Attention cependant, l'avantage que vous pensez avoir sur ceux qui n'ont jamais programmé est assez faible. La difficulté de ce concours est surtout d'ordre stratégique. Le langage Netlogo étant à la fois simple et puissant, les débutants arrivent très rapidement au niveau des candidats confirmés.

**Pensez aussi que vous n'est pas seul!** Lors de votre inscription un accès au forum vous a été ouvert, vous pouvez donc postez vos questions. Pour cela une équipe de 10 étudiants de l'ESIEA sont là pour vous aider. De plus les candidats sont encouragés à s'entraider. Le concours est fait de manière à que vous ne risquez rien à aider un concurrent. Il suffit d'avoir réussi à bien se classer une seule fois pour obtenir un ticket pour la finale. Ce ticket vous est acquis quelque soit la suite de l'évolution de votre classement. Vous serez tenu au courant de l'obtention de ce ticket au moment de votre login sur le site web <http://www.wingineer.fr>



```
File Edit Tools Zoom Tabs Help
Interface Information Procedures myCode.n
Find... Check Close Procedures
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; code candidat : équipe A (ble
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to myCode1
  move
  right 2
end ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to myCode2
  move
  fire
end ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to myCode3
  same-as 1
end ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to myCode4
  same-as 1
end ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

to myCode5
  same-as 1
end ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

## Bibliothèque de codes

Nous proposons ici des codes simples (défensif, agressif, neutre etc....) permettant aux candidats 100% débutants de ne commencer que par des problématique stratégiques, c'est à dire : le choix des unités (tank ou bulldozer), leur comportement, leur position initiales, ainsi que les positions des premiers bunkers.

En piochant des codes d'exemple ici vous pouvez déjà constituer un ensemble d'unités pour vous lancer dans la compétition.

## Attaque

Description du comportement	code
<p><b>« tank risque tout » :</b> Faire face au drapeau ennemi, tirer à la fréquence basse). Si on peut avancer sans risque alors avancer, sinon faire face à l'endroit le moins risqué et avancer. Note : la notion de risque est tout simplement le fait qu'il y ai une explosions ou non devant l'unité en question.</p>	<pre>face enemy-flag fire-one ifelse safe-move? [ move] [ face safer-place move]</pre> <p>Voir une vidéo d'exemple utilisant ce code : <a href="http://www.youtube.com/watch?v=mubxHEWhY6E">http://www.youtube.com/watch?v=mubxHEWhY6E</a></p>
<p><b>« tank nettoyeur de bunkers » :</b> Faire face au drapeau ennemi. Identifier tous les bunkers qui sont dans un cone de longueur 20 cases et d'un angle de 30 degrés (c'est à dire globalement devant). Si il y en a (des bunkers), faire face à l'un d'eux, et si il n'y a pas de rocket sur le chemin(c'est à dire si on a pas déjà tiré sur ce bunker), alors tirer. Ensuite, Si on peut avancer sans danger, on avance, sinon on cherche une zone sans danger. Note : ce tank avancera globalement moins vite que le « risque tout » car il fera « le ménage » devant lui...</p>	<pre>face enemy-flag let b bunkers in-cone 20 30 ifelse any? b [ face one-of b fire 1 ] [ifelse safe-move? [ move] [ face safer-place move]]</pre> <p>Voir une vidéo d'exemple utilisant ce code : <a href="http://www.youtube.com/watch?v=CDkJFOT5RIU">http://www.youtube.com/watch?v=CDkJFOT5RIU</a></p>
<p><b>« tank prudent » :</b> Ce tank fuit les ennemis qui sont devant lui (cône de 25 cases de rayon et 45 degrés d'angle), ...non sans leur avoir envoyer une rocket avant de changer de direction. Par défaut il tire devant lui un missile (pour s'ouvrir la voie).</p>	<pre>face enemy-flag let danger enemies in-cone 25 45 if any? danger [ face one-of danger if safe-fire? [fire 2] right 45 ] if safe-fire? [fire 1]</pre>

<p>Note : ce tank est assez lent (car il fait un détour dès qu'il croise un ennemi), et n'ouvre pas vraiment de voies pour les autres, mais, sur la durée, il a plus de chances d'arriver au but...</p>	<p><i>ifelse safe-move? [ move] [ face safer-place move]</i></p> <p>Voir une vidéo d'exemple utilisant ce code :  <a href="http://www.youtube.com/watch?v=EWpPS7QdBgI">http://www.youtube.com/watch?v=EWpPS7QdBgI</a></p>
---	---

## Défense

Description du comportement	code
<p>« <b>bulldozers constructeur de remparts</b> » :</p> <p>Il construit un rempart autour du drapeau, c'est le choix du positionnement initial qui décidera du rayon du rempart.</p>	<p><i>face friend-flag right 90 build move</i></p> <p>Voir une vidéo d'exemple utilisant ce code :  <a href="http://www.youtube.com/watch?v=9kDSYM0TfZY">http://www.youtube.com/watch?v=9kDSYM0TfZY</a></p>
<p>« <b>tank de défense fixe</b> » :</p> <p>Ce tank ne bouge pas, il faut le positionner assez près du drapeau, dès qu'il voit un danger devant lui et sur les cotés (200 degrés), que ce soit un ennemi ou une rocket (ennemies), il tire dessus. Le fait qu'il ne bouge pas et ne s'occupe que de ce qui est devant lui et sur les coté lui évite de tirer sur son propre drapeau... Son coté statique le rend par contre assez vulnérable...</p>	<p><i>face friend-flag right 180 let d enemies in-cone 20 200 let r rockets in-cone 20 200 with [enemy?] if safe-fire? [ if any? r [face one-of r fire-all] if any? d [face one-of d fire-all] ]</i></p> <p>Voir une vidéo d'exemple utilisant ce code :  <a href="http://www.youtube.com/watch?v=H0FYhCmR5XQ">http://www.youtube.com/watch?v=H0FYhCmR5XQ</a></p>
<p>« <b>tank de défense mobile</b> » :</p> <p>Ce tank surveille autour de lui dans un rayon de 40 cases, identifie le ennemi le plus proche, et se rapproche de lui jusqu'à une distance de 15 cases. Il s'arrête alors et tire sur l'adversaire.</p> <p>Attention le coté à la fois mobile et acharné de ce tank le pousse parfois à se s'éloigner un peu trop du drapeau qu'il est censé défendre...</p>	<p><i>let danger enemies in-radius 40 if any? danger [ let proche min-one-of danger [distance myself] face proche let d distance proche ifelse d &gt; 15 [move] [ if safe-fire? [fire 1]] ]</i></p> <p>Voir une vidéo d'exemple utilisant ce code :  <a href="http://www.youtube.com/watch?v=A8vmYhH5NGI">http://www.youtube.com/watch?v=A8vmYhH5NGI</a></p>

# Programmation

Dans cette section nous allons voir pas à pas tout ce que vous avez besoin de comprendre pour écrire vous même vos stratégies.

Pour s'orienter dans l'arène il existe un certain nombre de points stratégiques que l'on peut nommer.

*friend-flag* : le drapeau de son équipe  
*enemy-flag* : le drapeau de l'équipe adverse  
*friend* suivit du numéro d'identification (1-> 5) : une unité de son équipe  
*enemy* suivit du numéro d'identification (1-> 5) : une unité adverse.  
etc...

Ainsi pour se déplacer vers le drapeau adverse cela s'écrit :

```
face enemy-flag  
move
```

Pour suivre l'unité No 2 de son équipe, cela s'écrit :

```
face friend 2  
move
```

On se rend rapidement compte que les bunkers gênent la progression, il n'y a que deux façons de faire pour résoudre le problème qu'ils posent :

- Solution 1 : contourner le bunker

```
face enemy-flag  
ifelse move? [ move ] [left 90 move]
```

Cela peut se traduire par : « faire face au drapeau adverse. si je peux avancer j'avance, sinon je tourne à gauche de 90 degrés et j'avance ».

Note 1 : l'instruction *ifelse* permet de choisir quel action à mener en fonction de tests. Le test, dans cet exemple, est de savoir si on peut avancer. Si le test est vrai alors le premier bloc de code est exécuté, si il est faux c'est le deuxième qui le sera.

Note 2 : On peut regrouper des instructions dans un « bloc » grâce aux crochets. Dans le cas où on ne peut pas avancer, le programme effectue deux opérations : « tourner à gauche » et « avancer ». Vous pouvez enchaîner autant d'instructions que vous le désirez.

Remarque : rien ne nous assure que l'instruction « move » (après avoir tourné de 90 degrés) sera possible, cela veut dire que cette technique d'évitement est perfectible, mais dans des cas simples, c'est fonctionnel.

Pour plus de clarté , il est conseillé de mettre en forme ce code comme ceci :

```
face enemy-flag  
ifelse move? [ move ]  
[ left 90  
move]
```

On identifie alors mieux le fait qu'il y a plusieurs instructions dans le deuxième bloc de code.

- Solution 2 : exploser le bunker qui nous bloque...

```
face enemy-flag
```

*ifelse move? [ move ] [ fire 1]*

C'est maladroit car les explosions détruisent autant notre tank que le bunker... Pour limiter ce genre de problématiques il y a une fonction qui permet de savoir si le tir est sûr ou pas...

*face enemy-flag  
ifelse move? [ move ]  
[ if safe-fire? [fire 1] ]*

Note 1 : l'instruction *if* permet de conditionner un seul bloc d'action à un test.

Remarque : on reste bloqué, car en effet le tir dans cette situation est dangereux donc il n'est pas effectué... Il faudrait être capable de voir plus tôt que l'on va être bloqué. Pour cela des instructions très pratique permettent de connaître ce qu'il y a dans l'environnement d'une unité. Par exemple si on veut savoir si il y a un bunker devant notre unité a une distance donnée, cela s'écrit :

*face enemy-flag  
ifelse any? bunkers in-cone 15 30 [ fire 1] [move]*



Ce qu'on peut traduire par : « si il y a un bunker dans un cône de rayon de 15 et d'angle 30 degrés, alors tirer, sinon avancer. » Le cône est visualisé en rouge sur cette illustration.

Note 1 : la suite instruction « *bunkers in-cone 15 30* » désigne un ensemble d'agents, c'est à dire tous les élément qui se trouvent à cet endroit. L'instruction « *any?* » qui la précède, transforme alors tout l'expression en quelque chose qui est soit vrai, soit faux : « y a-t-il quelque chose dans le cône de 15 case et 30 degrés devant moi ? ». On peut alors utiliser l'instruction *if* (ou *ifelse*) qui exige d'être suivit par quelque chose qui est soit vrai soit faux (« *true* » ou « *false* »).

On aurait aussi pu utiliser l'instruction *count*, pour être plus précis... Dans ce cas ont peut décider de contourner si il y a plusieurs bunkers et tirer si il n'y en a qu'un... Cela s'écrit :

*face enemy-flag  
ifelse count bunkers in-cone 15 30 > 1 [ right 90 ] [ fire-one]  
move*

Remarquez que ce qui suit le *ifelse* , c'est à dire : « *count bunkers in-cone 15 30 > 1* » est finalement aussi quelque chose qui est soit vrai soit faux. ( Le décompte de ces éléments est strictement supérieur à un ou non).

Note 2 : pour avoir une notion de distance, sachez qu'il y a 60 unité de distances entre les deux drapeaux.

Remarque : la fréquence de tir est à la fois trop importante et non-constante. Elle est trop importante car on demande de tirer tant qu'il y a un bloc, or une fois le premier tir parti, les suivants ne sont peut être pas nécessaires car les bunkers seront détruits dans quelques secondes. La fréquence n'est pas constante car le tank chauffe à chaque tir, et quand il atteint une température limite il ne peut plus tirer... C'est cette information de température devra être contrôlée si on ne veut pas être pris au dépourvu lors d'une attaque...

*face enemy-flag  
ifelse any? bunkers in-cone 15 30 [ if heat < 10 [fire 3]] [move]*

à chaque tir la chaleur (*heat*) augmente de 10, si elle atteint 50, il n'est plus possible de tirer. Seul le fait d'attendre sans tirer permet de faire baisser cette température, et donc de pouvoir tirer à nouveau. Si on met 50 comme limite de chauffe il essaiera alors de tirer le nombre de rockets demandées, cela donne une grosse rafale mais est elle est courte et après il ne sera pas possible de tirer pendant un bon moment ... Vous pouvez bien sûr utiliser toutes les valeurs entre 10 et 50 pour obtenir la fréquence de tir voulue, et surtout qui vous laissera suffisamment de marge pour pouvoir vous défendre si il y a besoin. Si cela vous paraît un peu compliqué pour le moment, vous pouvez utiliser l'instruction *fire-one*, qui tire avec la fréquence la plus basse.

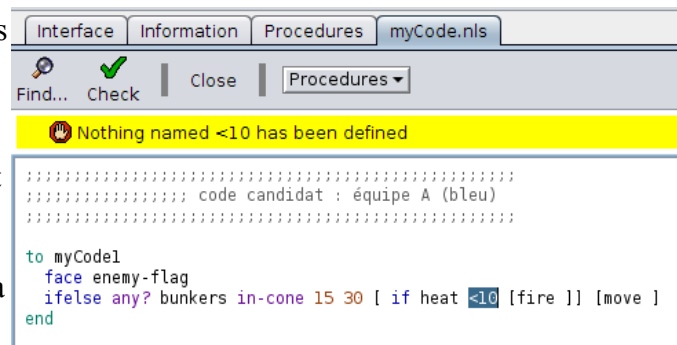
```
face enemy-flag
ifelse any? bunkers in-cone 15 30 [fire-one] [move]
```

Il existe aussi une fonction qui s'appelle *fire-all*, qui au contraire, tire tant qu'elle peut mais essaye de répartir les tir pour couvrir un certain angle.

Notez qu'on peut cumuler plusieurs tests grâce aux « opérateur logique » (*and* et *or* ). On peut vouloir tirer à une cadence donnée, (c'est à dire que si la température est inférieure à un seuil donné) ET que si le tir n'est pas directement dangereux pour son équipe. Vous allez voir que cela se fait très naturellement.

```
if heat < 20 and safe-fire? [ fire 3]
```

**Note importante :** faites attentions aux espaces entre les instructions, il faut absolument que toutes les instructions soient séparées par un espace, sinon l'interpréteur du langage Netlogo a du mal à comprendre le programme. Dans cet exemple, le simple oubli d'un espace entre le symbole « < » et « 10 » rend le programme incompréhensible par Netlogo, et amène alors à un message d'erreur.



Il est temps maintenant à apprendre à se défendre... Voilà une proposition : « si un ennemi est trop proche de moi, lui faire face et tirer... ». Cela peut s'écrire comme cela :

```
if any? enemies in-radius 20 [
  face one-of enemies in-radius 20
  fire 1
]
```

Quelques explications sont nécessaires :

- *enemies* désigne l'ensemble des unités ennemies
- *in-radius* permet de « filtrer » uniquement les unités qui sont positionnées à une distance inférieures à un seuil donnée (ici 20).  
Note : 20 correspond à la portée des rockets.
- *one-of* permet de ne sélectionner qu'une seule unité. En effet quand on demande de faire face à quelque chose, il faut qu'il y ait une seule chose, or « *enemies in-radius 40* » peut tout à fait désigner deux unités différentes qui peuvent être dans des directions opposées...  
Ainsi si vous oubliez le « one-of » Netlogo vous dira qu'il y a une erreur et qu'il ne peut pas

interpréter la commande *face*.

Note : *one-of* choisit aléatoirement une unité dans la liste, si on veut faire une sélection sur un critère donné il faut le spécifier. Voici un exemple de code permettant de sélectionner l'unité ennemie la plus près (donc vraisemblablement la plus menaçante).

```
face min-one-of enemies [ distance myself]
```

*myself* désigne tout simplement l'unité que vous êtes entrain de commander.

On peut vouloir sélectionner l'unité la plus faible : (*shield* est la valeur du bouclier, quand cette valeur est nulle l'unité disparaît.)

```
face min-one-of enemies [ shield ]
```

La présence de crochets, indique ici non pas un bloc de code mais plutôt l'accès à une information provenant d'autre chose que l'unité que vous commandez. La variable *shield* dans ce cas désigne la valeur d'état du bouclier d'une unité ennemie, pas du bouclier de l'unité que vous commandez. Donc si vous voulez sélectionner l'unité ennemie la plus proche de votre drapeau, cela s'écrit :

```
face min-one-of x [ distance enemy-flag ]
```

Vous auriez certainement mis *friend-flag* au lieu d'*enemy-flag*, cela semblerait plus logique... Cependant il faut comprendre que ce qui est entre crochet, dans ce cas, est exécuté selon le point de vue de l'ennemi, ainsi quand on demande à l'ennemi comment il désigne votre drapeau il est naturel qu'il le désigne comme le drapeau adverse...

Pour faciliter la lecture et aussi la vitesse d'exécution de votre programme, il est préférable d'éviter d'écrire plusieurs fois la même chose. Le code précédent peut donc se re-écrire plus élégamment comme ceci :

```
let x enemies in-radius 40  
if any? x [  
  face one-of x  
  fire 1  
]
```

La première ligne a pour effet de déclarer une variable qui stocke la liste de tous les ennemis dans un rayon de 40. La suite du code est la même que dans l'exemple précédent, mais on re-utilise *x* pour nommer cette liste. Il est possible de donner une nouvelle valeur à la variable *x* comme ceci :

```
set x rockets in-radius 20
```

En fait l'instruction *let* permet de déclarer une variable (et de lui donner une valeur), l'instruction *set* permet d'en modifier sa valeur.

On a vu comment sélectionner des éléments du jeu en fonction de leur position et de leur type. Il y a aussi un critère très important c'est les variables internes de ces éléments. Par exemple, il eut être stratégiquement intéressant de choisir de tirer sur une unité ennemie qui a un indice de chauffe élevé. Cela veut dire , à priori, qu'il sera bientôt en surchauffe, et donc dans l'impossibilité de riposter. Voilà comment cela s'écrit :

```
let p enemies with [heat > 40 ]  
if any? p [  
  face one-of p  
  fire-one]
```

Note : Les crochets autour du critère « *heat > 40* » servent à bien signaler qu'on parle de la variable *heat* des ennemis et non de sa propre variable *heat*.

Remarque : il est possible de combiner les deux type de sélection, à la fois la sélection en fonction d'une position et en fonction de variables internes. Par exemple il est inutile de sélectionner les ennemis qui sont au delà de 20 cases de distance car les rockets ne porterons pas au delà. Cela s'écrit alors :

```
let p enemies in-radius 20 with [heat > 40 ]
if any? p [
  face one-of p
  fire-one]
```

## **Bouclier**

Vous l'avez déjà remarquer une seule rocket est rarement suffisante pour détruire un objectif, en effet chaque élément du jeu à un bouclier (variable *shield*), tant que cette valeur est positive ou nulle l'objet résiste.

Il peut donc être stratégie que surveiller son propre bouclier, ainsi que celui de ses ennemis...

Voici un exemple de code qui décide de changer d'objectif si son bouclier est inférieur à un seuil donné...

```
ifelse shield > 50 [face enemy-flag] [ face friend-flag]
move
```

Voici un autre exemple permettant de sélectionner l'ennemi le plus fragile pour lui tirer dessus.

```
let p enemies in-radius 10
if any? p [
  face min-one-of p [shield]
  fire 1
]
```

Ce code nécessite quelques explications :

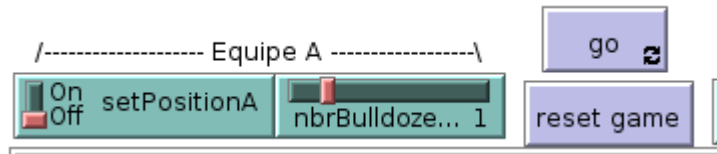
- l'instruction *min-one-of*, comme l'instruction *one-of* qu'on a déjà vu précédemment, permet de sélectionner un seul élément. Au lieux de choisir un élément au hasard elle prend le plus petit selon le critère définit entre crochets. Notez qu'il existe aussi l'instruction *max-one-of*.
- Le bloc *[shield]* n'est pas à comprendre comme un bloc de code conditionné à l'instruction précédente, mais comme l'extraction d'une variable hors de son contexte. Dans cet exemple on va sélectionner dans la liste *p*, l'élément qui a le bouclier le plus faible. La paire de crochet sert à signaler que la variable *shield* dont on parle ici n'est pas le bouclier de votre unité mais de chacune des unités contenu dans la variable *p*.

Notez que tous les éléments du jeu ont un bouclier (unités mobiles, drapeaux, bunkers), et vous pouvez accéder à cette information de la même manière.

## Soyons plus « constructif »...(piloter les bulldozers)

Nous avons vu comment commander les unités de type tank, mais vous avez pu constater qu'il existe un autre type d'unité, les bulldozers ... Dans l'interface vous pouvez choisir de constituer votre équipe de tanks et de bulldozers, vous êtes libre de la constituer comme vous le voulez. Notez seulement que les bulldozers ne peuvent pas tirer, il est donc peu probable d'arriver à gagner avec une équipe constituée exclusivement de bulldozers...

Pour avoir des bulldozer dans son équipe il faut modifier l'élément d'interface qui s'appelle « *nbrBulldozers* » et mettre sur « ON » l'interrupteur « *setPositionA* ».



Cliquez ensuite sur le bouton « *setup* »,

l'interface vous permettra alors de positionner toutes vos unités ainsi que quelques bunkers. Une fois ces positions choisies, remettez l'interrupteur « *setPositionA* » sur OFF, sinon il vous redemandera de positionner toutes vos éléments à chaque redémarrage de parties.

Note : identifiez bien quelles unités sont les bulldozer et quels unités sont les tanks, car l'instruction « *fire* » n'aura aucune action dans le code d'un bulldozer.

Les bulldozers ne peuvent pas tirer mais ils peuvent construire des bunkers pour protéger votre drapeau ou simplement pour ralentir la progression de l'ennemi.

Voici un code plutôt simple permettant de faire construire un rempart autour de votre drapeau :

```
face friend-flag
right 90
build
move
```

Ce qui peut se traduire en : « faire face à notre drapeau, tourner de 90 degrés, construire un bunker, avancer. »

Notez alors qu'une fois que le bulldozer a fini sa boucle il se trouve alors au point de départ et n'ajoute plus de réelle protection (il ne fait que détruire et reconstruire les mêmes bunkers). Ce qu'on voudrait faire c'est se rendre compte qu'on a fini, et s'en souvenir... Car se rendre compte qu'on a fini c'est facile, il suffit de voir qu'il y a un bunker devant le bulldozer. (« Devant » se traduit ici par un cône de rayon 5 et d'angle 90 degrés).

```
ifelseany? bunkers in-cone 5 90 [ ; il y a un bunker en face (donc on a fini le rempart)
```

```
face enemy-flag
move
```

```
]
```

```
[ face friend-flag ; on est dans le bloc « else », il n'y a pas bunker en face
```

```
right 90 ; ... il faut continuer à construire le rempart.
```

```
build
```

```
move
```

```
]
```

On espère alors qu'après avoir fait un tour il s'arrête de construire et se dirige vers le drapeau ennemi. Mais cela ne marche pas car dès que le bulldozer fait face au drapeau ennemi il n'y a plus de bunker en face de lui, il « oublie » alors qu'il a fini un tour... Une solution à ce problème est d'utiliser la mémoire des unités, pour y stocker l'information qu'on a fini un travail, et qu'on peut alors passer à un autre...

Cela se fait grâce à la variable *memory*, cette variable est différente des variables que vous pouvez créer avec l'instruction *let*, car elle garde sa valeur d'une exécution à l'autre de votre programme. Au début elle vaut 0, une fois que vous changez sa valeur avec l'instruction *set*, la valeur est conservée jusqu'à une prochaine mise à jour. Voici un exemple d'utilisation pour stopper la création du rempart une fois celui-ci fini :

```
if memory = 0 [ ; première tâche : construire un rempart
  ifelse any? bunkers in-cone 5 90
    [ set memory 1]
    [ face friend-flag
      right 90
      build
      move ]
]
if memory = 1 [ ; deuxième tâche s'approcher du drapeau ennemi
  face enemy-flag
  move
]
```

Note 1 : L'instruction « *set memory 1* » est exécutée quand l'unité trouve un bunker devant elle, c'est à dire quand elle a fini le rempart.

Note 2 : Vous pouvez donc imaginer plusieurs missions différentes pour une même unité.

Note 3 : On peut mettre n'importe quel type d'information dans la variable *memory*, on peut y mettre par exemple l'identité d'une unité de l'équipe adverse. Imaginons qu'on veuille créer un tank « rancunier »...

```
ifelse memory = 0 [ ; si on a encore identifié personne... on va à l'attaque
  face enemy-flag
  move
  let x enemies in-radius 10 ; on regarde si il y a des ennemis proches
  if any? x [ set memory one-of x] ; si il y a des ennemis proches, on en mémorise un
]
[ face memory ; on fait face à l'ennemi mémorisé, et on l'attaque...
  move
  fire-one
]
```

Note 4 : attention cet exemple de code ne fonctionnera pas avec les bulldozers car ils ne peuvent pas tirer.

## « Inspection des troupes »

Netlogo fournit une interface très pratique pour développer ses programmes, la notion « d'inspection ». À tout moment dans le jeu vous pouvez faire un clic du bouton droit de la souris sur une unité, vous voyez alors apparaître un menu.

(Pour plus de facilité, je vous conseille d'arrêter la simulation en cliquant sur le bouton « go ».)

Dans ce menu, sélectionnez l'unité qui vous intéresse, et choisissez l'option « inspect »....



S'ouvre alors une fenêtre vous donnant accès à 3 choses très pratique pour programmer :

1. un zoom sur l'unité en question. Vous pouvez ajuster ce zoom à votre convenance en vous servant du curseur qui est juste sous l'image.
2. Une vue sur toutes les variables de cette unité (dont son niveau de bouclier ou de ce chauffe, etc...)
3. ...et tout en bas, une zone permettant de tester des bouts de code. Cela permet de tester des lignes de code, étape par étape, et de manière très interactive.



## Bibliothèque de fonctions utiles

- *fire n* : tir n rockets (si il n'y a pas surchauffe). Le nombre *n* donné en paramètre est le nombre de rockets que le tank essayera de tirer. Attention, le décompte de rockets est tout simplement le nombre de rockets « en l'air ». Cela veut dire que si l'on tire très près de l'objectif, il en tirera plus puisque les rockets exploseront très tôt...
- *fire-one* : Cette fonction permet de tirer au débit le plus bas et donc de limiter la chauffe.
- *fire-all* : De même que la fonction *fire* cette fonction tir jusqu'à atteindre la surchauffe, mais en plus elle « ventile » les tirs sur des angles différents pour éviter que la première explosion ne déclenche toute les autres, ce qui finirait par infliger des dégâts au tireur...
- *enemy n* : identifie l'unité ennemie numéro *n*  
Exemple, pour suivre l'unité ennemie 3, il suffit d'écrire :  
*face enemy 3*  
*move*
- *enemies* : désigne tous les ennemis (tanks et bulldozers)

Exemple : voici un code qui fait choisir un enemy au hasard et leur tire dessus  
*face one-of enemies*  
*fire 1*

- *safe-fire?* Répond par *true* ou *false* si le tir qu'on s'apprête à faire risque de nuire à l'équipe (par exemple si il y a une unité amie trop proche)

Exemple, pour limiter les tirs dangereux :

```
if safe-fire? [fire 1]
```

Attention, cette fonction n'est pas parfaite il se peut que dans certains cas quelques unités amies soient touchées quand même. De même il y aura peut-être des cas où il sera stratégiquement plus important de tirer quitte à sacrifier une unité...

- *alive? x* Cette fonction répond à la question est-ce que l'unité x est en vie ?

Exemple :

```
let x friend 3 ; soit la variable x contenant l'unité Numéro 3
```

```
if alive? X [ ; si elle est en vie, alors la suivre
```

```
face x
```

```
move
```

```
]
```

- *friend-flag, enemy-flag* : désigne le drapeau ami ou ennemi

- *friend n* : identifie l'unité amie numéro n

Exemple, pour suivre l'unité 3 amie , il suffit d'écrire :

```
face friend 3
```

```
move
```

(Évidemment il ne faut pas mettre ce code dans la fonction gérant l'unité No 3...)

- *friend? x* : Cette fonction répond à la question est-ce que l'unité x est une unité amie ?

Exemple : s'éloigner des autres tanks amis (pour ne pas offrir une cible facile)

```
let x one-of other tanks in-radius 10 ; soit x un des tanks inférieur à une distance de 10
```

```
if friend? x [ ; si c'est un ami
```

```
face x ; lui faire face
```

```
right 180 ; faire demi tour
```

```
move ; bouger
```

```
]
```

- *enemy? x* : Cette fonction répond à la question est-ce que l'unité x est une unité ennemie ?  
Exemple : voir l'exemple précédent...

```
let x one-of other tanks in-radius 10 ; soit x un des tanks inférieur à une distance de 10
```

```
if enemy? x [ ; si c'est un ennemi
```

```
face x ; lui faire face
```

```
fire 1 ; tirer...
```

```
]
```

- *safe-move?* : cette fonction répond à la question : « puis-je avancer sans danger », cette fonction permet d'éviter d'avancer dans le souffle d'une explosion.

Exemple :

```
if safe-move? [move]
```

- *safe-here?* : cette fonction répond à la question « suis-je en danger », cette fonction répond

« *true* », c'est à dire vrai, si l'unité en question est dans le souffle d'une explosion et est entrain de perdre des points de bouclier.

Exemple : Si on est en danger se déplacer, attention rien n'indique ici qu'on se déplace vers un endroit moins dangereux, pour cela regarder la fonction « *safer-place* ».

```
if not safe-here [ move ]
```

Notez ici l'utilisation de l'instruction *not* qui permet d'inverser le sens logique dans un test, ici on ne décide de se déplacer que si le lieu actuel n'est **pas** sûr.

- *safer-place* : cette fonction donne l'orientation présentant le moins de danger.  
Exemple : faire une unité « lâche », qui ne fait que fuir le danger

```
face safer-place  
move
```

Note : quand il n'y a aucun danger, l'orientation donnée est aléatoire, mais ce n'est pas la même orientation qui sera donnée d'un appel à l'autre, donc si il n'y a pas d'explosion proche les déplacements créés par ce code seront aléatoires. Ainsi l'utilisation la plus efficace est de ce type :

```
face enemy-flag  
if not safe-move? [face safer-place]  
move
```

Qui peut se traduire en : « faire face au drapeau ennemi, si on ne peut pas avancer sans danger, alors identifier l'orientation la plus sûre (là où l'explosion est la moins forte) et se diriger vers elle.

- *facing-me* : Cette fonction est à utiliser en conjonction avec l'instruction *with*, elle permet alors de sélectionner, les dangers les plus menaçants... En effet la proximité n'est pas l'unique critère de danger. Par exemple un missile qui passe proche mais n'est pas dirigé vers nous n'est pas réellement dangereux... La fonction « *facing-me* » donne l'écart d'angle avec l'angle « optimal », c'est à dire que si la valeur est 10 ça veut dire que l'élément en question devrait changer de 10 degrés (en plus ou en moins) sa direction pour être pile face à vous. Dit autrement plus cette valeur est forte moins le danger est fort.  
Voici un exemple d'utilisation :

```
let r rockets in-radius 15 with [facing-me < 10]  
if any? r [face one-of r  
left 90  
move  
]
```

Qui peut se traduire en : « sélectionner les rockets dans un rayon de 15 cases (et) qui sont face à moi (à 10 degrés près), Si il y en a, en choisir une, lui faire face, tourner à gauche (pour l'esquiver) puis avancer. »

## Conclusion

Vous avez là déjà quelques bases pour élaborer des stratégies intéressantes, à vous de jouer ;-)

Vous trouverez encore bien d'autres aides et conseils sur le forum du site Wingineer.

Vous pourrez trouver un **dictionnaire des fonctions disponible en Netlogo** en explorant le menu « help » de l'interface ou à cette adresse : <http://ccl.northwestern.edu/netlogo/docs/primindex.html>.

Attention bien sûr à bien respecter **la règle du jeu** :

Vous êtes autorisé à lire toute information de la simulation, vous êtes autorisés à agir sur la simulation mais uniquement avec les instructions : *move, fire, right, left, face, build* appliquées à vos unités. Bien sûr vous avez le droit d'utiliser toutes les instructions de Netlogo vous aidant à produire les décisions de votre robot. Vous ne devez par contre en aucun cas agir directement sur le jeu (en essayant de modifier directement son score, par exemple), cela serait considéré comme de la triche, votre code sera alors supprimé et votre score mis à 0 jusqu'à soumission d'un nouveau code respectant les règles.

### TODO

Comment exporter son code

- sauvegarder !!
- utiliser « export »