



Créer, initialiser, exécuter un programme

Lancez RobotProg en double-cliquant sur l'icône



Liste des blocs :
D: Début,
F: Fin,
?: test logique,
TG: Tourne à Gauche
A: Avance d'1 case,
TD: Tourne à Droite

Liaison
 Sélection
//c Commentaires

A. Dessiner un organigramme

- Pour choisir un **bloc** dans la palette d'outils, **cliquer** sur le bloc ; puis **déplacer** le pointeur de souris dans la fenêtre de programme ; puis **cliquer** de nouveau pour y placer le bloc
- Pour effacer un bloc, **cliquer** sur la **gomme** puis **cliquer** sur le bloc à effacer
- pour **lier** les blocs entre eux, **cliquer** sur l'**outil liaison** dans la palette , puis **cliquer** sur une sortie de bloc, **déplacer** la souris et **cliquer** sur l'entrée du bloc suivant.
- on peut aussi joindre directement un bloc à un autre au moment du placement mais on verra plus tard qu'il est préférable d'utiliser les liaisons. (*voir ci-dessus*)



Un organigramme **doit contenir un seul bloc début** pour indiquer où le programme doit commencer, **mais peut comporter un ou plusieurs blocs fin.**

Exercice 1 :

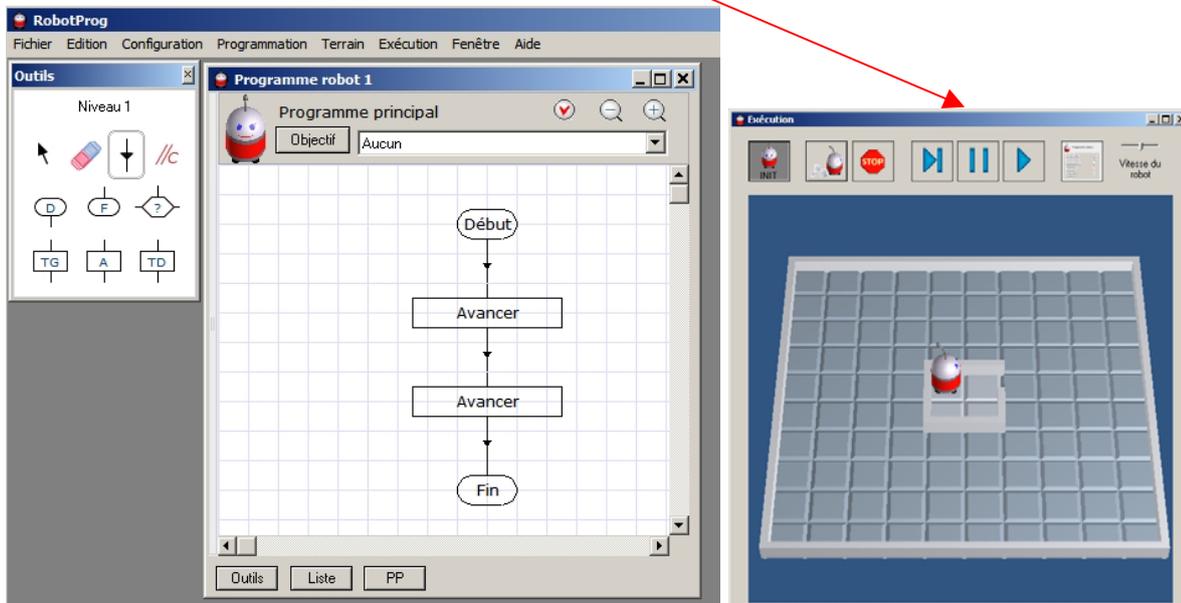
Reproduire l'organigramme ci-dessus

Vérifier la validité de l'organigramme en cliquant sur le bouton



B. Affichage du terrain du robot

Pour visualiser l'exécution du programme, il faut faire apparaître le terrain.
Cliquer sur le menu **Fenêtre > Exécution..**



C. Initialisation du programme



Cliquez sur le bouton **INIT** (ou **Exécution > Initialisation**). Le programme créé d'après l'organigramme est alors vérifié.

Si le programme ne contient pas d'erreur, le bouton **INIT** affiche alors une image du robot à initialiser.



Si le programme contient une erreur, vous ne pourrez pas lancer l'exécution, vous devrez d'abord corriger l'erreur.

Il faut maintenant positionner correctement le robot dans une case de départ.

- Pour positionner le robot sur une case, cliquez sur la case choisie.
- Pour le tourner d'un quart de tour, cliquez sur le robot.**

D. Lancement du programme

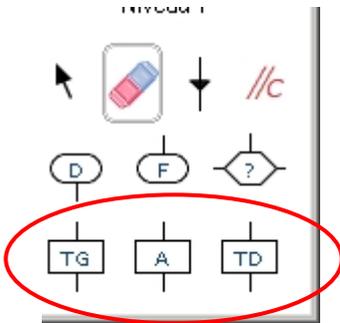


- Cliquez sur le bouton  ou bien choisissez le menu **Exécution > Lancer**. (Impossible de cliquer sur ce bouton si vous n'avez pas tout d'abord initialisé le programme.)
- Initialisez et lancez votre programme. (un rectangle rouge qui se déplace dans la **fenêtre de programme**.)
- Essayez plusieurs positions de départ du robot et plusieurs orientations (G/D).
- Créer un dossier **Robotprog_1415_NOM** et
- Sauvegardez votre programme (menu **Fichier**) dans votre dossier sous le nom **PROG1**
- Quittez le programme (menu **Fichier-FERMER**).



Les déplacements du robot

Pour déplacer le robot, on dispose de trois commandes:



Avancer (A) = avance d'une case devant lui

Tourner à droite (TD) = 1/4 de tour à Droite

Tourner à gauche (TG) = 1/4 de tour à Gauche

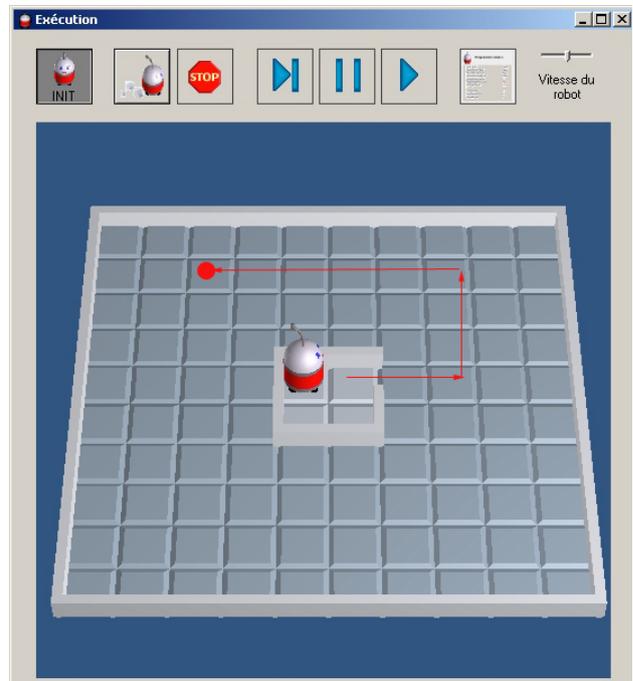
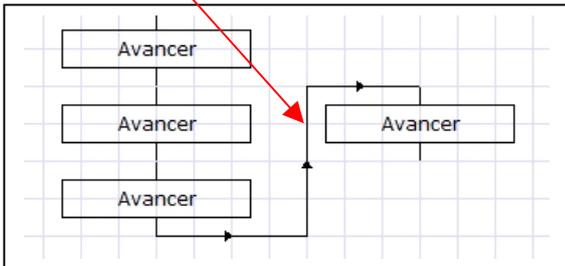


Si le robot se heurte au mur, on dit qu'il y a une erreur d'exécution : le programme s'arrête.

Exercice 2a

Le robot doit parcourir le trajet ci-dessous. Écrivez le programme. Sauvegardez ensuite votre programme sous le nom **Trajet1**.

Si la fenêtre est trop petite, utilisez les liaisons comme dans cet exemple



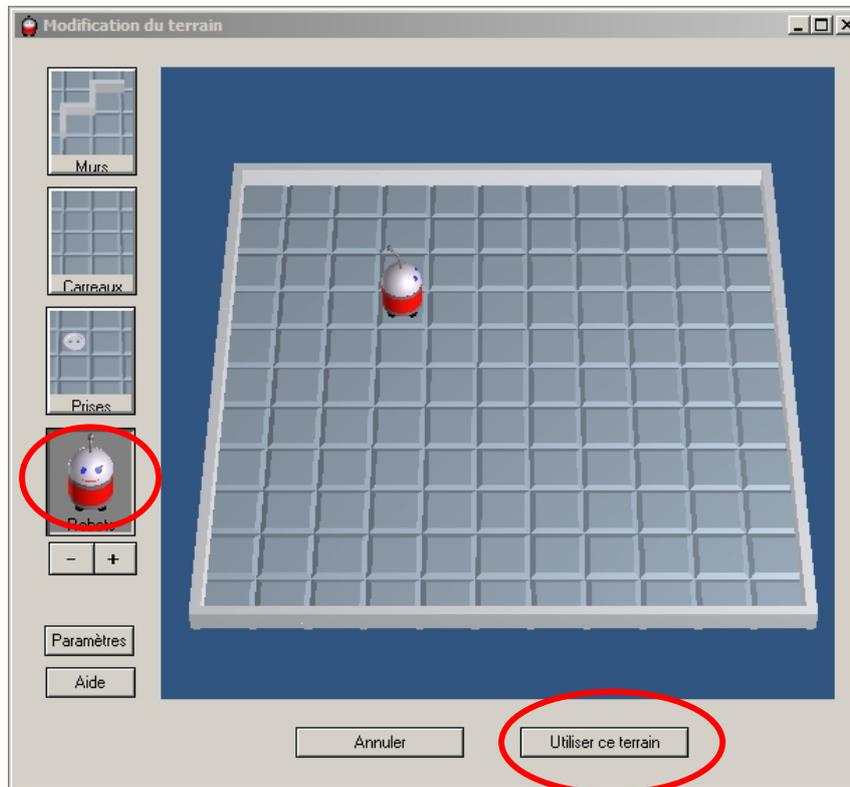
Exercice 2b

- Écrivez un nouveau programme où le robot avance de trois cases, fait demi-tour, et retourne à sa case de départ, **en position initiale exacte**. Exécutez-le pour vérifier. (attention, lisez bien la consigne)
- Sauvegardez ensuite votre programme sous le nom de **Trajet2**.



Modifications du terrain

Le terrain peut être modifié, mais avant l'exécution d'un programme.
On peut ajouter ou retirer des murs à l'intérieur du terrain (mais pas ceux l'entourant).



1. Menu **Terrain > Modifier**. La **fenêtre d'édition de terrain** apparaît.
2. Menu **Terrain > Nouveau** pour créer un nouveau terrain. La fenêtre des paramètres du terrain apparaît.
3. Choisir une largeur et une hauteur de 11 cases
4. Choisissez 1 seul robot et une énergie de 1000.
5. Validez en cliquant le bouton **OK**.
6. Cliquez sur le bouton **Robots** puis cliquez sur le bouton **+** en dessous: un robot apparaît sur le terrain.
7. Cliquez sur une case pour placer le robot à sa position initiale pour le début de l'exécution.
8. Enregistrer (**Terrain-Enregistrer sous**) sous le nom **Terrain11x11SansObstacles**
9. Fermez la fenêtre en cliquant sur le bouton **Utiliser ce terrain**. Si vous affichez le terrain avec le menu **Fenêtre > Fenêtre exécution**, vous constaterez que le terrain a bel et bien été modifié.

Exercice 3

Construisez le terrain comme décrit ci-dessus et enregistrez-le (**Terrain-Enregistrer sous**) sous le nom **Terrain11x11SansObstacles**.



Conditions logiques et tests

Objectifs : faire tester un obstacle au robot.

1. Conditions logiques

Une condition logique est une expression donnant un résultat vrai ou faux.

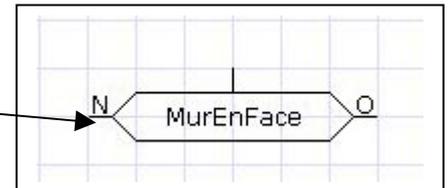
On verra par la suite que l'on peut combiner différents mots-clé avec les **opérateurs logiques Et, Ou, Non**, par exemple: (*MurEnFace Et MurADroite*) **Ou** Non *MurAGauche*.

2. Bloc test

Dans l'organigramme, un **bloc test** a cette forme

Pour modifier le texte contenu dans un bloc test, il faut choisir

l'outil sélection  et faire un double-clic sur le bloc.

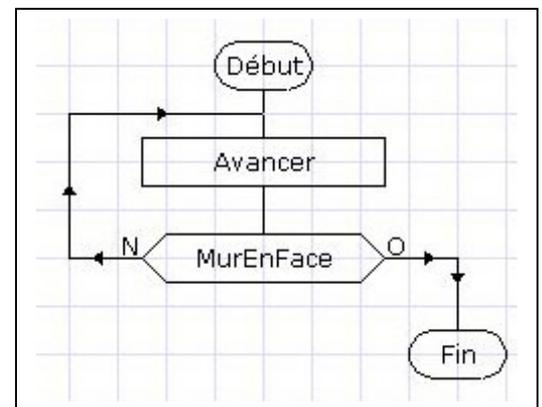


Si le résultat est vrai, l'exécution se poursuit après la sortie marquée **O** (Oui signifie vrai) ;

Si le résultat est faux, l'exécution se poursuit après la sortie marqué **N** (non ou faux).

Dans l'exemple ci-contre, on utilise la condition logique *MurEnFace*. (à écrire dans le bloc)

Si le robot est en face d'un mur, **Alors** le programme ira vers Oui et donc s'arrêtera (Fin), **Sinon** le programme ira vers Non et donc le robot avancera et recommencera le test jusqu'à ce qu'il rencontre un mur.



Exercice 4a

Objectif : Aller jusqu'à un mur et s'arrêter

Vous utiliserez le terrain Terrain11x11SansObstacles.

Le programme doit fonctionner **quelles que soient la position et la direction initiale du robot**, sans erreur d'exécution.

Enregistrer le programme sous le nom **TestMur**

Visualisation de l'exécution pas à pas Pendant l'exécution d'un programme, vous pouvez cliquer sur le bouton Pause  dans la fenêtre exécution, puis faire exécuter les instructions les unes après les autres en cliquant sur le bouton Exécution pas à pas . Vous trouverez la liste complète des mots-clés dans le document accessible par le menu Aide > Résumé du langage du robot

Exercice 4b

Objectif : Modifier le programme TestMur.pour Aller dans un coin et s'arrêter

Vous utiliserez le terrain Terrain11x11SansObstacles.

Le programme doit fonctionner **quelles que soient la position et la direction initiale du robot**, sans erreur d'exécution. **Testez plusieurs orientations initiales du robot**

Enregistrer le programme sous le nom **TestCoin**

Exercice 4c

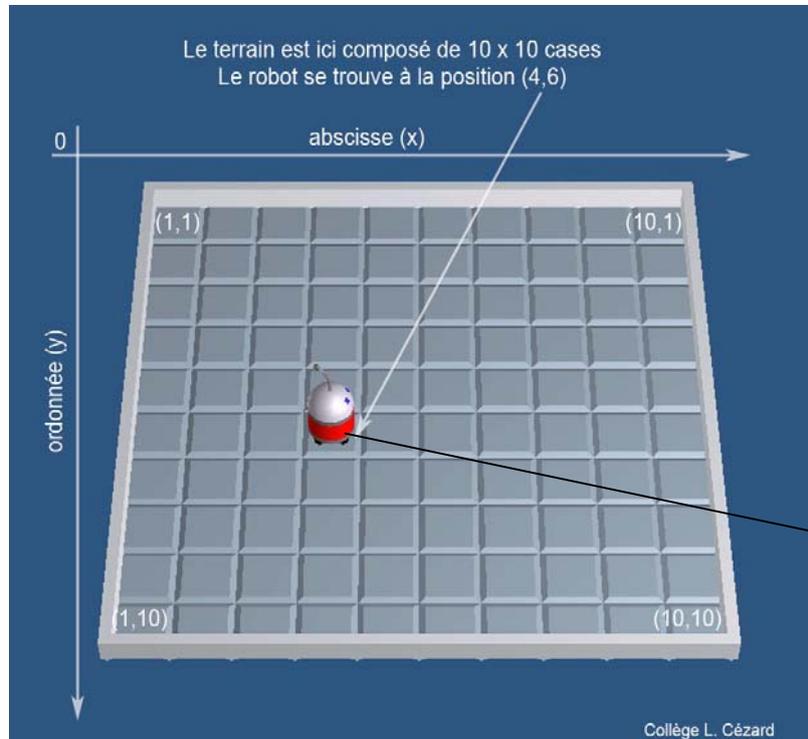
Objectif : Modifier le programme TestCoin pour créer une programme qui permet au robot d'aller dans les 4 coins et s'arrêter

Modifiez la position du robot et testez que quelle que soit sa position, il passe par les 4 coins. Enregistrer le programme sous le nom **Test4Coins**.

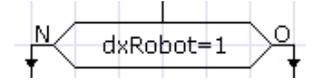


Les déplacements du robot

Une case du terrain est repérée par ses coordonnées (x, y);
x et y sont des nombres entiers positifs.



Pour utiliser dans un bloc



xRobot =
yRobot =
dxRobot = 1
dyRobot = 0

Passer
impérativement au
niveau 2
Menu Configuration
Niveau

Position
xRobot = 4
yRobot = 6
Direction
dxRobot = 1
dyRobot = 0

1. Position du robot (x, y)

- La **position** du robot est donnée par les deux mots-clef **xRobot** et **yRobot**.
- Pendant l'exécution, **xRobot** et **yRobot** ont les valeurs x, y de la case occupée par le robot.
- Dans l'exemple ci-dessus, **xRobot = 4** et **yRobot = 6**.

2. Direction du robot (dx, dy)

La **direction** suivant laquelle le robot est orienté est donnée par les deux mots-clef **dxRobot** et **dyRobot**.

Les valeurs de **dxRobot** et **dyRobot** varient de **xRobot** et de **yRobot** quand le robot avance d'une case devant lui:

- si le robot est tourné **vers la droite** du terrain: **dxRobot** vaut **1** et **dyRobot** vaut **0**
- si le robot est tourné **vers la gauche** du terrain: **dxRobot** vaut **-1** et **dyRobot** vaut **0**
- si le robot est tourné **vers l'avant** du terrain: **dxRobot** vaut **0** et **dyRobot** vaut **1**
- si le robot est tourné **vers l'arrière** du terrain: **dxRobot** vaut **0** et **dyRobot** vaut **-1**



dxRobot et **dyRobot** n'ont comme valeurs possibles que **0** ou **1** ou **-1**.

L'une des deux valeurs est nulle et l'autre non nulle. (voir ci-dessus)

Exercice 5a

Écrivez un programme qui oriente le robot vers la gauche, quelle que soit son orientation initiale.

Sauvegardez ce programme sous le nom S OrienterAGauche.

Exercice 5b

Vous utiliserez pour ce deuxième exercice le terrain Terrain11x11SansObstacles défini à la fiche 3.

Écrivez un programme qui déplace le robot sur une case quelconque de la colonne centrale (x=5), quelles que soient sa direction et sa position initiales.

Sauvegardez ce programme sous le nom AllerColonneCentrale.



Création d'un sous-programme

1. Pourquoi des sous-programmes ?

Un programme est écrit pour résoudre un problème qui peut être très complexe. On commence donc habituellement par analyser le problème posé et **on le divise en problèmes plus petits** et donc plus faciles à résoudre.

Les sous-programmes permettent ainsi **de décomposer un programme en plusieurs parties**. L'avantage est qu'un **même sous-programme peut être utilisé à plusieurs reprises**, ce qui évite de réécrire plusieurs fois le même code.

2. Créer un sous-programme

Fichier > Nouveau programme. puis Programmation > Nouveau sous-programme

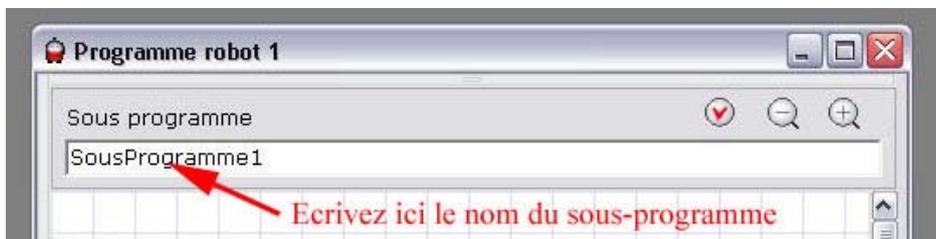
Le sous-programme est maintenant affiché dans la **fenêtre programme** à la place du programme principal.

Par défaut, quand le sous-programme est créé, il reçoit le nom de SousProgramme1. Remplacez ce nom par **SeTournerVersLaGauche**.

Les sous-programmes sont disponibles à partir du niveau 2.

Le niveau est affiché dans la **palette d'outil**. Si vous êtes au niveau 1, passez au niveau 2 en utilisant le menu **Configuration > Niveau**.

Un nom de sous-programme est formé **de lettres et de chiffres sans espaces**, doit commencer par une lettre et contenir au maximum 32 caractères. Il doit être différent des mots-clé du langage du robot. Les lettres majuscules et minuscules sont considérées comme étant identiques



Exercice 6 Copiez-collez l'organigramme **SorienterAGauche** que vous avez construit à la [leçon 5](#) dans le sous-programme **SeTournerVersLaGauche**.



Afficher le programme principal et les sous-programmes

- Cliquez sur le bouton **Liste** en bas de la fenêtre programme pour afficher (ou masquer) la liste des sous-programme. La liste apparaît à gauche de la fenêtre.
- Le premier élément de cette liste est le programme principal. Son nom est le nom de la fenêtre (c'est aussi le nom du fichier associé si le programme a été enregistré). Les éléments suivants sont les noms des sous-programmes créés. Pour afficher l'un de ces éléments il suffit de cliquer sur son nom dans la liste.
- Quand un sous programme est affiché, vous pouvez aussi revenir au programme principal en cliquant sur le bouton **PP** en bas de la fenêtre.
- Vous pouvez aussi afficher simultanément le programme principal et un sous-programme en faisant glisser le séparateur horizontal qui est placé en haut ou en bas de la fenêtre.



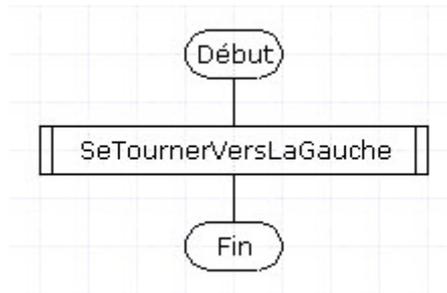
Appeler un sous-programme

Pour qu'un sous-programme soit exécuté, il faut l'appeler à partir du programme principal ou bien d'un autre sous-programme. Ceci se réalise avec une instruction d'appel qui figure dans un bloc d'appel de sous-programme



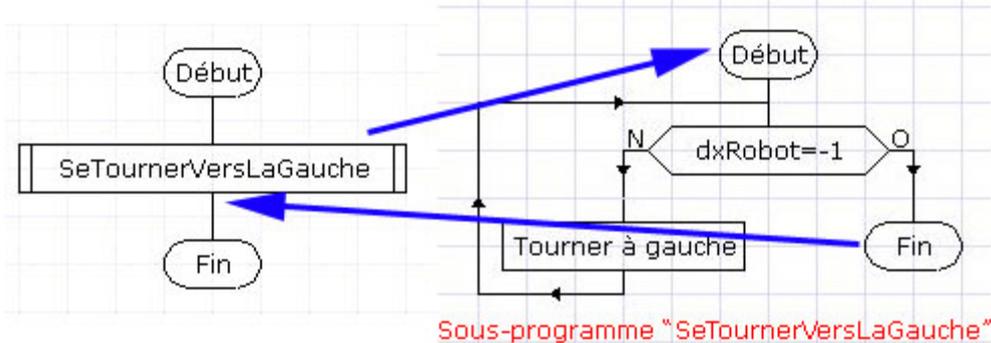
Pour tester l'exécution du sous-programme **SeTournerVersLaGauche**, vous allez écrire le programme principal contenant l'appel du sous-programme.

Affichez le programme principal et construisez l'organigramme suivant:



1. Déroulement de l'exécution

L'exécution d'un appel de sous-programme provoque le passage à l'exécution du bloc début du sous-programme, puis le sous-programme est exécuté jusqu'à son bloc fin, et ensuite l'exécution se poursuit dans le programme appelant au bloc qui suit le bloc d'appel de sous-programme.



Sous-programme "SeTournerVersLaGauche"

Exercice 7a Ajoutez et faites exécuter un nouveau sous-programme de nom SeTournerVersLaDroite.

Conseil: Copiez-collez l'organigramme de SeTournerVersLaGauche et modifiez-le.

Exercice 7b Écrivez un programme qui déplace le robot sur la case centrale du terrain, de coordonnées (5,5). Vous utiliserez le terrain Terrain9x9SansObstacles défini à la fiche 3.

Ce programme doit fonctionner correctement quelles que soient la position et l'orientation initiales du robot.

Vous écrirez pour cela les 6 sous-programmes :

SeTournerVersLaDroite – SeTournerVersLaGauche - SeTournerVersLAvant, SeTournerVersLArriere – AllerALigneCentrale - AllerAColonneCentrale.



Variables et expressions

1. Les Variables

Les données manipulées par un programme sont stockées dans la mémoire de l'ordinateur. Pour accéder à la zone de mémoire correspondant à une donnée, le programmeur associe une **variable** à la donnée.

Le **nom d'une variable** est défini par le programmeur.

Un nom de variable est formé de lettres et de chiffres sans espaces. Le nom doit commencer par une lettre et peut contenir au maximum 32 caractères.

Il doit être différent des mots-clés du langage du robot. (Les lettres majuscules et minuscules sont considérées comme étant identiques.)

Dans cette leçon, nous déterminerons le nombre de pas effectués par le robot. Pour cela nous utiliserons une variable de nom *nbPas*. On n'aurait pas pu utiliser le nom *Pas*, car c'est un mot-clé du langage.

2. La valeur d'une variable

Pour simplifier, on peut voir les variables comme des "boîtes". Le nom de la variable permet d'identifier la boîte. La valeur de la boîte est son contenu.

Dans RobotProg les variables sont définies pour l'ensemble du programme et des sous-programmes, **les valeurs des variables sont des nombres entiers**. Toutes les variables ont la valeur 0 au début de l'exécution du programme.

3. Expressions numériques

Les **expressions numériques** sont des formules de calcul pouvant **contenir des variables, des nombres entiers, des parenthèses et des signes d'opération**. Le résultat du calcul est un nombre entier. En supposant que la variable *nbPas* contient la valeur **5**, l'expression ***nbPas* + 2** donnera comme résultat la valeur **7**.

Une expression numérique peut aussi contenir des fonctions numériques prédéfinies par des mots-clés, comme *DistanceMur* qui donne le nombre de cases entre le robot et le mur en face de lui, ou bien *xRobot* et *yRobot* qui donnent la position du robot. Si, par exemple, le robot se trouve à 4 cases du mur en face, l'expression $2 * \text{DistanceMur}$ donnera comme résultat 8.

Affectation d'une valeur à une variable

On change la valeur d'une variable avec l'instruction d'affectation qui a la forme suivante : *nom de la variable* = *expression numérique*. Par exemple : *nbPas* = *DistanceMur* + 2.

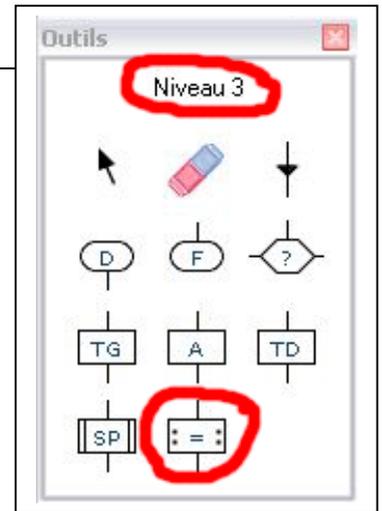
Quand l'instruction est exécutée, la valeur de l'expression est calculée, puis cette valeur est placée dans la variable dont le nom figure à gauche du signe =

Le signe = n'a pas la même signification que le signe = utilisé en mathématiques. On peut ainsi écrire une instruction de la forme suivante : *nbPas* = *nbPas* + 1.

Exemple :

En supposant que *nbPas* a la valeur **5** avant l'exécution de l'instruction, l'expression ***nbPas* + 1** est calculée, ce qui donne **6**, puis ce résultat **6** est affecté à la variable *nbPas*. On a ainsi augmenté la valeur de *nbPas* d'une unité, cette opération s'appelle une **incrément**.

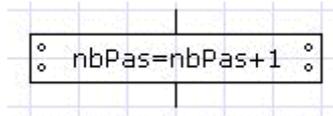
Si on reprend l'image des boîtes: on va chercher la boîte de nom *nbPas*, on regarde son contenu, on y ajoute 1, et on remet dans la même boîte la nouvelle valeur, qui remplacera l'ancienne





Fiche n° 8:

Dans un organigramme, l'instruction d'affectation s'écrit dans un **bloc affectation**:



L'affectation est utilisable à partir du niveau 3: choisissez en conséquence un niveau supérieur ou égal à 3. Utilisez le terrain Terrain9x9SansObstacles.

Exercice 8	<p>Reprenez votre programme de la fiche 4 qui faisait aller votre robot dans un coin, et modifiez-le pour que le robot compte le nombre de ses pas depuis sa position initiale jusqu'à sa position finale. Le principe est simple: chaque fois que le robot avancera, on incrémentera la variable <i>nbPas</i>. Sauvegardez ce programme sous le nom AllerCoinEnComptant.</p> <p>Lancez l'exécution et cliquez sur le bouton Montrer les variables  pour afficher l'état courant. Vous pourrez ainsi suivre l'évolution de la valeur de la variable <i>nbPas</i> pendant les déplacements du robot.</p>
-------------------	---

