Découvrir

MSWLogo

Un langage informatique simple et puissant

pour aborder la programmation en s'amusant.

Décembre 2003

Par J N Saillet

j-n.saillet@tele2.fr

Vous avez peut-être déjà entendu parler de Logo, ce langage créé dans les années 80 pour initier les enfants à la programmation informatique ?

MSWLogo est issu directement de Logo, mais il est beaucoup plus puissant et surtout, il fonctionne dans l'environnement Windows et peut être utilisé sous Linux avec un émulateur comme wine. De plus, c'est un logiciel libre et gratuit sous licence GNU GPL.

Ce document a pour but de vous faire découvrir ce fabuleux programme, de vous montrer qu'il n'a absolument pas vieilli et qu'il peut être un excellent point d'entrée pour une première initiation à la programmation informatique à l'école élémentaire, au collège voire au lycée.

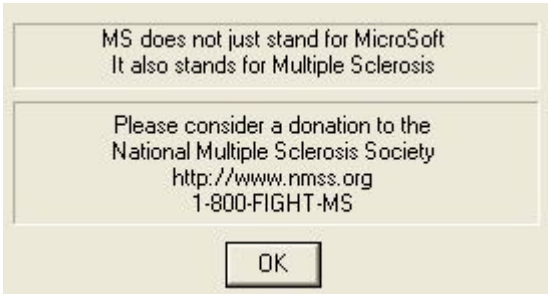
Table des matières :

Installation de MSWLogo :

L'installation de MSWLogo ne pose aucun problème particulier, aussi je ne m'étendrai pas sur ce sujet.
Vous trouverez MSWLogo à cette adresse : http://www.softronix.com/logo.html
C'est un logiciel sous licence GNU GPL. Téléchargez l'archive (mswlogo65.exe) puis lancez l'exécution du programme d'installation en faisant ou double clic du bouton gauche sur l'icône du fichier mswlogo65.exe.
Par défaut, le programme s'installe dans le répertoire C:\Program Files. Si vous désirez l'installer à un autre endroit, vous devrez créer le répertoire destiné à l'accueillir avant de commencer l'installation.
Le programme d'installation place un raccourci vers le programme dans le menu Démarrer et sur le bureau de Windows.

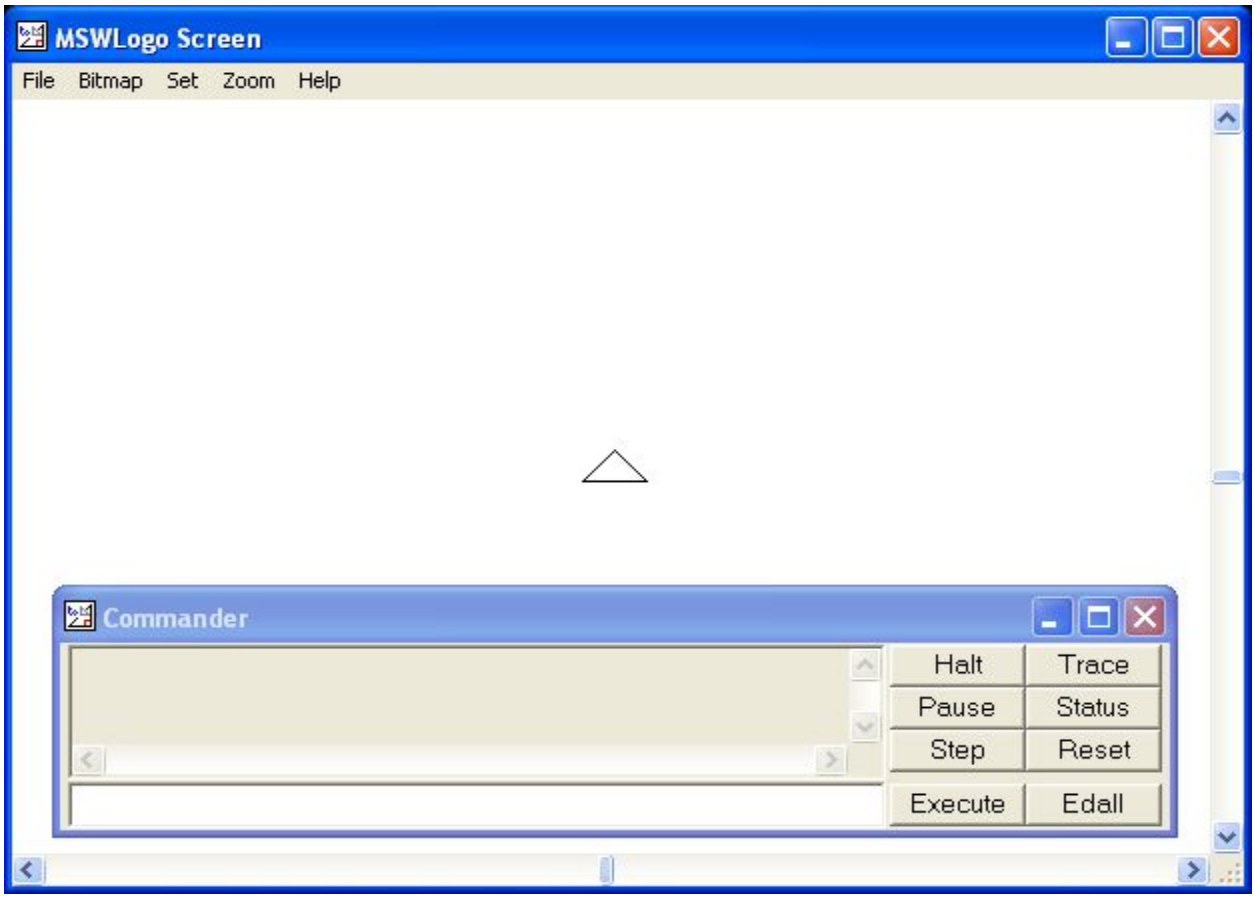
Premier démarrage du programme :

Lors du tout premier premier démarrage, cette fenêtre s'affiche :



Elle n'apparaît que lors du premier lancement du programme et rappelle l'engagement personnel de l'auteur.
Puis, une seconde fenêtre s'affiche :

Celle-ci n'est affichée que des 10 premières utilisations du programme.
Cliquez sur OK. L'écran de MSWLogo apparaît, il est composé de deux fenêtres superposées intitulées respectivement « **MSWLogo Screen** » et « **Commander** » :



MSWLogo Screen :

C'est **l'écran graphique** dans lequel évolue la **tortue** Logo.

D'où vient ce nom ?
A l'origine, le langage Logo servait à piloter un petit robot qui se déplaçait au sol et dont la forme faisait penser à une tortue. Ce robot était équipé d'un crayon qu'il pouvait baisser et lever à volonté et en combinant les déplacements au sol et la position du crayon, il pouvait tracer des graphismes sur une feuille de papier. Par la suite, pour des raisons de coût, on a utilisé l'écran de l'ordinateur pour représenter ce robot et le plan sur lequel il se déplace mais le nom tortue est resté.

La tortue est donc représentée par le triangle isocèle placé au centre de la fenêtre. Cette fenêtre est censée représenter la feuille de papier blanc sur laquelle la tortue va évoluer et tracer des graphismes. On peut également y placer des composants Windows tels que des boîtes de dialogue, des boutons, des listes, des ascenseurs des cases à cocher etc... pour rendre un programme Logo encore plus interactif.

Cette fenêtre comporte une barre de menu en anglais que nous allons examiner de plus près :

Pour cela, cliquons sur le menu **File** :

New permet d'effacer les procédures et les variables contenues dans la mémoire de Logo pour commencer un nouveau projet.

Un peu de vocabulaire :

***Une procédure** est une groupe d'instructions qui permettent de faire exécuter une tache simple et bien précise à l'ordinateur. L'ensemble des procédures qui permettent d'obtenir le résultat final attendu, une tâche généralement très complexe, est le **programme**.*
Un programme est donc constitué de plusieurs dizaines (voire centaines) de procédures.
***Une variable** permet de stocker une valeur dans la mémoire de l'ordinateur. Cette valeur est utilisée par le programme pour réaliser la tâche pour laquelle il a été conçu. Nous verrons plus loin comment définir et utiliser des variables.*

Load... permet de charger en mémoire un programme présent sur le disque.

Save permet de sauvegarder sur le disque un programme qui a été chargé en mémoire auparavant. Toutes les procédures et les variables contenues dans la mémoire sont enregistrées sur le disque.

SaveAs... sauvegarde aussi le contenu de la mémoire mais en permettant à l'utilisateur de choisir le nom du fichier.

Edit... permet d'éditer une ou toutes les procédures du programme dans l'éditeur de texte de MSWLogo.

Erase... permet d'effacer une ou plusieurs procédures du programme.

Exit enfin, permet de quitter MSWLogo.

Cliquons maintenant sur le menu **Bitmap** :

New efface le tracé de la tortue sur l'écran graphique.

Load... permet de charger une image bitmap sur l'écran graphique.

Save permet de sauvegarder une image qui a été chargée en mémoire auparavant (après modification par exemple).

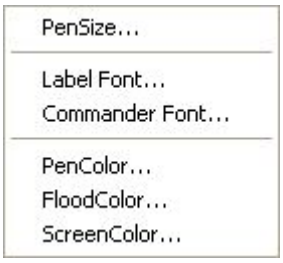
SaveAs... permet de sauvegarder une image en laissant le choix du nom du fichier à l'utilisateur.

Print... lance l'impression de l'écran graphique sur l'imprimante.

Printer setup permet de paramétrer l'impression de la feuille (format de la feuille de papier, marge etc...)

Active Area... définit la zone de l'écran graphique qui sera imprimée sur la feuille.

Maintenant, cliquons sur le menu **Set** :



PenSize... permet de définir la largeur de trait du crayon utilisé par la tortue pour tracer des graphismes à l'écran.

Label Font... permet de définir la police de caractères qui sera utilisée par la tortue pour écrire du texte sur l'écran graphique.

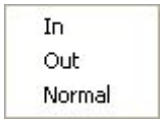
Commander Font... permet de choisir la police de caractères qui sera utilisée pour afficher le texte dans la fenêtre « Commander ».

PenColor permet de définir la couleur du crayon.

FloodColor... permet de définir la couleur de remplissage des figures.

Et enfin **ScreenColor...** pour définir la couleur du fond de l'écran graphique.

Voyons maintenant le menu **Zoom** :



In active le grossissement du contenu de l'écran graphique. La taille de la tortue et de tout ce qui apparaît à l'écran est multiplié par 2 à chaque fois que vous cliquez sur **In**.

Out diminue la taille du contenu de l'écran graphique. La taille de tout ce qui apparaît à l'écran est divisée par 2 à chaque fois que vous cliquez sur **Out**.

Normal ramène la taille du contenu de l'écran graphique à sa taille initiale.

Examinons enfin le menu **Help** :

Index affiche l'index du fichier d'aide de MSWLogo (en anglais). Vous utiliserez ce menu à chaque fois que vous voudrez consulter le manuel de référence pour avoir de l'aide sur une **primitive** Logo.

*Une **primitive** est une tâche élémentaire simple prédéfinie par le concepteur du logiciel. MSWLogo possède un grand nombre de primitives c'est à dire d'outils permettant à l'utilisateur de donner des instructions à l'ordinateur. En combinant ces tâches différentes élémentaires simples, vous écrirez des tâches plus complexes que sont les procédures et en combinant des procédures vous obtiendrez une tâche complexe qu'est un programme.*
Exemples : print (écris), forward (avance), right (tourne à droite), buttoncreate (crée un bouton) etc... sont des primitives.

MCI affiche le fichier d'aide des fonctions multimédia de MSWLogo (en anglais).

Using Help est en quelque sorte l'aide sur l'aide.

Tutorial affiche la partie du manuel de référence expliquant comment démarrer avec Logo pour un débutant (en anglais).

Demo lance un programme Logo de démonstration très complet qui donne un aperçu des possibilités de MSWLogo.

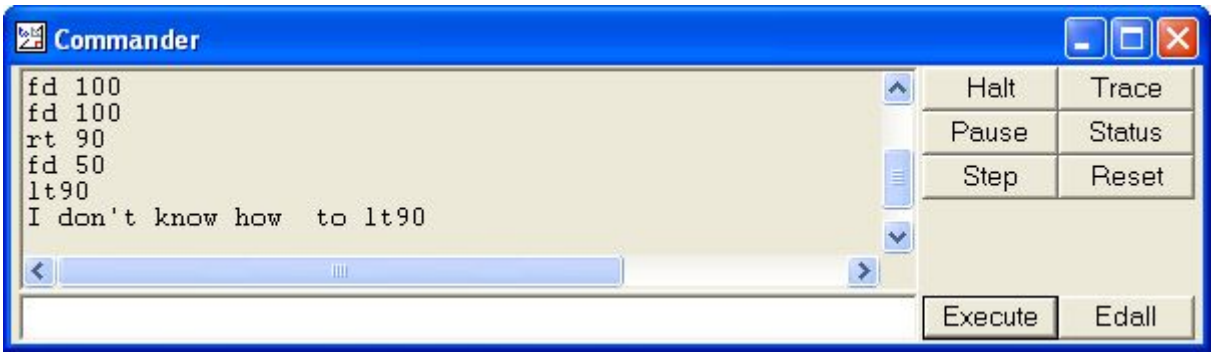
Examples affiche un fichier texte où sont répertoriés tous les exemples de programmes qui se trouvent dans le répertoire Examples avec leur description.

Release Notes affiche un fichier texte qui contient les notes de l'auteur.

About Logo et **About MS...** affiche les fenêtres « à propos » que nous avons déjà vues lors du premier démarrage

Le « Commander » :

Examinons cette copie du Commander en cours d'utilisation :



La partie inférieure du **Commander** comporte une zone où on peut taper du texte. Elle permet d'entrer une ligne de commande en mode direct. Pour valider cette ligne de commande, il suffit de cliquer sur le bouton **Execute** ou de taper sur la touche **Entrée** du clavier.

La zone de texte qui se trouve au dessus contient dans l'ordre toutes les lignes de commandes qui ont été entrées ainsi que les messages d'erreurs. Notez que si vous cliquez sur une ligne de texte de cette zone, le texte est automatiquement recopié dans la zone de saisie.

*Dans notre exemple, **fd 100** est une instruction qui fait avancer la tortue graphique de 100 pas.*
***rt 90** est une instruction qui fait pivoter la tortue graphique de 90 degrés vers la droite.*
***lt90** est une instruction qui comporte une faute de frappe (il manque un espace ent lt et 90) et qui a donc provoqué une erreur.*
***I don't know how to lt90** (je ne sais pas comment faire lt90) est le message d'erreur qui correspond à la ligne ci-dessus.*

Examinons maintenant la série de boutons qui se trouvent dans la partie droite du Commander : Le bouton **Execute** lance l'exécution de la ligne de commande qui a été tapée, il a le même effet qu'une frappe sur la touche Entrée.

Edall affiche le contenu de la mémoire (procédures et variables) dans l'éditeur. Je décrirai le fonctionnement de l'éditeur plus loin dans ce document.

Halt permet d'interrompre l'exécution d'un programme.

Pause permet de figer le programme au cours de son exécution pour repérer une erreur en examinant le contenu d'une variable par exemple.

Step permet d'exécuter le programme ligne par ligne pour repérer des erreurs.

Trace affiche une trace de toutes les actions effectuées par le programme dans le Commander. Peut être d'une aide précieuse pour une erreur dans un programme.

Status affiche la fenêtre ci-dessous qui donne des renseignements à un moment donné sur MSWLogo :

Et enfin le bouton **Reset** dont on se servira souvent permet de réinitialiser MSWLogo en effaçant l'écran graphique et tous les objets Windows qui s'y trouvent.

*Le terme **objets Windows** désigne les fenêtres et tout ce qu'on peut placer dedans (les boutons, les cases à cocher, les zones de liste, les zones de choix, les ascenseurs, les étiquettes).*

L'éditeur de texte de MSWLogo

MSWLogo possède un puissant éditeur de texte. C'est dans l'éditeur que l'on écrit ou que l'on modifie les procédures qui composent le programme. Il y a plusieurs manières d'accéder à l'éditeur :

1 – La manière la plus simple d'afficher l'éditeur est de cliquer sur le bouton **Edall** du Commander. Tout ce qui est en mémoire, procédures et variables est alors affiché dans l'éditeur.

2 - En cliquant sur le menu **File / Edit** vous obtenez cette boîte de dialogue contenant la liste de tous les noms de procédures qui sont déjà définies dans votre programme ou une boîte de dialogue avec une liste vide si aucune procédure n'est définie :

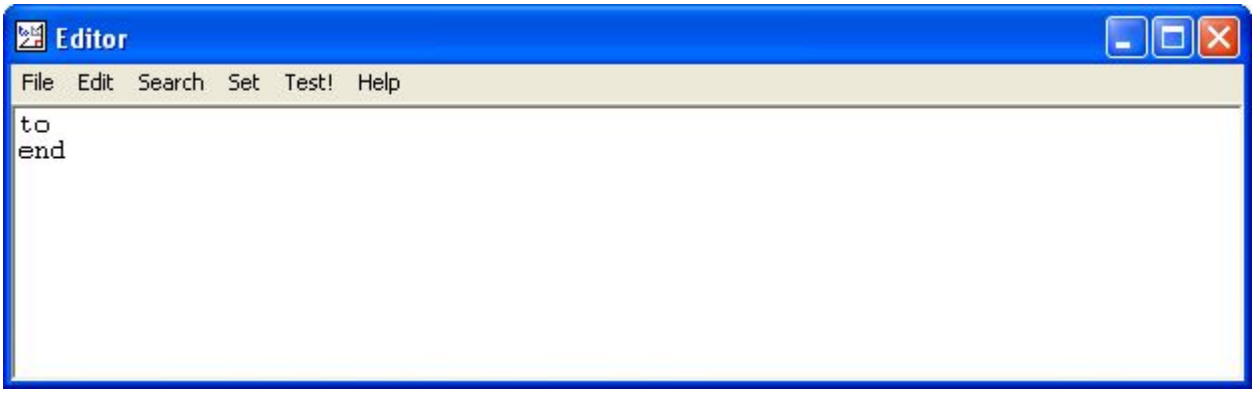
Si on veut entrer dans un éditeur vide, il faut cliquer sur le bouton **OK**.
Si on veut éditer une procédure en particulier, il suffit de la sélectionner dans la liste en cliquant sur son nom puis de cliquer sur le bouton **OK**.
Si on veut éditer toutes les procédures du programme, il faut cliquer sur le bouton **All**.
Le bouton **Cancel** permet d'abandonner l'opération.

3 – Tapez la commande **`edall`** dans la zone de saisie de texte du Commander puis tapez la touche <Entrée> ou cliquez sur le bouton **Execute**.

4 – Tapez **ed** [liste des procédures que vous voulez éditer] dans la zone de saisie de texte du Commander puis tapez <Entrée> ou cliquez sur le bouton **Execute**.

Mon tout premier programme :

Nous allons maintenant découvrir l'éditeur en écrivant quelques procédures simples pour réaliser des graphismes à l'écran.
Utilisons la manière la plus rapide d'entrer dans l'éditeur, cliquons sur le bouton **Edall** du Commander :



Une nouvelle fenêtre intitulée **Editor** s'affiche à l'écran. Elle comporte une menu et vous voyez apparaître 2 mots placés sur 2 lignes différentes : **to** et **end**.

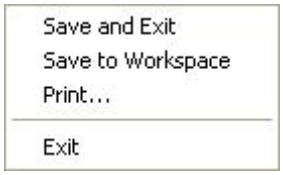
L'éditeur est l'endroit où on définit les procédures. Une procédure est délimitée par les 2 mots réservés **to** et **end** qui marquent respectivement le début et la fin de la procédure. Le premier et le dernier mot de la procédure sont donc affichés automatiquement, pratique non ?
Pour définir une procédure, vous procédez ainsi :
Sur la première ligne, après le mot **to**, placez le nom que vous avez choisi pour votre procédure.
Sur la ou les lignes suivantes ajoutez la ou les lignes d'instructions.
Et sur la dernière ligne qui marque la fin de la procédure laissez le mot **end** seul.

Exemple d'une procédure qui trace un triangle équilatéral à l'écran. Tapez le texte suivant dans l'éditeur :
(*Vous pouvez remplacer **forward** et **right** par leur abréviation : **fd** et **rt**.)*

```
to triangle
repeat 3 [forward 100 right 120]
end
```

*Ce que je traduirais en bon français par : Répète 3 fois de suite la liste d'instructions suivante : avance de 100 pas et pivote de 120 degrés vers la droite. Avec Logo, une liste est toujours délimitée par des crochets : **[** et **]**. «[» s'obtient en appuyant simultanément sur les touches AltGr et 6 et «]» s'obtient avec la combinaison AltGr et °.*

Nous venons d'écrire notre première procédure. Comme vous le voyez, elle est très courte mais elle peut aussi être fort longue puisque théoriquement, il n'y a pas de limite à la longueur d'une procédure. Nous vérifierons tout à l'heure que le résultat de cette procédure est bien ce que nous attendons. En attendant, observons le menu **File** de l'éditeur :



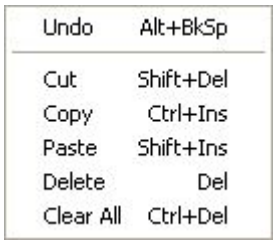
Save and Exit permet de sortir de l'éditeur en enregistrant en mémoire la ou les procédures définies.

Save to Workspace est réservé à un usage avancé de l'éditeur, il n'est pas disponible lors d'un usage normal. Pour activer cet item, il faut passer un paramètre spécial à Logo au moment du lancement du programme.

Print... permet d'imprimer le texte contenu dans l'éditeur sur votre imprimante.

Exit pour quitter l'éditeur.

Ouvrons maintenant le menu **Edit** :



Undo permet d'annuler la dernière action effectuée dans l'éditeur.

Cut pour couper le texte sélectionné dans l'éditeur.

Copy pour copier dans le presse-papiers le texte sélectionné dans l'éditeur.

Paste pour coller dans l'éditeur le texte contenu dans le presse-papiers.

Delete pour effacer le texte sélectionné dans l'éditeur.

Clear All pour effacer le contenu dans l'éditeur (pour obtenir un éditeur complètement vide).

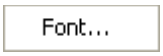
Examinons maintenant le menu **Search** :

Find... permet la recherche de texte dans l'éditeur.

Replace... pour remplacer une expression par une autre expression.

Next pour passer à l'occurrence suivante de l'expression recherchée.

Au tour du menu **Set** maintenant :



Font... permet de sélectionner la police de caractère de l'éditeur. Je vous conseille l'utilisation d'une police de caractères non proportionnelle qui met mieux en évidence les espaces entre les mots.

Le menu **Test!** maintenant. Vous constatez qu'aucun menu ne s'affiche. Son rôle est d'envoyer à l'interpréteur Logo le texte qui est sélectionné dans l'éditeur pour tester sa validité sans avoir à enregistrer la procédure en mémoire ou à taper ce texte dans la zone de saisie du Commander.

L'interpréteur : En informatique, il existe plusieurs familles de langage de programmation. Dans les langages compilés, comme le langage C, le programme tel qu'il a été écrit par le programmeur est converti par un autre programme (qu'on appelle compilateur) en un fichier écrit dans un langage uniquement compréhensible par la machine (ce qu'on appelle un exécutable). Les programmes compilés sont ceux qui s'exécutent le plus rapidement. Mais c'est aussi sur ces programmes que certains virus s'implantent. Comme le code est totalement incompréhensible pour un humain, impossible de repérer facilement un morceau de code qui a été ajouté au programme pour nuire à l'utilisateur.
Il existe aussi des langages interprétés, comme Logo ou Basic où le texte du programme est lu par un programme spécial appelé interpréteur qui analyse chaque ligne du programme pendant l'exécution pour la transformer en code compréhensible par la machine. C'est donc plus lent puisque le programme est traduit pendant la lecture. Par contre, le programme reste lisible pour un humain; il est donc plus difficile d'y cacher une portion de code destinée à nuire.
Il existe enfin des langages mixtes qui sont partiellement compilés et partiellement interprétés comme le langage Java par exemple.

Cliquons enfin sur le menu **Help :**

Index permet d'accéder à l'index du fichier d'aide de MSWLogo.

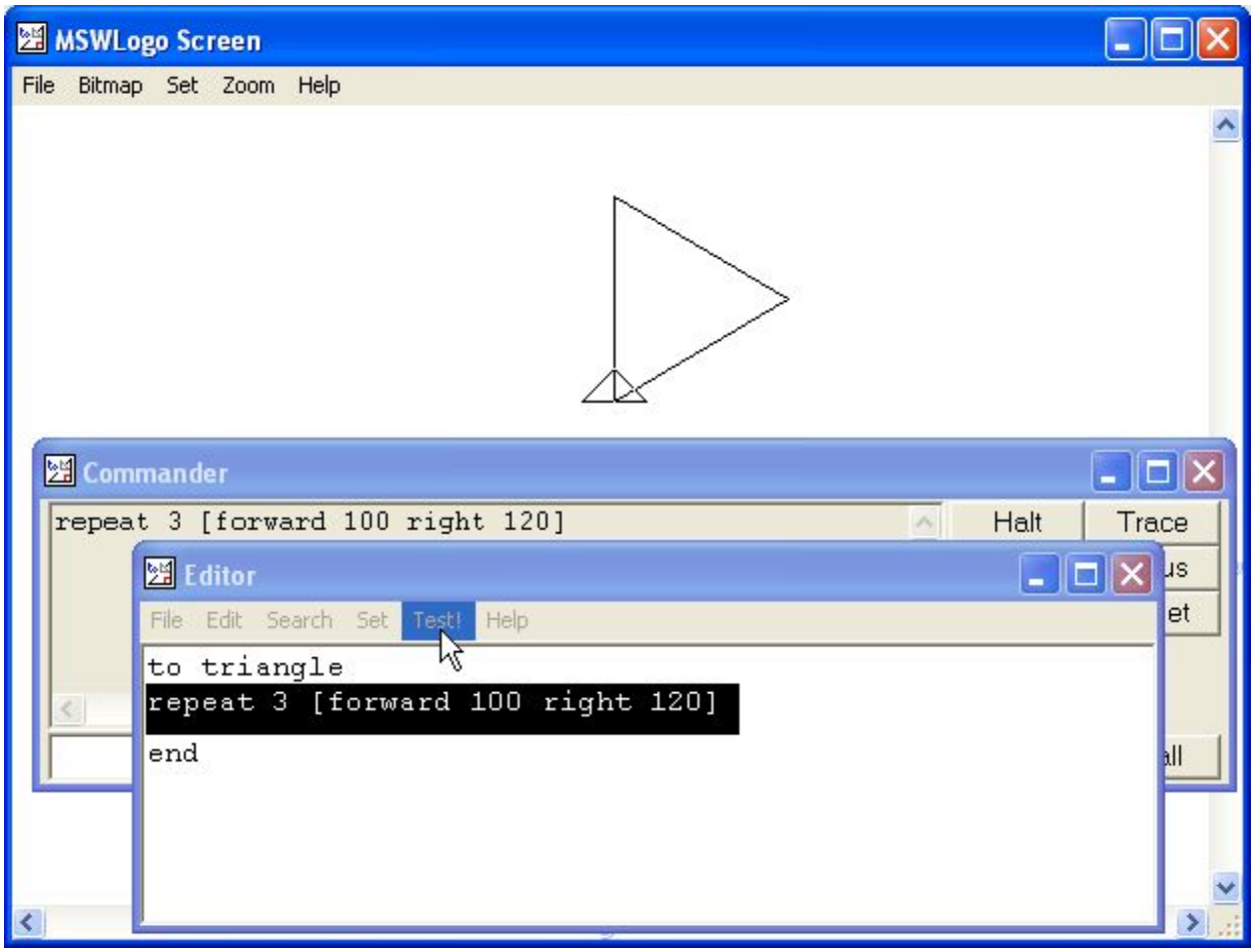
Editor affiche l'aide relative à l'éditeur.

Topic Search permet d'entrer dans le manuel de référence de MSWLogo.

Mais revenons à la procédure triangle que nous avons écrite dans l'éditeur :
Avec la souris, sélectionnons le texte de la ligne `repeat 3 [forward 100 right 60]`en la mettant en surbrillance puis cliquons sur **Test!**

Logo a exécuté la ligne en mode direct exactement comme si on l'avait tapée dans le Commander. Cette méthode évite de taper la ligne dans le Commander pour la tester.

Examinons l'écran ci-dessous :



Le résultat obtenu est bien celui que nous escomptions, nous pouvons donc enregistrer la procédure en mémoire pour la réutiliser plus tard.
Cliquons sur le menu **File / Save and Exit** de l'éditeur. L'éditeur disparaît. La procédure est maintenant enregistrée en mémoire et à chaque fois que nous voudrons tracer un triangle à l'écran, il suffira d'invoquer la procédure en tapant **triangle** <Entrée> dans le Commander ou encore d'invoquer son nom dans une autre procédure.

Et si ça ne marche pas?

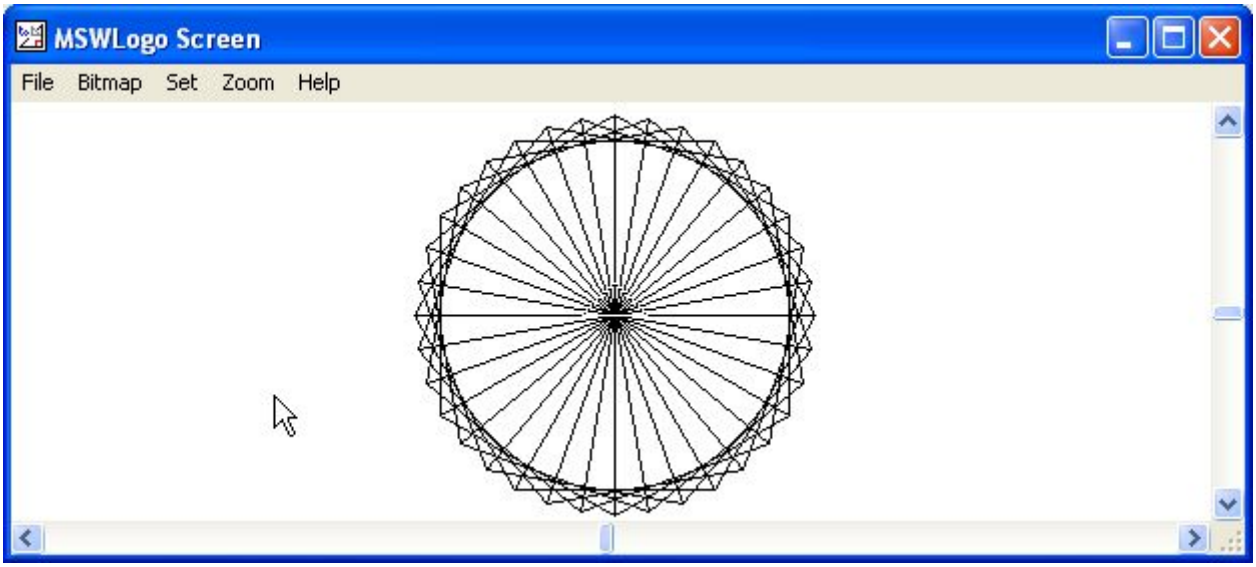
Plusieurs causes sont possibles :
*Vous avez fait une faute de frappe dans la rédaction de la ligne et Logo a rencontré un mot qui lui est inconnu. Il vous prévient en affichant un message du type **I don't know how to** ... (je ne sais pas comment faire pour ...) dans le Commander.*
*Vous avez oublié de taper le paramètre après **repeat**, ou **forward** ou **right**. L'instruction est incomplète et Logo vous prévient qu'il manque un paramètre dans la ligne en affichant le message **not enough inputs to ...**(il manque une donnée pour l'instruction ...)*
*Vous avez oublié de taper le crochet fermant (]) à la fin de la ligne ou vous avez oublié de taper le nom de la procédure après le mot **to** : Lorsque vous cliquez sur le menu **File / Save and Exit** le*

message suivant s'affiche :

Ce message vous indique que la procédure n'a pas pu être définie. Regardez dans le ***Commander****, il y a peut-être un message d'erreur pour vous aider à trouver la cause de l'erreur. Cliquez sur le bouton* ***OK*** *pour retourner à l'éditeur et corrigez avant de renouveler l' opération.*

A vous maintenant : Définissez la procédure nommée **rosace** qui affichera une rosace constituée de 36 triangles.
Je vous aide un peu en vous indiquant la marche à suivre en français :
La tortue trace le premier triangle, elle pivote de 10 degrés vers la droite, elle trace le second triangle, elle pivote de 10 degrés vers la droite, elle trace un 3ème triangle et ainsi de suite 36 fois ...
Ce qui peut se dire :
Répète 36 fois de suite les instructions suivantes : trace un triangle, pivote de 10 degrés vers la droite.
Souvenez-vous, il faut cliquer sur le bouton **Edall** pour accéder à l'éditeur et prenez modèle sur la procédure triangle pour définir rosace.. (N'oubliez pas : Le menu **File / Save and Exit** pour l'enregistrer, tapez rosace <Entrée> pour la tester...)

Vous avez trouvé ?



Voici une solution parmi d'autres :

```
to rosace
repeat 36 [triangle right 10]
end
```

le résultat est époustouflant, n'est-ce pas ?

Enregistrez la procédure en mémoire. (Menu **File / Save and Exit** de l'éditeur). Cliquez sur le bouton **Reset** du Commander pour effacer l'écran. Et invoquez la procédure rosace pour vérifier qu'elle est bien en mémoire. Vous souvenez-vous comment faire ?
Tapez **rosace** <Entrée> dans le Commander.

Vous avez peut-être envie de sauvegarder votre travail pour le retrouver la prochaine fois ?

Cliquez sur le menu **File / SaveAs...** de MSWLogo Screen pour obtenir la boîte de dialogue de sauvegarde. Là, vous savez faire je pense...

Nous venons de définir des procédures simples. Quand nous invoquons triangle ou rosace, nous obtenons toujours le même triangle ou la même rosace. Je vous propose maintenant de modifier la procédure triangle de manière à obtenir des triangles et des rosaces de différentes tailles. Pour cela, nous allons changer la taille des triangles qui composent la rosace en passant **un paramètre** (*un renseignement qui peut varier*) à la procédure triangle et qui indiquera à Logo la longueur du côté du triangle.

Cliquons sur le bouton **Edall**.

Modifions la procédure triangle ainsi :

```
to triangle :cote
repeat 3 [forward :cote right 120]
end
```

Et cliquons sur le menu **File / Save and Exit** pour enregistrer les modifications de la procédure en mémoire. Vous remarquez que nous avons ajouté le mot **:cote** précédé de 2 points après le nom de la procédure ainsi que dans la ligne d'instruction. C'est ce qu'on appelle **une variable**. Nous aurions pu l'appeler truc ou machin, nous sommes entièrement libre dans le choix du nom, il faut seulement obligatoirement mettre « **:** » devant le nom sans laisser d'espace. Logo reconnaît une variable quand il voit un mot qui commence par « **:** »

Voyons maintenant comment nous servir de cette variable :
Effaçons l'écran. Vous vous souvenez ? (**Reset**)

Tapons dans le Commander **triangle 20** <Entrée>. (Là, inutile de sélectionner la ligne dans l'éditeur et de cliquer sur Test! comme je vous l'ai expliqué précédemment, ça ne marchera pas parce qu'en mode direct, Logo ne saura pas à quelle valeur correspond :cote).
Essayons maintenant **triangle 30** puis **triangle 50**. Vous voyez, Logo trace des triangles de différentes tailles 20 pas de côté pour le premier, 30 pas pour le second, 50 pas pour le dernier.

Et si ça ne marche pas ?
Dans la première ligne, vous avez peut-être oublié d'ajouter :cote après le nom de la procédure, Logo affiche le message suivant dans le Commander :
cote has no value in triangle
[repeat 3 [forward :cote right 120]]
(La variable :cote n'a pas de valeur dans la procédure triangle et Logo affiche la ligne qui a provoqué l'erreur).
Vous avez bien ajouté :cote après le nom de la procédure mais vous avez oublié de remplacer le 100 par :cote dans la deuxième ligne de la procédure : Là pas de message, Logo trace un triangle de 100 pas de côté quelle que soit la valeur que vous passez en paramètre. Non, Logo ne peut pas tout deviner ! Sinon, qu'est-ce qui vous empêcherait de lui demander de tondre votre gazon ? Bien que, en y réfléchissant...

Revenons dans l'éditeur, je ne vous dis plus comment faire...

Maintenant, modifions la procédure rosace ainsi :

```
to rosace :c
repeat 36 [triangle :c right 10]
end
```

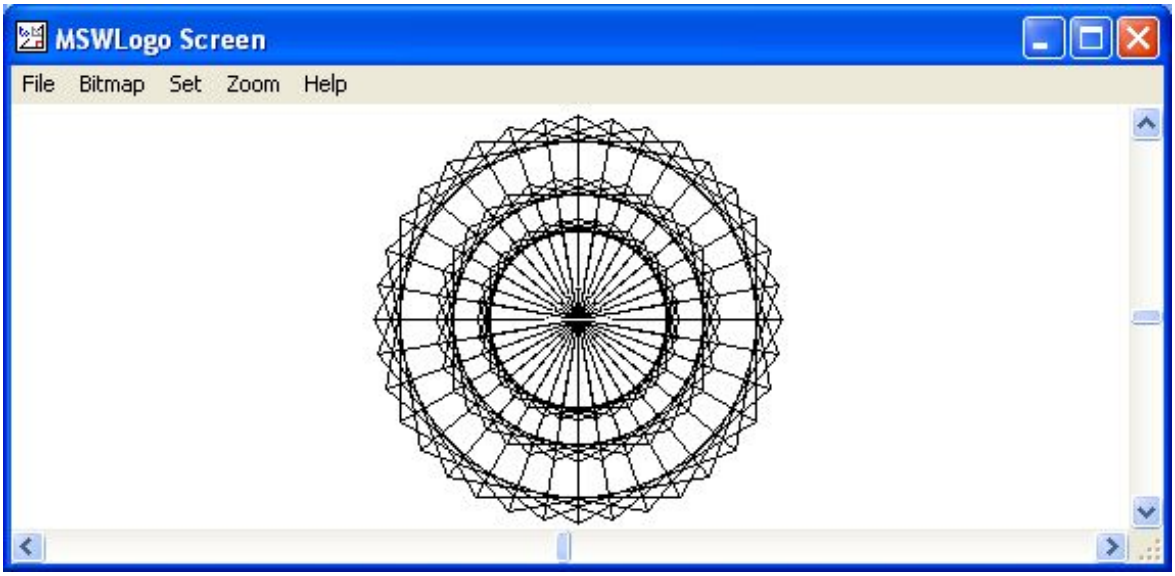
Enregistrons la modification en mémoire, vous vous souvenez comment ? Allez je vous le dis pour

la dernière fois : Cliquez sur le menu **File / Save and Exit** de l'éditeur.
Là aussi nous avons ajouté une variable appelée :c après le titre de la procédure et après le mot triangle. Cette variable va nous permettre d'indiquer la longueur du côté des triangles qui composent la rosace.

Maintenant testons la procédure rosace :

Reset pour effacer l'écran...
```
rosace 70
rosace 100
rosace 50
```
Vous obtenez des rosaces de différentes dimensions qui se superposent sur l'écran graphique.

*Notez que vous pouvez taper indifféremment **rosace** 70 ou bien **Rosace** 70 ou en core **ROSACE** 70 ou même **RoSaCe** 70 ! Logo ne fait pas la différence entre majuscules et minuscules. Mais attention aux lettres accentuées !*



A vous maintenant, cliquez sur Edall et en vous inspirant de ce que nous avons fait pour créer la procédure triangle, créez une nouvelle procédure nommée **carre** qui tracera un carré dont la longueur du côté sera contenue dans la variable :cote. Allez, je vous aide un peu : pour cela, il faut quatre fois de suite [avancer de :cote, pivoter vers la droite de 90 degrés]
N'oubliez pas d'enregistrer la procédure en mémoire avant d'appeler **carre.**

Voici une solution :
```
to carre :cote
repeat 4 [forward :cote right 90]
end
```

Maintenant essayez de modifier la procédure rosace pour remplacer les triangles par des carrés.
N'oubliez pas d'enregistrer la procédure en mémoire avant d'appeler **rosace.**

Voici une solution :
```
to rosace :c
repeat 36 [carre :c right 10]
end
```

Je vous propose maintenant de modifier la procédure rosace pour tracer la rosace avec au choix des triangles ou des carrés.

Editez la procédure rosace et modifiez-la ainsi :

```
to rosace :c :forme
```

```
if equalp :forme "carre [repeat 36 [carre :c right 10]]
if equalp :forme "triangle [repeat 36 [triangle :c right 10]]
end
```

Enregistrez ces modifications en mémoire et testez :

```
rosace 100 "triangle
```
Reset
```
rosace 150 "carre
```

Essayez d'autres valeurs pour :c

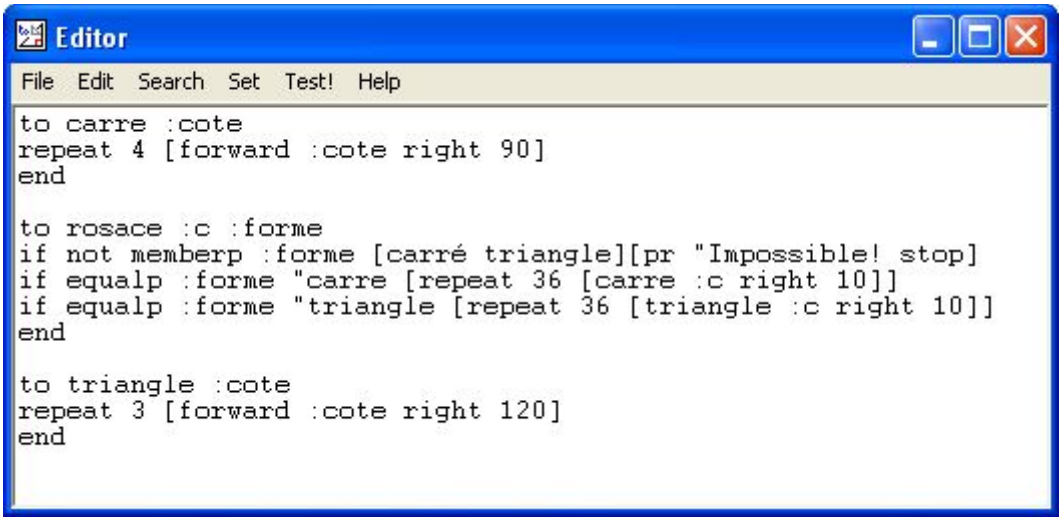
Je vous explique :
Cette fois, nous passons 2 paramètres à la procédure rosace : la longueur du côté (:c) et la forme de la figure (:forme). Ensuite nous faisons **un test** :
Si le contenu de la variable :forme est égal au mot "carré, alors on répète 36 fois de suite la liste suivante [trace le carré de côté :c et pivote vers la droite de 10 degrés].
Si le contenu de la variable :forme est égal au mot "triangle, alors on répète 36 fois de suite la liste suivante [trace le triangle de côté :c et pivote vers la droite de 10 degrés].

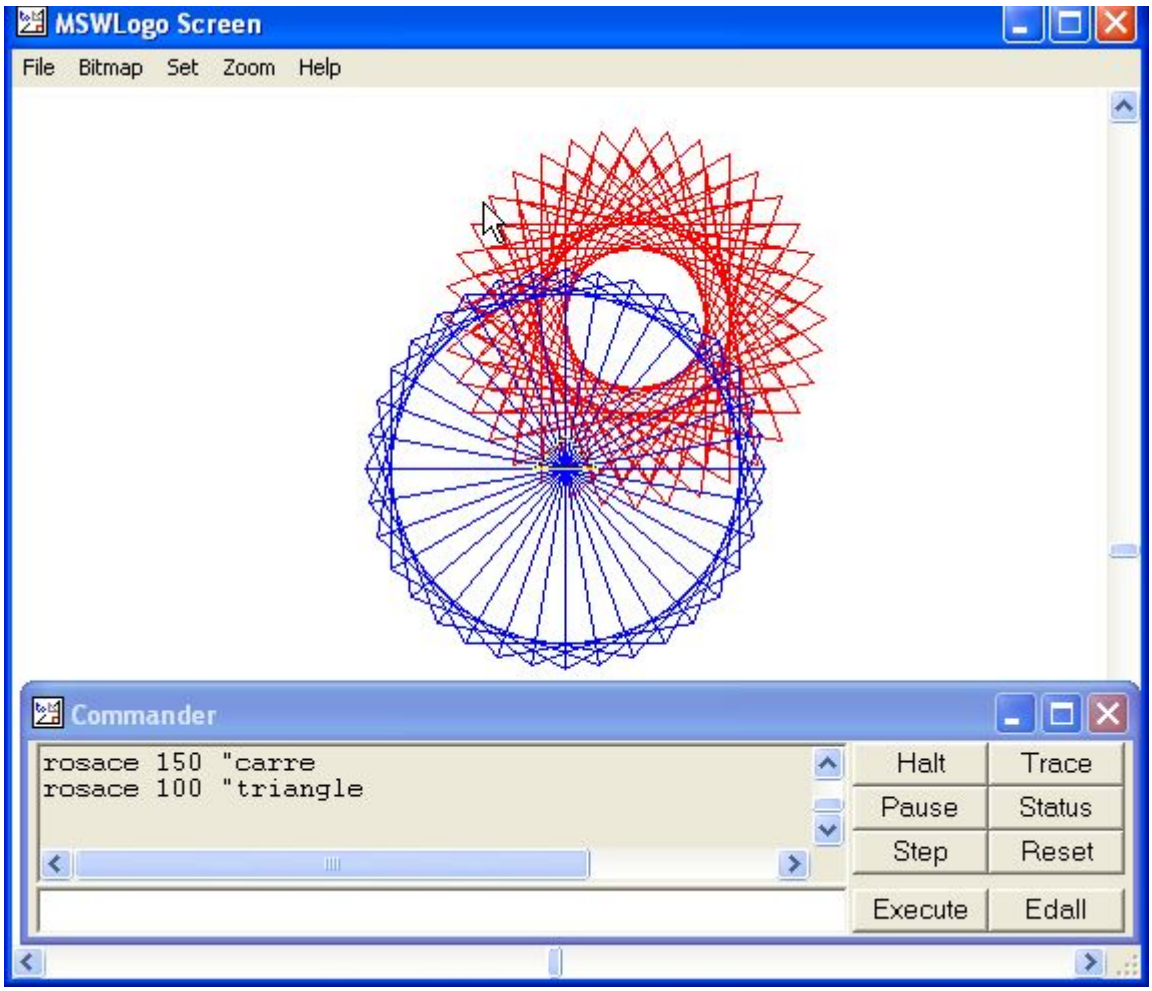
Le mot qui permet de faire un test est le mot **if** (si).
Il est employé dans une expression composée de **if** suivi du mot **equalp** (égal?) puis des 2 expressions à comparer : la valeur de la variable **:forme** et le mot **"carre** ou le mot **"triangle**.
Logo évalue cette expression et rend **true** (vrai) ou bien **false** (faux).
Si c'est vrai, Logo exécute la liste d'instructions qui suit (entre crochets), dans le cas contraire, Logo ne fait rien et passe à la ligne suivante.

Bien entendu si vous appelez rosace 100 "losange il ne se passera rien car les seuls mots attendus sont carré et triangle ! Alors, ajoutons un autre test pour en avertir l'utilisateur :
Modifions une dernière fois la procédure rosace de la manière suivante :

```
to rosace :c :forme
if not memberp :forme [carré triangle][pr "Impossible! stop]
if equalp :forme "carre [repeat 36 [carre :c right 10]]
if equalp :forme "triangle [repeat 36 [triangle :c right 10]]
end
```

Ce qui signifie : Si le paramètre passé par l'intermédiaire de la variable forme ne fait pas partie de la liste de mots [carré triangle], alors écris le mot "impossible" et arrête-toi.
Enregistrez la modification. Maintenant, si vous tapez rosace 100 "losange, Logo vous répond "Impossible!"





Ouf, ça commence à être compliqué! Mais ça en vaut la peine ! Connaissez-vous un autre langage de programmation qui soit capable de réaliser de tels graphismes en si peu de lignes ?

Encore plus fort et encore plus loin, accrochez-vous !
De nos jours, plus personne n'a envie d' utiliser un logiciel qui se lance à partir d'une ligne de commande.
Maintenant, tous les programmes se lancent et s'utilisent avec la souris. C'est ce que je vous propose de faire maintenant. MSWLogo permet de créer des programmes qui utilisent la souris pour gérer des fenêtres, des boutons, des boîtes de dialogue, des ascenseurs comme ceux qu'on rencontre habituellement dans un programme Windows, Linux ou Mac OS.

Entrons dans l'éditeur et ajoutons les 2 procédures suivantes à notre programme:

```
to mon_appli
;dessine l'interface de commande du programme
windowcreate "root "appli [Mon dessin] 0 25 112 120 [objets_mon_appli]
end


to objets_mon_appli
;dessine les objets contenus dans la fenêtre
listboxcreate "appli "formes 5 5 35 50
listboxaddstring "formes "carré
listboxaddstring "formes "triangle
listboxcreate "appli "tailles 65 5 35 50
listboxaddstring "tailles 50
listboxaddstring "tailles 75
listboxaddstring "tailles 100
listboxaddstring "tailles 125
```

```
listboxaddstring "tailles 150
listboxaddstring "tailles 175
listboxaddstring "tailles 200
buttoncreate "appli "btn_dessiner "Dessine 5 70 30 16 [rosace first
(fin ligne ci-dessus :) listboxgetselect "tailles first listboxgetselect "formes]
buttoncreate "appli "btn_effacer "Efface 40 70 30 16 [cs]
buttoncreate "appli "btn_quitter "Quitter 75 70 30 16 [windowdelete "appli]
end
```

Examinons ces procédures :
Avec Logo, on peut ajouter des lignes de commentaires dans le programme pour le rendre plus lisible. Pour cela il suffit de commencer la ligne par **un point virgule**.
La procédure mon_appli lance l'affichage de la fenêtre. Elle ne comporte qu'une seule instruction : **windowcreate** qui permet de créer une fenêtre à l'écran, elle est suivie des paramètres suivants : le nom de la fenêtre parent (root) le nom de la fenêtre (appli) le titre de la fenêtre (Mon dessin) les coordonnées x et y à l'écran en partant du coin supérieur gauche de l'écran (0 et 25), la largeur de la fenêtre (112), sa hauteur (120) et enfin la liste des objets à placer dans cette fenêtre. Ici, comme la liste est assez longue, j'ai préféré placer ces objets dans une procédure (objets_mon_appli)

La procédure objets_mon_appli affiche et initialise les éléments contenus dans la fenêtre :

listboxcreate crée une liste de choix (listbox) elle est suivie des paramètres suivants : le nom de la fenêtre (appli), le nom attribué à la listbox (formes ou tailles), les coordonnées x et y dans la fenêtre (5, 5 ou 65, 5), la largeur de la listbox (35) et sa hauteur (50).

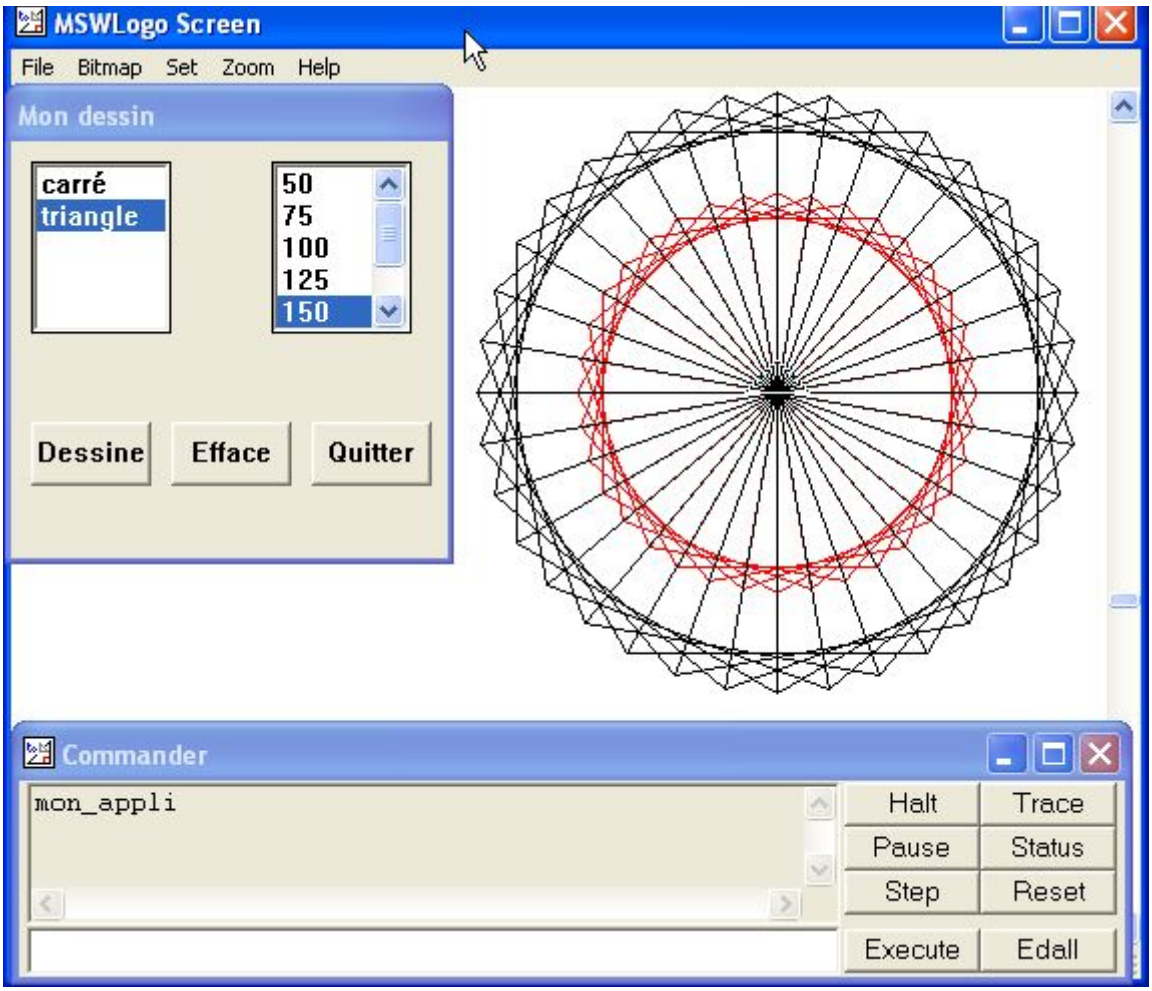
L'instruction **listboxaddstring** permet de compléter la listbox avec des mots et des éléments. Elle a besoin de 2 paramètres : le nom de la listbox (formes ou tailles), l'expression à afficher dans la listbox (carré, triangle, 100, 175 etc...)

L'instruction **buttoncreate** permet d'afficher un bouton dans une fenêtre. Ses paramètres sont le nom de la fenêtre qui porte le bouton (appli), le nom donné au bouton (btn_effacer, btn_quitter, btn_dessiner), le texte à afficher sur le bouton (Dessine, Efface, Quitter) les coordonnées x et y du bouton dans la fenêtre, la largeur du bouton (35), la hauteur du bouton (16) et enfin la liste d'instructions (entre crochets []) à exécuter lorsqu'on clique sur le bouton.

L'instruction **listboxgetselect** suivie du nom de la listbox rend le texte qui est sélectionné dans la listbox.

L'instruction **first** rend le premier élément d'une liste ou d'un mot. Logo fait la différence entre les mots et les listes et n'aime pas qu'on lui fournisse une liste quand il attend un mot. Comme **listboxgetselect** rend une liste, on prend le premier élément de cette liste pour être sûr d'avoir affaire à un mot.

Et enfin **windowdelete** suivi du nom de la fenêtre efface la fenêtre et tous ses éléments.
Enregistrons ces procédures en mémoire et testons le programme : **mon_appli** <Entrée>. Testons son fonctionnement : ça marche !



Et si ça ne marche pas ?
Faites attention quand vous rentrez l'intitulé qui doit figurer sur un bouton, Logo est intransigeant en ce qui concerne les mots et les listes :
Quand c'est un mot, il est précédé de guillemets exemple ***"Quitter***
Quand c'est une liste, les mots sont entre crochets exemple ***[Mon programme]***
Faites attention aux crochets et aux parenthèses, ils doivent impérativement être appariés ! Si ce n'est pas le cas, Logo proteste et vous propose de retourner dans l'éditeur mais sans pointer l'erreur.

Et si je définis une nouvelle procédure portant le nom d'une primitive ou d'une procédure qui existe déjà dans mon programme, est-ce que la primitive ou la procédure seront écrasées?
Non, lorsque vous cliquerez sur le menu ***File / Save and Exi****t de l'éditeur, le message suivant s'affichera à l'écran:*

Et le Commander contiendra le message suivant : ... ***is already defined.***

A chaque fois que nous invoquons la procédure **mon_appli**, le programme démarre, mais vous remarquerez que si vous essayez de lancer une 2ème session de mon_appli, le message d'erreur suivant s'affiche. Deux objets portant le même nom ne peuvent pas être affichés simultanément à l'écran**.**

Maintenant, avant de sauvegarder notre programme, nous allons faire en sorte qu'il démarre automatiquement à chaque fois qu'on le charge en mémoire sans qu'il y ait besoin d'invoquer la procédure **mon_appli.**

Pour cela tapons dans le Commander :

```
make "startup [mon_appli]
```

make sert à affecter une valeur à une variable. Ici, c'est une variable spéciale nommée **startup** qui est chargée avec le nom de la procédure à invoquer aussitôt que le programme sera chargé en mémoire.

Maintenant sauvegardez le programme avec le menu **File /Save.**

La prochaine fois que nous chargerons le programme mon_appli dans la mémoire de MSWLogo, il démarrera tout seul !

Quelques exemples de programmes :

Si vous êtes arrivés jusqu'ici, c'est que ma petite démo vous a intéressé, je ne saurais trop vous encourager à persévérer, car c'est un langage de programmation fabuleux et extraordinairement puissant, pour l'instant, je ne vous ai guère fait découvrir que 1% de ses possibilités. Je programme avec MSWLogo depuis pas mal d'années et je n'ai pas encore fait le tour de toutes ses possibilités.

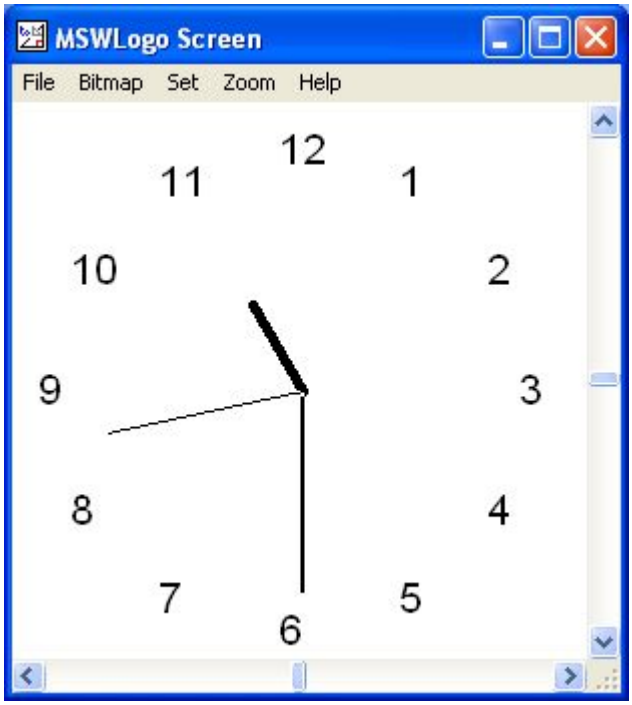
Bien sûr, vous ne serez pas capable de créer une application avec des fenêtres comme celle que nous venons d'écrire dès le premier essai, mais vous y arriverez petit à petit à force de persévérance, de rigueur et en observant comment d'autres s'y sont pris. MSWLogo est fourni avec de nombreux programmes d'exemples qui pourront vous servir de point de départ. Je suis convaincu que c'est réellement le langage informatique le mieux adapté pour la découverte de la programmation.

MSWLogo sait presque tout faire. Et je vais en apporter la preuve au travers de quelques exemples:

Tapez **`chdir "Examples`** <Entrée> dans le Commander
Logo affiche : **`Pushed to C:\Mswlogo\examples`**
*La commande **chdir** permet de changer de répertoire.*

Votre montre est en panne ?

Tapez **`chdir "Misc`** <Entrée> pour entrer dans le sous répertoire Examples\Misc.
Tapez maintenant **`load "Clock.lgo`** <Entrée> pour charger le programme Clock.lgo en mémoire (vous l'avez compris, **load** permet de charger un programme). Le programme démarre automatiquement comme je vous l'ai expliqué plus haut. Répondez **oui** à la question posée, si vous répondez **non**, le listing du programme est affiché. Mais si, cette horloge fonctionne !
Notez que vous pouvez également charger le programme en utilisant le menu **File / Load**.



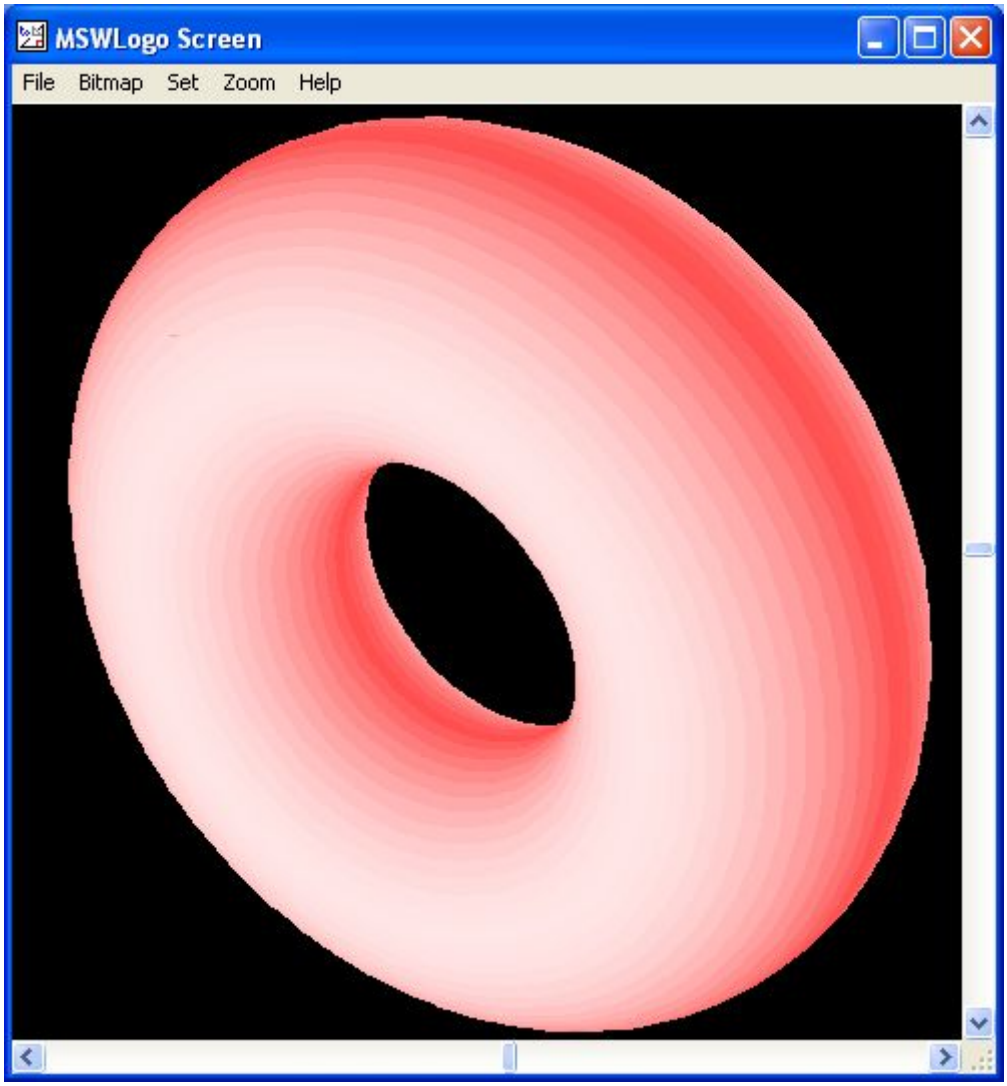
Votre piano n'est plus accordé? (là c'est sous sous réserve que votre carte son fonctionne et que le son du haut parleur ne soit pas coupé)

Cliquez sur **Reset.**
Tapez **`popdir`** <Entrée> pour revenir dans le répertoire Examples

popdir *permet de revenir au répertoire supérieur.*
puis **`chdir "Multimed`** <Entrée> pour entrer dans le sous répertoire Multimed.
Tapez maintenant **`load "Midi.lgo`** <Entrée> . Cliquez une fois sur le fond de l'écran graphique pour vous assurer que c'est bien MSWLogo Screen qui est la fenêtre active. Appuyez sur les touches X, C, V, B, N... étonnant, n'est-ce pas ?
Vous n'êtes pas convaincus ?

Un pneu de votre voiture est crevé et votre roue de secours est dégonflée ? (là, c'est sans garantie pour la voiture)

Cliquez sur **Reset.**
Tapez **`popdir`** <Entrée> pour revenir dans le répertoire Examples.
Tapez **`chdir "3d`** <Entrée> pour entrer dans le sous répertoire 3d.
Tapez maintenant **`load "Torus.lgo`** <Entrée> .



Cliquez sur **Reset**.
Vous avez besoin de boules pour décorer votre sapin de Noël ?
Tapez **`load "Sphere.lgo`** <Entrée> . Cliquez sur **Reset** pour finir.

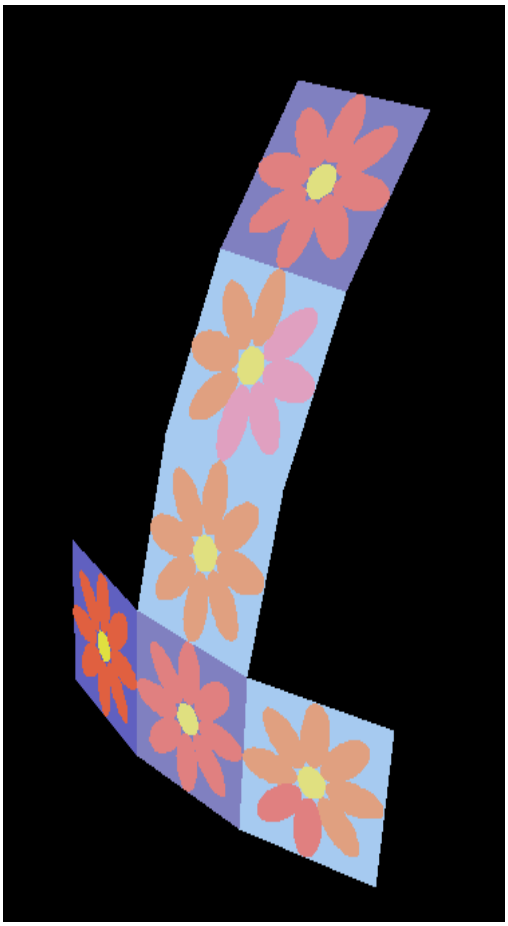
Vous avez besoin d'une représentation du système solaire pour votre prochain cours ?
Tapez **`load "Solar.lgo`** <Entrée> . Cliquez sur **Reset** pour finir.

Vous avez besoin d'une navette spatiale dont les plans ont été conçu avec Autocad pour vous déplacer dans le système solaire ?

Tapez **`load "Shuttle.lgo`** <Entrée> . Cliquez sur **Reset** pour finir.

Si vous préférez voir la même navette en relief (avec des lunettes adéquates, bien sûr) :
Tapez **`load "Stereo.lgo`** <Entrée> . Cliquez sur **Reset** pour finir.

Vous désirez créer une image gif animée pour votre site internet ?
Tapez **`load "3dmovie.lgo`** <Entrée> . Cliquez sur **Reset** pour finir.



Pour voir l'animation, rendez-vous dans le répertoire **Examples/3d**. Avec votre navigateur internet préféré, ouvrez le fichier **"3dmovie.gif** que Logo vient de créer pour vous.

Vous voulez que votre belle-mère vous f.... la paix ? Tapez... non, là, ça ne marchera pas.

Le répertoire **Examples** contient des dizaines d'exemples des plus spectaculaires aux plus sophistiqués, des plus simples aux plus compliqués qui vous aideront "à comprendre comment ça marche" et à créer vos propres réalisations.

Et si ça ne marche pas ?

Si vous n'arrivez pas à charger les programmes ci-dessus, vérifiez que vous avez mis des guillemets devant le nom du fichier et que vous avez bien tapé le nom du fichier avec son extension (.lgo). Vérifiez encore que vous n'avez pas fait une faute de frappe dans le nom du fichier.
Si ça ne marche toujours pas, c'est que vous n'êtes pas dans le bon répertoire (gardez un oeil sur les messsages du Commander).

Passons aux choses sérieuses !

La réalisation qui suit nécessite une carte son en ordre de marche et un microphone. *Assurez-vous que vous disposez de ces équipements avant de vous lancer dans la programmation. Notez aussi qu'actuellement, avec MSWLogo utilisé* ***sous Linux avec l'émulateur wine****, les fonctions sonores ne fonctionnent pas. (Ne désespérez pas,* ***wine*** *est un projet qui évolue très vite et il est possible qu'au moment où vous lirez ces lignes, tout fonctionne normalement.)*

Tapez **bye** pour quitter MSWLogo et relancez-le pour sortir du **mode perspective** qui a permis les graphismes ci-dessus. Le mode perspective permet de tracer des graphismes en 3D, la tortue se transformant en avion... C'est pour cela qu'elle n'a plus la forme d'un triangle isocèle quand on efface l'écran avec le bouton Reset, le triangle est affiché en perspective.

Je vous propose maintenant d'écrire un programme utilitaire qui pourra servir en classe : un dictaphone. Pourquoi écrire un dictaphone, alors que Windows comprend déjà un magnétophone ? Essayez d'utiliser le magnétophone de Windows comme dictaphone pour écrire un texte d'une dizaines de phrases. Rapidement, vous vous rendrez compte que c'est galère pour positionner la bande au début de la phrase qu'on veut réécouter si elle est située dans le milieu de l'enregistrement. Le dictaphone que je vous propose d'écrire permettra l'enregistrement des phrases une par une et en lecture nous pourrons naviguer facilement parmi ces phrases en cliquant simplement sur des boutons et écouter plusieurs fois la même phrase sans être obligé de repositionner la bande au début de la phrase.

Avant de nous lancer dans l'écriture de procédures, réfléchissons à ce que nous voulons obtenir et surtout comment l'obtenir... C'est l'étape la plus importante et la plus délicate dans la programmation d'un logiciel ! Savoir quoi faire et comment le faire !

Nous avons besoin d'un module d'enregistrement des phrases, d'un module de lecture des phrases enregistrées et d'une interface qui proposera le choix entre l'enregistrement et la lecture.

Logo possède des primitives qui permettent l'enregistrement et la lecture de fichiers sonores au format .wav, c'est donc ce format que nous utiliserons. Chaque fragment de la dictée sera en fait un fichier. Le nom du fichier contiendra le numéro de l'enregistrement. Tous les fichiers d'une dictée seront placés dans un même sous-répertoire. Il y aura donc autant de sous-répertoires que de dictées. Pour chaque dictée, nous créerons un fichier vide dont le nom sera celui de la dictée et dont le rôle sera de déterminer le nom du sous-répertoire qu'il faudra lire. C'est le nom de ce fichier qui apparaîtra dans la boîte de dialogue d'ouverture de fichier.
En résumé, pour chaque dictée, on créera un fichier vide et un sous-répertoire contenant les enregistrements sonores de la dictée.

Maintenant que nous avons une idée précise de la manière dont nous allons procéder et que nous savons que c'est techniquement possible, nous pouvons commencer :

Conseils :
Plus un programme est lisible, plus il est facile de comprendre son fonctionnement et par conséquent de trouver les erreurs. **Lorsqu'on crée une procédure ou une variable, il faut lui choisir un nom qui évoquera le plus possible ce qu'elle fait ou ce qu'elle contient**. Les noms doivent toujours être des mots, utilisez le caractère « _ » pour relier les mots si le nom est composé de plusieurs mots.
Evitez d'écrire des procédures « fleuves », il est toujours très difficile d'y repérer une erreur.
N'hésitez pas à ajouter des lignes de commentaire dans le programme, elles vous aideront à retrouver ce à quoi sert tel morceau de programme quand ce n'est pas évident à première vue.

<u>Convention adoptée pour la rédaction des procédures dans les pages qui suivent</u> :
Dans l'éditeur Logo, la longueur de ligne est pratiquement illimitée, un ascenseur horizontal permet de se déplacer dans le texte si la ligne est trop longue. Dans ce document, la longueur de ligne est limitée par la largeur de la feuille de papier. Il arrivera souvent qu'une ligne de programme tienne sur 2 ou 3 lignes de ce document. Pour éviter les erreurs, je placerai systématiquement un « ► » au début de chaque nouvelle ligne. N'essayez pas de taper ce « ► », il n'est pas directement accessible au clavier. Vous saurez simplement que lorsqu'une ligne ne commence pas par ce signe, <u>c'est qu'il faut la taper à la suite de la ligne précédente</u>.

La fenêtre de choix de l'activité :

C'est la fenêtre principale de notre programme, nous reviendrons à cette fenêtre après chaque activité. Elle comporte une zone de texte **Que désires-tu faire?** et 3 boutons :
Enregistrer une dictée lancera le module d'enregistrement.
Faire une dictée lancera le module de lecture.
Quitter nous fera sortir du programme et de Logo.

En ce qui me concerne, j'ai adopté les règles suivantes pour choisir les noms de procédures : Le nom d'une procédure qui se rapporte à une fenêtre se termine toujours par le nom de la fenêtre. Par exemple, la procédure qui dessine les objets de la fenêtre nommée **demarrage** s'appelle **objets_demarrage,** la procédure qui est appelée quand on clique sur le bouton **Quitter** de la fenêtre nommée **demarrage** s'appelle **clic_sortie_demarrage,** la procédure appelée quand on clique sur le bouton nommé **enregistrer** de la fenêtre **demarrage** s'appelle donc **clic_enregistrer_demarrage**... Ainsi, c'est plus facile de s'y retrouver !

```
►to demarrage
►windowcreate "root "demarrage [Dictaphone] 100 60  115 90
[objects_demarrage]
►end

►to objets_demarrage
►buttoncreate "demarrage [sortie] [Quitter] 9 52 94 12
[clic_sortie_demarrage]
►buttoncreate "demarrage [faire_dictee] [Faire une dictée] 9 36 94 12
[clic_fairedictee_demarrage]
►buttoncreate "demarrage [enregistrer] [Enregistrer une dictée] 9 20 94
12 [clic_enregistrer_demarrage]
►staticcreate "demarrage [texte] [Que désires-tu faire ?] 9 4 94 12
►end
```

La procédure **demarrage** est le point d'entré dans notre programme, elle crée la fenêtre principale et appelle la procédure **objects_demarrage** qui dessine tous les objets de la fenêtre.

Astuce : Pour afficher toutes les informations sur un mot clé du programme par exemple ***buttoncreate,*** *mettez-le en surbrillance d'un double clic puis tapez sur la touche F1, le manuel d'aide de MSWLogo s'ouvrira à la page de ce mot.*

Lorsqu'on clique sur le bouton **Sortie**, on efface la fenêtre principale et on quitte Logo.

```
►to clic_sortie_demarrage
►windowdelete "demarrage
►bye
►end
```

Lorsqu'on clique sur le bouton **Faire une dictée**, on efface la fenêtre principale et on affiche la fenêtre de lecture.

```
►to clic_fairedictee_demarrage
►windowdelete "demarrage
►choix_dictee
►end
```

Lorsqu'on clique sur le bouton **Enregistrer**, on efface la fenêtre principale et on affiche la fenêtre d'enregistrement.

```
►to clic_enregistrer_demarrage
►windowdelete "demarrage
►enregistrement
►end
```

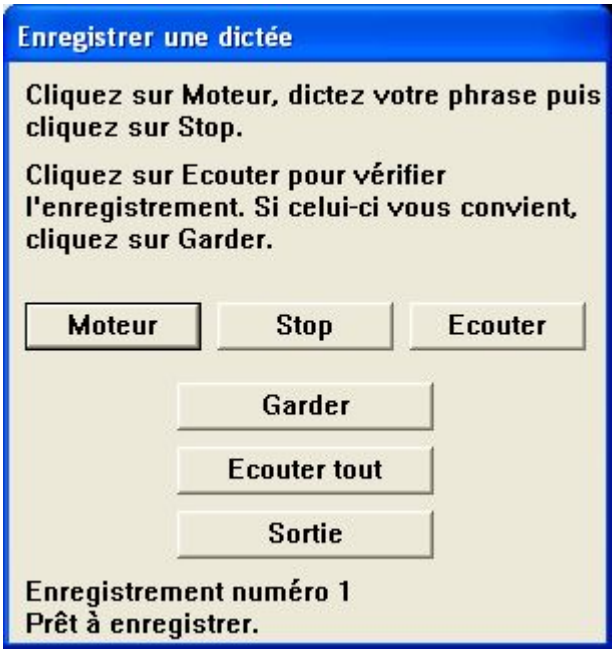
Testez cette portion de code :
Cliquez sur le menu **File / Save and Exit** puis tapez **demarrage** <Entrée> dans le Commander.

Lorsque vous cliquez sur les boutons **Enregistrer une dictée** et **Faire une dictée**, Logo affiche les messages d'erreur **I don't know how to do choix_dictee** et **I don't know how to do enregistrement**. C'est tout à fait normal car les procédures **choix_dictee** et **enregistrement** ne sont pas encore définies.
Lorsque vous cliquez sur le bouton **Quitter**, la fenêtre principale s'efface.

Le module d'enregistrement :

Ce sera une fenêtre avec 6 boutons. Une zone de texte pour afficher le mode d'emploi dans la partie haute. Une autre zone de texte en bas de la fenêtre qui servira de barre d'état.

Comme ceci:



Le bouton **Moteur** démarre un enregistrement.
Le bouton **Stop** arrête l'enregistrement en cours.
Le bouton **Ecouter** permet d'entendre l'enregistrement en mémoire.
Le bouton **Garder** écrit l'enregistrement sur le disque dur et passe à l'enregistrement suivant.
Le bouton **Ecouter tout** permet d'entendre tous les enregistrements depuis le début.
Le bouton **Sortie** permet de quitter le module.

Entrons dans l'éditeur de MSWLogo : **Edall**.
Tapez toutes les procédures suivantes :

Point d'entrée du module d'enregistrement.

```
►to enregistrement
►demande_le_nom
►corrige_le_nom
►existe_deja?
►cree_dictee
►cree_fenetre_enregistrement
►end
```

Affichage de la boîte de dialogue d'enregistrement de fichiers. Seuls les fichiers portant l'extension .dictee sont listés.

```
►to demande_le_nom
►make "nom_dict dialogfilesave "*.dictee
►end
```

C'est le point qui permet de repérer l'extension dans le nom du fichier. Il faut s'assurer que le nom de la dictée porte bien l'extension .dictee. Pour cela, on supprime l'extension existante, (c'est plus facile que de la vérifier) et on ajoute l'extension attendue à la fin du nom.

```
►to corrige_le_nom
```

```
►make "nom_dict ajoute_extension :nom_dict
►end
```
word permet d'assembler 2 mots en un seul : **word "essai ".dictee** rend **"essai.dictee**. Pour Logo un mot commence toujours par des guillemets mais contrairement à d'autres langages de programmation, il n'y en a pas à la fin du mot.

```
►to ajoute_extension :mot
►if not memberp ". :mot [op word :mot ".dictee]
►op word enleve_extension :mot "dictee
►end
►to enleve_extension :mot
►if equalp last :mot ". [op :mot]
►op enleve_extension bl :mot
►end
```

Il faut vérifier que le nom choisi n'est pas celui d'une dictée existante. Si c'est le cas, retour à la case départ...
Lorsqu'une instruction est longue et qu'elle contient elle même plusieurs instructions, on peut créer **une instruction multilignes** comme c'est le cas ci dessous. Toute la partie du code :
if emptyp error ~
[~
messagebox "Erreur [Il existe déjà une dictée portant ce nom. Choisis un autre nom car il n'est pas possible de l'écraser.]~
closeall~
enregistrement~
stop~
]
est en réalité une seule ligne d'instructions pour Logo.
Pour indiquer à Logo qu'il a affaire à une instruction écrite sur plusieurs lignes, il faut placer le caractère **tilde** (~) à la fin de chaque ligne. Ce caractère s'obtient par la combinaison de touches **AltGr 2** suivi d'une frappe sur la touche **espace.**
catch (attrape) est une instruction qui permet de capturer une erreur : En temps normal, lorsque Logo rencontre une erreur dans une procédure, il arrête l'exécution du programme et affiche un message d'erreur. Catch permet de continuer l'exécution de la procédure malgré l'erreur. L'erreur est placée dans la variable **error**, ce qui permet son traitement.
messagebox est une instruction qui affiche un message à l'écran. L'exécution du programme est suspendue jusqu'à ce que l'utilisateur clique sur un des boutons OK ou Annuler.

```
►to existe_deja?
►catch "error [openread :nom_dict]
►if emptyp error ~
►  [~
►  messagebox "Erreur [Il existe déjà une dictée portant ce nom. Choisis
un autre nom car il n'est pas possible de l'écraser.]~
►  closeall~
►  enregistrement~
►  ]
►end
```

Création du fichier et du sous-répertoire :

```
►to cree_dictee
►;création du fichier vide
►openwrite :nom_dict
```

```
►setwrite :nom_dict
►print "
►setwrite []
►close :nom_dict
►;création du répertoire
►mkdir word :nom_dict "_fichiers
►end
```

Affichage de la fenêtre d'enregistrement.

make permet de d'affecter une valeur à une variable.
buttoncreate crée un bouton.
staticcreate crée une étiquette.
buttonenable permet d'activer ou de désactiver un bouton. Lorsqu'il est désactivé, le texte du bouton est en gris.
staticupdate permet de modifier le texte d'une étiquette existante.
setwrite permet d'indiquer le nom du fichier dans lequel on va écrire.

```
►to cree_fenetre_enregistrement
►windowcreate "ed_ph "magneto  [Enregistrer une dictée]  100 50 152 161
[objets_magneto]
►make "numero_enregistrement 1
►end

►to objets_magneto
►buttoncreate "magneto [moteur] [Moteur] 4 60 44 12 [clic_moteur_magneto]
►buttoncreate "magneto [stop] [Stop] 52 60 44 12 [clic_stop_magneto]
►buttoncreate "magneto [ecouter] [Ecouter] 100 60 44 12
clic_ecouter_magneto]
►staticcreate "magneto [texte1] [Cliquez sur Moteur, dictez votre phrase
puis cliquez sur Stop.] 4 4 146 16
►staticcreate "magneto [texte2] [Cliquez sur Ecouter pour vérifier
l'enregistrement. Si celui-ci vous convient, cliquez sur Garder.] 4 24
140 30
►staticcreate "magneto [etat] [] 4 136 140 8
►staticcreate "magneto [fragment] [] 4 128 140 8
►buttoncreate "magneto [garder] [Garder] 42 80 64 12
[clic_garder_magneto]
►buttoncreate "magneto [sortie] [Sortie] 42 112 64 12
[clic_sortie_magneto]
►buttoncreate "magneto [ecoutetout] [Ecouter tout] 42 96 64 12
[clic_ecoutetout_magneto]
►;gestion des boutons
►buttonenable "ecouter "false
►buttonenable "garder "false
►buttonenable "sortie "true
►buttonenable "moteur "true
►buttonenable "stop "false
►buttonenable "ecoutetout "false
►;actualiser la ligne d'état
►staticupdate "etat [Prêt à enregistrer.]
►end
```

Définissons maintenant les procédures qui sont exécutées lorsqu'on clique sur les boutons de la fenêtre.
ern permet d'effacer une variable.
ifelse (si sinon) est une instruction conditionnelle. Si l'expression qui suit est vraie, on exécute la première liste d'instructions, si elle est fausse, on exécute la seconde liste d'instructions.

localmake permet d'affecter une valeur à une variable locale, c'est à dire que cette variable n'est visible que par la procédure en cours et elle est effacée lorsque le programme sort de cette procédure alors que **make** permet d'affecter une valeur à une variable visible par tout le programme.
closeall ferme tous les fichiers ouverts. Si on tente d'ouvrir un fichier qui est déjà ouvert, une erreur se produit.
mci permet d'accéder aux fonctions multimédia de MSWLogo. **mci** est suivi d'une liste d'instructions.
yesnobox crée une boîte de dialogue où la réponse attendue est **oui** ou **non**. Si la réponse est oui, on exécute la première liste d'instructions, si la réponse est non, on exécute le seconde liste d'instructions.

```
►to clic_ecoutetout_magneto
►make "numero_fichier 0
►staticupdate "etat [Lecture en cours.]
►;gestion des boutons
►buttonenable "ecouter "false
►buttonenable "garder "false
►buttonenable "sortie "false
►buttonenable "moteur "false
►buttonenable "stop "false
►buttonenable "ecoutetout "false
►;ecoute...
►ecoute_tout
►ern "numero_fichier
►staticupdate "etat [Prêt à enregistrer.]
►;gestion des boutons
►ifelse il_y_a_un_enregistrement_en_memoire [buttonenable "ecouter "true]
[buttonenable "ecouter "false]
►ifelse il_y_a_un_enregistrement_en_memoire [buttonenable "garder "true]
[buttonenable "garder "false]
►buttonenable "sortie "true
►buttonenable "moteur "true
►buttonenable "stop "false
►buttonenable "ecoutetout "true
►end
```

stop permet de mettre fin à l'exécution de la procédure en cours. La procédure appelante poursuit son exécution. Dans l'exemple ci-dessous, s'il y a un message d'erreur, c'est que le fichier suivant n'a pas été trouvé, alors on arrête la lecture...

```
►to ecoute_tout
►make "numero_fichier :numero_fichier + 1
►;vérifier que le fichier existe
►catch "error [openread (word "Fragment :numero_fichier ".wav)]
►if not emptyp error [messagebox [Dictée] [Lecture terminée !] stop]
►closeall
►;lecture
►localmake  "rr mci (list "open (word "Fragment :numero_fichier ".wav)
"type "waveaudio "alias "sound)
►mci [seek sound to start]
►(mci [play sound notify])
►attends_la_fin_de_la_lecture "sound
►mci [close sound]
►;boucle
►ecoute_tout
►end
```

Lorsqu'une instruction multilignes contient plusieurs conditions imbriquées, on a intérêt à **indenter** le code pour améliorer sa lisibilité...

```
►to clic_sortie_magneto
►;avant de sortir, avertir l'utilisateur s'il y a un enregistrement en
mémoire
►if namep "record ~
►  [~
►  if il_y_a_un_enregistrement_en_memoire ~
►    [~
►    ifelse yesnobox [Attention][L'enregistrement en mémoire va être
perdu. Faut-il continuer ?]~
►      [~
►      mci [close mysound]~
►      ern "record ~
►      ]~
►      [~
►      stop~
►      ]~
►    ]~
►  ]
►windowdelete "magneto
►;retour à la case départ...
►ern "numero_enregistrement
►demarrage
►end
```

Une procédure qui rend un résultat est **une fonction**.
op est l'abréviation de **output** (rend).
namep est une fonction qui rend **true** si le mot qui suit est un nom, **false** dans le cas contraire.
not est la négation (non). **not namep** signifie « n'est pas un nom ».
equalp est une fonction qui rend **true** si les deux expressions qui suivent sont égales, **false** dans le cas contraire.

```
►to il_y_a_un_enregistrement_en_memoire
►;Fonction qui teste si un enregistrement est présent en mémoire.
►catch "error [localmake "longueur mci [status mysound length]]
►if not namep "longueur [op "false]
►ifelse equalp :longueur [0] [op "true] [op "false]
►end
```

La ligne **localmake "nomfich (word "fragment :numero_enregistrement ".wav)** détermine le nom de sauvegarde du fichier.
Ce nom est composé du mot **fragment** suivi du numéro d'enregistrement et de l'extension **.wav**

```
►to clic_garder_magneto
►;si l'enregistrement est vide, erreur !
►localmake "e mci [status mysound length]
►if  equalp :e 0 [messagebox [Erreur][L'enregistrement est vide !] stop]
►;enregistrer sur le disque le fragment en mémoire
►localmake "nomfich (word "fragment :numero_enregistrement ".wav)
►mci (list "save "mysound  :nomfich)
►mci [close mysound]
►ern "record
►;incrémenter le compteur de fragments
►make "numero_enregistrement :numero_enregistrement + 1
►;gestion des boutons
```

```
►buttonenable "ecouter "false
►buttonenable "garder "false
►buttonenable "sortie "true
►buttonenable "moteur "true
►buttonenable "stop "false
►buttonenable "ecoutetout "true
►;actualiser la ligne d'état
►staticupdate "etat [Prêt à enregistrer.]
►end

►to clic_ecouter_magneto
►;gestion des boutons
►buttonenable "ecouter "false
►buttonenable "garder "false
►buttonenable "sortie "false
►buttonenable "moteur "false
►buttonenable "stop "false
►buttonenable "ecoutetout "false
►;actualiser la ligne d'état
►staticupdate "etat [Lecture en cours.]
►;démarrage de la lecture
►mci [cue mysound output]
►mci [seek mysound to start]
►mci [play mysound]
►attends_la_fin_de_la_lecture "mysound
►;gestion des boutons
►buttonenable "ecouter "true
►buttonenable "garder "true
►buttonenable "sortie "true
►buttonenable "moteur "true
►buttonenable "stop "false
►buttonenable "ecoutetout "true
►;actualiser la ligne d'état
►staticupdate "etat [Prêt à enregistrer.]
►end
```

La procédure **attends_la_fin_de_la_lecture** fait **une boucle** jusqu'à ce que la variable **:etat** contienne la valeur **"stopped.**

```
►to attends_la_fin_de_la_lecture :alias
►;fait une boucle jusqu'à ce que le fichier .wav soit joué en entier
►localmake "etat mci (list "status :alias "mode)
►if not equalp :etat [stopped] [attends_la_fin_de_la_lecture :alias]
►end

►to clic_stop_magneto
►mci [stop mysound]
►;gestion des boutons
►buttonenable "ecouter "true
►buttonenable "garder "true
►buttonenable "sortie "true
►buttonenable "moteur "true
►buttonenable "stop "false
►buttonenable "ecoutetout "true
►;actualiser la ligne d'état
►staticupdate "etat [Prêt à enregistrer.]
►end
```

Quelques mot sur les commandes multimédia (MCI) :
open permet d'ouvrir un périphérique MCI, **close** le ferme. Le périphérique doit avoir un nom (un **alias,** ici je l'ai appelé mysound), il est possible d'en ouvrir plusieurs simultanément.
Logo possède de nombreux périphériques MCI capables de lire des fichiers son, des CD audio, des fichiers .avi, des disques vidéo, des séquences midi... Il faut indiquer à Logo le type de ce périphérique MCI , **waveaudio** pour le son**.**
Il faut aussi indiquer à Logo si on utilise le périphérique MCI pour enregistrer ou pour restituer :
cue ... input pour utiliser un périphérique d'entrée, ici le microphone.
cue ... output pour utiliser un périphérique de sortie, ici le haut parleur.
Les commandes maintenant :
seek ... to start pour positionner la bande au début...
record pour enregistrer.
play pour lire un enregistrement.
save pour enregistrer sur le disque.
status pour connaître l'état du périphérique MCI.

```
►to clic_moteur_magneto
►;gestion des boutons
►buttonenable "ecouter "false
►buttonenable "garder "false
►buttonenable "sortie "false
►buttonenable "moteur "false
►buttonenable "stop "true
►buttonenable "ecoutetout "false
►;ouverture du périphérique audio
►if namep "record [mci [close mysound]]
►localmake "uu mci [open new type waveaudio alias mysound buffer 6]
►;démarrage de l'enregistrement
►make "record "true
►mci [cue mysound input]
►mci [record mysound]
►;actualiser les lignes d'état
►staticupdate "etat [Enregistrement en cours.]
►staticupdate "fragment se [Enregistrement numéro] :numero_enregistrement
►end
```

Il est temps de tester ce module. Tapons **enregistrement** <Entrée> dans le Commander. Si tout est correct, on voit d'abord s'afficher la fenêtre de sauvegarde de fichier. Tapons un nom de dictée avec une extension volontairement erronée pour tester le dispositif de correction du nom, par exemple essai.dic puis cliquons sur **Enregistrer**. La fenêtre Enregistrer une dictée doit normalement apparaître.
Vérifions que le fichier **essai.dictee** et le sous-répertoire **essai.dictee_fichiers** ont bien été créés.

Testez le fonctionnement du module en dictant quelques phrases. N'oubliez pas de cliquer sur le bouton **Garder** pour enregistrer chaque phrase sur le disque dur ! Testez bien tous les boutons. Si un message d'erreur s'affiche dans le Commander, c'est qu'il y a une erreur. Lisez attentivement les messages d'erreurs, ils contiennent les informations qui vous permettront de localiser l'erreur. Pour chaque erreur, retournez dans l'éditeur avec le bouton **Edall**, corrigez-la et enregistrer les modifications en mémoire en cliquant sur le menu **File / Save and Exit** de l'éditeur.

Tout fonctionne correctement ? C'est parfait, passons maintenant à la programmation du module de lecture de notre dictée.

Le module de lecture :

Nous voulons lui donner cette apparence:

Ecouter pour réécouter le fragment en cours.
Flèche à droite pour écouter le fragment suivant.
Flèche à gauche pour revenir au fragment précédent.
Flèche en butée vers la gauche pour revenir au début de la dictée.
Sortie pour revenir à l'écran principal de choix de l'activité.
La ligne d'état indique en permanence l'état du lecteur.

Le point d'entrée dans ce module est la procédure **choix_dictée**. Elle affiche d'abord la boîte de dialogue d'ouverture de fichier. Quand le nom est saisi, elle vérifie que la dictée existe, qu'il y a au moins un enregistrement dans le sous-répertoire correspondant et enfin, elle affiche la fenêtre de lecture de la dictée.
emptyp est une fonction qui rend **true** si la variable qui suit est une liste vide, **false** dans le cas contraire.
chdir permet de se placer dans un répertoire déterminé.
openread ouvre un fichier en lecture.
bye fait quitter Logo (si on clique sur le bouton Annuler du dialogue d'ouverture de fichier).

```
►to choix_dictee
►make "nom_dict dialogfileopen "*.dictee
►if emptyp :nom_dict [bye]
►;vérifier que la dictée existe :
►catch "error [openread :nom_dict]
►if not emptyp error [messagebox [:nom_dict] [N'existe pas !] stop]
►chdir word :nom_dict "_fichiers
►;attention, si le répertoire n'existe pas, il y aura affichage d'un
message d'erreur MCI à l'écran.
►;C'est tout bon !
►make "maxi compte_fichiers 1
►if equalp :maxi 0 [messagebox [Erreur !][Cette dictée ne comporte aucun
enregistrement.] choix_dictee stop]
►make "numero_enregistrement 1
►window_dictaphone
►end
```

La procédure compte_fichiers est une fonction qui rend le nombre de fichiers .wav contenu dans le répertoire.

```
►to compte_fichiers :n
►catch "error [openread (word "Fragment :n ".wav) close (word "Fragment :
n ".wav)]
►if not emptyp error [closeall op :n-1]
►op compte_fichiers :n + 1
►end
```

Création de la fenêtre de lecture :

```
►to Window_dictaphone
►windowcreate "root "dictaphone  [Faire une dictée]  0 0 168 46
[objets_dictaphone]
►end
```

Création des objets qu'elle contient :

```
►to objets_dictaphone
►buttoncreate "dictaphone [ecouter] [Ecouter] 4 4 30 12
[clic_ecouter_dictaphone]
►buttoncreate "dictaphone [precedent] [<---] 36 4 30 12
[clic_precedent_dictaphone]
►buttoncreate "dictaphone [suivant] [---►] 68 4 30 12
[clic_suivant_dictaphone]
►buttoncreate "dictaphone [debut] [\|<---] 100 4 30 12
[clic_debut_dictaphone]
►buttoncreate "dictaphone [sortie] [Sortie] 132 4 30 12
[clic_sortie_dictaphone]
►staticcreate "dictaphone "etat [] 5 20 156 8
►staticupdate "etat [Prêt...]
►buttonenable "suivant "false
►buttonenable "debut "false
►buttonenable "precedent "false
►end
```

Clic sur le bouton **Ecouter** :

```
►to clic_ecouter_dictaphone
►catch "error [joue_fichier_audio (word "Fragment :numero_enregistrement
".wav)]
►;gestion des boutons
►ifelse emptyp error ~
►  [~
►  buttonenable "ecouter "false~
►  buttonenable "suivant "false~
►  buttonenable "debut "false~
►  buttonenable "sortie "false~
►  buttonenable "precedent "false~
►  staticupdate "etat [Lecture...]~
►  ]~
►  [~
►  buttonenable "ecouter "false~
►  buttonenable "suivant "false~
►  buttonenable "debut "true~
►  buttonenable "sortie "true~
►  buttonenable "precedent "true~
►  stop
►  ]
►attends_la_fin_de_la_lecture "mysound
►staticupdate "etat [Prêt...]
►mci [close mysound]
►;gestion des boutons
►buttonenable "ecouter "true ;bouton écouter
►ifelse lessp :numero_enregistrement :maxi [buttonenable "suivant "true]
[buttonenable "suivant "false]
►ifelse greaterp :numero_enregistrement 1 [buttonenable "debut "true]
```

```
[buttonenable "debut "false]
►buttonenable "sortie "true
►ifelse greaterp :numero_enregistrement 1 [buttonenable "precedent "true]
[buttonenable "precedent "false]
►if equalp :numero_enregistrement :maxi [messagebox [Dictée][La dictée
est terminée.]]
►end
```

clic sur suivant...

```
►to clic_suivant_dictaphone
►make "numero_enregistrement :numero_enregistrement + 1
►;tester si le fichier est lisible
►catch "error [openread (word "Fragment :numero_enregistrement ".wav)]
►if not emptyp error ~
►  [~
►  buttonenable "ecouter "false~
►  buttonenable "suivant "false~
►  buttonenable "debut "true~
►  buttonenable "sortie "true~
►  buttonenable "precedent "true~
►  make "numero_enregistrement :numero_enregistrement - 1~
►  messagebox [Dictée] [Erreur de lecture !]~
►  stop~
►  ]~
►closeall
►staticupdate "etat [Lecture...]
►clic_ecouter_dictaphone
►end
```

clic sur précédent...

```
►to clic_precedent_dictaphone
►make "numero_enregistrement :numero_enregistrement - 1
►staticupdate "etat [Lecture...]
►clic_ecouter_dictaphone
►end
```

clic sur début...

```
►to clic_debut_dictaphone
►make "numero_enregistrement 1
►;lecture
►staticupdate "etat [Lecture...]
►clic_ecouter_dictaphone
►end
```

clic sur **Sortie**...

```
►to clic_sortie_dictaphone
►windowdelete "dictaphone
►ern "numero_enregistrement
►demarrage
►end
```

La procédure **joue_fichier_audio** lit le fichier dont le nom est contenu dans la variable **:fich.**

```
►to joue_fichier_audio :fich
►localmake  "rr mci (list "open :fich "type "waveaudio "alias "mysound)
►mci [seek mysound to start]
```

```
►mci [play mysound notify]
►end
```

Voilà, notre programme est complet, il n'est pas si long... Il ne vous reste plus qu'à tester le module de lecture pour vérifier que tout fonctionne :

Tapez **demarrage** <Entrée> dans le Commander puis choisissez l'activité **Faire une dictée**. Choisissez la dictée **"essai.dictee** que nous avons enregistrée lors de l'essai du module d'enregistrement. Vous devez entendre ce que vous avez enregistré. Testez les autres boutons en surveillant les messages d'erreur qui pourraient s'afficher dans le Commander.
Tout fonctionne parfaitement ? C'est très bien.

Maintenant que notre programme fonctionne, que nous sonmmes certains qu'il n'y a plus d'erreurs, nous allons modifier quelques procédures pour rendre notre programme indépendant des fenêtres MSWLogo Screen et Commander. Nous allons tout simplement supprimer ces fenêtres. Si nous ne l'avons pas fait dès le départ, c'est parce que nous avions besoin de voir les messages d'erreur du Commander et que nous voulions conserver le contrôle de Logo. Maintenant que nous avons longuement testé notre programme et qu'il ne semble plus contenir d'erreurs nous pouvons le faire sans risque. Pour cela, éditons la procédure demarrage dans l'éditeur et ajoutons ces 2 lignes :

```
►to demarrage
►windowcreate "root "demarrage [Dictaphone] 100 60  115 90
[objects_demarrage]
►windowset [MSWLogo Screen] 0
►windowset "Commander 0
►end
```

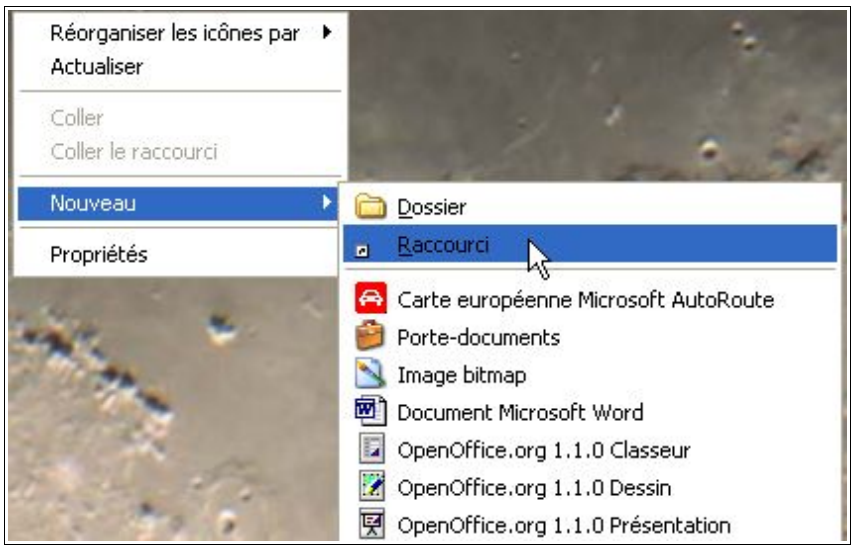
windowset permet de modifier l'état d'une fenêtre affichée à l'écran. En fixant la valeur à 0, on la rend invisible (pour les valeurs de ce paramètre, voir l'aide de MSWLogo). On donne le titre de la fenêtre en paramètre, en faisant bien la différence entre majuscules et minuscules. Si deux fenêtres affichées à l'écran ont le même titre, vous imaginez tout de suite que cela pose problème. Notez que cette instruction fonctionne également avec des fenêtres étrangères à Logo. Vous pouvez, à partir de Logo, masquer une fenêtre Notepad ou Open Office sans problème !

Nous allons maintenant faire en sorte que le programme démarre automatiquement dès son chargement en mémoire, vous vous souvenez comment faire ?

Tapez **make "startup "demarrage** <Entrée> dans le Commander puis cliquez sur le menu **File / Save.**
Saisissez le nom du programme : **dictaphone.lgo** puis cliquez sur **Enregistrer.**

Placer un raccourci sur le bureau

Nous allons créer un raccourci que nous placerons sur le bureau de Windows ou dans le menu **Démarrer** : Cliquons avec bouton droit de la souris sur le bureau de manière à créer un nouveau raccourci.

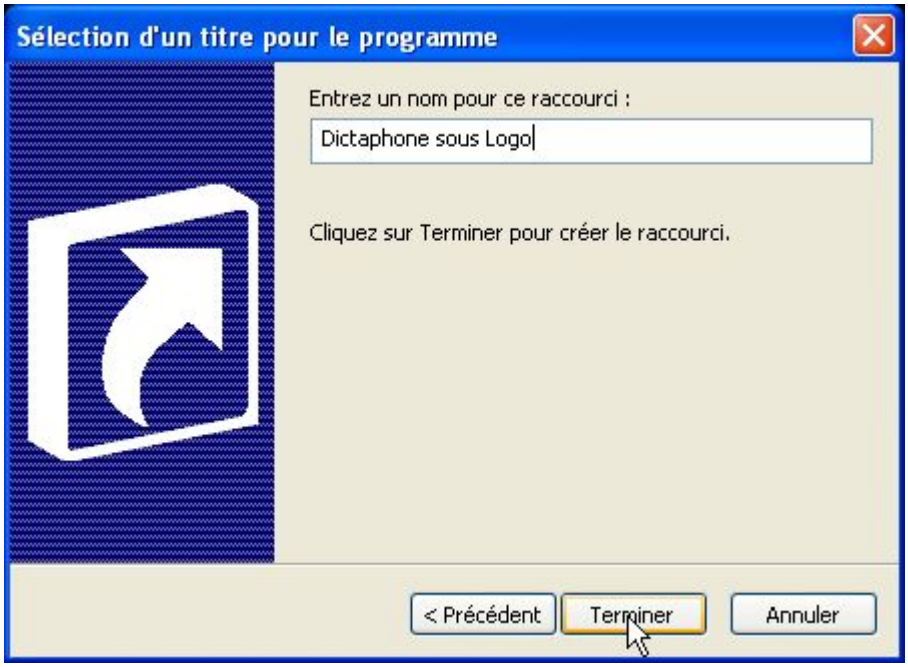


Cette fenêtre de recherche de dossier apparaît :

Choisissons **Microsoft Windows Logo**. Le chemin menant à l'exécutable de MSWLogo est alors

affiché dans la zone de saisie de la boîte de dialogue. Ajoutons **`-l dictaphone.lgo`** (**l** pour **load**) à la suite de ce chemin (il faut un espace entre Logo32.exe et -l) et cliquons sur **Suivant**.

Maintenant indiquons le nom de ce raccourci en remplaçant Logo32.exe par **Dictaphone sous Logo** et cliquons sur **Terminer**.



Le nouveau raccourci **Dictaphone sous Logo** avec l'icône de MSWLogo apparaît sur le bureau.
Lorsqu'on fait un double clic sur ce raccourci, Logo démarre et charge le programme dictaphone.lgo dans la foulée.
Vous pouvez copier ce raccourci et en placer une copie dans le menu démarrer.
C'est terminé !

Et si ça ne marche pas ?

Voici quelques messages d'erreurs que vous risquez de rencontrer :

*Dans l'éditeur Logo, lorsque vous cliquez sur le menu **File / Save and Exit**, vous obtenez le message d'erreur suivant:*

C'est probablement parce qu'il manque un crochet ou une parenthèse dans une ligne. Regardez tout de même dans le Commander si un message d'erreur est présent. Cliquez sur OK pour retourner dans l'éditeur. L'erreur est située dans la procédure qui se trouve juste après le curseur de texte. Corrigez et recommencez.

Voici quelques messages d'erreurs que Logo risque d'afficher dans le Commander et leurs causes possibles :

***I don't know how to do...** c'est sans doute parce que vous avez fait une faute de frappe en tapant le nom d'une procédure ou d'une primitive. Logo a rencontré un mot qu'il ne connaît pas. Cherchez et corrigez le mot mal orthographié et recommencez.*

***Not enough inputs to ...** c'est sans doute parce que vous avez oublié de passer un paramètre à la procédure ou à la primitive (c'est le cas quand on tape **forward** en omettant le nombre de pas).*

***... doesn't like ... as input,** c'est sans doute parce que vous avez passé un paramètre qui ne convient pas à une procédure ou à une primitive (c'est le cas quand on tape **forward** "droite car forward attend un nombre, pas un mot).*

***Too much inside ( )'s** ou bien**Unexpected ')'** vous indique qu'il y a un problème de correspondance entre les parenthèses.*

***... is already defined,** lorsque vous donnez un nom de primitive à une de vos procédures, changez de nom !*

***Assuming you mean IFELSE, not IF**, quand vous avez employé IF à la place de IFELSE. C'est seulement une mise en garde qui n'interrompt pas l'exécution de la procédure.*

***...has no value**, lorsqu'une variable qui n'a pas été définie est invoquée dans une procédure. Il faut d'abord définir la variable au moyen de **make** ou de **localmake**.*

***Can't use TO inside a procedure** quand on utilise le mot réservé **to** à l'intérieur d'une procédure ou qu'on oublie de préciser le nom de la procédure dans l'éditeur.*

***... didn't output to ...** lorsque vous passez le nom d'une procédure qui n'est pas une fonction comme paramètre à une procédure qui a besoin d'un paramètre (forward right 90 par exemple, forward attend un nombre comme paramètre et trouve right qui ne rend rien du tout à la place...)*

***I don't know what to do with ...** Losque Logo trouve une valeur dont il ne sait que faire dans une ligne (vous avez tapé **forward 20 90** par exemple, forward attend un nombre, 20 mais Logo ne sait pas ce qu'il doit faire avec le 90...)*

***File system error** lorsque Logo échoue lors d'une opération sur les fichiers. C'est peut-être tout*

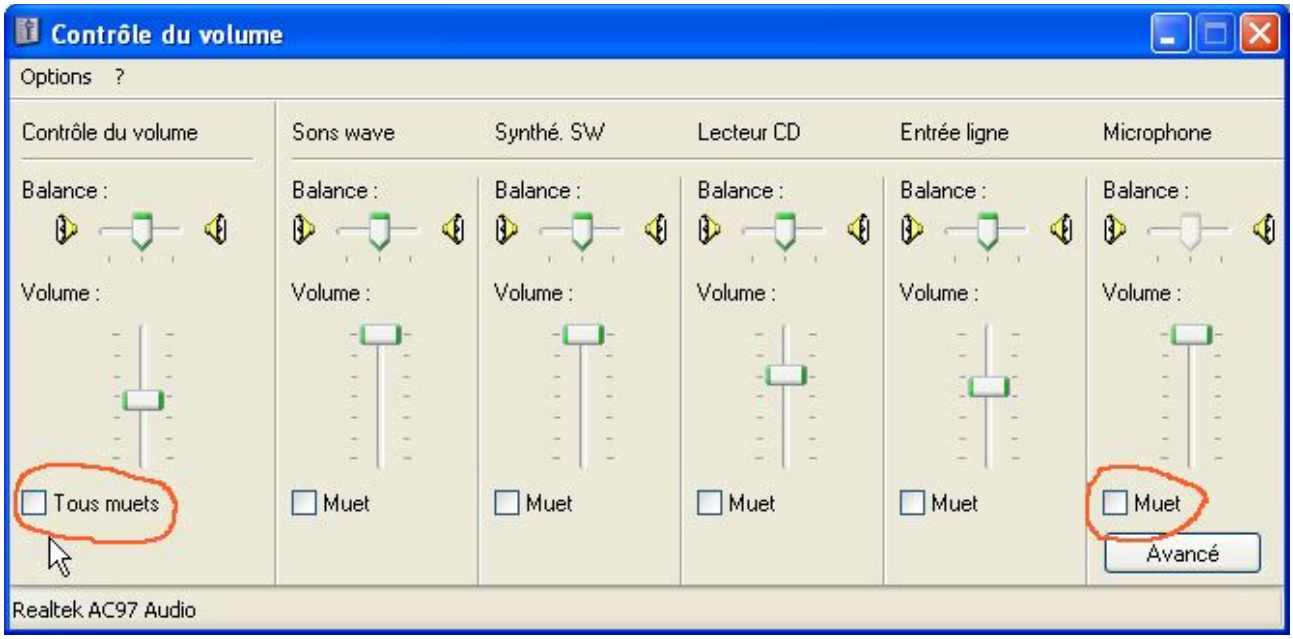
simplement parce que vous essayez d'ouvrir un fichier déjà ouvert lors d'une précédente tentative. Tapez ***closeall*** *et essayez à nouveau.*

Ce ne sont pas les seuls messages d'erreur de Logo, mais ce sont ceux que vous risquez de rencontrer en tapant ce programme. La liste complète des messages se trouve dans l'aide de MSWLogo.

Si vous n'arrivez pas à obtenir de son :
Normalement, si vous arrivez à enregistrer et à restituer des sons avec le magnétophone de Windows, le dictaphone devrait fonctionner correctement. Cependant si aucun son ne sort:

- Vérifiez que les hauts parleurs ne sont pas coupés.
- Vérifiez que le volume du son est suffisant.

Ouvrez la fenêtre ***Contrôle du volume du son*** *:*
- Vérifiez que le volume du ***Microphone*** *n'est pas à 0.*
- Vérifiez que l'option Muet du ***Microphone*** *n'est pas cochée.*
- Vérifiez que l'option Muet du ***Contrôle du volume*** *n'est pas cochée.*



Bien sûr, dans ces quelques pages nous n'avons fait qu'effleurer les possibilités de MSWLogo, car nous n'avons guère utilisé qu'une quarantaine de primitives différentes (MSWLogo en possède plus de 300 au total).

J'espère que ce document aura éveillé votre curiosité et qu'il vous aura donné l'envie d'aller plus loin.

Amusez-vous bien et bienvenue au club !

Si vous n'arrivez pas à faire fonctionner le dictaphone ou si vous voulez l'utiliser mais que vous n'avez pas le temps de le taper, il est disponible au téléchargement sur mon site internet à l'adresse suivante :

http://jeannoel.saillet.free.fr/download/dictaphone.zip

C'est un minuscule fichier zip de 3 Ko (petit mais costaud !) qu'il faut décompresser dans le répertoire qui contient l'exécutable de MSWLogo (Logo32.exe).