

Tutoriel de Création et de Programmation d'un robot

Voilà ! Le jeu est installé depuis un certain temps. Pour vous, il n'y a plus de secret pour naviguer dans l'application, vous savez sélectionner des robots existants et lancer une « battle » et là je vous rappelle que nous sommes dans du virtuel, alors n'allez pas vous battre avec votre voisin ou vos camarades !

Et maintenant vous aimeriez avoir votre « char » Ben-Hur ! Et bien pas de problèmes, c'est ce que nous allons apprendre ensemble dans les chapitres suivants :

- **Partie 1 : Créer un robot dans l'application**
- **Partie 2 : Sauvegarder et compiler votre robot**
- **Partie 3 : Définir le design de votre robot**
- **Partie 4 : Les déplacements du robot**
- **Partie 5 : Les actions des éléments du robot**
- **partie 6 : Les réactions du robot face aux événements**

Vous n'avez plus qu'une chose à faire : lisez les chapitres suivants dans l'ordre pour découvrir le monde fascinant de la stratégie militaire ! Il ne faut peut être pas abuser quand même !

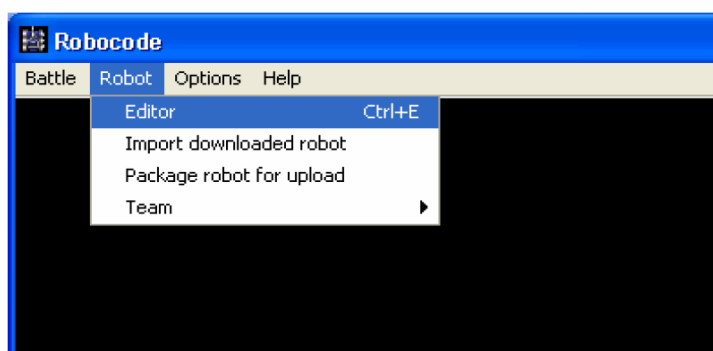
Partie 1 : Créer un robot dans l'application

Créer un robot dans Robocode est assez facile, mais le rendre invincible ne l'est pas.

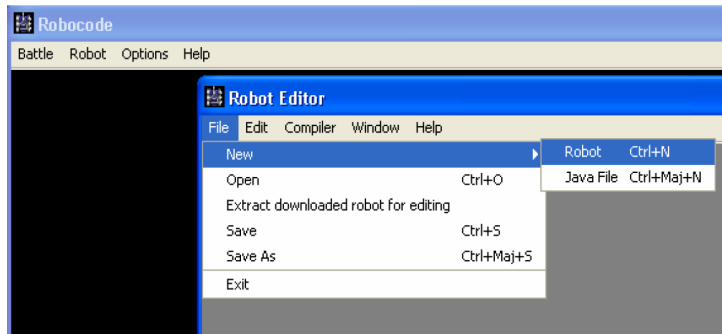
Sa création peut vous prendre quelques minutes, ou alors des mois et des mois si vous développez des stratégies avec de plus en plus d'intelligence artificielle.

La mise au point d'un robot peut devenir une drogue ! Les améliorations apportées à votre robot ne se feront pas sans erreurs et difficultés et il y aura plus d'un tir manqué pendant vos battles. Mais au fur et à mesure de l'expérience acquise, vous apprendrez à votre robot où aller, comment réagir, qui éviter, et dans quelle direction tirer.

Pour créer un robot, lancer l'application Robocode si ce n'est pas déjà fait et sélectionner dans le menu « Robot / Editor »



Ensuite, sélectionner « File / New / Robot » :



Dans les différents champs qui se présenteront successivement à vous, écrivez :
le nom de votre robot comme « I_am_the_best » par exemple mais cela restera à prouver.



Attention à ne pas mettre d'espace dans votre nom mais plutôt des tirets bas (underscore) et la consigne est valable d'une manière générale pour les noms de dossier ou répertoire.

Donner un nom de dossier pour y placer votre robot. Validez.

Vous arrivez maintenant dans la fenêtre « Editing – nom_de_votre_robot ». Il apparaît dans cette fenêtre le code java succinct de votre Robot.

```
1 package mesRobots;
2 import robocode.*;
3 //import java.awt.Color;
4 // API help : http://robocode.sourceforge.net/docs/robocode/robocode/Robot.html
5 /**
6  * Test - a robot by (your name here)
7  */
8 public class Test extends Robot
9 {
10     /**
11     * run: Test's default behavior
12     */
13     public void run() {
14         // Initialization of the robot should be put here
15
16         // After trying out your robot, try uncommenting the import at the top,
17         // and the next line:
18
19         // setColors(Color.red,Color.blue,Color.green); // body,gun,radar
20
21         // Robot main loop
22         while(true) {
23             // Replace the next 4 lines with any behavior you would like
24             ahead(100);
25             turnGunRight(360);
26             back(100);
27             turnGunRight(360);
28         }
29     }
30     /**
31     * onScannedRobot: What to do when you see another robot
32     */
33     public void onScannedRobot(ScannedRobotEvent e) {
34         // Replace the next line with any behavior you would like
35         fire(1);
36     }
37 }
```

Line: 4 Column: 80

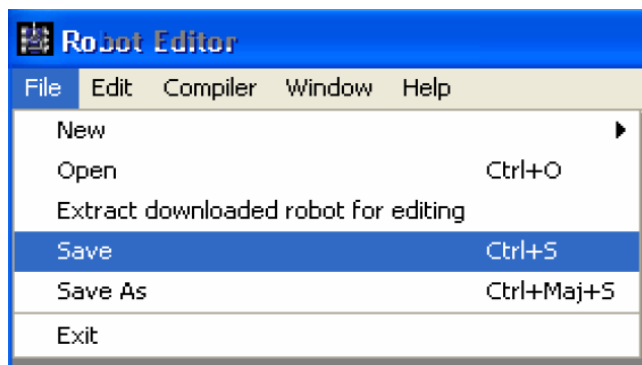
Et là, je vous entends déjà. Au secours ! Qu'est-ce que c'est que ce charabia ? Du calme. Tout va bien se passer. Nous allons décortiquer tranquillement tout cela ensemble.

Allez, on commence fort. Vous allez devenir l'auteur de ce charabia en remplaçant ligne 6 « (your name here) » par votre nom. Non! Pas le mien, j'ai bien dit le votre, je ne veux pas être responsable de l'accident, d'une telle tuerie par la suite !

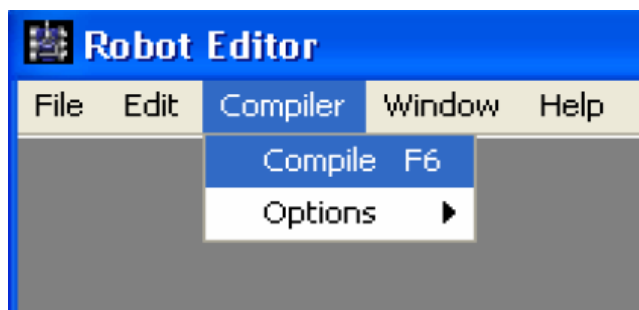
Comme vous êtes certain de vouloir garder votre œuvre avant de l'améliorer, je vous propose rapidement de savoir la sauvegarder et surtout compiler votre code... question ? Lisez la suite

partie 2 : Sauvegarder et compiler votre robot

Avant de tester votre robot ainsi créé, il faut tout d'abord le sauvegarder, et ce n'est pas très compliqué si vous ne le savez pas encore :

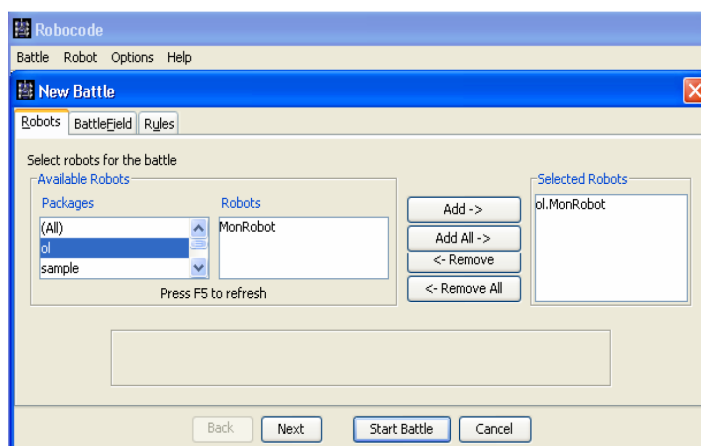


Il reste juste à le compiler, c'est à dire à le transformer en code exécutable pour que l'application puisse l'utiliser :



Pour tester votre robot, il ne reste plus qu'à créer une nouvelle bataille, et cela vous savez le faire depuis un certain temps.

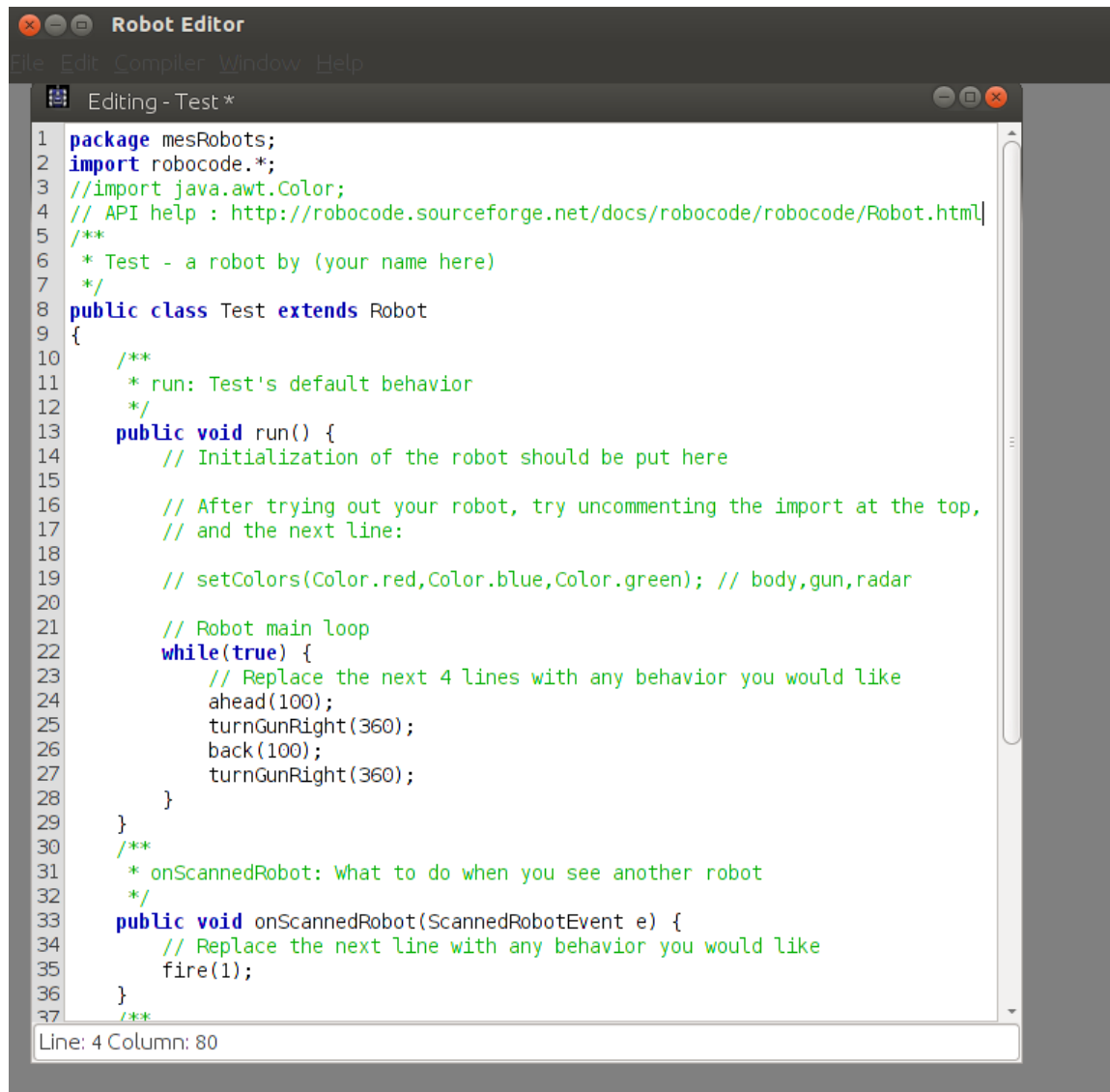
Allez au cas où, et pour le même prix :



Vous ne trouvez pas cela génial, votre robot ne fait pas grand chose... Vous ne vous rappelez plus que vous avez juste programmer le nom de votre robot. Nous allons attaquer les choses sérieuses. Rappelez-vous le charabia dans la fenêtre « Editing » :



Retenez bien cette fenêtre et comment y accéder, nous allons souvent y revenir puisque nous allons y écrire toutes les commandes désirées.



The screenshot shows the 'Robot Editor' application window. The title bar says 'Robot Editor'. Below it is a menu bar with 'File', 'Edit', 'Compiler', 'Window', and 'Help'. The main window has a tab titled 'Editing - Test *'. The code editor displays the following Java code:

```
1 package mesRobots;
2 import robocode.*;
3 //import java.awt.Color;
4 // API help : http://robocode.sourceforge.net/docs/robocode/robocode/Robot.html
5 /**
6  * Test - a robot by (your name here)
7  */
8 public class Test extends Robot
9 {
10     /**
11     * run: Test's default behavior
12     */
13     public void run() {
14         // Initialization of the robot should be put here
15
16         // After trying out your robot, try uncommenting the import at the top,
17         // and the next line:
18
19         // setColors(Color.red,Color.blue,Color.green); // body,gun,radar
20
21         // Robot main loop
22         while(true) {
23             // Replace the next 4 lines with any behavior you would like
24             ahead(100);
25             turnGunRight(360);
26             back(100);
27             turnGunRight(360);
28         }
29     }
30     /**
31     * onScannedRobot: What to do when you see another robot
32     */
33     public void onScannedRobot(ScannedRobotEvent e) {
34         // Replace the next line with any behavior you would like
35         fire(1);
36     }
37     /**
```

At the bottom of the editor, a status bar shows 'Line: 4 Column: 80'.



Pourquoi y a-t'il plusieurs couleurs dans la fenêtre de création de mon robot ?

Si vous ne le savez pas déjà, sachez que tout ce qui est écrit en vert représente du commentaire pour expliquer ce qui est écrit en noir. Et là miracle ! Vous avez le droit de le traduire en français pour améliorer votre anglais. Le programme fonctionnera tout pareil ensuite. Seul ce qui est écrit en noir est important pour se faire comprendre par l'ordinateur.

Au fait ! Ce langage a aussi un nom, c'est le langage Java qui est un langage de programmation parmi tant d'autres.

Si une seule petite erreur se glisse parmi les lignes écrites en noir, votre robot ne pourra pas être créé, l'ordinateur ne pourra pas traduire votre prose. Et une erreur, cela peut être pas grand chose : comme l'oubli d'un point virgule par exemple, ou son inversion avec un point, etc...

Maintenant, nous allons revenir plus en détail sur cette fenêtre « Editing-nom-robot » et étudier les différentes parties du code java que vous allez pouvoir modifier par la suite. N'oubliez pas cette fenêtre, nous allons très souvent y faire référence par la suite, elle contient presque tout pour devenir le champion des battles. Et là, faut-il me croire ? On verra bien plus tard...

Sachez tout d'abord que tout ce qui est écrit en noir donc, de la ligne 13 à l'avant dernière ligne, est ce qui sera exécuté à chaque tour de jeu dans les futures battles où vous testerez votre robot. Nous appelons cela en Java une méthode et elle s'appelle ici :

```
public void run() { ... ...
```

A l'intérieur de cette méthode existe une autre méthode définie entre les accolades qui la suivent. Elle s'appelle `while(true)` et celle-ci permet de définir tous les déplacements principaux de votre robot, tout pendant qu'il est en vie pendant une bataille et lorsqu'il ne se passe rien de particulier pour lui. D'ailleurs si on le voulait, on pourrait très bien ne pas le faire déplacer lorsqu'il ne se passe rien de particulier.

A la création du robot, le programme génère quelques déplacements simples que nous allons détailler ensuite. Mais je ne vous est pas tout dit : cette méthode permet aussi de définir des actions de votre robot comme tourner son canon par exemple.

Voilà, vous en savez déjà plus et voici donc cette méthode prédéfinie à la création de votre robot qui je vous rappelle pourra être modifiée ensuite. D'ailleurs, c'est tout l'intérêt du jeu.

Ligne 22 à 28 :

```
while (true) {  
    //replace the next 4 lines with any behavior you would like  
    ahead(100) ;          avancer de 100 pixels  
    turnGunRight(360);    tourner le canon à droite de 360°  
    back(100) ;           reculer de 100 pixels  
    turnGunRight(360);    tourner le canon à droite de 360°  
}
```

Dans cette méthode `run()` qui je vous rappelle est appelé lors des battles pour connaître les déplacements et actions de votre robot, il existe aussi imbriquée à l'intérieur d'autres méthodes qui permettent de réagir aux événements qui se produisent au cours du jeu. Celles-ci sont évidemment très importantes pour la survie de votre robot, pour se défendre et attaquer de manière efficace. Lorsque ces événements qui sont prévus dans l'application de Robocode se produiront lors de battles contre d'autres robots, votre progéniture pourra réagir. Vous pouvez donc prévoir la réaction de votre robot en lui indiquant au préalable lors de sa création les déplacements et actions qu'il pourra faire.

Ligne 33 à 36, est définie une méthode qui indique au robot ce qu'il doit faire lorsque son scanner a repéré un autre robot :

```
public void onScannedRobot(scannedRobotEvent e) {  
    //replace the next line with any behavior you would like  
    fire(1) ; tire un obus à la force 1  
}
```

Ici, le robot tirera un obus avec une énergie de niveau 1 à chaque fois qu'il repérera un autre robot avec son scanner.

Je suppose que vous vous dites actuellement :



Mais comment écrit-on les déplacements que l'on aimerait programmer à notre robot ?
Quelles sont précisément les déplacements possibles ? Et les actions possibles ?

Et bien, je vous comprends bien ! Et si j'étais à votre place je me dirai la même chose. Je voulais juste m'assurer que vous maîtrisez bien les principes de conception et de fonctionnement de Robocode avant de rentrer dans le détail des fonctions ou plutôt méthodes que vous pouvez utiliser. Et ceci sera tout le contenu des prochains chapitres.

Il faut bien prendre note qu'un robot est défini par :



- son design
- ses déplacements
- ses actions

et en fonction :

- des événements qui peuvent se produire

Avant donc de rentrer dans le vif du sujet, je vois que certains sont impatients, il faut bien se rappeler ce que l'on a vu dans ce chapitre :

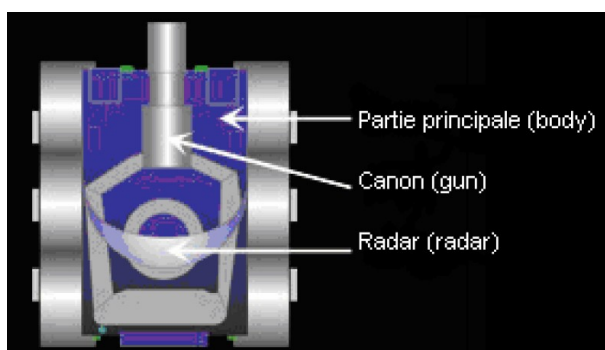
En résumé

dans cette partie, nous avons vu :

- comment utiliser Editor pour créer un nouveau robot, lui donner un nom et un répertoire
- la fenêtre Editing qui permet en langage java de découvrir et modifier ce robot
- la principale méthode run() qui identifie le comportement de votre robot
- la méthode while(true) qui permet de définir le comportement du robot lorsqu'il ne se passe rien de particulier comme événements.
- Que le robot a la possibilité de se déplacer, d'effectuer des actions et de réagir à des événements particuliers, tout ceci encore à l'aide de méthodes
- certaines de ces méthodes sont imbriquées dans la méthode while(true) qui elle-même est imbriquée dans la méthode run().

Partie 3 : Définir le design de votre robot

Tout d'abord, pour créer le design de votre robot, il faut connaître les différents éléments qui le compose : le châssis (ou body), le canon (gun) et le radar (scanner). Il existe aussi le faisceau du radar lorsque celui-ci est utilisé.



pour chacun de ses éléments, vous pouvez choisir et définir sa couleur. Pour cela il existe plusieurs méthodes quelques peut différentes que l'on peut rajouter dans le code avant la méthode `while(true)`, c'est à dire avant la ligne 22 « v'là les flics ! » Non, là je délire, n'importe quoi !
Et si vous êtes observateur, vous avez vu ligne 19, en commentaire, l'exemple d'une méthode possible pour colorier (pas au crayon mon cher Watson) les 3 éléments principaux décrits ci-dessus.

```
// setColors (Color.red, Color.blue, Color.green) //body, gun, radar
```

Et bien, si on ôtait les deux premiers « // » de cette ligne, elle passerait en noir et deviendrait plus un commentaire mais une ligne de programmation qui utiliserait la méthode `setColors()`.
Vous pouvez essayer, vous allez voir si je raconte n'importe quoi.

Dans l'exemple ci-dessus, `Color.red` correspond à la couleur du châssis (ou Body), `Color.blue` à la couleur du canon (gun) et `Color.green` à la couleur du radar.



Quelles sont les couleurs que je peux utiliser avec cette méthode ?

Et bien les voici dans le tableau ci-dessous :

Black	blue	cyan	darkGray	gray	green	lightGray	magenta
	orange	pink	red	white	yellow		

Mais il existe d'autres méthodes possibles dans Robocode pour définir les couleurs de vos robots :

```
setAllColors (Color.couleur)
```

méthode qui permet de définir la même couleur pour toutes les éléments du robot. Il suffit de remplacer « couleur » par votre choix dans les propositions ci-dessus.

```
setColors (Couleur châssis, Couleur canon, Couleur radar, Couleur  
Obus , Couleur faisceau)
```

Celle-ci permet en plus de définir la couleur des obus (lorsqu'il est tiré) et la couleur du faisceau du radar (lorsqu'il est utilisé).

Une méthode existe pour chaque partie du robot, les voici et elles sont assez simples à utiliser.

<code>setBodyColor (Color.couleur)</code>	pour le châssis
<code>setGunColor (Color.couleur)</code>	pour le canon
<code>setRadarColor (Color.couleur)</code>	pour le radar
<code>setScanColor (Color.couleur)</code>	pour le faisceau du radar
<code>setBulletColor (Color.couleur)</code>	pour l'obus

Pour terminer sur ce sujet, il existe d'autres méthodes un petit peu plus complexes qui permettent d'avoir encore plus de choix de couleurs en utilisant le pourcentage des trois couleurs de base, mais nous ne les étudierons pas ici car pas très nécessaire à mon avis. Sachez juste qu'elles existent.

Alors ! Il va être beau votre robot pour le défilé du 14 Juillet ! Oui mais c'est pas tout, encore faut-il qu'il puisse s'y rendre à ce défilé, et qu'il puisse s'y déplacer précisément sans écraser votre belle-mère dans le public qui sera venu exprès pour vous admirer.

Bonne transition pour la partie suivante, hein, mon capitaine !

Partie 4 : Les déplacements du robot

Les méthodes que nous allons voir ici servent à effectuer un déplacement du châssis du robot, elles ne peuvent évidemment pas être utilisées pour le canon ou le radar. Chacun de ces éléments a aussi ses propres méthodes et nous les verrons par la suite.

<code>ahead (distance)</code>	Fait avancer le robot de la distance donnée qui correspond au nombre de pixel
<code>back (distance)</code>	Fait reculer le robot de la distance donnée qui correspond au nombre de pixel
<code>doNothing ()</code>	Ne fait rien (peut être intéressant à utiliser quelquefois)
<code>turnLeft (degrés)</code>	Tourne le châssis vers la gauche de x degrés passés en paramètre
<code>turnRight (degrés)</code>	Tourne le châssis vers la droite de x degrés passés en paramètre



Attention ! Je vous rappelle qu'il faut utiliser exactement la syntaxe pour que votre robot puisse fonctionner. Qu'après avoir défini un déplacement, il faut terminer par un point virgule

partie 5 : Les actions des éléments du robot

Ah! Enfin, allez-vous me dire... On va pouvoir exterminer toute la planète ! Heu... Peace and love mon ami !

Le robot possède donc un radar et un canon, cela doit donc bien pouvoir servir à quelque chose.



**Quelles méthodes puis-je utiliser avec mon canon et mon radar ?
Comment est-ce que je peux m'en servir ? Et pour quoi faire ?**

Alors là, les possibilités vont devenir grandissantes et je vois bien votre soif d'apprendre. N'hésitez pas, prenez un grand verre d'eau et jetons-nous à l'eau !

Évidemment, un canon cela doit bien pouvoir tourner, même peut être bien pouvoir tirer des obus. Et certainement d'autres subtilités très intéressantes. Alors voilà, on se lance (pas la lance de l'homme de Cro Magnon qui a précédé le canon) puis ensuite on « s'attaquera » au radar.

Le canon

```
turnGunLeft(degrés)
```

Tourne le canon vers la gauche de x degrés passés en paramètre

```
turnGunRight(degrés)
```

Tourne le canon vers la droite de x degrés passés en paramètre

```
fire(puissance)
```

Le robot tire des obus de la puissance donnée en paramètre (de 1 à 3)

Mais il existe aussi une méthode très intéressante qui permet de désolidariser le canon des mouvements du châssis :

```
setAdjustGunForRobotTurn(true ou false)
```

Définit le comportement du canon pour qu'il continue à viser dans la direction initiale même si le châssis bouge (Par défaut le canon tourne avec le châssis) . Avec `true` cette méthode est activée et avec `false`, elle est désactivée.



Faite toujours attention à la syntaxe, on ne le dira jamais assez souvent ! Sinon vous ne pourrez pas compiler votre robot, c'est à dire traduire votre programme dans le langage de l'ordinateur.

Pour information, je vais vous donner un bref aperçu de trois autres méthodes qui existent, sans trop rentrer dans les détails car elles ne vous seront utiles que pour un robot déjà bien sophistiqué.

```
getGunCoolingRate()
```

Retourne le taux auquel l'arme à feu se refroidira.

```
getGunHeading()
```

Retourne en degré la direction où pointe le canon

```
getGunHeat()
```

Retourne la chaleur actuelle du canon si elle est à 0

Maintenant comme promis, nous allons voir les méthodes du radar.

Le radar

Pour ce qui est du radar, c'est assez simple pour certaines méthodes puis un peu plus compliqué ensuite :

```
turnRadarLeft (degrés)
```

Tourne le radar vers la gauche de x degrés passés en paramètre

```
turnRadarRight (degrés)
```

Tourne le radar vers la droite de x degrés passés en paramètre

Le radar lui est posé sur le canon et bouge donc avec lui. De la même manière que le canon, il peut être désolidarisé du canon et du châssis avec les méthodes suivantes :

```
setAdjustRadarForGunTurn (true ou false)
```

Définit le comportement du radar pour qu'il continue à viser dans la direction initiale même si le canon bouge (Par défaut le radar tourne avec le canon)

```
setAdjustRadarForRobotTurn (true ou false)
```

Définit le comportement du radar pour qu'il continue à viser dans la direction initiale même si le châssis bouge (Par défaut le radar tourne avec le châssis)

Et pour terminer, une dernière méthode qui peut être utilisée pour les créateurs de robots très ingénieux, donc pas pour vous. Ah ! Quelle susceptibilité ! C' était de l'humour ... En tout cas, à cet instant, moi je ne l'ai pas encore utilisée.

```
getRadarHeading ()
```

Retourne en degré la direction du radar

Voilà, vous avez donc appris les méthodes importantes pour piloter votre « char ». Je préfère ce nom à celui employé dans le manuel de Robocode.



Notez bien que la programmation de votre robot se fait lors de sa création ou lors d'une modification, mais dans ce cas il faudra le ré-compiler à chaque modification. Cette programmation déterminera son comportement face aux événements lors des battles pendant lesquelles vous ne pourrez pas modifier le code.

Nous allons arriver maintenant dans le domaine le plus passionnant et je ne mâche pas mes mots. Celui qui permet à votre robot de réagir à tous les événements que vous pouvez anticiper lors de sa création. Et là cela va se compliquer encore. Et il faut cela pour devenir un génie. Donc, on peut passer à la prochaine partie.

partie 6 : Les réactions du robot face aux événements

Nous arrivons ici dans un assez vaste domaine où beaucoup de choses vont être possible de faire avec une bonne maîtrise des méthodes que nous allons voir progressivement. Mais sachez que le travail ne sera jamais abouti, on pourra toujours faire mieux... Et oui, la perfection n'existe pas ! Même si on essaie de s'en rapprocher.

Voici donc un premier jet de méthodes pré-définies que vous pouvez utiliser pour réagir en fonction des événements :

```
onBulletHit (évènement)
    Définit le comportement du robot lorsqu'il a tiré et touché un autre robot

onBulletHitBullet ()
    Définit le comportement du robot lorsqu'il a tiré et touché un autre obus

onBulletMissed (évènement)
    Définit le comportement du robot lorsqu'il a tiré et touché un mur

onDeath (évènement)
    Définit le comportement du robot pour le "round" suivant lorsqu'il est mort

onHitByBullet (évènement)
    Définit le comportement du robot lorsqu'il est touché par un autre robot

onHitRobot (évènement)
    Définit le comportement du robot lorsqu'il est heurté par un autre robot

onHitWall (évènement)
    Définit le comportement du robot lorsqu'il touche un mur
```



**Mais comment fait-on pour réagir à ces différents événements ?
Doit-on envisager tous les cas ?**

Et bien là, c'est tout votre savoir faire et tout votre génie qui va pouvoir s'exprimer. Vous n'êtes pas obligé d'envisager une réaction à chaque événement. C'est à vous de décider ceux qui vous paraissent importants. Au fur et à mesure de votre savoir faire, vous complétez votre panoplie de Zoro pour être plus crédible face à vos futurs adversaires.

C'est encore un peu vague pour vous, vous avez du mal à percevoir comment il faut écrire les réactions voulues ? Je vous propose de voir un exemple ensemble avec explication de texte.

```
public void onScannedRobot (ScannedRobotEvent e) {
    fire (1) ;
}
```

Dans cet exemple, à chaque fois que le radar repère un autre robot, un obus est tiré à la force 1, en sachant que la force va de 1 à 3, que plus la force est importante, plus il fera de dégâts mais plus il dépensera de l'énergie. Et il faut savoir que l'énergie n'est pas inépuisable.



Notez bien que ces méthodes pré-définies doivent être complétées entre les accolades. Que vous pouvez y placer plusieurs autres méthodes (pour déplacer votre robot par exemple) sans oublier le point virgule à la fin de chaque méthode utilisée.

Voici un deuxième exemple avec plusieurs méthodes utilisées en réaction à un événement :

```
public void onHitByBullet(HitByBulletEvent e) {  
    turnLeft(90);  
    ahead(100) ;  
}
```

Ici, à chaque fois que le robot sera touché par un obus adverse, il tournera à gauche de 90° et avancera de 100 pixels.

En conclusion

Bon ! Les choses sont bien dégrossies j'espère. Et vous devez maintenant avoir une bonne idée de la puissance de développement et de conceptualisation que propose l'application Robocode.

Vous voyez bien que le champ d'investigation est énorme, et il faut que je vous le dise : nous n'avons pas abordé toutes les méthodes que propose cette application. Il en existe bien d'autres, mais cela ne pouvait pas se voir dans ce manuel de premières créations de robots.

Vous pourrez lorsque vous le désirerez, approfondir de nouvelles méthodes en piochant dans la javadoc de cette application, essentiellement dans la Classe Robot, même s'il existe d'autres classes, plus sophistiquées encore, avec encore plus de méthodes.

Vous voyez, les ressources sont inépuisables... et un long chemin peut encore être parcouru durant lequel vous apprendrez encore plein de choses. Ainsi vous serez de plus en plus un expert, avec un sens développé pour la stratégie...

Ou alors, tout ceci est pour vous de la « rigolade », très facile, il n'y a pratiquement plus de mystères. Vous êtes déjà devenu grand ! Et bientôt vous allez vouloir donner des cours ! C'est bien aussi, de donner des cours, et il faut bien que certains le fassent. C'est utile, non ? Ce petit passage était pour moi, cela me fait plaisir et il ne faut pas se priver.

Je vous souhaite plein de courage, et de grandes aventures dans le monde de Robocode.
A bientôt