

# MANUEL de PROGRAMMATION

## Pour aller plus loin dans la création d'un Robot

Si vous avez suivi le « **TUTORIEL de création et de programmation d'un robot** » vous devez donc savoir créer un robot basique. C'est déjà un bon début ! Mais peut être que maintenant, vous voulez combattre ou simplement jouer dans la cour des grands... Ce manuel est donc fait pour vous. Nous allons aborder ensemble des fonctions (ou méthodes) un peu plus élaborées et quelques techniques pratiques pour que votre robot ne ressemble plus à un « punching ball ».

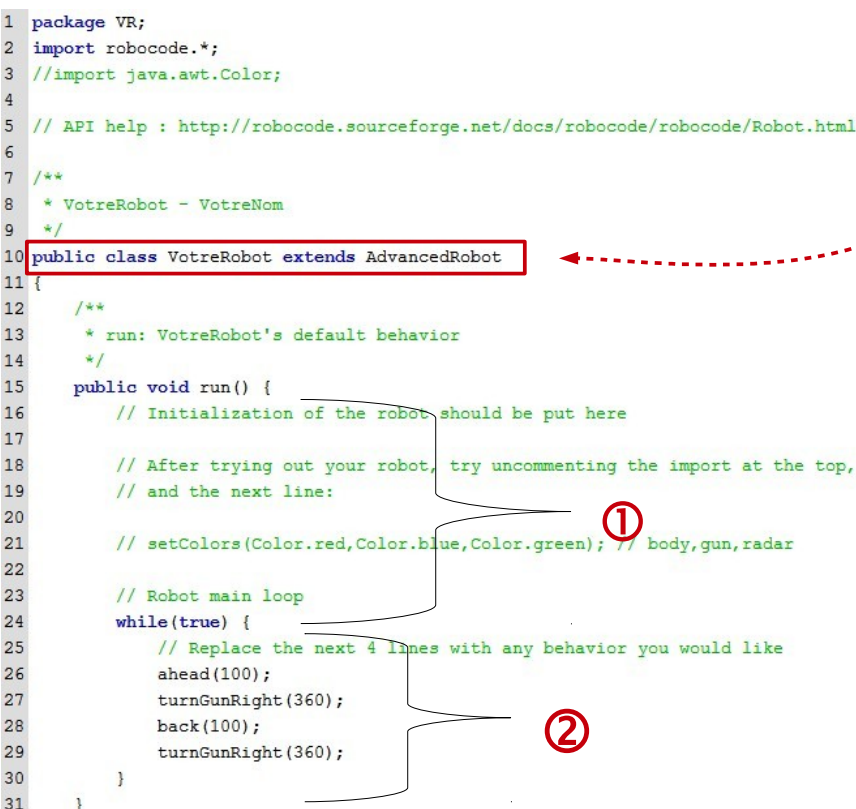
On va passer directement à la partie codage de la création de votre robot. Si vous ne savez pas ce qu'est la partie codage, référez-vous à la « **Partie 1 : créer un robot dans l'application** » du « **Tutoriel de création et de programmation d'un robot** ».

Lorsque la création de votre robot est lancée dans l'application, Il faut savoir qu'il existe deux classes permettant la gestion de ce dernier, la classe Robot et AdvancedRobot. La classe AdvancedRobot possède toutes les fonctions basiques de la classe Robot mais aussi des fonctions supplémentaires plus avancées.

Puisque l'on veut créer un robot efficace, on utilisera la classe AdvancedRobot. Il faut donc changer la ligne 10 :

`public class VotreRobot extends Robot` → `public class VotreRobot extends AdvancedRobot`

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



Pour le moment nous allons nous intéresser à la méthode `run()` et la méthode `while()`.

### **méthode `run()` ① :**

Le code qui est ici va être exécuté une seule fois au début de la partie. On peut donc récupérer des informations qui peuvent être utiles pour mettre en place certaines techniques.

Par exemple, si vous voulez, dès le début de la partie, vous coller contre un mur, vous déplacer dans une certaine direction, vous placer à un endroit stratégique. Et bien c'est ici qu'il faudra écrire le code, avant la méthode `while()`.

`getBattleFieldHeight()` :  
Retourne la hauteur du champ de bataille  
`getBattleFieldWidth()` :  
Retourne la largeur du champ de bataille  
`getX()` :  
Retourne la position x du robot  
`getY()` :  
Retourne la position y du robot  
`turnLeft(degrés)` :  
Tourne le châssis vers la gauche de x degrés passés en paramètre  
`turnRight(degrés)` :  
Tourne le châssis vers la droite de x degrés passés en paramètre  
`getHeading()` :  
Retourne en degré la direction du châssis  
`ahead(distance)` :  
Fait avancer le robot en ligne droite, d'une distance de x pixels passée en paramètre

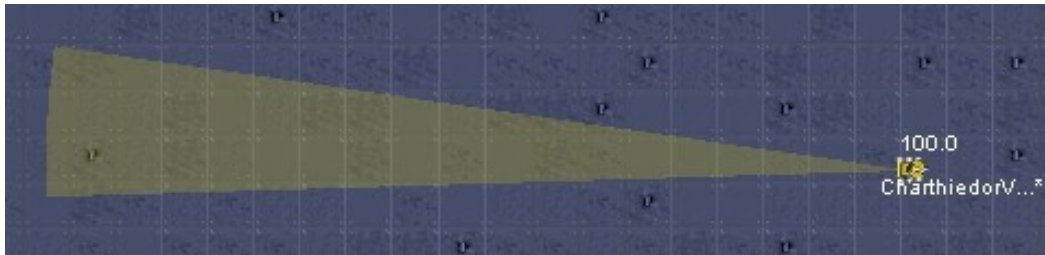
Dans cette partie, il est possible de personnaliser le design de son robot. Il n'est pas nécessaire d'affecter une couleur à votre robot à chaque tour, c'est pour cela que je vous conseil de mettre ces paramètres dans la partie ①.

### **méthode `while()` ② :**

Le code qui se trouve dans cette partie sera exécuté à chaque tour. Je vous propose de mettre en place une technique qui permet de tourner le canon de 10° à chaque tour. Ce permettra de scanner un robot. On écrira le code suivant :

```
// Boucle principale
while(true)
{
    turnRadarRight(10) ; //On tourne le radar vers la droite de 10 degrés
}
```

Le problème avec ce code c'est que l'on avance jamais. Sur un petit champs de bataille le problème ne se pose pas, le robot scannera obligatoirement un ennemi. En revanche, sur un grand champs de bataille, puisque la longueur du radar est limité, le robot ne détectera peut être jamais un autre robot, et en restant passif il finira pas exploser.



Pour remédier à ce problème, il faut avancer lorsque l'on aura atteint 360° sans scanner ou heurter d'ennemi. C'est ce que produit le code suivant :

```
// Boucle principale
while(true)
{
    turnRadarRight(10); // On tourne le radar vers la droite de 10 degrés
    compteurNbDegre += 10;

    // Si on à fait un tour complet sans avoir scanné ou heurté un ennemi, on avance
    if (compteurNbDegre > 360)
        setAhead(getBattleFieldWidth()*0.1);
    // si notre vitesse est nulle (lorsque l'on heurte un mur), on tourne
    if (getVelocity()==0) setTurnRight(120);
}
```



Maintenant que notre début de partie est configuré, on va pouvoir s'attaquer aux événements. Une bonne partie des événements sont expliqués dans le « **Tutoriel de création et de programmation d'un robot** ». Je vais donc vous proposer des comportements possibles à adopter lorsqu'ils sont déclenchés:

### **onDeath(événement)**

Événement déclenché lorsque votre robot est tué

Cet événement peut être utilisé pour changer de stratégie. Le code inscrit dans cet cette fonction, prendra effet au round suivant.

Par exemple, vous dirigez votre robot dans un coin au début de la partie. Après la mort du robot, vous pouvez lui indiquer un changement de direction pour le diriger vers un autre coin.

On pourrait aussi changer la puissance de tire ou la vitesse de déplacement.

```

int corner = 0 ;

// On se tourne face au mur
turnRight(normalRelativeAngleDegrees(corner - getHeading()));
// On avance
ahead(5000);
// On se tourne vers le coin
turnLeft(90);
// On avance
ahead(5000);
// On tourne le canon vers le champs de bataille
turnGunLeft(90);

```



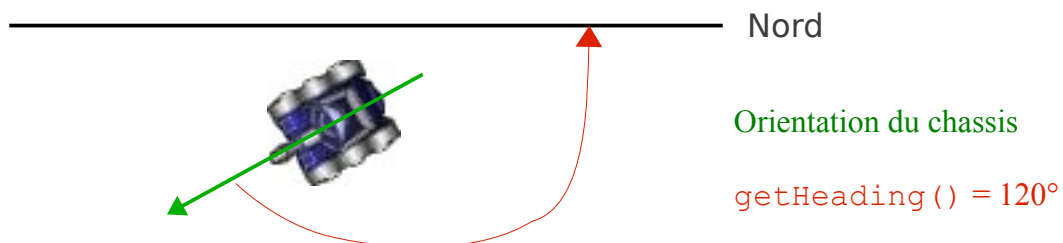
Ce code va permettre à votre robot d'aller se positionner dans le coin gauche.

### Détails :

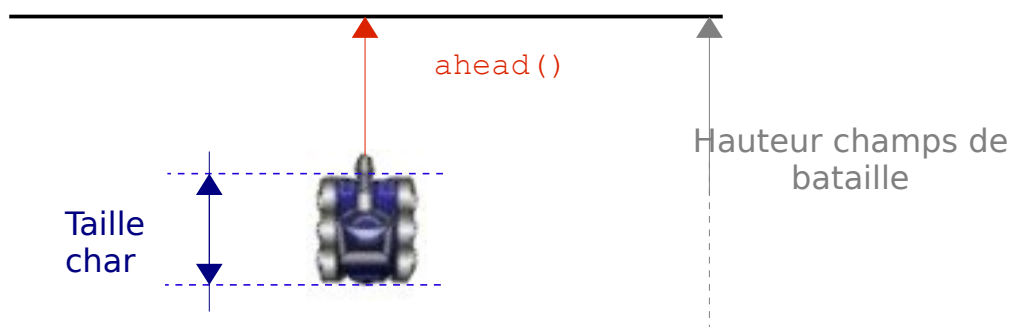
Premièrement, on s'oriente vers le mur du haut.

```
turnLeft(getHeading());
```

La fonction `getHeading()` retourne en degré la direction du châssis par rapport au Nord



Dans le cas du schéma, on va tourner à gauche de 120°.



Une fois en direction du nord, on avance de :  
 (hauteur du champs de bataille - (position Y du robot + **taille char**))

Comme vous avez pu le constater, à chaque méthode faisant référence à un type d'événement, il est possible d'y insérer du code pour que votre robot réagisse précisément en fonction d'une stratégie que vous auriez définie.

Maintenant de manière générale, nous allons citer d'autres méthodes événementielles avec présentation succincte de quelques réactions possibles.

### **onBulletHit(événement)**

Événement déclenché lorsque l'obus tiré par votre robot touche un ennemi.

On peut ici, par exemple, augmenter la puissance de tir lorsque notre robot a touché un ennemi.

### **onBulletHitBullet(événement)**

Événement déclenché lorsque l'obus tiré par votre robot percute un autre obus.

Si l'obus tiré par notre robot touche un obus ennemi, c'est que nous sommes alignés avec l'ennemi. Donc pourquoi pas se décaler rapidement latéralement ou de modifier un peu l'angle de tir pour essayer de toucher l'ennemi et non pas son obus.

### **onBulletMissed(événement)**

Événement déclenché lorsqu'un des obus tirés par votre robot percute un mur.

Si vous vous trouvez dans ce cas, c'est que vous visez mal ! Alors ajustez votre tir ou relancez un scan.

### **onHitByBullet(événement)**

Événement déclenché lorsque votre robot est touché par un obus.

Ici, par exemple, on pourrait riposter en orientant notre canon en direction de l'origine de l'obus percuté de manière à toucher l'ennemi.

### **onRobotDeath()**

Événement déclenché lorsqu'un ennemi meurt.

Vous pouvez changer de stratégie lorsqu'il y a de moins en moins d'ennemi sur le champ de bataille. Par exemple, vous pouvez réduire votre vitesse pour être plus précis. Puisqu'il y a moins d'ennemi, vous avez moins de chance d'être touché.

### **onRoundEnded()**

Événement déclenché lorsqu'un round est fini.

Si vous avez en tête une bonne poignée de stratégie, c'est assez difficile de les fusionner pour en faire une seule. Avec cette fonction vous pourrez changer de stratégie à chaque fin de round. Votre robot sera alors très varié et les ennemis ne pourrons pas prédire vos actions.

### **onScannedRobot(événement)**

Événement déclenché lorsque votre robot scan un ennemi.

« Char en vu ! Rapprochez-vous, encerclez et canardez l'ennemi ! » Il faut mettre en place une stratégie efficace pour que le robot scanné n'est aucune chance de s'en sortir.

### **onWin(événement)**

Événement déclenché lorsque votre robot gagne

Un événement plutôt amusant, vous pouvez effectuer une petite danse de la victoire pour faire rager vos ennemis.

Il faut quand même savoir que pour coder ces fonctions événementielles, nous pouvons faire appel à plein d'autres méthodes qui ont été principalement définies dans la classe « Robot » et dans la classe « AdvancedRobot ».

Si le Robot que vous avez créé hérite de la classe « AdvancedRobot », vous pourrez utiliser en plus des méthodes définies dans cette classe, toutes celles définies dans la classe « Robot ».

A l'inverse, si votre robot hérite de la classe « Robot », vous ne pourrez pas utiliser les méthodes de la classe « AdvancedRobot ».

Voici donc maintenant les fonctions les plus courantes que l'on peut utiliser dans ces deux classes.

## **Classe Robot**

### **ahead(distance)**

Fait avancer le robot en ligne droite, d'une distance de x pixels passée en paramètre

### **back(distance)**

Fait reculer le robot en ligne droite, d'une distance de x pixels passée en paramètre

`doNothing()`

Le robot ne fait rien pendant le tour, il passe son tour

`fire(puissance)`

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

`fireBullet(puissance)`

Le robot tire des obus de la puissance donnée en paramètre. Permet de connaître les caractéristiques de l'obus

`getBattleFieldHeight()`

Retourne la hauteur du champ de bataille

`getBattleFieldWidth()`

Retourne la largeur du champ de bataille

`getEnergy()`

Retourne l'énergie du robot

`getGunHeading()`

Retourne en degré la direction ou pointe le canon

`getGunHeat()`

Retourne la chaleur actuel du canon si elle est à 0

`getHeading()`

Retourne en degré la direction du châssis

`getHeight()`

Retourne en pixel la hauteur du robot

`getName()`

Retourne le nom du robot

`getNumRounds()`

Retourne le nombre de round dans la bataille actuel

`getOthers()`

Retourne le nombre d'ennemi restant

`getRadarHeading()`

Retourne en degré la direction du radar

`getRoundNum()`

Retourne le round actuel de la bataille

`getTime()`

Retourne le numero du tour actuel

`getVelocity()`

Retourne la vitesse pixel/tour => max 8 pixels/turn

`getWidth()`

Retourne en pixel la largeur du robot

`getX()`

Retourne la position x du robot

`getY()`

Retourne la position y du robot

`scan()`

Lance le repérage des autres robots => Cf onScannedRobot()

`setAdjustGunForRobotTurn(independent)`

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

`setAdjustRadarForGunTurn(independent)`

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

`setAdjustRadarForRobotTurn(independent)`

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

`setAllColors(couleur)`

Définit la couleur du robot avec la couleur passée en paramètre

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

Existe aussi pour chaque partie de manière séparée

Définit les couleurs du châssis, du canon, du radar, de l'obus, du faisceau radar avec les couleurs passées en paramètre

`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

`turnLeft(degrés)`

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



`turnRight(degrés)`

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

`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

## **Classe AdvancedRobot**

`addCustomEvent(condition)`

enregistre un événement quand une condition est respectée.

`clearAllEvents()`

supprime tous les événements en attente concernant le robot.

`execute()`

execute toutes les actions en attente tout en continuant celles commencées par le processus.

`getAllEvents()`

retourne un "vector" contenant tous les événements.

`getBulletHitBulletEvents()`

retourne un "vector" contenant tous les événements concernant la collision d'obus.

`getBulletHitEvents()`

retourne un "vector" contenant tous les événements concernant la collision d'un obus avec un autre robot.

`getBulletMissedEvents()`

retourne un "vector" contenant tous les événements concernant la collision d'un obus avec un mur.

`getEventPriority(eventClasse)`

retourne l'événement prioritaire.

`getGunTurnRemaining()`

retourne l'angle qu'il reste à tourner en degrés.

`getHitByBulletEvents()`

retourne un "vector" contenant tous les événements concernant la frappe d'un obus sur le robot.

`getHitRobotEvents()`

retourne un “vector” contenant tous les événements concernant la collision du robot avec d’autres robots.

#### `getHitWallEvents()`

retourne un “vector” contenant tous les événements concernant la collision du robot avec les murs.

#### `getRadarTurnRemaining()`

retourne en degré, l’angle du radar qu’il reste à tourner. Angle négatif si tourne vers la gauche, positif vers la droite et nul si rotation est effectuée complètement

#### `getRobotDeathEvents()`

retourne un “vector” contenant tous les événements concernant mort du robot.

#### `getScannedRobotEvents()`

retourne un “vector” contenant tous les événements concernant les scans du robot.

#### `getStatusEvents()`

retourne un “vector” contenant tous les événements concernant l’état du robot à la fin de chaque tour.

#### `getTurnRemaining()`

retourne l’angle qu’il reste à tourner pour le châssis du robot.

#### `isAdjustGunForRobotTurn()`

retourne vrai ou faux si la rotation du canon est liée à celle du châssis.

#### `isAdjustRadarForGunTurn()`

retourne vrai ou faux si la rotation du radar est liée à celle du canon.

#### `isAdjustRadarForRobotTurn()`

retourne vrai ou faux si la rotation du radar est liée à celle du châssis.

#### `onCustomEvent(événement)`

effectue cette méthode si l’événement a lieu.

#### `onDeath(événement)`

effectue cette méthode si le robot est mort.

#### `onSkippedTurn(événement)`

Cette méthode est appelée si le robot utilise trop de temps entre des actions. Quand cet événement arrive, le tour du robot est sauté, signifiant qu’il ne peut pas agir désormais à ce tour.

#### `removeCustomEvent(condition)`

supprime l’événement lorsque la condition est vérifiée.

#### `setAhead(distance)`

avancera le robot de la distance donnée à l’exécution suivante.

#### `setBack(distance)`

reculera le robot de la distance donnée à l'exécution suivante.

`setEventPriority(eventClass, priorité)`

précise la priorité donnée pour un type d'événement

`setFire(puissance)`

Mettre la puissance de l'obus lors de la prochaine exécution

`setFireBullet(puissance)`

Mettre la puissance de l'obus lors de la prochaine exécution et retourne les caractéristique de l'obus

`setMaxTurnRate(newMaxTurnRate)`

Met la rotation maximum effectuée par le robot => Max 10 degré/tour.

`setMaxVelocity(newMaxVelocity)`

Met la vitesse du robot => Max 8 pixel/tour.

`setResume()`

Reprend le mouvement du robot arrêté par `stp()` ou `setStop()`. Elle ne s'exécute pas tant que l'on a pas appelé `execute()`.

`setStop()`

ressemble à `stop()` *Robot.java* mais donne son retour directement. Et ne s'exécute pas tant qu'on a pas appelé `execute()`.

`setStop(booléen)`

ressemble à `stop()` *Robot.java* mais donne son retour directement. Et ne s'exécute pas tant qu'on a pas appelé `execute()` . Différences avec `setStop()`?!?!

`setTurnGunLeft(degré)`

Fait tourner le canon de x degré vers la gauche quand la prochaine exécution à lieu.

`setTurnGunRight(degré)`

Fait tourner le canon de x degré vers la droite quand la prochaine exécution à lieu.

`setTurnLeft(degré)`

Fait tourner le châssis de x degré vers la gauche quand la prochaine exécution à lieu.

`setTurnRadarLeft(degré)`

Fait tourner le radar de x degré vers la gauche quand la prochaine exécution à lieu.

`setTurnRadarRigth(degré)`

Fait tourner le radar de x degré vers la droite quand la prochaine exécution à lieu.

`setTurnRigth(degré)`

Fait tourner le châssis de x degré vers la droite quand la prochaine exécution à lieu.

`turnGunLeft(radian)`

Fait tourner le canon de x radian vers la gauche immédiatement.

`turnGunRight(radian)`

Fait tourner le canon de x radian vers la droite immédiatement.

`turnLeft(radian)`

Fait tourner le châssis de x radian vers la gauche immédiatement.

`turnRadarLeft(radian)`

Fait tourner le radar de x radian vers la gauche immédiatement.

`turnRadarRigth(radian)`

Fait tourner le radar de x radian vers la droite immédiatement.

`turnRigth(radian)`

Fait tourner le châssis de x radian vers la droite immédiatement.

Pour terminer, il faut savoir aussi qu'il existe des méthodes que l'on peut utiliser dans la classe « TeamRobot ». Ces méthodes permettent principalement de passer des messages entre robots dans le cas où l'on voudrait mettre en place des combats par équipes de robots.