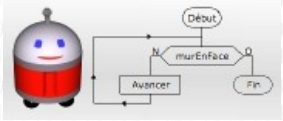
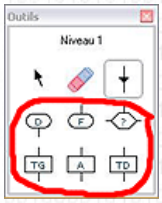


LEÇON 1 :

CRÉER, INITIALISER ET LANCER UN PROGRAMME

Lancez RobotProg. Vous voyez deux fenêtres affichées: la fenêtre de programme où vous allez dessiner l'organigramme et une palette d'outils permettant de construire l'organigramme.

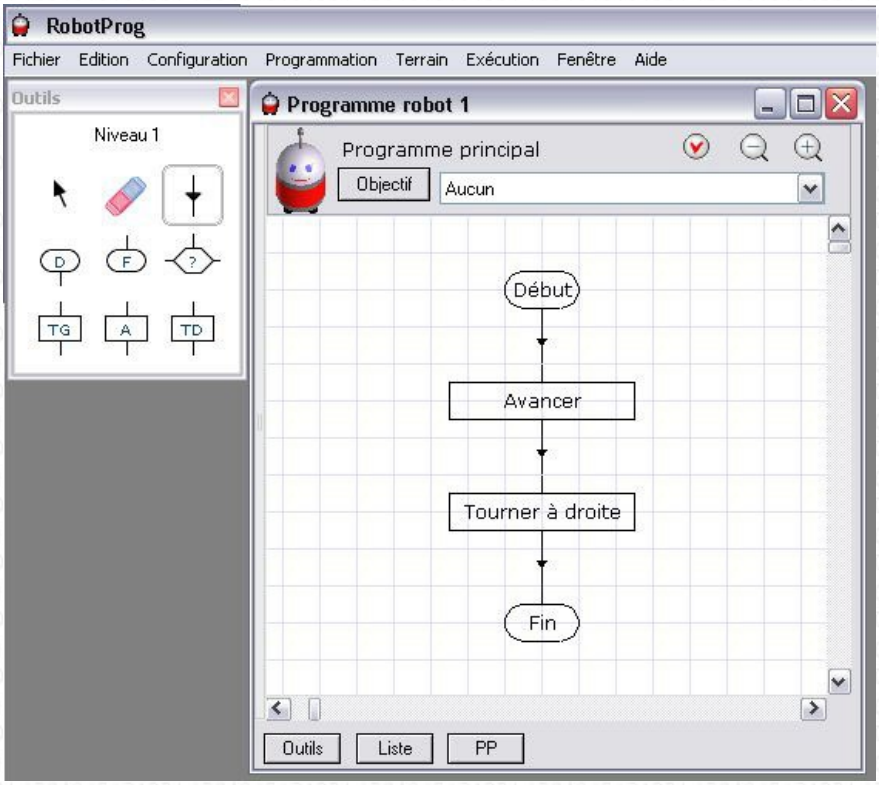
Dessiner un organigramme



Blocs

Pour construire l'organigramme représenté ci-dessous :

- pour chaque bloc de l'organigramme: prenez le bloc dans la palette d'outils (D: Début, F: Fin, ?: test logique, TG: Tourner à Gauche, A: Avancer, TD: Tourner à Droite) et cliquez ensuite dans la fenêtre de programme pour y placer le bloc;
- pour lier les blocs entre eux: choisissez l'outil lien dans la palette (la flèche vers le bas), puis cliquez sur une sortie de bloc, déplacez la souris et cliquez sur l'entrée du bloc suivant. *Remarque:* on peut aussi joindre directement un bloc à un autre au moment de placement; il suffit que la sortie d'un bloc touche l'entrée d'un autre.

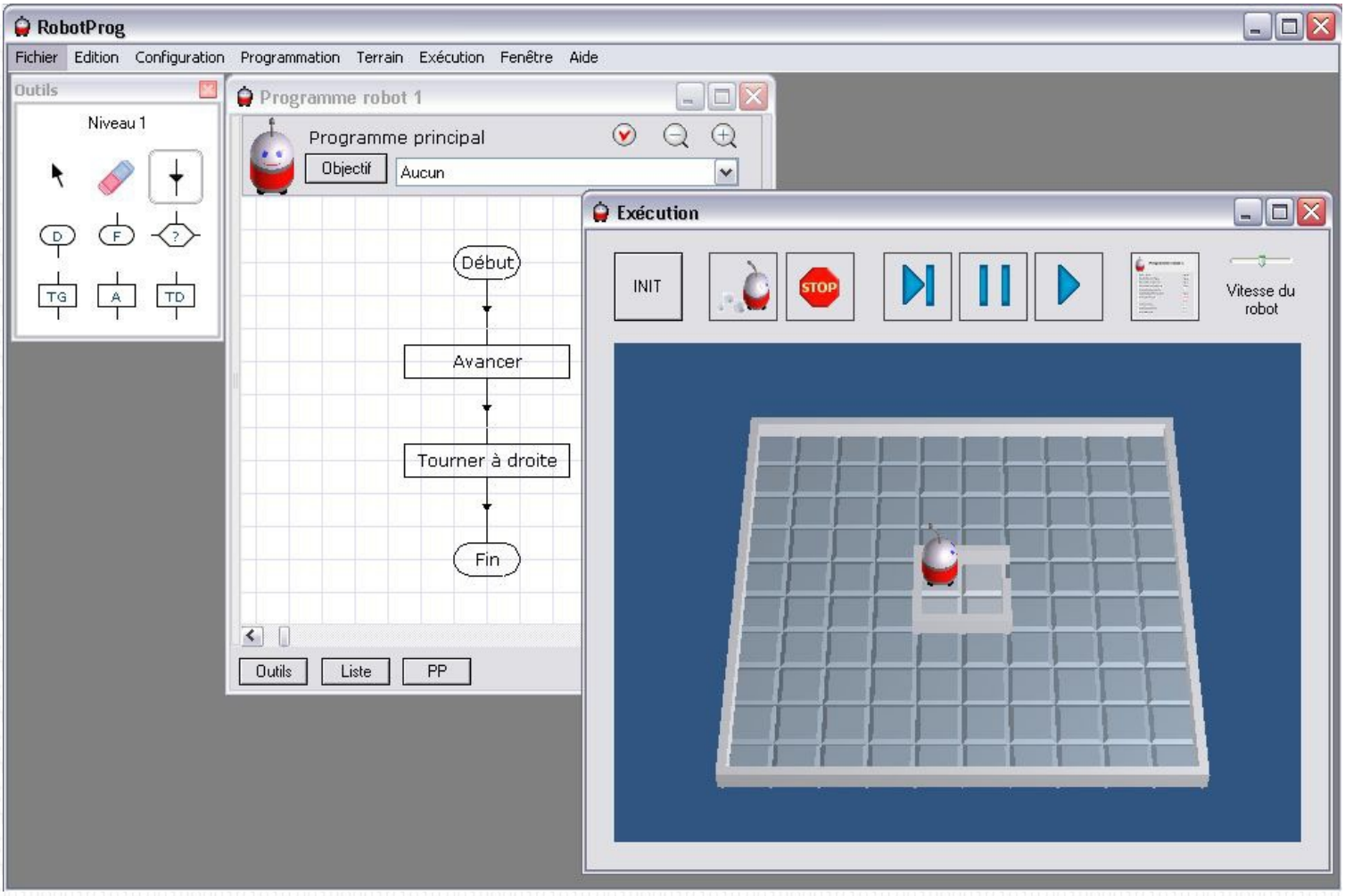


Un organigramme doit contenir un bloc début et un seul pour indiquer où le programme doit commencer, et un ou plusieurs blocs fin.


Reproduisez l'organigramme ci-dessus.

Affichage du terrain du robot

Choisissez le menu **Fenêtre > Exécution**. Le terrain apparaît dans une autre fenêtre avec des boutons permettant de contrôler l'exécution.



Initialisation du programme

Cliquez sur le bouton **INIT** ou bien choisissez le menu **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 devez d'abord corriger l'erreur. Cliquez sur une case pour désigner la position initiale du robot et cliquez sur le robot pour le tourner d'un quart de tour.

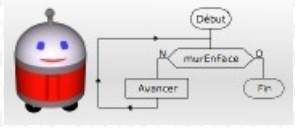
Lancement du programme

Cliquez sur le bouton  ou bien choisissez le menu **Exécution > Lancer**.

Vous ne pourrez pas cliquer sur ce bouton si vous n'avez pas tout d'abord initialisé le programme.

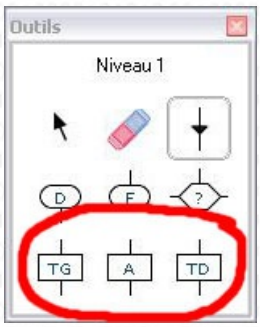
Initialisez et lancez votre programme. Vous remarquerez un rectangle rouge qui se déplace dans la **fenêtre de programme**.
Essayez plusieurs positions de départ du robot. Placez entre autres votre robot contre le mur de droite et dirigé vers la droite.
Sauvegardez ensuite votre programme (cliquez d'abord sur la **fenêtre de programme** puis allez dans le **menu fichier**).
Quittez le programme (**menu fichier**).

Didier Müller, 4.3.05



LEÇON 2 :

LES DÉPLACEMENTS DU ROBOT



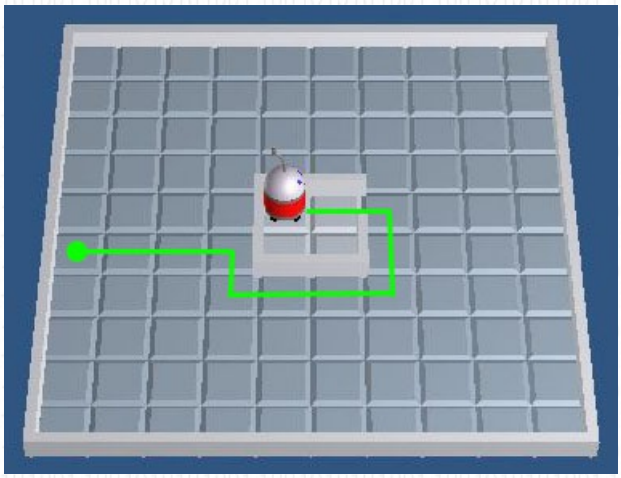
Pour déplacer le robot, vous disposez de trois commandes: *Avancer*, *Tourner à droite*, *Tourner à gauche*. Ces commandes sont écrites dans des blocs de forme rectangulaire disponibles dans la palette d'outils (voir ci-contre).

La commande *Avancer* fait avancer le robot d'une case devant lui.
Comme vous l'avez vu à la leçon 1, si le robot est en face d'un mur quand cette commande est exécutée, il s'écrase contre le mur. C'est une erreur d'exécution, le programme s'arrête.

Les commandes *Tourner à droite* et *Tourner à gauche* font faire un quart de tour au robot vers sa droite ou sa gauche. Le robot reste sur la même case.

Faites parcourir à votre robot le trajet vert ci-dessous. Dans la **fenêtre programme**, vous pouvez sélectionner l'**objectif** *Aller devant un mur*. A la fin de l'exécution, RobotProg vérifiera si l'objectif a été atteint.
Sauvegardez ensuite votre programme.

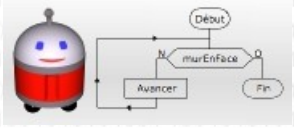
Parcours.bop



Ecrivez un nouveau programme où le robot avance de deux cases, fait demi-tour, et retourne à sa case de départ. Exécutez ce programme.
Dans la **fenêtre programme**, vous pouvez sélectionner l'**objectif** *Faire un demi-tour*.
Sauvegardez ensuite votre programme.

AllerRetour.bop

Didier Müller, 4.3.05

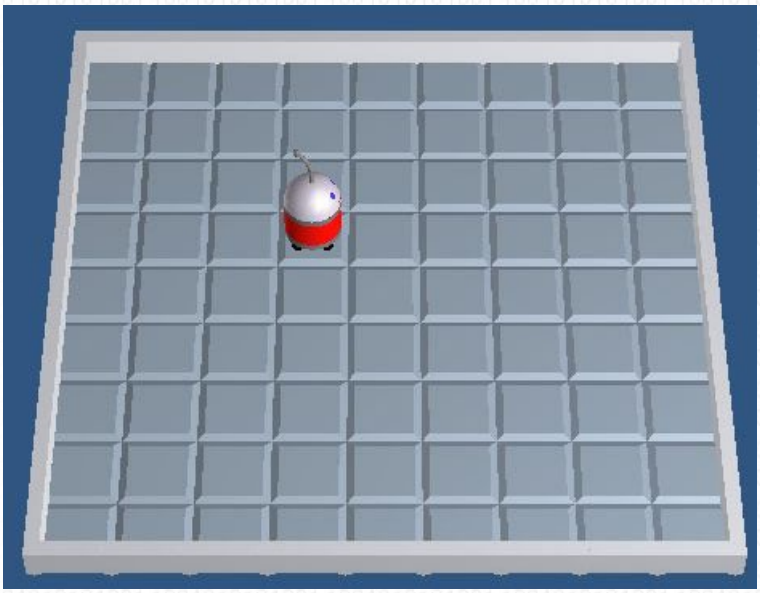


LEÇON 3 :

MODIFICATIONS DU TERRAIN

Le terrain peut être modifié, mais avant l'exécution d'un programme.
Vous pouvez en particulier ajouter ou retirer des murs à l'intérieur du terrain. Les murs encadrant le terrain sont fixes et ne peuvent pas être supprimés. Certains objectifs apparemment simples peuvent être beaucoup plus difficiles à atteindre dans le cas où le terrain comporte beaucoup d'obstacles.

Vous allez maintenant construire un terrain qui sera souvent utilisé plus tard. C'est le terrain représenté ci-dessous.



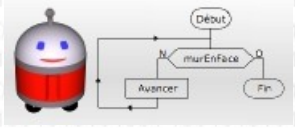
1. Choisissez le menu **Terrain > Modifier**. La fenêtre d'édition de terrain apparaît.
2. Choisissez le menu **Terrain > Nouveau** pour créer un nouveau terrain. La fenêtre des paramètres du terrain apparaît.
3. Choisissez une largeur et une hauteur de 9 cases et validez en cliquant le bouton **OK**.
4. Cliquez sur le bouton **Robots** puis cliquez sur le bouton **+** en dessous: un robot apparaît sur le terrain.
5. Cliquez sur une case pour placer le robot à sa position initiale pour le début de l'exécution.
6. 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.

Avec la fenêtre de modification du terrain, vous pouvez aussi ajouter ou supprimer des murs et des prises d'énergie, enregistrer et ouvrir des fichiers terrain.

Construisez le terrain comme décrit ci-dessus et enregistrez-le sous le nom Terrain9x9SansObstacles.

Terrain9x9SansObstacles.bog

Didier Müller, 4.3.05



LEÇON 4 :

CONDITIONS LOGIQUES ET TESTS

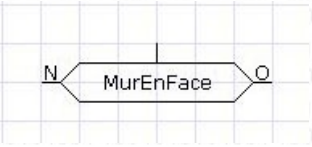
Conditions logiques

Une condition logique est une expression donnant un résultat vrai ou faux. On peut en particulier combiner différents mots-clef avec les opérateurs logiques *Et*, *Ou*, *Non*, par exemple: *(MurEnFace Et MurADroite) Ou Non MurAGauche*.

Vous trouverez une description détaillée dans la documentation, chapitre *langage du robot, rubriques fonctions prédéfinies et expressions*. La documentation s'affiche avec le menu **Aide > Documentation**. Pour l'instant, il vous suffit de connaître trois conditions: *MurEnFace*, *MurADroite*, *MurAGauche*.

Bloc test

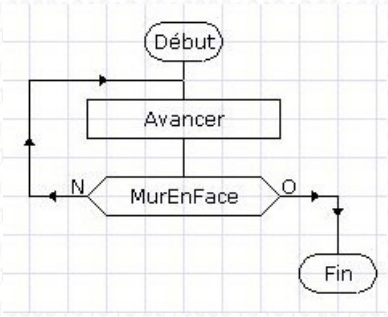
Dans l'organigramme, un bloc test a cette forme:



Pour modifier le texte contenu dans un bloc test, il faut choisir l'outil sélection et faire un double-clic sur le bloc.

La condition logique contenue dans le bloc test est évaluée quand le bloc est exécuté. Le résultat peut être vrai ou faux. Si le résultat est vrai, l'exécution se poursuit après la sortie marquée O (oui ou vrai); si le résultat est faux, l'exécution se poursuit après la sortie marqué N (non ou faux).

Dans l'exemple ci-dessous, on utilise la condition logique *MurEnFace*. C'est un mot-clef du langage du robot qui fournit un résultat de type logique (vrai ou faux) en fonction de la position du robot au moment où elle est évaluée. Si le robot est en face d'un mur, le programme s'arrêtera, sinon le robot avancera et recommencera le test.



Exécution pas à pas et visualisation de l'état courant

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**.

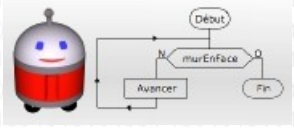
Vous pouvez aussi afficher l'état courant du robot en cliquant sur le bouton **Afficher l'état**. Vous verrez alors apparaître une liste contenant en particulier des mots-clef utilisables dans les tests.

Vous trouverez la liste complète des mots-clefs dans le document accessible par le menu **Aide > Résumé du langage du robot**

Dans cet exercice, vous allez essayer d'atteindre l'**objectif** *Aller dans un coin*(n'importe lequel, mais si le robot se trouve dans un coin au début du programme, il doit aller dans un autre coin).
Vous utiliserez le terrain Terrain9x9SansObstacles défini à la leçon 3.
Attention! Votre programme doit fonctionner sans erreur d'exécution, quelles que soient la position et la direction initiale du robot. A vous de trouver les situations initiales à problèmes.

AllerCoin.bop

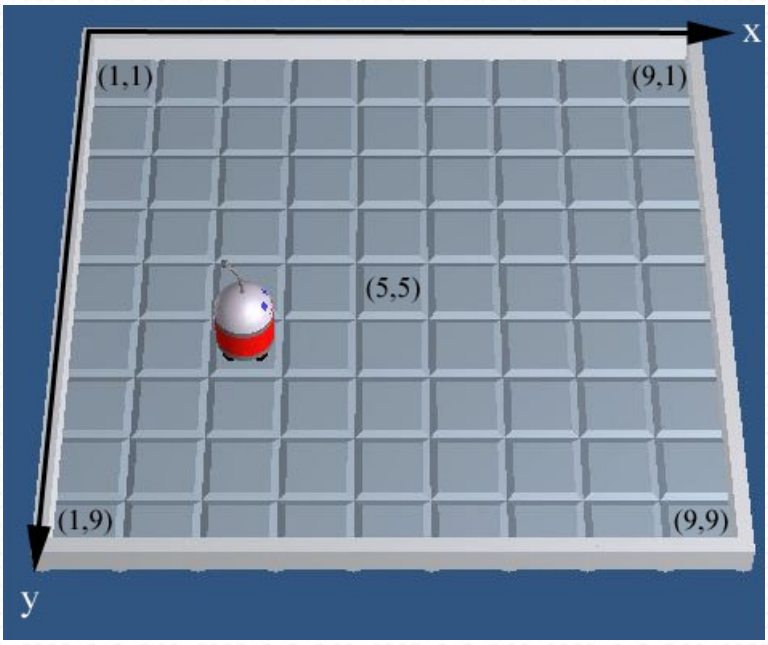
Didier Müller, 4.3.05



LEÇON 5:

LES DÉPLACEMENTS DU ROBOT

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



Position du robot

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* = 3 et *yRobot* = 6.

Direction du robot

La direction suivant laquelle le robot est orienté est donnée par les deux mots-clef *dxRobot* et *dyRobot*. Les valeurs de *dxRobot* et *dyRobot* correspondent à la variation 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

Dans l'exemple ci-dessus, *dxRobot* vaut 1 et *dyRobot* vaut 0.

Comme on peut le remarquer, *dxRobot* et *dyRobot* n'ont comme valeurs possibles que 0, 1 et -1. L'une des deux valeurs est nulle et l'autre non nulle.

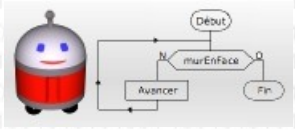
Ecrivez un programme qui oriente le robot vers la gauche, quelle que soit son orientation initiale.
Sauvegardez ce programme sous le nom SOrienterAGauche.

SOrienterAGauche.bop

Vous utiliserez pour ce deuxième exercice le terrain Terrain9x9SansObstacles défini à la leçon 3.
Ecrivez 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.

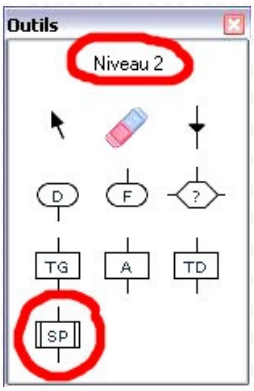
AllerColonneCentrale.bop

Didier Müller, 4.3.05



LEÇON 6:

CRÉATION D'UN SOUS-PROGRAMME



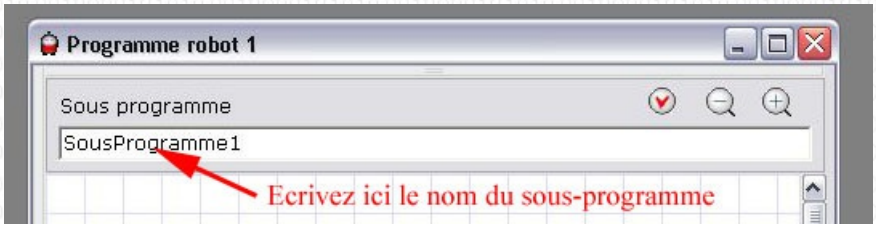
Pourquoi des sous-programmes ?

En général, 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 simples et donc plus faciles à résoudre. C'est un des principes de base de la programmation: diviser pour régner. Les sous-programmes permettent ainsi de décomposer un programme en plusieurs parties. Par ailleurs, un même sous-programme peut être utilisé à plusieurs reprises, ceci évitant de réécrire plusieurs fois le même code.

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**.

Créer un sous-programme

Avant de créer un sous-programme, vous devez avoir un nouveau programme vide. Si ce n'est pas le cas, fermez la **fenêtre Programme** et choisissez le menu **Fichier > Nouveau programme**.
Pour créer un sous-programme, choisissez le menu **Programmation > Nouveau sous-programme**. Le sous-programme est maintenant affiché dans la **fenêtre programme** à la place du programme principal.
Chaque sous-programme est identifié par un nom que vous lui attribuez. Par défaut, quand le sous-programme est créé, il reçoit le nom SousProgramme1. Remplacez ce nom par SeTournerVersLaGauche.



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-clef du langage du robot. Les lettres majuscules et minuscules sont considérées comme étant identiques.

Copiez-collez l'organigramme SOrienterAGauche que vous avez construit à la leçon 5 dans le sous-programme SeTournerVersLaGauche.

Affichage du programme principal et des sous-programmes

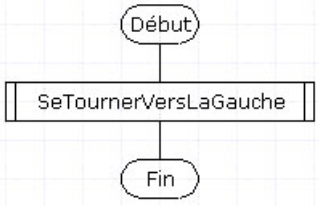
- Cliquez sur le bouton **Liste** en bas de la fenêtre programme pour afficher (ou masquer) la liste des sous-programme. La liste apparaît à gauche de la fenêtre. Le premier élément de cette liste est le programme principal. Son nom est le nom de la fenêtre (c'est aussi le nom du fichier associé si le programme a été enregistré). Les éléments suivants sont les noms des sous-programmes crées. 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** 

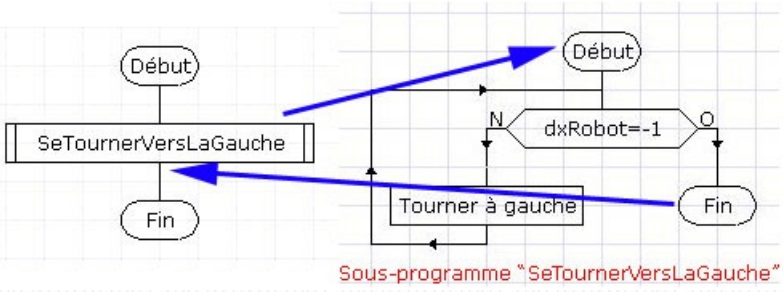
.

Pour tester l'exécution du sous-programme SeTournerVersLaGauche, vous aller écrire le programme principal contenant l'appel du sous-programme. Affichez le programme principal et construisez l'organigramme suivant:



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"

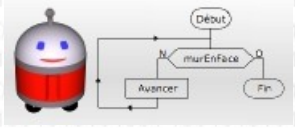
Ajoutez et faites exécuter un nouveau sous-programme de nom SeTournerVersLaDroite.

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

Ecrivez un programme qui déplace le robot sur la case centrale du terrain, de coordonnées (5,5). Vous utiliserez le terrain Terrain9x9SansObstacles défini à la leçon 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.

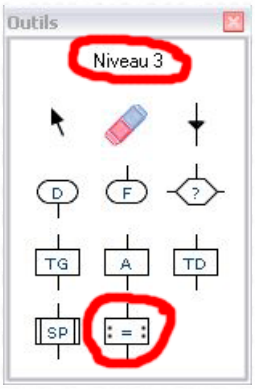
CaseCentrale.bop

Didier Müller, 4.3.05



LEÇON 7:

VARIABLES ET EXPRESSIONS



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. Ce nom permet à l'ordinateur de retrouver la donnée associée dans sa mémoire. Le nom correspond à l'adresse de la donnée dans la mémoire. Un nom de variable 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-clefs 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-clef du langage.

La valeur d'une variable est le contenu de la zone de mémoire référencée par le nom de la 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.

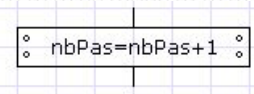
Expressions numériques

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-clefs, 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 * *DistanceMur* donnera comme résultat 8.

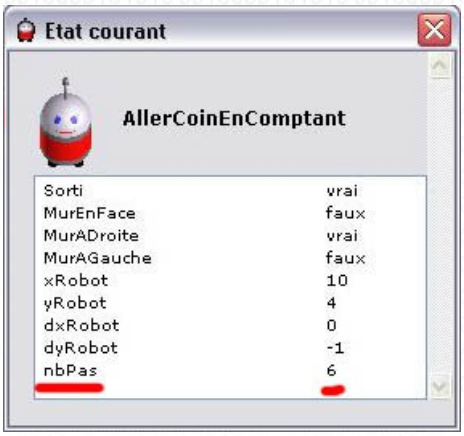
Affectation d'une valeur à une variable

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 = est utilisé dans de nombreux langages de programmation pour l'instruction d'affectation. Mais attention! Ce 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.
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 incrémentation.
Si on reprend l'image des boîtes, cela donne ceci: 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.
Dans un organigramme, l'instruction d'affectation s'écrit dans un bloc affectation:

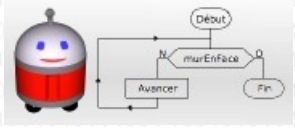


L'affectation est utilisable à partir du niveau 3: choisissez en conséquence un niveau supérieur ou égal à 3. Utilisez le terrain Terrain9x9SansObstacles.
Reprenez votre programme de la leçon 4 qui faisait aller votre robot dans un coin, et modifiez-le pour que le robot compte le nombre de ses pas depuis sa position initiale jusqu'à sa position finale. Le principe est simple: chaque fois que le robot avancera, on incrémentera la variable *nbPas*.
Sauvegardez ce programme sous le nom AllerCoinEnComptant.
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 *nbPas* pendant les déplacements du robot.



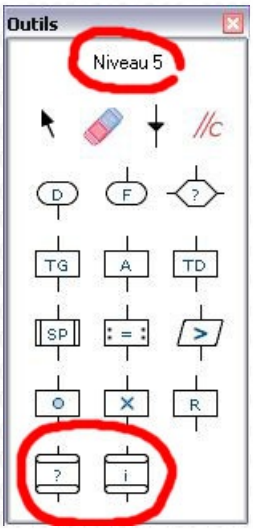
AllerCoinEnComptant.bop

Didier Müller, 8.3.05



LEÇON 8:

BOUCLE POUR



Utilité des boucles

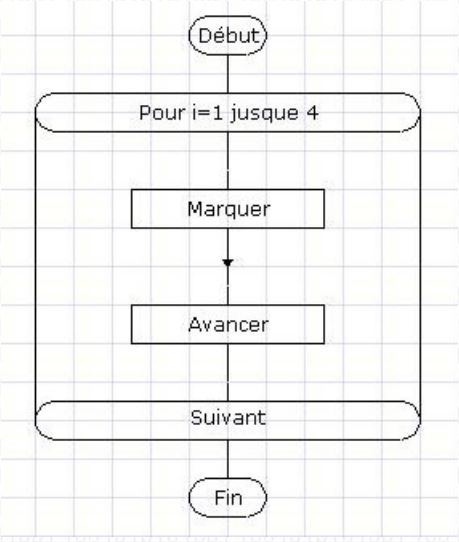
Dans cette leçon, vous allez faire dessiner par le robot un carré de 5 cases de coté. Pour marquer une case d'un point rouge, le robot utilisera l'instruction *Marquer*  .

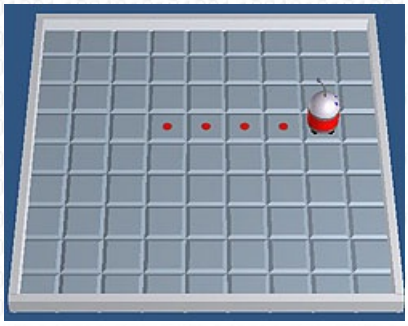
Le périmètre du carré est formé de 16 cases. Au total, le robot devra avancer et marquer 16 fois et il devra tourner 4 fois à chaque coin: il faudrait donc au moins 36 (16+16+4) blocs pour faire cela.... C'est long, surtout pour des actions qui se répètent. Et si on veut dessiner un carré de 6 cases de coté, tout est à refaire!
Il existe une solution adaptée à des actions répétitives: les boucles qui permettent de répéter un nombre de fois donné une même partie de programme.
Une boucle est représentée dans l'organigramme par un bloc dans lequel on peut insérer d'autres blocs correspondant à la partie de programme à répéter.

Boucle Pour 

Recopiez le sous-programme ci-dessous qui fait marquer au robot 4 cases alignées.

1. choisissez le menu **Programmation > Nouveau sous-programme**
2. donnez au sous-programme le nom DessinerUnCote
3. construisez l'organigramme du sous-programme; utilisez la boucle de droite (Boucle Pour)
4. vérifiez le résultat obtenu





L'en-tête de la boucle contient l'instruction: *Pour i = 1 jusque* 4.
Pour et *jusque* sont deux mots-clefs obligatoires dans cette instruction.
i est une variable du programme, utilisée ici comme variable de la boucle.
1 est la valeur initiale de la variable et 4 est sa valeur finale; ces valeurs peuvent être des nombres entiers ou des expressions numériques.
Suivant est un mot-clef qui indique la fin de la boucle.

Déroulement de l'exécution :

1. la première fois que la boucle est exécutée, la variable de boucle reçoit la valeur initiale: *i* a au départ la valeur 1
2. ensuite le corps de la boucle (organigramme contenu dans la boucle) est exécuté: le robot dessine une marque sur la case qu'il occupe, puis avance d'une case
3. ensuite l'instruction *Suivant* est exécutée, elle a pour effet de ramener l'exécution à l'en-tête de la boucle
4. la deuxième fois que l'en-tête est exécuté, la variable de la boucle est augmentée par défaut d'une unité: la valeur de *i* devient 2 (On peut augmenter la variable de plusieurs unités grâce à l'instruction *pas*. Par exemple, avec l'en-tête *Pour j*=5 *jusque* 25 *pas* 10, la variable *j* aura successivement les valeurs 5, 15 et 25)
5. ensuite le corps de la boucle est à nouveau exécuté, puis l'instruction *Suivant* et ainsi de suite jusqu'à ce que la valeur de la variable dépasse la valeur finale. La boucle sera terminée quand *i* aura la valeur 5: le robot aura alors marqué 4 cases. L'exécution du programme se poursuit au bloc qui suit la boucle (ici le bloc de fin).

Les boucles sont disponibles à partir du niveau 5: choisissez en conséquence un niveau supérieur ou égal à 5. Utilisez le terrain Terrain9x9SansObstacles. Sélectionnez l'**objectif** *Dessiner un carré sur le sol*.

Complétez l'organigramme de cette leçon pour faire dessiner un carré de 5 cases de côtés:

1. avec le robot orienté vers la droite et positionné sur la case (1,1). Pour mémoire, les coordonnées sont expliquées à la leçon 5.
2. quelles que soient la position et l'orientation initiales du robot.

DessinerCarre.bop

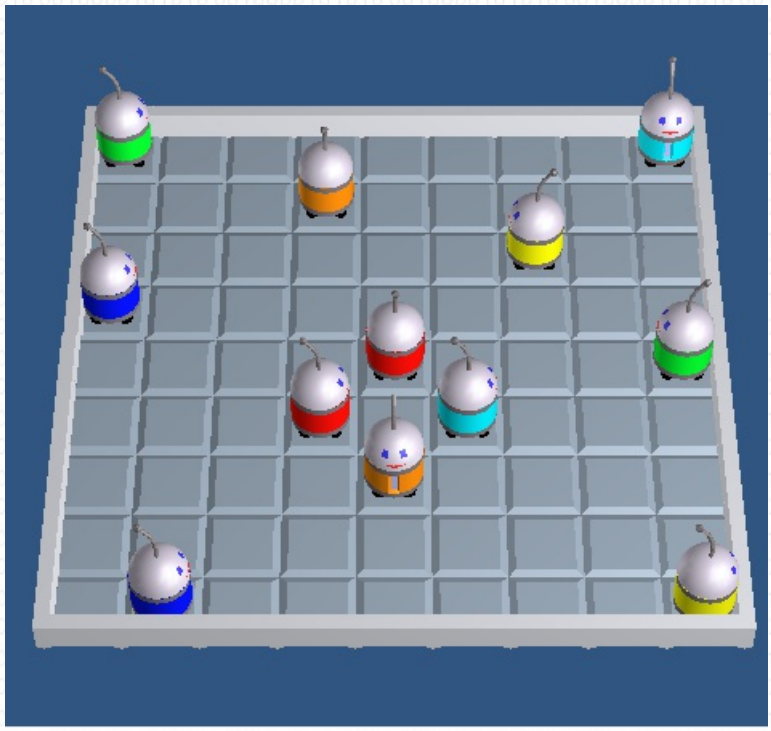
COMPÉTITION

Dans ce projet, le robot devra marquer les 81 cases d'un terrain 9x9 sans obstacles le plus vite possible. La position et l'orientation initiales du robot sont aléatoires.
Vous compterez le nombre de mouvements (à l'aide d'un compteur dans le programme) ainsi: 1 mouvement = tourner à droite, tourner à gauche ou avancer. Les autres instructions ne comptent pas pour un mouvement.
Vous devrez rendre un rapport, le 21 octobre 2015, qui sera noté.

P.S. Le plus rapide d'entre vous recevra un petit quelque chose...

Tests

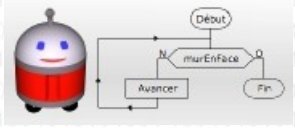
Pour tester votre programme, utilisez les positions initiales données par les 12 robots ci-dessous. Comptez pour chaque robot le nombre total de mouvements (avancer ou tourner d'un quart de tour) pour colorer tout le damier.



Structure du rapport (suggestion)

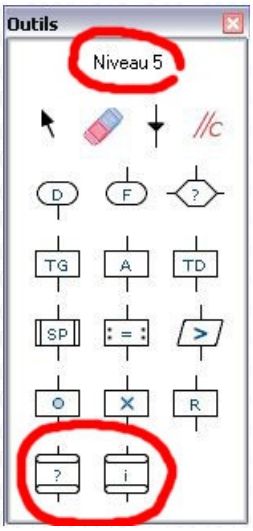
1. Enoncé (voir ci-dessus).
2. Analyse du problème
    1. Stratégie(s) générale(s) expliquées en français. Des schémas sont aussi bienvenus.
    2. Décomposition du problème. Descriptions des sous-programmes.
    3. Pire des cas.
    4. Meilleur des cas.
3. Sous-programmes et programme principal (programme à envoyer par email).
   Je dois pouvoir comprendre le programme aisément, donc il faut choisir de bons noms pour les sous-programmes. Je conseille de faire suffisamment de sous-programmes pour ne pas avoir de problèmes de place sur la feuille de RobotProg.
4. Analyse des résultats (faire partir le robot d'un coin, d'un bord, de la case centrale, d'une case quelconque,...)

Didier Müller, 1.9.15



LEÇON 9:

BOUCLE TANT QUE



Boucle Tant Que 

A la leçon précédente, nous avons vu la boucle *Pour*, qui permet de répéter un certain nombre de fois un sous-programme. Nous connaissions alors *a priori* le nombre de fois que le sous-programme devait être exécuté. Il n'en est pas toujours ainsi. Il arrive que l'on sache la condition de sortie de la boucle, mais pas le nombre d'itérations. Dans ce cas, on utilisera une boucle *TantQue*.

Dans cette leçon, nous allons écrire un sous-programme AllerALaCaseXY qui déplace le robot vers une case de coordonnées (x,y), *x* et *y* étant deux variables dont les valeurs auront été affectées avant d'appeler le sous-programme. Pour faire aller le robot à la case (x,y), on le déplacera vers la position (x,y) tant qu'il ne sera pas arrivé à destination. On utilisera pour cela une boucle *TantQue* et un autre sous-programme AllerVersXY.

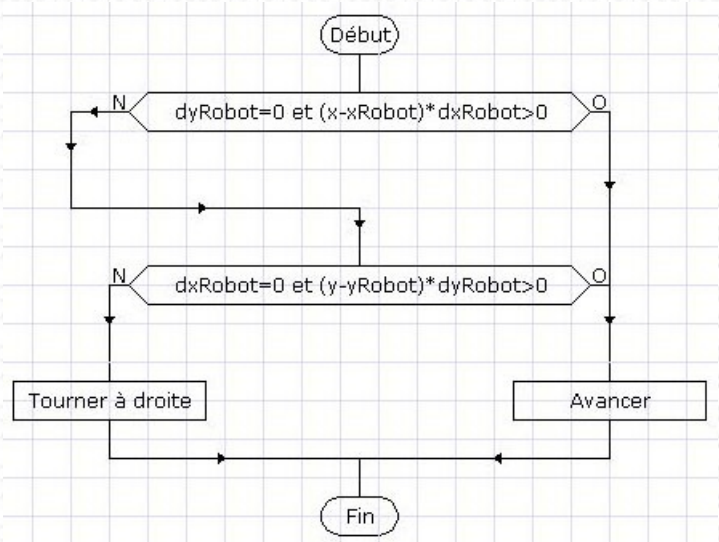
Sous-programme AllerVersXY

Ce sous-programme permettra:

1. de le faire tourner sur lui-même s'il est mal orienté
2. de faire avancer le robot d'une case s'il est bien orienté.

Les boucles sont utilisables à partir du niveau 5: choisissez en conséquence un niveau supérieur ou égal à 5. Utilisez le terrain Terrain9x9SansObstacles.

- Créez un nouveau sous-programme auquel vous donnerez le nom AllerVersXY.
- Construisez l'organigramme ci-dessous.
- Essayez de comprendre la signification de la condition logique *dyRobot*=0 *et* (*x-xRobot*)**dxRobot* > 0.
- Vérifiez que l'organigramme est correct en cliquant le bouton **Vérifier organigramme**  .



Conditions logiques utilisées pour tester la direction du robot

dyRobot=0 *et* (*x-xRobot*)**dxRobot* > 0:

- la condition *dyRobot* = 0 est vraie si le robot est orienté suivant l'axe des x; *dxRobot* vaut alors 1 pour une orientation vers la droite et -1 pour une orientation vers la gauche (voir leçon 5);
- la valeur *x-xRobot* représente le parcours à effectuer suivant l'axe des x;
- le robot est orienté vers la colonne x à atteindre si sa direction *dxRobot* et le parcours à effectuer sont de même signe;
- le produit des deux doit alors être positif, ce qui est traduit par la condition (*x-xRobot*)**dxRobot*> 0.

De la même façon, la condition *dxRobot*=0 *et* (*y-yRobot*)**dyRobot* > 0 permet de tester l'orientation du robot suivant l'axe des y.

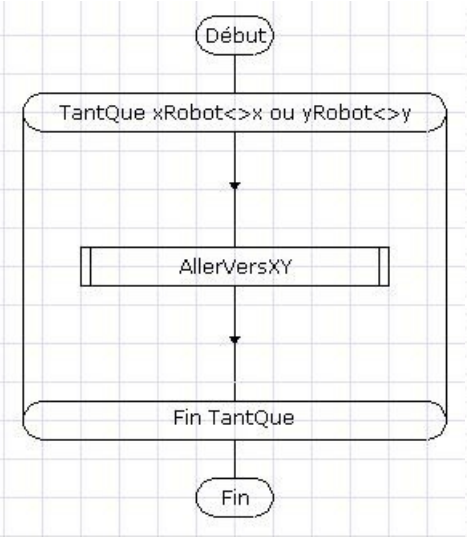
Sous-programme AllerALaCaseXY

La boucle *TantQue* est représentée par un bloc boucle de même forme que la boucle Pour (voir leçon 8). L'en-tête de la boucle contient l'instruction *TantQue* qui est de la forme:

TantQue condition logique

Si la condition logique donne le résultat Vrai, l'organigramme présent dans le corps de la boucle est exécuté. Puis l'instruction *Fin TantQue* ramène l'exécution à l'en-tête de la boucle, la condition est à nouveau évaluée, etc.
Si la condtion logique donne le résultat Faux, le corps de la boucle n'est pas exécuté et l'exécution se poursuit au bloc suivant la boucle.

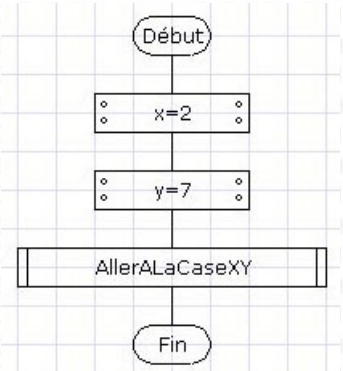
- Créez un nouveau sous-programme auquel vous donnerez le nom AllerALaCaseXY
- Construisez l'organigramme ci-dessous
- Vérifiez que l'organigramme est correct en cliquant le bouton **Vérifier organigramme** 



Dans le sous-programme AllerALaCaseXY, le corps de la boucle, c'est-à-dire l'appel du sous-programme AllerVersXY, sera exécuté tant que la condition *xRobot* <> *x ou yRobot* <> *y* sera vraie; *xRobot* <> *x* est une expression logique qui compare *xRobot* et *x*, le résultat est vrai si *xRobot* est différent de *x*; de même, *yRobot* <> *y* est vraie si *yRobot* est différent de *y*; en conséquence la condition *xRobot* <> *x* ou *yRobot* <> *y* sera vraie si le robot n'est pas dans la case (x,y).

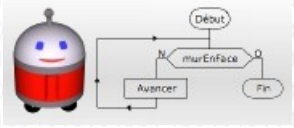
Programme principal

- Entrez l'organigramme ci-dessous dans le programme principal.
- Lancez l'exécution et vérifiez que la robot arrive bien dans la case indiquée par les variables *x* et *y*.
- Changez la position de départ du robot et relancez le programme.
- Changez les valeurs de *x* et *y* et relancez le programme.



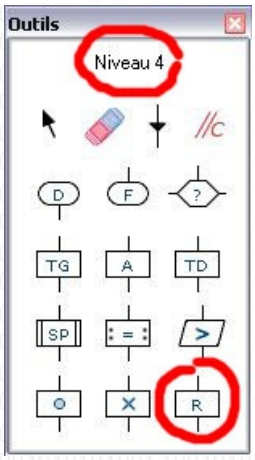
AllerSurUneCase.bop

Didier Müller, 30.4.05



LEÇON 10:

ENERGIE DU ROBOT



Etat des batteries

Au début du programme, le robot a une énergie de 1000 (cette énergie initiale peut être modifiée quand on modifie un terrain, en cliquant sur la case **Robots**). Cette énergie diminue avec le temps. Le mot-clef *Energie* permet de connaître le niveau des batteries. Si le robot n'a plus d'énergie, le programme s'arrête avec un message d'erreur.

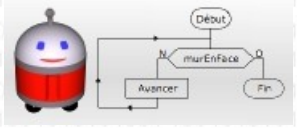
Prise d'énergie

On peut placer sur le terrain une ou plusieurs prises d'énergie.
La condition logique *RobotSurUnePrise* est vraie si le robot sur trouve une prise d'énergie.
La commande *Recharger* (symbole entouré ci-contre) recharge complètement les batteries instantanément, si le robot se trouve sur une prise d'énergie, évidemment.

L'énergie est utilisable à partir du niveau 4: choisissez le niveau 5.
Créez un nouveau terrain: reprenez le terrain Terrain9x9SansObstacles auquel vous ajouterez une prise d'énergie sur la case centrale (5;5). Appelez ce nouveau terrain Terrain9x9SansObstaclesAvecPrise.
Reprenez le programme principal que vous avez écrit à la leçon 9.
Modifiez-le pour que le robot fasse des aller-retour entre les cases (2;7) et (8;3) tant que son énergie est supérieure à 300. Quand son énergie tombera en dessous de 300, faites que le robot aille recharger ses batteries sur la case centrale, puis il retourne sur la case où il avait abandonné son circuit et recommence ses allers-retours. La troisième fois que le robot va sur la prise, faites s'arrêter le programme.
Pendant l'exécution du programme, affichez l'état  pour voir comment évolue l'énergie.

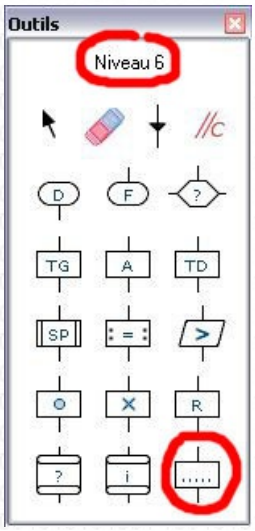
Terrain9x9SansObstaclesAvecPrise.bog, Energie.bop

Didier Müller, 12.6.05



LEÇON 11:

UTILISATION DU BALLON



Programmation du ballon

Le ballon apparaît sur le terrain uniquement quand le programme utilise un mot-clef relatif au ballon. Au début de l'exécution, le ballon est placé sur une case déterminée au hasard. La position du ballon sur le terrain est donnée par les deux mots-clef *xBallon* et *yBallon*. Ainsi, pour faire apparaître le ballon, il suffit d'écrire *x = xBallon* et *y = yBallon*.

On peut tester si le robot est dans la même case que le ballon en utilisant la condition : *xRobot = xBallon* et *yRobot = yBallon*. La condition logique *BallonSurLeSol* est vraie si le ballon est... sur le sol.

Les commandes du robot relatives au ballon sont : *PrendreBallon*, *PoserBallon* et *LancerBallon*. Ces commandes doivent être écrites dans des blocs de commande éditables (symbole entouré ci-contre).
PrendreBallon : le robot prend le ballon s'il se trouve sur la même case que le ballon.
PoserBallon : le robot pose le ballon sur la case qu'il occupe.
LancerBallon : le robot lance le ballon trois cases devant lui. Si le ballon sort du terrain, il est relancé automatiquement sur une case déterminée au hasard.

Un robot joue au ballon

Fermez d'abord la fenêtre **Programme Robot 1**. La raison est expliquée au chapitre 12.
L'utilisation du ballon n'est possible qu'au niveau 6. Choisissez ce niveau maintenant.
Utilisez le terrain Terrain9x9SansObstacles.
Ouvrez le programme principal que vous avez écrit à la leçon 9 et modifiez-le pour que le robot trouve le ballon et le lance. Appelez ce programme LancerUnBallon.

LancerUnBallon.bop

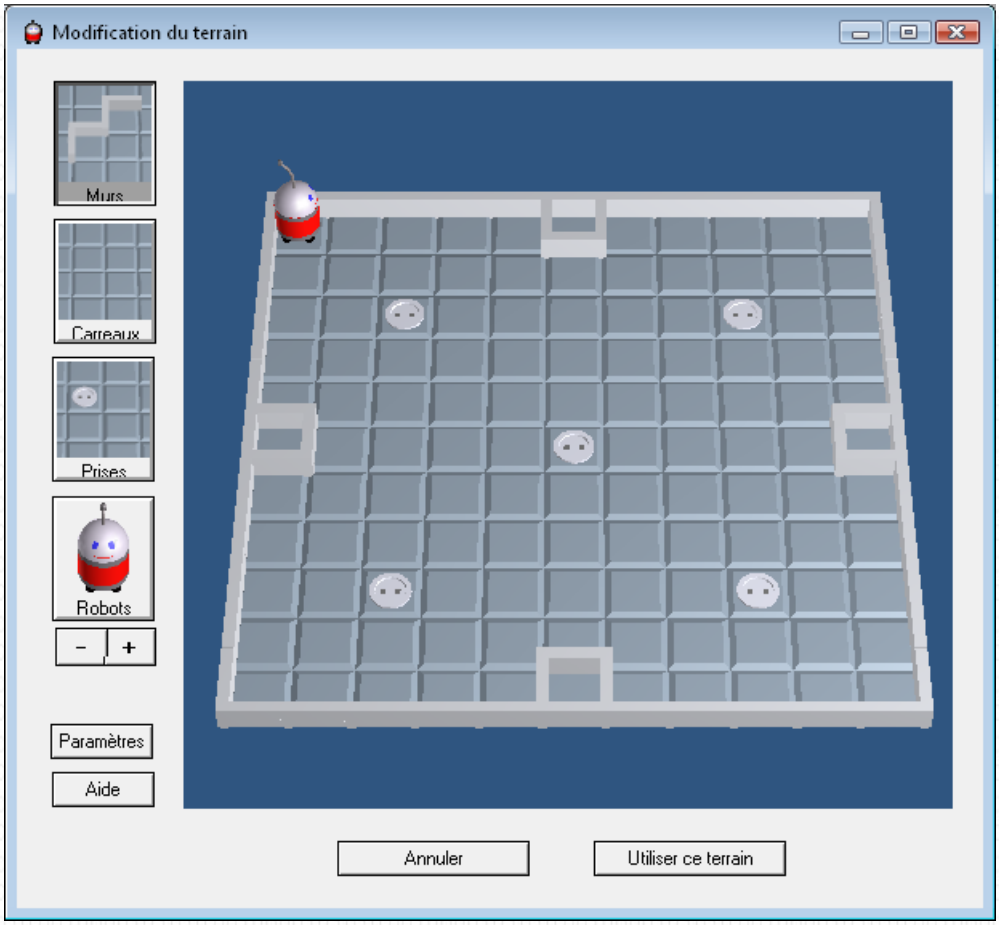
Au niveau 6, il est possible aussi de faire jouer les robots à différents jeux. Ici les robots vont jouer au basket.
- choisissez le menu **Configuration > Choisir un jeu**. La fenêtre de choix de jeu apparaît.
- sélectionnez le jeu de **basket**
- consultez les règles de ce jeu en cliquant sur le bouton **Règles du jeu**
- validez le choix en cliquant sur le bouton **OK**

Vous constaterez que l'**objectif** du programme est devenu "Lancer le ballon dans le panier": c'est l'objectif du jeu et il n'est plus modifiable tant qu'un jeu est en cours.
Le terrain est aussi fixé et non modifiable, vous pouvez voir le terrain de basket en affichant la fenêtre exécution. Le panier est la case centrale entourée de murs.

Modifiez le programme LancerUnBallon pour que le robot lance le ballon dans le panier. Le robot ne doit pas s'écraser contre les murs du panier!
Appelez ce programme JouerBasket.

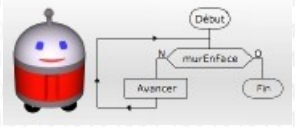
JouerBasket.bop

Pour l'exercice suivant, utilisez le terrain ci-dessous. Vous appelerez votre programme Jordan.



- Un ballon est lancé au hasard sur le terrain.
- Le robot va le chercher et le lance dans le panier le plus proche, en faisant le moins de mouvements possible (bien choisir la case d'où lancer le ballon).
- En cas de besoin, le robot doit recharger ses batteries. Au départ, le niveau de la batterie sera de 350

Didier Müller, 12.6.05



LEÇON 12:

PROGRAMMER PLUSIEURS ROBOTS

Au niveau 6, on peut faire exécuter simultanément les programmes de plusieurs robots sur le même terrain. Chacun des robots est associé à une fenêtre de programme.
Quand le niveau est inférieur à 6, l'ouverture d'un programme ou la création d'un nouveau programme entraîne la fermeture du programme courant: il ne peut y avoir qu'un seul programme utilisé à la fois. Au niveau 6, quand vous ouvrez un autre programme, les autres précédemment ouverts le restent.

Projet

L'ensemble des programmes des robots constituent un projet. Vous pouvez voir la liste de ces programmes dans la fenêtre du projet avec le menu **Fenêtre > Projet**. Cette fenêtre vous permet aussi d'afficher les programmes masqués ou de supprimer des programmes du projet. Au niveau 6, quand une fenêtre de programme est fermée, le programme proprement dit reste dans le projet même s'il n'est plus affiché.

Jeu de basket à plusieurs robots

- fermez d'abord la fenêtre **Programme Robot 1**.
- choisissez le niveau 6.
- choisissez le menu **Configuration > Choisir un jeu**. La fenêtre de choix de jeu apparaît.
- sélectionnez le jeu de **basket**
- consultez les règles de ce jeu en cliquant sur le bouton **Règles du jeu**.
- validez le choix en cliquant sur le bouton **OK**

Pour essayer de jouer à plusieurs robots, vous pouvez dupliquer le fichier JouerBasket et ouvrir le fichier dupliqué: vous aurez deux robots joueurs suivant le même programme.
Mais il y a encore des problèmes! En effet, les robots peuvent s'écraser les uns contre les autres, il faut donc tester la présence d'un autre robot avec le mot-clef *CaseDevantOccupée*. D'autre part, quand un robot a pris le ballon, le ballon change de place avec le robot.

Régle avancée

- si la case devant le robot qui a le ballon est occupée par un autre robot, le premier doit immédiatement lancer le ballon.

Didier Müller, 13.6.05