

Apprendre les bases de la programmation
avec un dérivé de LOGO ou se perfectionner
en bâtissant l'interpréteur de A à Z...

GVLOGO

Réaliser un interpréteur en
Pascal V1.0.0 - 07/08/2014

Gilles Vasseur © 2014

TABLE DES MATIERES

Le projet	4
 Les objectifs du projet	4
 Qu'est-ce que GVLOGO ?.....	4
 En quoi ce projet peut-il être utile ?	4
 Plan de l'ouvrage	5
 Conventions.....	5
 Me contacter	5
Les objets de GVLOGO	6
 Les mots	6
Définitions	6
Exemples de mots.....	6
Opérations sur les mots	7
Implémentation des mots.....	10
 Les listes.....	18
Définitions	18
Exemples de listes	18
Opérations sur les listes	18
Implémentation des listes.....	22
 Les listes de propriétés.....	27
Définitions	27
Exemples de listes de propriétés	28
Opérations sur les listes de propriétés	28
Implémentation des listes de propriétés.....	29
La tortue graphique.....	33
Présentation	33
Opérations avec la tortue.....	34
Implémentation de la tortue.....	41
Récréation : EasyTurtle (logiciel de dessin).....	54
 Le projet EasyTurtle.....	54
 Mode d'emploi rapide.....	54
L'écran d'accueil	54
La tortue	55
Couleurs et formes	56
Ordres généraux	56
L'aide.....	57
Boîte « À propos »	57
Boîte des préférences.....	58

Autres éléments	58
La programmation.....	59
La fiche principale	60
Les autres fiches	65
Les outils de programmation	66
Les piles.....	66
Définition	66
Exemples.....	66
Opérations sur les piles (et les queues)	69
Implémentation des piles.....	70
L'évaluation d'une expression mathématique.....	76
Définition	76
Opérations dans l'évaluateur.....	76
Implémentation de l'évaluateur	76
Les composants du langage	77
Les variables	77
Les procédures	77
Les paquets	77
Le noyau	77
Les primitives	77
Le programme final	78
L'interpréteur	78
L'interface utilisateur	78
Mode d'emploi de GVLOGO.....	79
Installer GVLOGO	79
Interface et menus	79
Créer un programme.....	79
Exécuter un programme	79
Sauvegarder et récupérer un programme	79
Modifier un programme.....	79
Les messages d'erreur.....	79
Le débogage	79
Mettre à jour le logiciel	79

Programmes exemples.....	80
Travailler avec la souris	80
Travailler avec les listes.....	80
Travailler avec les listes de propriétés	80
Manipuler l'espace de travail.....	80
Licence GNU	81
Table des figures	94

LE PROJET

LES OBJECTIFS DU PROJET

Le projet **GVLOGO** est né au début de cette année 2014. Il s'agissait d'implémenter le langage LOGO en Pascal grâce à Lazarus (support de Free Pascal) ou Delphi (diffusé par Embarcadero).

À présent que l'écriture du logiciel est terminée, l'objectif est double :

- ✓ offrir un langage de programmation adapté à des enfants, mais dont les possibilités sont étendues grâce au traitement des listes ;
- ✓ proposer aux programmeurs ou apprentis programmeurs la réalisation complète d'un interpréteur et de son environnement à l'aide d'outils gratuits ou payants stables et facilement disponibles.

QU'EST-CE QUE GVLOGO ?

GVLOGO est un langage de programmation. Il descend de LOGO, lui-même apparu dans les années 60 à la suite de recherches menées par des universitaires du MIT (U.S.A.) autour du mathématicien Seymour Papert. S'appuyant sur des travaux de Piaget relatifs à l'acquisition de connaissances et de méthodes chez l'enfant, la visée était pédagogique : l'objectif essentiel était de rendre l'élève acteur de son apprentissage y compris et surtout lorsque cet apprentissage devait passer par un ordinateur. En cela, il s'opposait à l'Enseignement Assisté par Ordinateur alors en vogue qui se contentait le plus souvent d'un scénario figé.

« Dans bien des écoles, aujourd'hui, l'expression "Enseignement Assisté par Ordinateur" signifie que l'ordinateur est programmé pour enseigner à l'enfant. On pourrait dire que l'ordinateur sert à programmer l'enfant. Dans ma vision des choses, l'enfant programme l'ordinateur et, ce faisant, acquiert la maîtrise de l'un des éléments de la technologie la plus moderne et la plus puissante, tout en établissant un contact intime avec certaines des notions les plus profondes de la science, des mathématiques, et de l'art de bâtir des modèles intellectuels. »

(S. Papert - [JAILLISSEMENT DE L'ESPRIT, ORDINATEURS ET APPRENTISSAGE](#), Flammarion 1981)

Peut-être avez-vous vous-même utilisé ce langage au cours de votre scolarité en pilotant sur l'écran de l'ordinateur un triangle (baptisé "tortue") grâce à une série de commandes telles que : **AVANCE, RECULE, GAUCHE, DROITE...** Comme cette tortue pouvait laisser une trace de son passage, vous obteniez des dessins en programmant ses déplacements. Beaucoup plus tard, peut-être avez-vous aussi manipulé les listes LOGO pour traiter des problèmes d'intelligence artificielle... Et quand bien même vous n'auriez rien fait de tout cela, peu importe, car programmer en LOGO est très simple !

EN QUOI CE PROJET PEUT-IL ETRE UTILE ?

Si vous ne connaissez rien à la programmation, **GVLOGO** vous initiera à ses grands principes.

Si vous êtes en contact avec des enfants ou des adolescents, **GVLOGO** leur permettra une première approche de la programmation tout en développant leur sens logique, leur représentation dans l'espace et leur capacité à résoudre des problèmes par eux-mêmes.

Si vous êtes programmeur ou si vous aspirez à l'être, vous pourrez être intéressé par le logiciel lui-même dont les programmes sources sont fournis et largement commentés. S'il ne s'agit pas d'un

tutoriel d'apprentissage de la programmation, ce développement met en œuvre de nombreuses techniques *in vivo*. Le langage de programmation choisi est un dérivé de PASCAL (Free Pascal sous Lazarus) qui est connu pour sa grande lisibilité et sa puissance. Pascal a lui aussi des vertus pédagogiques ! De plus, il fonctionne sur de nombreuses plates-formes (dont Windows et Linux) et est gratuit. Les sources sont par ailleurs exploitables avec Delphi, un autre environnement plus puissant que le premier mais payant.

PLAN DE L'OUVRAGE

La réalisation de **GVLOGO** est modulaire. Dans un premier temps seront étudiés les objets **GVLOGO** et leur implémentation : les **mots**, les **listes**, les **listes de propriétés** et la **tortue graphique**. Les objets sont en quelque sorte les briques qui fondent le langage.

L'étape suivante consistera à implémenter les outils nécessaires à l'interpréteur, à savoir les **piles** et un **évaluateur d'expressions mathématiques**.

Une place conséquente sera alors consacrée au **noyau**. Comme la mission de ce dernier est de gérer le **contenu des éditeurs**, les **variables**, les **procédures** et les **paquets**, tous ces éléments seront tour à tour étudiés. Pour clore ce qui se rapporte aux fonctionnalités, les **primitives** seront regroupées dans leur propre unité.

Viendra alors l'**interpréteur** lui-même dont la tâche sera grandement simplifiée par tout le travail en amont. Il restera à s'occuper de l'architecture de l'**interface**, depuis l'**éditeur** jusqu'au **débogage**, en passant par la **zone de saisie**, celle de **texte** et celle réservée à la **tortue**.

Enfin, une fois le logiciel opérationnel, un mode d'emploi du logiciel créé et des **exemples** écrits en **GVLOGO** illustreront quelques-unes de ses capacités.

CONVENTIONS

Les éléments de programmation en **GVLOGO** sont écrits en **bleu**.

Les éléments de programmation de Pascal, en dehors des listings sont écrits en **rouge**.

Les unités utilisées ou créées sont en **noir gras**.



Ce pictogramme indique une astuce, un point intéressant pour faciliter la programmation.



Ce pictogramme indique un point à souligner.



Ce pictogramme attire l'attention sur un point sensible, une difficulté particulière.

ME CONTACTER

Il existe sans doute des méthodes plus efficaces que celles que j'ai employées pour résoudre certains problèmes liés à l'implémentation d'un interpréteur LOGO, mais je tenterai de justifier mes choix. Certains aspects du programme sont à améliorer : je pense notamment à la gestion des couleurs et du fond de la surface de dessin par la tortue graphique.

Il vous sera toujours possible de poser des questions, de critiquer les réalisations et de proposer des solutions plus élégantes que les miennes !

Pour cela, vous pouvez envoyer un mail à : gillesvasseur58@gmail.com

Vous pouvez aussi me joindre via mon site personnel : www.lettresenstock.org

Bonne lecture, bonne réflexion... et bon LOGO !

Gilles Vasseur, jeudi 7 août 2014.

LES OBJETS DE GVLOGO

LES MOTS

DEFINITIONS

GVLOGO travaille essentiellement à partir de mots et de listes. Ce chapitre se propose d'étudier les mots.

Un *mot* est une suite quelconque de caractères.

Un mot est délimité par un espace, les signes [,], (,). Ces cinq caractères particuliers sont appelés des *délimiteurs* : ils indiquent à **GVLOGO** comment séparer les éléments du langage.

Afin d'inclure un délimiteur dans un mot, on le fait précéder d'un autre caractère appelé *caractère d'échappement* : c'est le \$¹.



Il existe un mot particulier qui ne comprend aucun caractère : on l'appelle le *mot vide*. On le représente par un guillemet anglais suivi d'un espace.



Les mots "VRAI" et "FAUX", qui peuvent s'écrire simplement **VRAI** et **FAUX**, sont réservés au renvoi de valeurs booléennes.

EXEMPLES DE MOTS

MOTS SIMPLES

Voici des mots simples qui ne poseront pas de problèmes particuliers :

- ✓ libellule
- ✓ porte-avions
- ✓ entendre
- ✓ EnTenDRe

Les nombres ne sont que des mots particuliers qui seront traités comme des mots ordinaires ou comme des nombres suivant l'opération en cours :

- ✓ 1245
- ✓ 1E2²
- ✓ 13456,58

Les mots accentués et les caractères exotiques sont utilisables eux aussi³ :

- ✓ éléphant
- ✓ %&/ç\

¹ Si ce caractère d'échappement doit apparaître en tant que tel dans un mot, il faut le faire précéder de lui-même !

² Il s'agit de la notation scientifique pour 10^2 , soit 100.

³ Les caractères particuliers ne sont pas toujours bien gérés par les langages et les systèmes d'exploitation. Les normes se sont multipliées et des hiéroglyphes étranges peuvent parfois apparaître. Nous reviendrons à ce problème lorsque nous aborderons l'implémentation des mots en Pascal.

MOTS AVEC CARACTÈRE D'ÉCHAPPEMENT

Un caractère particulier peut toujours être introduit dans un mot s'il est précédé du caractère d'échappement :

- ✓ Victor\$ Hugo
- ✓ GVLOGO\$ est\$ un\$ langage\$ facile\$ à\$ apprendre
- ✓ \$[encore\$ un\$ mot\$]
- ✓ \$(un\$ autre\$)
- ✓ 15\$ \$\$

Lors de l'écriture des mots ainsi construits, le caractère d'échappement, sauf contre-ordre, n'apparaîtra pas à l'écran :

- ✓ [ECRIS \\$\[encore\\$ un\\$ mot\\$\]](#) donnera [encore un mot]
- ✓ [ECRIS 15\\$ \\$\\$](#) donnera 15 \$



Dès qu'une suite de mots est en jeu, l'emploi des listes est préconisé. Le caractère d'échappement est une commodité, sans plus. Il est plus facile d'écrire et de comprendre [ECRIS \[15 \\$\]](#) que [ECRIS 15\\$ \\$\\$](#).

AUTRES CARACTÈRES PARTICULIERS

En plus des délimiteurs, certains caractères sont utilisés par le langage lui-même et déterminent son comportement selon le contexte⁴ :

- ✓ le " (guillemet anglais) indique que le mot qui suit doit être pris tel quel, sans chercher à l'interpréter (par exemple, [ECRIS "Bonjour!"](#)) ;
- ✓ les : (deux-points) indiquent que le mot qui suit est une variable dont la valeur est à rechercher (par exemple, [ECRIS :salutation](#) affichera, si elle existe, la valeur qui correspond à la variable « salutation ») ;
- ✓ le . (point) en début de mot indique une primitive dont l'emploi est risqué, car elle touche au cœur de **GVLOGO** (par exemple [.EFFACETOUT](#) détruit tous les objets en réinitialisant le noyau).



Par ailleurs, le ? (point d'interrogation) est utilisé à la fin des primitives qui renvoient une valeur booléenne ([VRAI/FAUX](#)), mais c'est une habitude qui n'a pas de caractère obligatoire.

OPÉRATIONS SUR LES MOTS

La liste ci-après comprend les primitives connues de **GVLOGO** qui concernent les mots. Pour chaque primitive, avant sa définition, il est indiqué le nombre de paramètres attendus, leurs types et le type de valeur qu'elle renvoie. Un exemple illustre l'emploi de la primitive décrite.

Les primitives énumérées peuvent évidemment se combiner pour obtenir le résultat escompté : c'est même ainsi qu'elles prennent tout leur intérêt ! Nous y reviendrons dans la partie qui traitera de la réalisation de programmes en **GVLOGO**.

FABRIQUER DES MOTS

- **METSPREMIER** (raccourci : [MP](#)) : attend deux mots en entrée – renvoie un mot – le mot rendu est composé du premier paramètre placé avant le second.

Exemple :

[ECRIS METSPREMIER "tourne "dos](#) → tournedos

⁴ Ces caractères viennent compléter les délimiteurs déjà vus, mais ce ne sont pas des délimiteurs eux-mêmes.

- **METSDERNIER** (raccourci : **MD**) : attend deux mots en entrée – renvoie un mot – le mot rendu est composé du premier paramètre placé après le second.

Exemple :

ECRIS METSDERNIER "tourne "dé → détourne

- **MOT** : attend deux mots en entrée – renvoie un mot – le mot rendu est composé du second paramètre placé après le premier.

Exemple :

ECRIS MOT "dé "tourne → détourne



Il s'agit donc d'un synonyme de **METSDERNIER**. Cependant, ce dernier fonctionne aussi avec les listes alors que **MOT** déclenchera une erreur si une liste est donnée en entrée.

MODIFIER DES MOTS

- **INSERE** : attend un entier suivi deux mots en entrée – renvoie un mot – Le mot rendu est composé du deuxième paramètre inséré à la position précisée par l'entier dans le dernier mot.

Exemple :

ECRIS INSERE 4 "en "atttion → attention

- **INVERSE** : attend un mot en entrée – renvoie un mot – le mot rendu est celui d'entrée dont les lettres ont été inversées.

Exemple :

ECRIS INVERSE "billet → tellib

- **MAJUSCULES** : attend un mot en entrée – renvoie un mot – le mot rendu est en majuscules.

Exemple :

ECRIS MAJUCULES "éléphant → ÉLÉPHANT

- **MELANGE** : attend un mot en entrée – renvoie un mot – le mot rendu a les mêmes lettres que celui d'origine, mais dans un ordre aléatoire.

Exemple :

ECRIS MELANGE "alouette → eoetault

- **MINUSCULES** : attend un mot en entrée – renvoie un mot – le mot rendu est en minuscules.

Exemple :

ECRIS MINUSCULES "ESSAI → essai

- **REPLACE** : attend un entier suivi de deux mots en entrée – renvoie un mot – le mot rendu est composé de toutes les lettres du second mot en entrée, sauf celle visée par l'entier qui a été remplacée par le premier mot.

Exemple :

ECRIS REPLACE 3 "é "élément → élément

- **SANSECHAP** : attend un mot en entrée – renvoie une liste – le mot rendu est débarrassé de tous ses caractères d'échappement et placé dans une liste.

Exemple :

ECRIS SANSECHAP "GVLOGO\$ est\$ un\$ langage\$ facile\$ à\$ apprendre → GVLOGO est un langage facile à apprendre

- **TRIE** : attend un mot en entrée – renvoie un mot – les lettres du mot sont triées par ordre alphabétique⁵.

Exemple :

ECRIS TRIE "important → aimnoprtt

⁵ Le tri des caractères accentués sera correct avec Delphi, contrairement à Lazarus qui les classera après les caractères normaux.

- **ROTATION** : attend un mot en entrée – renvoie un mot – la primitive renvoie le mot dont le premier caractère est placé à la fin.

Exemple :

ECRIS ROTATION "essai → ssaie

EXTRAIRE DEPUIS DES MOTS

- **ELEMENT** : attend un entier suivi d'un mot en entrée – renvoie un mot – le mot rendu est composé de la lettre du mot en entrée indiquée par l'entier fourni.

Exemple :

ECRIS ELEMENT 4 "Zorro → r

- **DERNIER** (raccourci : **DER**) : attend un mot en entrée – renvoie un mot – le mot renvoyé est composé de la dernière lettre du mot en entrée.

Exemple :

ECRIS DERNIER "important → t

- **HASARD** : attend un mot en entrée – renvoie un mot – le mot renvoyé est composé d'une lettre du mot en entrée, tirée au hasard.

Exemple :

ECRIS HASARD "destin → i

- **PREMIER** (raccourci : **PREM**) : attend un mot en entrée – renvoie un mot – le mot renvoyé est composé de la première lettre du mot en entrée.

Exemple :

ECRIS PREMIER "important → i

- **SAUFDERNIER** (raccourci : **SD**) : attend un mot en entrée – renvoie un mot – le mot renvoyé est amputé de la dernière lettre du mot en entrée.

Exemple :

ECRIS SAUFDERNIER "important → importan

- **SAUFPREMIER** (raccourci : **SP**) : attend un mot en entrée – renvoie un mot – le mot renvoyé est amputé de la première lettre du mot en entrée.

Exemple :

ECRIS SAUFPREMIER "important → mportant

TESTER DES MOTS

- **APRES?** : attend deux mots en entrée – renvoie un booléen – la primitive renvoie "**VRAI**" si le premier mot vient strictement après le second selon l'ordre alphabétique, "**FAUX**" sinon.

Exemple :

ECRIS APRES? "petit "grand → FAUX



Comme **EGAL?** et **AVANT?**, **APRES?** se comporte différemment si les deux mots sont des nombres : ce sont leurs valeurs qui sont alors comparées.

- **AVANT?** : attend deux mots en entrée – renvoie un booléen – la primitive renvoie "**VRAI**" si le premier mot vient strictement avant le second selon l'ordre alphabétique, "**FAUX**" sinon.

Exemple :

ECRIS AVANT? "petit "grand → VRAI

- **COMPTE** : attend un mot en entrée – renvoie un entier – la primitive renvoie le nombre de caractères du mot en entrée.

Exemple :

ECRIS COMPTE "urticaire → 9

- **EGAL?** : attend deux mots en entrée – renvoie un booléen – la primitive renvoie "VRAI si le premier mot est égal au second selon l'ordre alphabétique, "FAUX sinon.

Exemple :

ECRIS EGAL? "100 "1E2 → VRAI

- **IDENTIFICATEUR?** : attend un mot en entrée – renvoie un booléen – la primitive renvoie "VRAI si le mot en entrée est un identificateur correct , "FAUX sinon.

Exemple :

ECRIS IDENTIFICATEUR? "MaFonction12 → VRAI

ECRIS IDENTIFICATEUR? "12MaFonction → FAUX



Un identificateur est correct quand il ne comprend que des lettres non accentuées, des chiffres non placés en première position, les signes _ (souligné) et . (point) en début de mot, ou le ? (point d'interrogation) à la fin.

- **MEMBRE?** : attend deux mots en entrée – renvoie un booléen – la primitive renvoie "VRAI si le premier mot est compris dans le second, "FAUX sinon.

Exemple :

ECRIS MEMBRE? "an "étonnant → VRAI

- **MOT?** : attend un objet en entrée – renvoie un booléen – la primitive renvoie "VRAI si l'objet est un mot, "FAUX sinon.

Exemple :

ECRIS MOT? [coucou] → FAUX

- **NOMBRE?** : attend un mot en entrée – renvoie un booléen – la primitive renvoie "VRAI si le mot est un nombre , "FAUX sinon.

Exemple :

ECRIS NOMBRE? "12458 → VRAI

- **PRECEDENT** : attend deux mots en entrée – renvoie un mot – la primitive renvoie le mot qui vient le premier selon l'ordre alphabétique.

Exemple :

ECRIS PRECEDENT "un "deux → deux

- **SUIVANT** : attend deux mots en entrée – renvoie un mot – la primitive renvoie le mot qui vient le dernier selon l'ordre alphabétique.

Exemple :

ECRIS SUIVANT "un "deux → un

- **VIDE?** : attend un mot en entrée – renvoie un booléen – la primitive renvoie "VRAI si le mot en entrée est le mot vide, "FAUX sinon.

Exemple :

ECRIS VIDE? " → VRAI

IMPLEMENTATION DES MOTS

L'implémentation des mots passe par une unité nommée **GVConsts** qui contient les constantes centralisées du projet et une unité baptisée **GVWords** qui abrite les classes nécessaires à la gestion proprement dite des mots.

CONSTANTES

L'unité **GVConsts** servira pour toutes les unités du projet. Pour une meilleure lisibilité, elle sera complétée au fur et à mesure de l'avancée du travail. En ce qui concerne son apport à l'unité **GVWords**, la partie interface se présente ainsi :

```

const
{ séparateurs }
CBlank = '';
CBeginList = '[';
CEndList = ']';
CBeginPar = '(';
CEndPar = ')';
CSeparators = [CBlank, CBeginList, CEndList, CBeginPar, CEndPar];
{ caractères spéciaux }
CLink = '$';
CUnderline = '_';
CDot = '.';
CAsk = '?';
CQuote = '"';
CColon = ':';
{ ensembles de caractères courants }
CLowAlpha = ['a' .. 'z'];
CHighAlpha = ['A' .. 'Z'];
CAlpha = CLowAlpha + CHighAlpha;
CDigit = ['0' .. '9'];
CAlphaNum = CAlpha + CDigit;

resourcestring
{ messages d'erreur }
ME_BadNumber = 'L"objet %s n''est pas un nombre correct。';
ME_BadInt = 'L"objet %s n''est pas un entier correct。';
ME_EmptyStr = 'Le mot vide ne convient pas pour la primitive %s。';
ME_BadChar = 'Le mot %s est trop court pour en traiter l''élément %d。';

{ primitives }
P_First = 'PREMIER';
P_Last = 'DERNIER';
P_ButFirst = 'SAUPREMIER';
P_ButLast = 'SAUDERNIER';
P_True = 'VRAI';
P_False = 'FAUX';

```

Sont donc définis les séparateurs, les caractères spéciaux, des ensembles de caractères couramment utilisés ainsi que quelques chaînes afin de gérer les erreurs. On notera qu'il est fait usage de chaînes de ressources qui permettent d'envisager une traduction du logiciel.

Les avantages d'une centralisation sont de deux types :

- ✓ il est possible modifier un élément à partir d'un seul point et donc inutile de parcourir tous les fichiers sources éparpillés dans plusieurs unités : on imagine le gain de temps dans la situation où l'on devrait modifier un caractère utilisé plusieurs dizaines de fois ;
- ✓ le risque d'erreurs difficiles à déceler est moindre puisqu'il n'y a pas à recopier un élément pour l'utiliser ailleurs : coder en dur une chaîne, par exemple, c'est s'exposer à une différence minime qui donnera des résultats inattendus lors de comparaisons.

On remarquera que la partie implémentation est vide, l'objectif de l'unité étant seulement de déclarer des types, des chaînes et des constantes.

LA CLASSE TGVSTRING

La difficulté essentielle dans le traitement des mots en **GVLOGO** réside dans la présence d'un caractère d'échappement dont il faut tenir compte pour toutes les opérations portant sur eux. La classe **TGVString** accomplit le travail de normaliser tout mot qui lui est proposé. Elle est capable de restituer le mot d'origine ainsi que celui transformé.

Voici sa partie interface :

```
{ TGVString }

EGVStringException = class(Exception);

TGVString = class
  strict private
    fRawStr: string; // chaîne brute interne
    fStr: string; // chaîne formatée interne
    function GetRawStr: string; // renvoie la chaîne brute
    function GetStr: string; // renvoie la chaîne formatée
    procedure SetStr(const St: string);
  protected
    function WithEsc(const St: string): string; // avec échappement
    function WithoutEsc(const St: string): string; // sans échappement
  public
    constructor Create; // constructeur
    destructor Destroy; override; // destructeur
    procedure Clear; // remise à zéro
    property Str: string read GetStr write SetStr; // chaîne formatée
    property RawStr: string read GetRawStr; // chaîne brute
  end;
```

Elle est bâtie de manière classique en proposant des propriétés dont l'implémentation est cachée via des méthodes privées. La propriété **Str** lit une chaîne afin de la transformer en chaîne normalisée et de la restituer ainsi. La propriété en lecture seule **RawStr** permet de retrouver la chaîne brute, telle qu'elle a été introduite.

La méthode **SetStr** est chargée de transformer la chaîne fournie en entrée. Elle invoque successivement **WithoutEsc** qui retire un éventuel formatage et **WithEsc** qui normalise la chaîne. Cette façon de procéder évite de formater une chaîne qui le serait déjà : en revanche, elle exige deux analyses de la chaîne et ne repère pas les mots normalisés par hasard⁶ !

L'algorithme utilisé par **WithEsc** est trivial. On remarquera cependant l'utilisation de l'énumération **for...in** qui balaie la chaîne en lieu et place d'une boucle traditionnelle **for...next** moins lisible.



On notera aussi une écriture qui reviendra chaque fois que Lazarus et Delphi ne sont pas tout à fait compatibles : le mot « Delphi » est défini ou non en tête de chaque unité. Ici, s'il est défini, on utilise la fonction **CharInSet** inconnue de Lazarus, sachant qu'un simple **in** déclenche un avertissement de dépréciation pour Delphi.

```
function TGVString.WithEsc(const St: string): string;
// *** normalise un mot en tenant compte du caractère d'échappement ***
var
  C: Char;
begin
  Result := EmptyStr; // chaîne vide par défaut
  if (St <> EmptyStr) then
    for C in St do
      // balaie une chaîne non vide
```

⁶ Encore un avantage à déléguer un travail particulier à une classe : un meilleur algorithme pourra être implémenté dans cette classe sans avoir à modifier la moindre ligne des autres fichiers. Autrement dit, une fois l'interface bien définie, il n'importe en rien aux classes qui s'en servent de savoir comment le problème est traité : seul le résultat compte. Il est même très judicieux qu'une classe utilisatrice n'ait aucun présupposé à faire avant d'utiliser une autre classe.

```

begin
{$IFDEF Delphi}
if CharInSet(C, CSeparators + [CLink]) then
{$ELSE}
if C in CSeparators + [CLink] then
{$ENDIF}
  // si séparateur ou échappement suit, on insère un échappement
  Result := Result + CLink;
  Result := Result + C; // caractère en cours traité
end;
end;

```

La méthode `WithoutEsc` n'intéressera que pour l'utilisation d'un drapeau (variable `Flag`) qui mémorise le fait d'avoir trouvé précédemment un caractère d'échappement. Ce drapeau est remis à zéro lorsqu'on a utilisé son repérage.

```

function TGVString.WithoutEsc(const St: string): string;
// *** sans caractère d'échappement ***
var
  C: Char;
  Flag: Boolean;
begin
  Result := EmptyStr;
  // chaîne vide par défaut
  Flag := False; // on n'a pas eu affaire à un caractère d'échappement
  for C in St do // on balaie la chaîne
    if (C <> CLink) or Flag then // caractère d'échappement initial ?
      begin
        Result := Result + C; // non : on ajoute simplement
        Flag := False; // on indique ce cas
      end
    else
      Flag := True; // échappement initial qu'on ignore
  end;
end;

```



Les utilisateurs de Delphi auront avantage à transformer cette classe en enregistrement. En effet, cet IDE accepte des enregistrements comprenant des méthodes. L'avantage sera de ne pas avoir à se préoccuper de la création et de la destruction des instances de la classe avec les méthodes `Create` et `Free`⁷.

LA CLASSE TGVNUMBER

La classe **TGVNumber** traite les chaînes susceptibles de représenter des nombres.

Voici sa partie interface :

```

{ TGVNumber }

EGVNumberException = class(Exception);

TGVNumber = class
strict private
  fNum: Double; // nombre de travail
  fSt: string; // chaîne brute d'entrée
  fValid: Boolean; // drapeau de validité
  function GetInt: Integer; // renvoie un entier
  function GetDouble: Double; // renvoie un réel
  function GetStr: string; // acquiert une chaîne à convertir en nombre
  procedure SetStr(const St: string); // forme une chaîne à partir d'un nombre
public

```

⁷ Pour rappel, si l'on définit bien le destructeur `Destroy`, c'est toujours la méthode `Free` qui est appelée pour le « nettoyage » de sortie.

```

constructor Create; // constructeur
destructor Destroy; override; // destructeur
procedure Clear; // remise à zéro
function IsValid: Boolean; // est-ce un nombre ?
function IsInt: Boolean; // est-ce un entier ?
function IsZero: Boolean; // nombre 0 ?
propertyAsString: string read GetStr write SetStr; // une chaîne de nombre
property AsDouble: Double read GetDouble; // un réel
property AsInt: Integer read GetInt; // un entier
end;

```

La partie publique comprend, outre les constructeurs et destructeurs, une procédure `Clear` qui remet à zéro le nombre interne, trois fonctions utiles et trois propriétés.

L'entrée dans la classe se fait par la propriété `AsString`. Si possible, la chaîne est de manière interne transformée en nombre. En cas d'échec, la fonction `IsValid` retournera `False` comme résultat. L'accès en lecture d'un nombre erroné déclenchera une exception.

La lecture peut être effectuée de trois manières : sous forme de chaîne (`AsString`), d'entier (`AsInt`) ou de nombre réel (`AsDouble`)⁸. Il est par ailleurs possible de tester le nombre interne pour savoir s'il s'agit d'un entier grâce à la fonction `IsInt`. Enfin, la fonction `IsZero` compare de manière correcte des nombres réels qui seraient jugés inégaux pour des raisons de représentation interne.

L'implémentation ne pose pas de problèmes particuliers. Voici celle de `SetStr` qui gère l'entrée d'une nouvelle chaîne à traduire :

```

procedure TGVNumber.SetStr(const St: string);
// *** forme un nombre si possible ***
begin
  fSt := St; // chaîne brute affectée
  fValid := TryStrToFloat(St, fNum); // transformation possible ?
end;

```

On conserve le nombre d'entrée et on teste la chaîne grâce à la fonction `TryStrToFloat`. Le drapeau privé `fValid` sera vérifié avant de renvoyer une valeur : l'utilisateur est ainsi assuré de toujours disposer d'une valeur correcte ou d'obtenir le déclenchement d'une exception. En règle générale, il faut éviter de court-circuiter les exceptions et/ou de ne pas avertir l'utilisateur que la valeur dont il dispose est erronée !

LA CLASSE TGVWORD

La classe **TGVWord** est de loin la plus fournie de cette unité. Elle gère les mots, qu'ils soient des chaînes de caractères ou des nombres, en reprenant toutes les fonctionnalités définies dans la description du langage **GVLOGO** qui concernent les mots.

En voici l'interface :

```

{ TGVWord }

EGVWordException = class(Exception);

TGVWord = class
strict private
  fWord, fWord2: TGVString; // mots de travail
  fNum, fNum2: TGVNumber; // nombre de travail
protected
  function Compare(const StFirst, StTwo: string): Integer; // comparaison de deux mots
public
  constructor Create; // constructeur
  destructor Destroy; override; // destructeur

```

⁸ Le choix du format `Double` provient d'une lacune de Delphi qui traite de manière très lente le format `Extended` en mode 64 bits. J'ai considéré que le mode format `Double` était bien suffisant.

```

function First(const St: string): string; // premier caractère d'un mot
function Last(const St: string): string; // dernier caractère d'un mot
function ButFirst(const St: string): string; // sauf le premier caractère d'un mot
function ButLast(const St: string): string; // sauf le dernier caractère d'un mot
function PutFirst(const StOne, StTwo: string): string; // concatène les deux mots, le second en premier
function PutLast(const StOne, StTwo: string): string; // concatène les deux mots, le premier d'abord
function WithoutQuote(const St: string): string; // supprime si nécessaire le " initial d'un mot
function WithoutColon(const St: string): string; // supprime si nécessaire le : initial d'un mot
function IsValidIdent(const St: string): Boolean; // est-ce un identificateur valide ?
function EqualP(const StFirst, StTwo: string): Boolean; // les deux mots sont-ils égaux ?
function LowerP(const StFirst, StTwo: string): Boolean; // le premier mot est-il à placer avant le second
par ordre alphabétique ?
function Lowest(const StFirst, StTwo: string): string; // renvoie le mot qui vient avant par ordre
alphabétique
function GreaterP(const StFirst, StTwo: string): Boolean; // le premier mot est-il à placer après le second
par ordre alphabétique ?
function Greatest(const StFirst, StTwo: string): string; // renvoie le mot qui vient après par ordre
alphabétique
function EmptyWordP(const St: string): Boolean; // le mot est-il vide ?
function MemberP(const St, SubSt: string): Boolean; // le mot est-il compris dans un autre ?
function Count(const St: string): Integer; // longueur du mot
function StrCount(const St: string): string; // longueur de mot en chaîne
function Item(const St: string; N: Integer): string; // élément N d'un mot
function Replace(const St, SubSt: string; N: Integer): string; // remplacement de l'élément N d'un mot
function Reverse(const St: string): string; // mot inversé
function Shuffle(const St: string): string; // mot mélangé
function AtRandom(const St: string): string; // lettre au hasard
function Uppercase(const St: string): string; // mot en majuscules
function Lowercase(const St: string): string; // mot en minuscules
function Insert(const St, SubSt: string; N: Integer): string; // insertion en position N
function Sort(const St: string): string; // tri des lettres du mot
function NumberP(const St: string): Boolean; // est-ce un nombre ?
function WithEsc(const St: string): string; // chaîne formatée
function WithoutEsc(const St: string): string; // chaîne brute
function IsValid(const St: string): Boolean; // le mot est-il valide sans traitement ?
function Rotate(const St: string): string; // rotation des caractères du mot
end;

```

La classe utilise deux champs privés de la classe **TGVString** et autant de la classe **TGVNumber** : ils servent d'outils de travail et de comparaisons. L'emploi de ces classes évite d'avoir constamment à vérifier la présence des caractères spéciaux.



En fait, la difficulté essentielle tient aux lacunes de Lazarus dans la gestion des caractères accentués. Les fonctions préfixées par UTF8 présentes dans l'unité **lazutf8** constituent une solution possible qui règle les problèmes, sauf ceux relatifs à l'ordre alphabétique : ainsi la fonction **Sort** ne triera pas correctement les lettres d'un mot, rejetant les caractères accentués après les caractères normaux. Delphi ne présente pas ce problème. On notera par ailleurs que la fonction **Sort** utilise un tri à bulles bien suffisant pour des quantités très limitées d'objets à trier.

La méthode protégée **Compare** s'occupe des comparaisons : il faut tenir compte de la comparaison de chaînes ordinaires et de nombres. En effet, 99 vient après 121 par ordre alphabétique, mais avant par ordre de grandeur mathématique !

```

function TGVWord.Compare(const StFirst, StTwo: string): Integer;
// *** comparaison de deux mots ***
begin
  fNumAsString := StFirst; // essai de conversion en nombre
  fNum2AsString := StTwo;
  // ce sont des nombres ?
  if fNum.IsValid and fNum2.IsValid then
    // si oui, on les compare
    Result := CompareValue(fNum.AsDouble, fNum2.AsDouble)
  else
    begin // les autres mots
      fWord.Str := StFirst; // mots normalisés

```

```

fWord2.Str := StTwo;
{$IFDEF Delphi}
Result := AnsiCompareText(fWord.Str, fWord2.Str); // comparaison
{$ELSE}
Result := UTF8CompareText(fWord.Str, fWord2.Str);
{$ENDIF}
end;
end;

```

TEST DE L'UNITÉ TGVWORDS

Le meilleur moyen de s'approprier les méthodes de la classe **TGVWord** est de les voir fonctionner dans un programme.

Les versions proposées sont au nombre de quatre :

- ✓ une version Delphi Win32 ;
- ✓ une version Delphi Win64 ;
- ✓ une version Lazarus Win32 ;
- ✓ une version Lazarus Linux.



Si Delphi offre de meilleures capacités à traiter les caractères Unicode et s'il facilite une présentation plus moderne grâce aux styles, Lazarus se montre à la hauteur dans sa portabilité sur différents OS : il n'a pas été nécessaire d'apporter une quelconque modification aux fichiers pour faire fonctionner le programme aussi bien sur Windows que sur Linux.

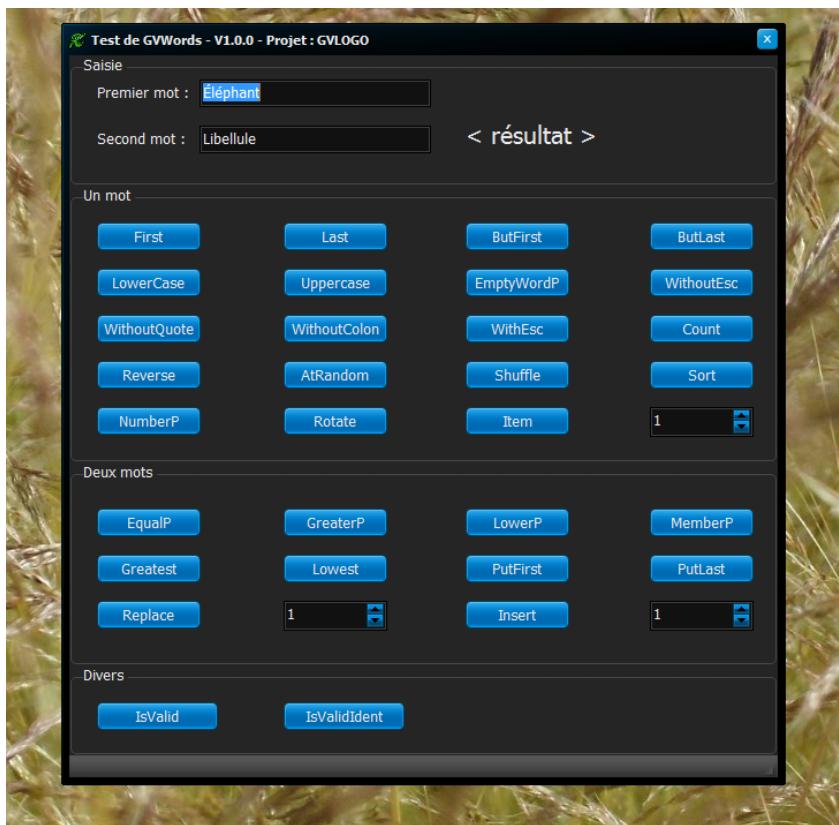


Figure 1 – Test de TGVWords avec Delphi

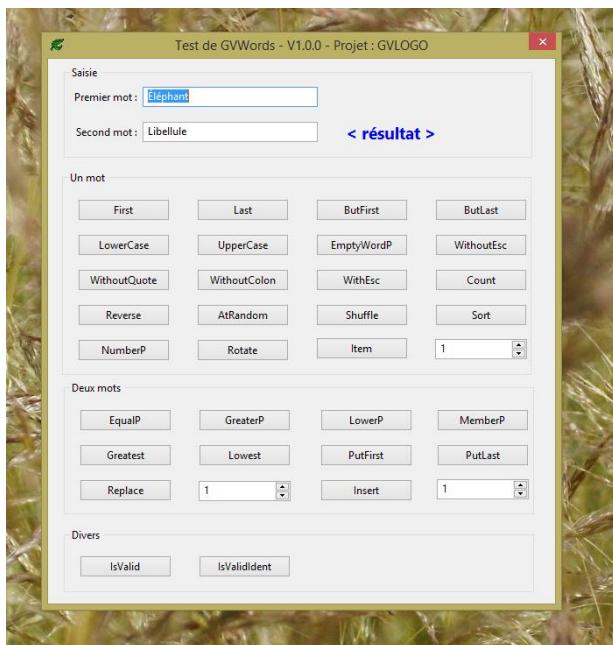


Figure 2 – Test de TGVWords avec Lazarus (Windows)

Le programme lui-même est trivial puisqu'il se contente d'affecter à un **TLabel** le résultat d'une méthode de **TGVWord**. C'est exactement ce qui était recherché : l'unité remplit son rôle en limitant au possible les complications pour l'utilisateur final.

Une fonction souvent méconnue est utilisée : il s'agit de **IfThen**. La version traitant les chaînes de caractères est définie dans l'unité **StrUtils**⁹. Elle prend deux ou trois paramètres en entrée : le premier est une valeur booléenne, les deux suivants une chaîne de caractères. Si le troisième est omis, il est par défaut défini à la chaîne vide. Le principe de fonctionnement est semblable à celui d'un test avec **if...then...else** : si la valeur booléenne est vraie, le second paramètre est renvoyé, sinon c'est le troisième.

Par exemple :

```
procedure TGVMainForm.btnIsValidIdentClick(Sender: TObject);
// test de ISVALIDIDENT
begin
  lblResult.Caption := IfThen(fWord.IsValidIdent(LabEdtFirst.Text),
    P_True, P_False);
end;
```

Le test de la fonction **IsValidIdent** est effectué sur le composant **TLabelEdit** : si le résultat est vrai, c'est la constante **P_True** qui est affectée au **Caption** du composant **TLabel**, sinon c'est **P_False**.

Il est intéressant de modifier les paramètres des éditeurs pour voir le comportement de l'unité. Par exemple, on pourra entrer des mots tels que "Essai, cou[cou... On peut aussi entrer une phrase complète et constater qu'elle sera normalisée pour être comprise comme un mot ! Dans tout test, il est conseillé de provoquer des erreurs afin de voir comment se comporte l'unité.



Toutes les unités produites seront ainsi testées par un programme autonome. Cette démarche permet d'isoler les éventuels problèmes : il sera bien plus facile de repérer une erreur dans un module que dans le programme final qui comportera des milliers de lignes !

⁹ Il existe aussi une version qui traite les nombres et qui est définie dans l'unité **math**.

LES LISTES

DEFINITIONS

Une *liste simple* est composée de mots séparés par des espaces¹⁰. Elle est délimitée par des crochets : [puis].

Une *liste complexe* comprend des mots et des listes. Les imbriques ne sont pas limitées, pourvu que l'ordre d'ouverture et de fermeture des listes imbriquées soit correct.



Il existe une liste particulière qui ne comprend ni mot ni liste. On la nomme *la liste vide* et elle est représentée par des crochets : [] accolés.

EXEMPLES DE LISTES

LISTES SIMPLES

Voici quelques listes simples :

- ✓ [un deux trois quatre cinq]
- ✓ [1 2 3 4 5]
- ✓ [GVLOGO est un langage simple.]
- ✓ [Des espaces à conserver : . C'est fait !]

LISTES IMBRIQUEES

Voici des listes imbriquées :

- ✓ [élément1 [liste imbriquée 1] élément3]
- ✓ [un [deux imbriquée [encore deux mais imbriquée]] trois]
- ✓ [[]]
- ✓ [un [deux[deux bis [deux ter]]] trois]
- ✓ [un (12*45+789) trois]



La dernière liste contient une expression. Une expression est une suite de caractères à interpréter comme un calcul à effectuer¹¹.

LISTES FAUTIVES

Une liste est fautive si elle ne contient pas des crochets correctement agencés et en bon nombre.

Voici des listes incorrectes :

- ✓ [oups ! [où est la fermeture ?]
- ✓ [fermée trop tôt][ceci est une autre liste]
- ✓ [le caractère d'échappement masque la fin de liste !\$]

OPERATIONS SUR LES LISTES

Les opérations possibles sur les listes sont en majorité celles possibles avec les mots. Elles sont mentionnées ici avec des exemples propres aux listes.

¹⁰ L'espace séparant les éléments d'une liste, une suite d'espaces sera par conséquent supprimée.

¹¹ On reviendra sur les expressions lors de l'étude de leur évaluation.



La primitive **ECRIS** retire les crochets lors de l'envoi d'une liste à l'écran. Ce comportement explique pourquoi les exemples qui utilisent cette primitive donnent des résultats sans crochets. Si l'on veut conserver les crochets, il faut utiliser la primitive **ECRIST**.

FABRIQUER DES LISTES

- **METSPREMIER** (raccourci : **MP**) : attend deux listes en entrée ou un mot et une liste – renvoie une liste – la liste rendue est composée du premier paramètre placé avant le second.

Exemples :

ECRIS METSPREMIER [Coucou] [Tortue !] → [Coucou] Tortue !

ECRIS METSPREMIER "Coucou [Tortue !] → Coucou Tortue !

- **METSDERNIER** (raccourci : **MD**) : attend deux listes en entrée ou un mot suivi d'une liste – renvoie une liste – la liste rendue est composée du premier paramètre placé après le second.

Exemples :

ECRIS METSDERNIER [Coucou] [Tortue !] → Tortue ! [Coucou]

ECRIS METSDERNIER "Coucou [Tortue !] → Tortue ! Coucou

- **PHRASE** (raccourci : **PH**) : attend deux listes ou deux mots ou une combinaison des deux en entrée – renvoie une liste – la liste rendue est composée des deux éléments séparés par un espace sans les crochets et/ou sans les guillemets anglais.

Exemples :

ECRIS PHRASE "je "viens → je viens

ECRIS PHRASE [je] [pars] → je pars

ECRIS PHRASE "je [reviens] → je reviens

ECRIS PHRASE [je] "repars → je repars

MODIFIER DES LISTES

- **INSERE** : attend un entier, un mot ou une liste, puis une liste en entrée – renvoie une liste – la liste rendue est composée du deuxième paramètre inséré à la position précisée par l'entier dans la liste finale.

Exemples :

ECRIS INSERE 2 "très [c'est bien] → c'est très bien

ECRIS INSERE 1 [zéro] [[un][deux][trois]] → [zéro][un][deux][trois]

- **INVERSE** : attend une liste en entrée – renvoie une liste – la liste rendue est celle d'entrée dont les éléments ont été inversés.

Exemple :

ECRIS INVERSE [un deux trois] → trois deux un

- **MAJUSCULES** : attend une liste en entrée – renvoie une liste – la liste rendue est en majuscules.

Exemple :

ECRIS MAJUCULES [le soleil se lève] → LE SOLEIL SE LÈVE

- **MELANGE** : attend une liste en entrée – renvoie une liste – la liste rendue a les mêmes éléments que celle d'origine, mais dans un ordre aléatoire.

Exemple :

ECRIS MELANGE [un deux [troisa troisb]] → deux [troisa troisb] un

- **MINUSCULES** : attend une liste en entrée – renvoie une liste – la liste rendue est en minuscules.

Exemple :

ECRIS MINUSCULES [LES ENFANTS SE LÈVENT] → Les enfants se lèvent

- **REPLACE** : attend un entier, une liste ou un mot, puis une liste en entrée – renvoie une liste – la liste rendue est composée de toutes les éléments de la liste finale, sauf celui visé par l'entier qui a été remplacé par le deuxième paramètre.

Exemples :

ECRIS REPLACE 4 "couchent [Les enfants se lèvent] → Les enfants se couchent

ECRIS REPLACE 2 [deuxième] [[premier][seconde][troisième]] →

[premier][deuxième][troisième]

- **TRIE** : attend une liste en entrée – renvoie une liste – les éléments de la liste sont triés par ordre alphabétique¹².

Exemple :

ECRIS TRIE [zèbre tartine accent dommage] → accent dommage tartine zèbre

- **ROTATION** : attend une liste en entrée – renvoie une liste – la primitive renvoie la liste dont le premier élément est placé à la fin.

Exemple :

ECRIS ROTATION [un deux trois quatre] → deux trois quatre un

EXTRAIRE DE LISTES

- **ELEMENT** : attend un entier suivi d'une liste en entrée – renvoie une liste ou un mot – l'objet rendu est l'élément de la liste en entrée indiqué par l'entier.

Exemple :

ECRIS ELEMENT 2 [un deux [trois]] → deux

- **DERNIER** (raccourci : **DER**) : attend une liste en entrée – renvoie un mot ou une liste – l'objet renvoyé est composé du dernier élément de la liste en entrée.

Exemple :

ECRIS DERNIER [un deux] → deux

- **HASARD** : attend une liste en entrée – renvoie un mot ou une liste – l'objet renvoyé est composé d'un élément de la liste en entrée, tiré au hasard.

Exemple :

ECRIS HASARD [un [deux] trois quatre] → deux

- **PREMIER** (raccourci : **PREM**) : attend une liste en entrée – renvoie un mot ou une liste – l'objet renvoyé est composé du premier élément de la liste en entrée.

Exemple :

ECRIS PREMIER [beau laid] → beau

- **PREMS** : attend une liste en entrée – renvoie une liste – l'objet renvoyé est composé de chaque premier élément des éléments de la liste en entrée.

Exemples :

ECRIS PREMS [beau [laid]] → b laid

ECRIS PREMS [[moi beau] [toi laid]] → [moi] [toi]

- **SAUFDERNIER** (raccourci : **SD**) : attend une liste en entrée – renvoie un mot ou une liste – l'objet renvoyé est composé de la liste en entrée sans son dernier élément.

Exemple :

ECRIS SAUFDERNIER [pas cette liste stupide] → pas cette liste

¹² Le tri des caractères accentués sera correct avec Delphi, contrairement à Lazarus qui les classera après les caractères normaux.

- **SAUFPremier** (raccourci : **SP**) : attend une liste en entrée – renvoie un mot ou une liste – l'objet renvoyé est composé de la liste en entrée sans son premier élément.

Exemple :

ECRIS SAUFPremier [pas cette liste stupide] → cette liste stupide

- **SAUFPREMs** : attend une liste en entrée – renvoie une liste – l'objet renvoyé est composé de la liste en entrée sans chaque premier élément de ses éléments.

Exemples :

ECRIS SAUFPREMs [pas cette liste stupide] → as ette iste tupide

ECRIS SAUFPREMs [[un chat] [une chouette]] → [chat] [chouette]

TESTER DES LISTES

- **APRES?** : attend deux listes en entrée – renvoie un booléen – la primitive renvoie "VRAI si la première liste vient strictement après la seconde selon l'ordre alphabétique, "FAUX sinon.

Exemple :

ECRIS APRES? [un petit chat] [un chat] → VRAI

- **AVANT?** : attend deux listes en entrée – renvoie un booléen – la primitive renvoie "VRAI si la première liste vient strictement avant la seconde selon l'ordre alphabétique, "FAUX sinon.

Exemple :

ECRIS AVANT? [le renard] [un renard] → VRAI

- **COMPTE** : attend une liste en entrée – renvoie un entier – la primitive renvoie le nombre d'éléments de la liste en entrée.

Exemple :

ECRIS COMPTE [un [deux encore deux] trois] → 3

- **EGAL?** : attend deux listes en entrée – renvoie un booléen – la primitive renvoie "VRAI si la première liste est identique à la seconde, "FAUX sinon.

Exemple :

ECRIS EGAL? [c'est la même] METSPremier "c'est [la même] → VRAI

- **MEMBRE?** : attend une liste ou un mot, puis une liste en entrée – renvoie un booléen – la primitive renvoie "VRAI si le premier objet est compris dans le second, "FAUX sinon.

Exemples :

ECRIS MEMBRE? [oui] [un deux [oui]] → VRAI

ECRIS MEMBRE? "oui [un deux [oui]] → FAUX

- **LISTE?** : attend un objet en entrée – renvoie un booléen – la primitive renvoie "VRAI si l'objet est une liste, "FAUX sinon.

Exemples :

ECRIS LISTE? [coucou] → VRAI

ECRIS LISTE? "coucou → FAUX

- **PRECEDENT** : attend deux listes en entrée – renvoie une liste – la primitive renvoie la liste qui vient la première selon l'ordre alphabétique.

Exemple :

ECRIS PRECEDENT [je suis premier] [je suis second] → je suis premier

- **SUIVANT** : attend deux listes en entrée – renvoie une liste – la primitive renvoie la liste qui vient la dernière selon l'ordre alphabétique.

Exemple :

ECRIS SUIVANT [je suis premier] [je suis second] → je suis second

- **VIDE?** : attend une liste en entrée – renvoie un booléen – la primitive renvoie "VRAI si la liste en entrée est la liste vide, "FAUX sinon.

Exemples :

ECRIS VIDE? [] → VRAI

ECRIS VIDE? SAUPREMIER [un] → VRAI

IMPLEMENTATION DES LISTES

L'implémentation des listes passe par un complément de l'unité **GVConsts** qui contient les constantes centralisées du projet et une unité baptisée **GVLists** qui abrite les classes nécessaires à la gestion proprement dite des listes. Cette unité utilise par ailleurs l'unité **GVWords** précédemment étudiée.

CONSTANTES

Les constantes ont simplement été complétées avec les chaînes nécessaires au signalement des erreurs et un type énuméré baptisé **TGVError** qui reprend les erreurs sous forme codée.

```
type
  { erreurs }
  TGVError = (
    C_None, // pas d'erreur
    C_BadNumber, // nombre incorrect
    C_BadInt, // entier incorrect
    C_EmptyStr, // mot vide interdit
    c_BadChar, // caractère incorrect
    C_BadList, // erreur dans une liste
    C_DellItem, // position incorrecte pour une suppression
    C_InsItem, // position incorrecte pour une insertion
    C_ReplacelItem, // position incorrecte pour un remplacement
    C_NoListWord, // ni un mot ni une liste
    C_TwoDelete // pas assez d'éléments pour en supprimer deux
  );

resourcestring
  { message d'erreur }

  ME_None = 'Pas d''erreur à signaler。';
  ME_BadNumber = 'L''objet %s n''est pas un nombre correct。';
  ME_BadInt = 'L''objet %s n''est pas un entier correct。';
  ME_EmptyStr = 'Le mot vide ne convient pas pour la primitive %s。';
  ME_BadChar = 'Le mot %s est trop court pour en traiter l''élément %d。';
  ME_BadList = 'La liste %s est incorrecte。';
  ME_DellItem = 'L''élément %d n''existe pas pour une suppression。';
  ME_InsItem = 'L''élément %d n''existe pas pour une insertion。';
  ME_ReplacelItem = 'L''élément %d n''existe pas pour un remplacement。';
  ME_NoListWord = '%s n''est ni une liste ni un mot corrects。';
  ME_TwoDelete = 'La liste ne contient pas assez d''éléments pour en supprimer deux à partir de %d。';
```

LA CLASSE TGVLISTUTILS

La classe **TGVListUtils** concentre les méthodes utiles au traitement des listes. Il aurait été possible de les implémenter en tant que procédures et fonctions indépendantes, mais l'utilisation d'une classe et de son interface permet une meilleure maîtrise des outils : la classe définit précisément ce pour quoi elle est faite.

En voici l'interface :

```
{ TGVListUtils }
```

```

EGVListUtilsException = class(Exception);

TGVListUtils = class
  strict private
    fWord: TGVWord; // mot de travail
    fSt: TGVString; // chaîne de travail
  public
    constructor Create; // constructeur
    destructor Destroy; override; // destructeur
    // conversion d'une liste en mot
    function ListToWord(const St: string): string;
    // conversion d'un mot en liste
    function WordToList(const St: string): string;
    // conversion d'une liste en chaîne
    function ListToStr(const St: string): string;
    // conversion d'une chaîne en liste
    function StrToList(const St: string): string;
    // retourne la liste vide
    function EmptyList: string;
    // vérifie la validité d'une liste
    function IsValid(const St: string): Boolean;
    // teste la validité d'une valeur (mot ou liste)
    procedure TestValue(const St: string);
    // teste la validité d'une valeur (mot ou liste) - sans exception
    function IsValidValue(const St: string): Boolean;
    // liste simple ?
    function IsSimpleList(const St: string): Boolean;
end;

```

En dehors de la fonction `EmptyList` qui se contente de renvoyer la liste vide, il s'agit essentiellement de méthodes de conversion et de validation.

Ainsi `ListToStr` et `StrToList` permettent de convertir une liste en chaîne et réciproquement. `ListToWord` et `WordToList` prennent en compte le caractère d'échappement. Pour ces quatre méthodes, un contrôle de la validité des listes est effectué.

Le contrôle de la validité d'une liste s'effectue grâce à `IsValid` qui est de loin la plus complexe. Elle analyse une chaîne et contrôle caractère par caractère que les règles relatives aux listes sont respectées : autant de crochet ouvrants que de crochets fermants, dans le bon ordre ; autant de parenthèses ouvrantes que de parenthèses fermantes, dans le bon ordre ; prise en compte du caractère d'échappement ; interdiction des sous-listes à l'intérieur d'une expression parenthésée.

Les autres méthodes de contrôle sont plus simples. `IsSimpleValue` se contente de vérifier la présence des crochets pour une liste. `IsValidValue` exploite l'unité **GVWords** pour tester un mot et la méthode précédente pour une liste. `TestValue` est une procédure qui déclenche une exception si la chaîne en entrée n'est ni un mot ni une liste au sens de **GVLOGO**.

LA CLASSE TGVLIST

La classe **TGVList** implémente la gestion des listes pour **GVLOGO**. Elle hérite de **TStringList** qui gère les listes de chaînes. En fait, une liste en LOGO est une chaîne particulière de caractères : elle nécessite d'être entourée de crochets et, si elle comprend d'autres listes, de comprendre à l'intérieur autant de crochets ouvrants que de crochets fermants (dans le bon ordre évidemment !). Cette dernière remarque s'applique aussi pour les éventuelles expressions imbriquées dans la liste. Pour parvenir à cet effet, il faut surcharger les méthodes d'entrée : `Add`, `LoadFromStream`, `Assign`, `Insert` et `Put`. Chaque élément de la liste (mot ou sous-liste) est stocké dans une chaîne de la `StringList`.

Toutes les nouvelles méthodes renvoient une chaîne ou une valeur sans modifier la liste d'entrée. En cas d'erreur lors de la construction de la liste, la liste interne est la chaîne vide et la propriété `LastErrorPos` renvoie l'indice de l'erreur détectée.

En voici l'interface¹³ :

```
{ TGVList }

EGVListException = class(Exception);

TGVList = class(TStringList)
strict private
  fError: TGVError; // erreur
  fErrorPos: Integer; // position d'une erreur
  fLoading: Boolean; // chargement en cours ?
  fNumLastItem: Integer; // dernier élément trouvé
  fWord: TGVWord; // mot de travail
  fUtil: TGVListUtils; // utilitaire pour liste
  function GetLastErrorPos: Integer; // position de la dernière erreur
  function GetLastError: TGVError; // dernière erreur
protected
  procedure Put(Index: Integer; const S: string); override; // assignation
public
  constructor Create; overload; // création
  destructor Destroy; override; // destruction
  function Add(const St: string): Integer; override; // ajout
  procedure LoadFromStream(Stream: TStream); overload; override; // chargement
  procedure Assign(Source: TPersistent); override; // assignation
  procedure Insert(Index: Integer; const S: string); override; // insertion
  // nouvelles méthodes (** ne modifient pas la liste interne **)
  // renvoie la liste sous forme de chaîne
  function ToStr: string;
  // renvoie la liste sous forme de chaîne sans crochets
  function ToWBStr: string;
  // la liste est-elle vide ?
  function IsEmpty: Boolean;
  // la liste est-elle la liste vide ?
  function IsEmptyList: Boolean;
  // renvoie le premier élément de la liste
  function First: string;
  // renvoie le dernier élément de la liste
  function Last: string;
  // sauf le premier de la liste
  function ButFirst: string;
  // sauf le dernier de la liste
  function ButLast: string;
  // supprime l'élément N
  function DeleteItem(N: Integer): string;
  // insertion d'un élément en position N
  function InsertItem(N: Integer; const St: string): string;
  // remplacement de l'élément N
  function ReplaceItem(N: Integer; const St: string): string;
  // met en premier
  function PutFirst(const St: string): string;
  // met en dernier
  function PutLast(const St: string): string;
  // phrase à droite
  function SentenceRight(const St: string): string;
  // phrase à gauche
  function SentenceLeft(const St: string): string;
  // tri des éléments
  function SortItems: string;
  // inversion des éléments
  function ReverseItems: string;
  // mélange des éléments
  function ShuffleItems: string;
  // membre présent ?
  function IsItem(const St: string): Boolean;
```

¹³ On remarquera que les primitives concernant les piles et les queues ne sont pas implémentées dans cette unité : elles le seront dans l'unité **TGVStacks**.

```

// ajout d'une paire
function TwoAdd(const St1, St2: string): string;
// suppression d'une paire
function TwoDelete(N: Integer): string;
// liste en majuscules
function UpperCase: string;
// liste en minuscules
function LowerCase: string;
// rotation de la liste
function Rotate: string;
// nouvelles propriétés
// dernière erreur
property LastError: TGSError read GetLastError default C_None;
// position de la dernière erreur
property LastErrorPos: Integer read GetLastErrorPos default -1;
// dernier élément traité
property LastItem: Integer read fNumLastItem default -1;
end;

```

La méthode la plus compliquée est certainement **Add**. Elle prend en charge l'ensemble du contrôle de la validité de la liste. Afin d'en simplifier la lecture, deux versions coexistent : celle pour Delphi et celle pour Lazarus. La raison en est encore une fois la gestion différente par les deux IDE des caractères accentués. Cette méthode comprend une partie générale qui teste les mots et les listes avant de renvoyer une éventuelle position d'erreur ou de stocker les chaînes trouvées, et une procédure imbriquée qui examine le contenu de la chaîne. Cette dernière lit les caractères un par un et répartit le travail suivant ce qu'elle rencontre : blanc, crochet ouvrant, parenthèse ouvrante, échappement, crochet et parenthèse fermants mais orphelins, autre caractère. Elle est quasi-identique à la méthode **IsValid** de **TGVListUtils**, sinon qu'elle stocke les éléments afin qu'ils puissent être exploités. Du point de vue de l'utilisateur, l'entrée d'une nouvelle liste se fait grâce à la propriété héritée **Text** de l'unité : il suffit d'affecter une chaîne à cette propriété pour qu'elle soit analysée et stockée.

La propriété **LastItem** n'est affectée que par l'utilisation de la méthode **IsItem**. Si cette dernière renvoie **True**, **LastItem** contiendra le numéro de l'élément de la liste qui correspond à celui recherché.



On notera que les éléments des listes en Pascal sont numérotés avec une base de 0 alors qu'en **GVLOGO** la base est de 1, ce qui paraîtra plus naturel à l'utilisateur.

TEST DE L'UNITE TGVLISTS

Comme lors du test de **TGVWords**, les versions proposées sont au nombre de quatre :

- ✓ une version Delphi Win32 ;
- ✓ une version Delphi Win64 ;
- ✓ une version Lazarus Win32 ;
- ✓ une version Lazarus Linux.

Le programme lui-même ne pose pas de problèmes particuliers. Il ne s'agit en effet que de mettre à jour la liste de travail et d'appliquer la méthode voulue.

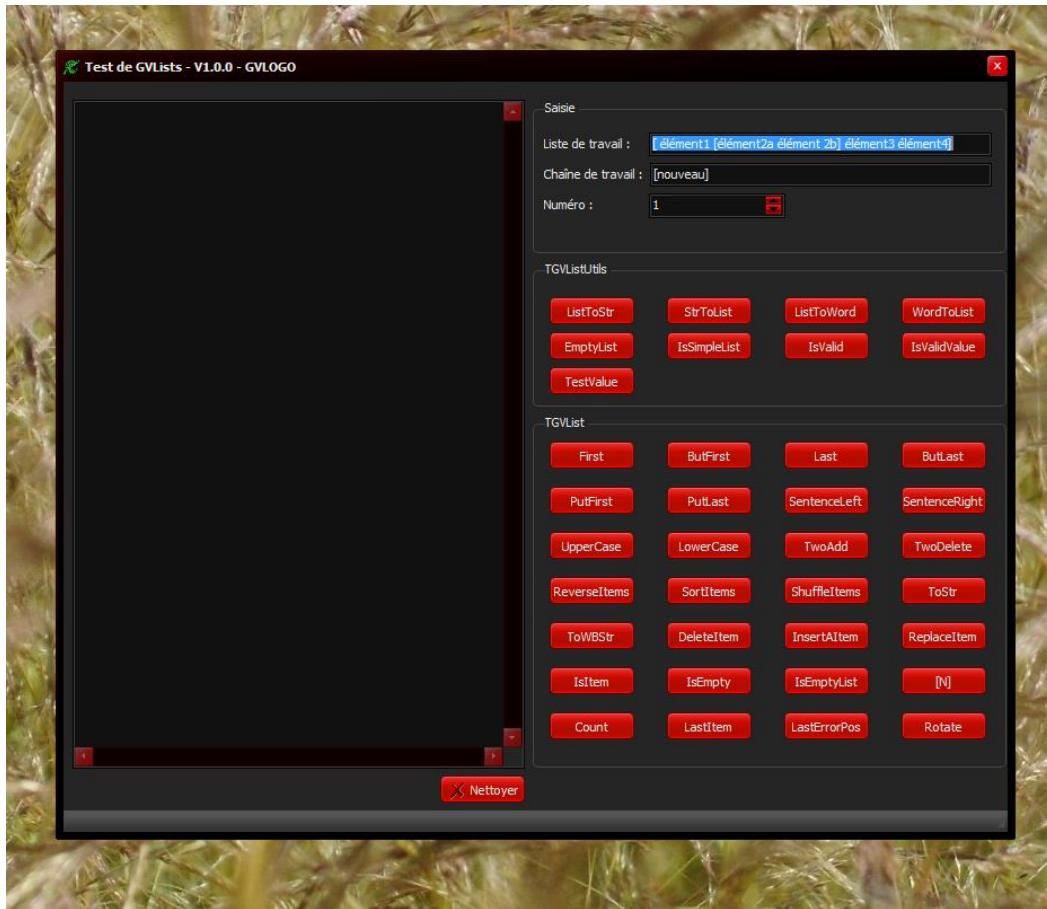


Figure 3 - Test de TGVLists avec Delphi

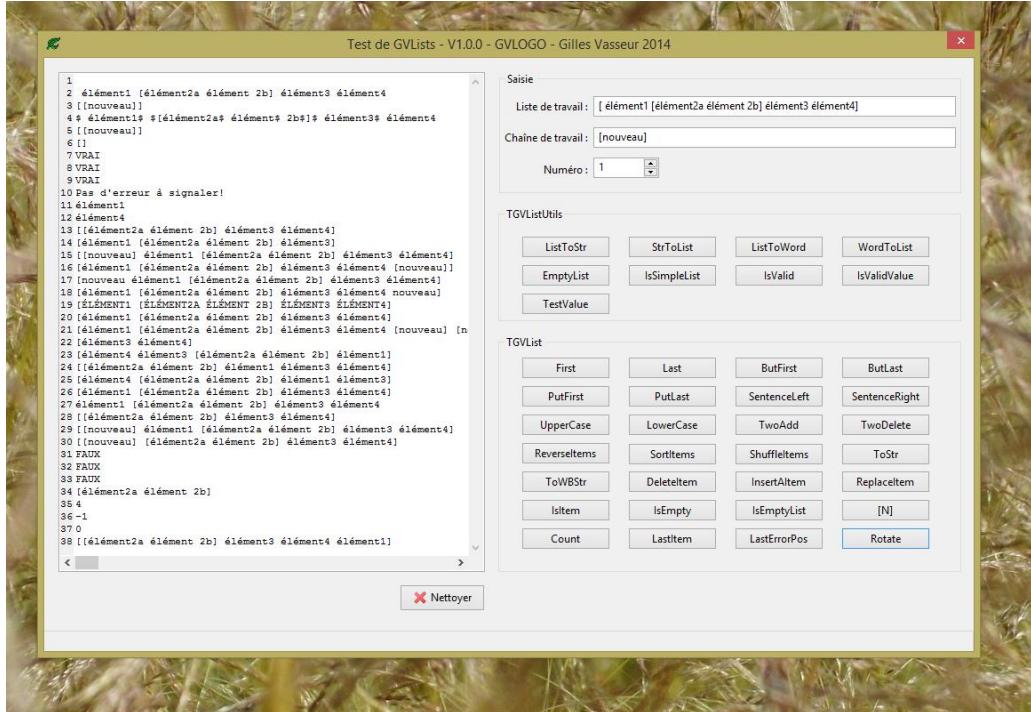


Figure 4 - Test de TGVLists avec Lazarus

Il est encore une fois conseillé de modifier les différents paramètres pour observer les réactions de l'unité.



Dans le dossier réservé à Lazarus, on trouvera un sous-dossier nommé « Length ». Il comprend un programme rudimentaire qui compare différentes possibilités de traitements d'une chaîne, avec ou sans l'unité **lazutf8**. Cette dernière est la seule à traiter correctement les caractères accentués.

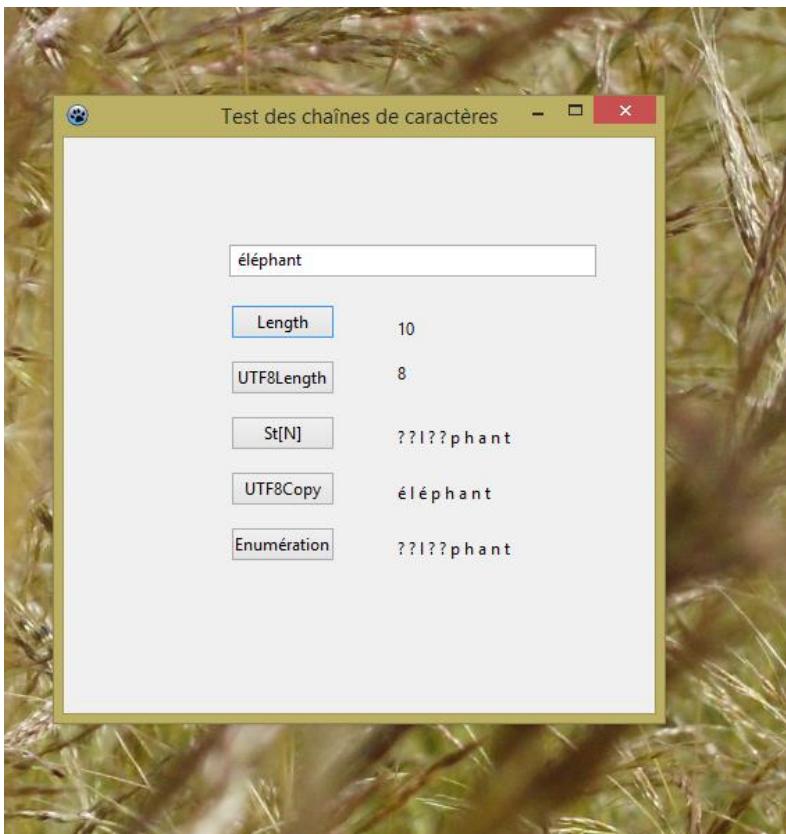


Figure 5 - Test des chaînes avec Lazarus

LES LISTES DE PROPRIETES

DEFINITIONS

Les listes de propriétés sont des listes particulières associant chacune à un nom des couples propriété-valeur. Par exemple, on peut imaginer répertorier des chiens dont les caractéristiques seront stockées : Zoé aura pour propriétés race, sexe, couleur, taille et âge dont les valeurs respectives seront caniche, femelle, blanc, très petit, 5. Médor aura les propriétés race, sexe, âge, caractère dont les valeurs seront teckel, mâle, 3, tranquille. On voit que l'agencement des propriétés est libre, contrairement à ce qui se passerait pour un tableau.



GVLOGO fait un usage interne intensif des listes de propriétés. En effet, aussi bien les variables, les procédures, les paquets que les lignes des éditeurs sont des listes de propriétés.



Les noms donnés aux propriétés tout comme le nom des listes est indifférent aux majuscules et minuscules. Par conséquent, les propriétés **MAPROP**, **MaProp** ou encore **maprop** sont équivalentes.

EXEMPLES DE LISTES DE PROPRIETES

- ✓ Liste associée à "caniche" : [couleur blanc taille [tout petit]]
- ✓ Liste associée à "labrador" : [couleur blanc vitesse [rapide] âge 8]



Les propriétés d'une liste de propriétés sont organisées en deux éléments consécutifs : le premier définit le nom de la propriété tandis que le second indique la valeur qui lui est associée. Le premier est toujours un mot unique pour la liste considérée alors que le second peut être un mot ou une liste quelconque.

OPERATIONS SUR LES LISTES DE PROPRIETES

- **DPROP** : attend deux mots en entrée suivis d'un mot ou une liste – ne renvoie rien – la primitive crée ou met à jour la propriété de la liste de propriétés fournie en paramètre. Le premier mot est la liste de propriétés et le second la propriété. Le dernier paramètre est soit un mot soit une liste qui définit la valeur attribuée à la propriété.

Exemple :

DPROP "MALISTE "MAPROP [une valeur] → -

- **RPROP** : attend deux mots en entrée – renvoie un mot ou une liste – la primitive renvoie la valeur de la propriété (second paramètre) de la liste visée (premier paramètre).

Exemple :

ECRIS RPROP "MALISTE "MAPROP → une valeur

- **ANNULEPROP** : attend deux mots en entrée – ne renvoie rien – la propriété (second paramètre) de la liste spécifiée (premier paramètre) est détruite.

Exemple :

ANNULEPROP "MALISTE "MAPROP → -

- **PROPS** : attend un mot en entrée – renvoie une liste – la primitive renvoie la liste des propriétés de la liste en entrée.

Exemple :

ECRIS PROPS "MALISTE → maprop

- **ANNULE** : attend un mot en entrée – ne renvoie rien – la liste spécifiée par le mot est détruite avec toutes ses propriétés.

Exemple :

ANNULE "MALISTE → -

- **COMPTEPROPS** : attend un mot en entrée – renvoie un entier – la primitive renvoie le nombre de propriétés associées à la liste en entrée.

Exemple :

COMPTEPROPS "MALISTE → 1

- **PROP?** : attend deux mots en entrée – renvoie un booléen – la primitive renvoie "VRAI si le second mot en entrée est une propriété du premier, "FAUX sinon

Exemple :

ECRIS PROP? "MALISTE "MAPROP → VRAI

- **LISTEPROP?** : attend un mot en entrée – renvoie un booléen – la primitive renvoie "VRAI si le mot en entrée est une liste de propriétés, "FAUX sinon.

Exemple :

ECRIS LISTEPROP? "MALISTE → VRAI

- **PROCEDURE?** : attend un mot en entrée – renvoie un booléen – la primitive renvoie "VRAI si le mot en entrée est une procédure, "FAUX sinon.

Exemple :

ECRIS PROCEDURE? "MALISTE → FAUX

- **PRIMITIVE?** : attend un mot en entrée – renvoie un booléen – la primitive renvoie "VRAI si le mot en entrée est une primitive, "FAUX sinon.

Exemple :

ECRIS PRIMITIVE? "ECRIS → VRAI



On notera que la primitive (ou la procédure) est précédée des guillemets anglais. Si elle ne l'était pas, **GVLOGO** chercherait à l'exécuter, ce qui n'est pas l'effet voulu.

- **NOM?** : attend un mot en entrée – renvoie un booléen – la primitive renvoie "VRAI si le mot en entrée est une variable, "FAUX sinon.

Exemple :

ECRIS NOM? "MALISTE → FAUX

IMPLEMENTATION DES LISTES DE PROPRIETES

L'unité **TGVPropLists** implémente essentiellement deux classes : **TGVPropListEnumerator** qui gère l'énumération et **TGVPropList** qui gère les listes elles-mêmes.

CONSTANTES

Comme pour les unités précédentes, l'unité **GVConsts** a été complétée afin de prendre en compte les nouveaux besoins. Les ajouts ont été effectués au niveau des constantes elles-mêmes.

En voici le listing :

```
{ listes de propriétés}

// extension pour les fichiers de listes de propriétés
CExtPI = '.GPL';
// en-tête de fichier
CHheader = '[GPL200 (c) GV 2014]';
// séparateur de liste de propriétés
CSep = '|';
```

On remarquera l'extension pour les fichiers et surtout l'en-tête qui permettra de vérifier qu'un fichier est conforme au format attendu : il précèdera les données proprement dites.

Les chaînes de ressources ont aussi été complétées :

```
ME_NoListWord = '%s n''est ni une liste ni un mot corrects。';
ME_TwoDelete = 'La liste ne contient pas assez d'éléments pour en supprimer deux à partir de %d。';
ME_BadListP = 'La liste de propriétés %d est introuvable。';
ME_BadFormat = 'Le format du fichier %s est incorrect : %。';
```

De même pour l'énumération des erreurs possibles :

```
C_NoListWord, // ni un mot ni une liste
C_TwoDelete, // pas assez d'éléments pour en supprimer deux
C_BadListP, // liste de propriétés incorrecte
C_BadFormat // fichier de format erroné
```

LA CLASSE TGVPROPLISTENUMERATOR

Comme son nom l'indique, l'énumération est un moyen d'énumérer une liste d'éléments contenus dans un objet. Aussi bien Lazarus que Delphi en font un usage étendu à travers les objets tels que les listes.

La technique d'implémentation, finalement assez simple, obéit à un schéma type : on doit fournir l'élément en cours (pointé par une propriété en lecture seule) et être capable d'indiquer si l'on peut se déplacer vers l'élément suivant.

Cela donne dans le cas des listes de propriétés l'interface suivante :

```
TGVPropListEnumerator = class(TObject)
private
  fLst: TStringList;
  fIndex: Integer;
protected
  function GetCurrent: string; virtual;
public
  constructor Create(const Value: TStrings);
  destructor Destroy; override;
  function MoveNext: Boolean;
  property Current: string read GetCurrent;
end;
```

Comme les listes de propriétés fonctionnent grâce à une liste interne, il suffit d'en manipuler les éléments grâce à un index.



La gestion se fait grâce au système intégré des listes (paires **Names/Values**) qui se sert d'un caractère séparateur personnalisé : ¹⁴. Ce caractère ne doit par conséquent pas figurer dans la liste elle-même.



Les utilisateurs de Delphi pourront imaginer une unité fondée sur la classe **TDictionary**. Ils devraient y gagner en rapidité d'exécution.

LA CLASSE TGVPROPLIST

La classe **TGVPropList** fonctionne grâce à un champ privé **fNames** de type **TStringList**. Les méthodes sont organisées en trois blocs : méthodes générales, celles concernant les listes de propriétés et celles concernant les propriétés.

En voici l'interface :

```
TGVPropList = class(TObject)
private
  fName: TStringList; // listes
  fOnchange: TNotifyEvent; // notification de changement
  function GetLPByNum(N: Integer): string; // liste par numéro
  function GetLPByName(const Name: string): string; // liste par nom
  procedure SetLPByName(const Name, AValue: string); // écriture par nom
protected
  procedure Change; dynamic; // changement
public
  // constructeur de la classe
  constructor Create;
  // destructeur de la classe
  destructor Destroy; override; // destructeur
  // énumération
  function GetEnumerator: TGVPropListEnumerator;
```

¹⁴ Ce caractère est celui qui figure sous le 6 du pavé alphanumérique. Il s'obtient en pressant la touche AltGr et la touche du 6. Il est choisi grâce à la propriété **NameValueSeparator**.

```

// nettoie les listes de propriétés
procedure Clear;

// *** listes de propriétés ***
// renvoie la liste des listes de propriétés
function ListP: string;
// la liste existe-t-elle ?
function IsListP(const Name: string): Boolean;
// renvoie le numéro d'une liste de propriétés
function NumListP(const Name: string): Integer;
// crée ou met à jour la liste de propriétés
function UpDateListP(const Name, Prop, Value: string): Boolean;
// renvoie la valeur d'une liste
function ValListP(const Name: string): string;
// destruction d'une liste de propriétés
function RemoveListP(const Name: string): Boolean;
// valeur d'une liste de propriétés par numéro
function ValNumListP(N: Integer; out Name, Value: string): Boolean;
// la propriété N existe-t-elle?
function IsListPByNum(N: Integer): Boolean;
// renvoie le nombre de listes de propriétés
function CountListP: Integer;
// chargement des listes
procedure LoadFromFile(const FileName: string);
// sauvegarde des listes
procedure SaveToFile(const FileName: string);

// *** propriétés
// la propriété existe-t-elle ?
function IsProp(const Name, Prop: string): Boolean;
// renvoie le numéro d'une propriété
function NumProp(const Name, Prop: string): Integer;
// valeur d'une propriété
function ValProp(const Name, Prop: string): string; overload;
function ValProp(const Name, Prop: string; out Value: string)
  : Boolean; overload;
// destruction d'une propriété
function RemoveProp(const Name, Prop: string): Boolean;
// renvoie le nombre de propriétés attachées à une liste
function CountProps(const Name: string): Integer;
// valeur d'une propriété par numéro
function ValNumProp(const Name: string; N: Integer; out Prop: string)
  : Boolean; overload;
function ValNumProp(const Name: string; N: Integer): string; overload;
// liste des propriétés d'une liste
function ListOfProps(const Name: string): string;
// nom d'une propriété par numéro
function NameOfProp(const Name: string; N: Integer): string; overload;
function NameOfProp(const Name: string; N: Integer; out Prop: string)
  : Boolean; overload;
// changement dans la liste de propriétés
property OnChange: TNotifyEvent read fOnchange write fOnchange;
// liste de propriétés par numéro
property ListPByNum[N: Integer]: string read GetLPByNum; default;
// liste de propriétés par nom
property LPByName[const Name: string]: string read GetLPByName write SetLPByName;
end;

```

On remarquera que la propriété `ListPByNum` est définie comme `default`, ce qui signifie qu'elle peut être omise en employant simplement le numéro de la liste entre crochets.

Parmi les difficultés, on relèvera :

- ✓ Les listes de propriétés sont numérotées à partir de 1 et non de 0 comme en Pascal ;
- ✓ La méthode `RemoveProp` détruit aussi la liste si elle ne contient plus aucune propriété ;

- ✓ La méthode `UpdateListP` crée et met à jour les listes et les propriétés ;
- ✓ Il est fait un usage intensif de la fonction `Odd` qui vérifie qu'un nombre est impair : en effet, comme les propriétés sont organisées par paires, il est nécessaire de bien différencier le nom de la propriété (élément impair) de sa valeur (élément pair).

TEST DE L'UNITE TGVPROPLISTS

Comme d'habitude, le programme de test est décliné en quatre versions :

- ✓ Lazarus pour Windows 32 ;
- ✓ Lazarus pour Linux ;
- ✓ Delphi pour Windows 32 ;
- ✓ Delphi pour Windows 64.

Un fichier de listes préenregistrées permet de tester immédiatement `LoadFromFile` : « `ListPs.GPL` ». Il peut être édité par n'importe quel lecteur de textes simples (genre NotePad).

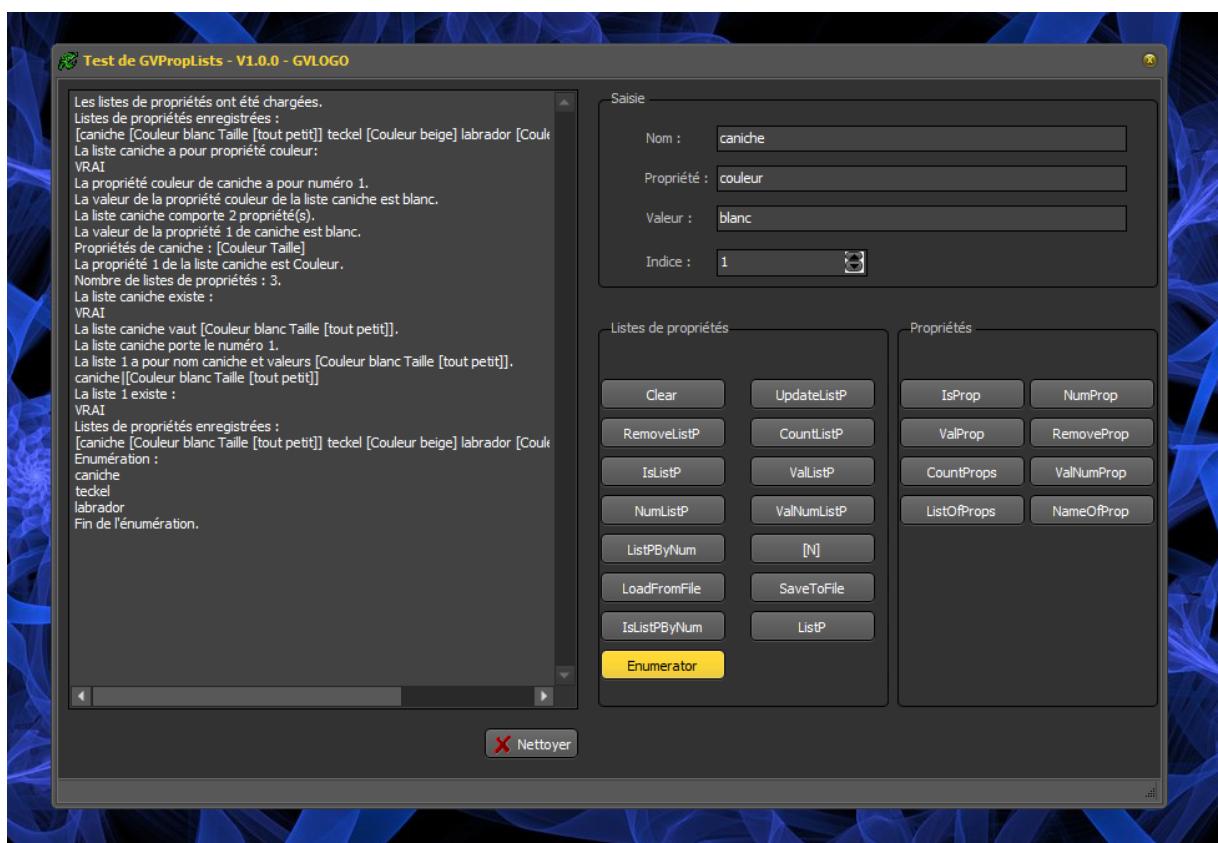


Figure 6 - Test des listes de propriétés avec Delphi

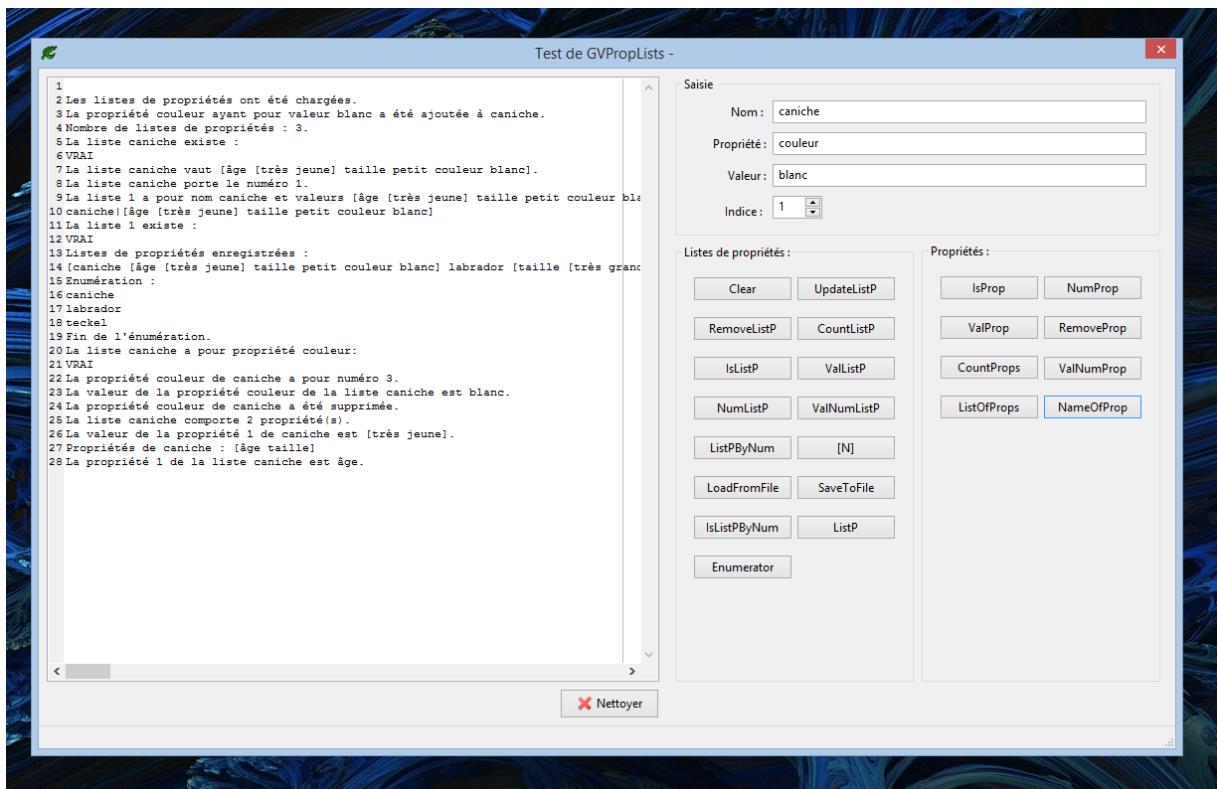


Figure 7 - Test des listes de propriétés avec Lazarus

LA TORTUE GRAPHIQUE

PRESENTATION

La tortue graphique est sans doute l'objet le plus connu du langage LOGO. Elle a fait son succès, mais aussi l'a limité très longtemps à une utilisation avec des enfants, conduisant les utilisateurs à négliger les listes qui permettent d'aborder des sujets plus ardus.

La tortue graphique est habituellement symbolisée par un triangle sur une surface de dessin. **GVLOGO** propose par ailleurs une tortue plus réalisée au format « png » ainsi qu'une tortue définie par l'utilisateur. Des ordres permettent de la faire dessiner. Les commandes disponibles sont multiples : l'utilisateur maîtrise son orientation, son trait, sa forme, sa couleur...

L'originalité de ce mode de dessin réside dans sa capacité à partir d'une orientation dans l'espace déterminée par la tortue elle-même : même s'il est possible de dessiner dans un repère orthonormé traditionnel, la plupart des commandes sont dépendantes de l'état en cours de la tortue. Ainsi, tourner de 90° à gauche se fera par rapport à l'orientation actuelle de la tortue : si son cap était de 45° par rapport à la surface de dessin, son nouveau cap sera de 135°.

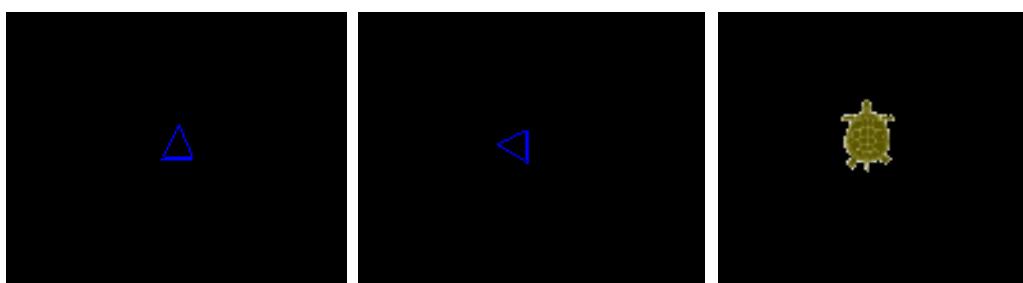


Figure 8 - Tortue avec cap 90

Tortue 180

Tortue 90 png

À l'origine, la tortue est au centre de la surface de dessin qui mesure 600 par 600 pixels : sa position d'origine est donc de [300 300]. Ses déplacements se comptent en pixels si les échelles des abscisses et des ordonnées sont fixées à 100 (valeur par défaut). Un déplacement négatif fait reculer la tortue.

Le cap de la tortue indique la direction des déplacements. Il est exprimé en degrés et va de 0 à 360. Le cap est de 90° par défaut et pointe vers le sommet de l'écran afin d'être en conformité avec le cercle trigonométrique. On visualise ce cap grâce à l'orientation du triangle qui symbolise la tortue : l'arrière de la tortue est un trait plus épais ; sa tête est donc le sommet du triangle qui marque l'intersection des deux lignes plus fines.

OPERATIONS AVEC LA TORTUE

CHAMP DE LA TORTUE

Le champ de la tortue est la surface sur laquelle elle peut évoluer. Le coin inférieur gauche de l'écran est la position d'abscisse 0 et d'ordonnée 0. Par défaut, le point supérieur droit a pour abscisse 600 et ordonnée 600¹⁵.

- **CLOS** : n'attend rien en entrée – ne renvoie rien – le champ de la tortue est limité à l'écran visible. Toute tentative de sortir de l'écran se soldera par un échec.

Exemple :

CLOS → -



Le champ est défini à **CLOS** par défaut.

- **ENROULE** : n'attend rien en entrée – ne renvoie rien – le champ de la tortue s'enroule sur lui-même : quand la tortue atteint un bord, elle réapparaît sur le bord opposé.

Exemple :

ENROULE → -

- **FENETRE** (raccourci : **FEN**) : n'attend rien en entrée – ne renvoie rien – le champ de la tortue s'étend au-delà de l'écran. La tortue peut donc disparaître du champ de vision de l'utilisateur.

Exemple :

FENETRE → -

- **ETATECRAN** : n'attend rien en entrée – renvoie un mot – le mot renvoyé indique l'état actuel de l'écran : **ENROULE**, **FENETRE** ou **CLOS**.

Exemple :

ECRIS ETATECRAN → CLOS

- **FIXEECHELLE** : attend une liste de deux entiers en entrée – ne renvoie rien – la primitive détermine les proportions du déplacement de la tortue. Par défaut, la valeur pour les abscisses (premier entier) et pour les ordonnées (second paramètre) est de 100. Une valeur moindre réduira les déplacements dans la proportion indiquée et une valeur supérieure l'augmentera. Par exemple, fixer le premier entier à 200 et le second à 50 fera que la tortue se déplacera deux fois plus vite horizontalement que la normale et deux fois plus lentement verticalement.

Exemple :

FIXECALEILLE [200 50] → -

¹⁵ En Pascal, les graphiques ont des coordonnées inversées par rapport aux habitudes en mathématiques. Ainsi le coin supérieur gauche a pour abscisse 0 et ordonnée 0. Plus l'ordonnée augmente plus le point visé se place vers le bas de l'écran.

- **FIXEECHELLEX** : attend un entier en entrée – ne renvoie rien – la primitive détermine la proportion du déplacement de la tortue selon l'axe des abscisses.

Exemple :

FIXEACHELLEX 200 → -

- **FIXEECHELLEY** : attend un entier en entrée – ne renvoie rien – la primitive détermine la proportion du déplacement de la tortue selon l'axe des ordonnées.

Exemple :

FIXEACHELLEY 50 → -

- **ECHELLE** : n'attend rien en entrée – renvoie une liste de deux entiers – la liste renvoyée contient un premier entier qui indique l'échelle des déplacements de la tortue selon l'axe des abscisses et un second entier qui indique l'échelle selon l'axe des ordonnées.

Exemple :

ECRIS ECHELLE → 100 100

- **VIDEECRAN** (raccourci : **VE**) : n'attend rien en entrée – ne renvoie rien – l'espace de la tortue est nettoyé et la tortue retrouve sa place et son cap d'origine, ainsi que toutes ses valeurs par défaut.

Exemple :

VIDEECRAN → -

- **ORIGINE** : n'attend rien en entrée – ne renvoie rien – la tortue retrouve sa position d'origine en [300 300] et son cap initial de 90. L'état du crayon n'est pas modifié. Si le crayon est baissé, le déplacement de la tortue laisse une trace.

Exemple :

ORIGINE → -

- **NETTOIE** : n'attend rien en entrée – ne renvoie rien – l'espace de la tortue est nettoyé, sans changer la position de la tortue ni son cap, pas plus que les valeurs du crayon.

Exemple :

NETTOIE → -

- **FIXECOULEURFOND** (raccourci : **FCF**) : attend un entier en entrée – ne renvoie rien – la primitive détermine la couleur de fond de l'écran de la tortue.

Exemple :

FIXECOULEURFOND 19 → -

Tableau 1 - Couleurs de fond de l'écran de la tortue

Noir	0	
Bleu ciel	1	
Bleu	2	
Crème	3	
Gris foncé	4	
Fuchsia	5	
Gris	6	
Vert	7	
Vert citron	8	
Gris clair	9	
Marron	10	
Gris moyen	11	
Vert menthe	12	
Bleu marine	13	
Vert olive	14	

Violet	15	
Rouge	16	
Argent	17	
Bleu ciel	18	
Sarcelle	19	
Blanc	20	
Jaune	21	



Du fait d'implémentations différentes, le résultat de **FIXECOULEURFOND** sera différent avec Delphi et Lazarus : le premier changera simplement le fond tandis que le second réinitialisera complètement le dessin¹⁶.

- **COULEURFOND** (raccourci : **CF**) : n'attend rien en entrée – renvoie un entier – l'entier renvoyé correspond à la couleur du fond de l'écran de la tortue.

Exemple :

ECRIS COULEURFOND → 19

DEPLACEMENTS DE LA TORTUE

- **AVANCE** (raccourci : **AV**) : attend un entier en entrée – ne renvoie rien – la tortue avance du nombre de pixels indiqués par le paramètre en entrée, sauf si les échelles ont été changées. Elle laisse une trace sur la zone de dessin à moins que son crayon ne soit levé ou qu'elle soit en mode gomme.

Exemple :

AVANCE 45 → -

- **RECULE** (raccourci : **RE**) : attend un entier en entrée – ne renvoie rien – la tortue recule du nombre de pixels indiqués par le paramètre en entrée, sauf si les échelles ont été changées. Elle laisse une trace sur la zone de dessin à moins que son crayon ne soit levé ou qu'elle soit en mode gomme.

Exemple :

RECULE 50 → -

- **GAUCHE** (raccourci : **TG**) : attend un réel en entrée – ne renvoie rien – la tortue tourne à gauche du nombre de degrés indiqué en entrée.

Exemple :

GAUCHE 90 → -



Pour rappel : la gauche indiquée est celle par rapport à la tortue et non la gauche de l'écran ! Cette remarque vaut évidemment pour la primitive **DROITE**.

- **DROITE** (raccourci : **TD**) : attend un réel en entrée – ne renvoie rien – la tortue tourne à droite du nombre de degrés indiqué en entrée.

Exemple :

DROITE 90 → -

- **FIXEPOS** (raccourci : **FPOS**) : attend une liste de deux entiers en entrée – ne renvoie rien – la tortue est envoyée à la position indiquée par la liste : le premier élément fixe l'abscisse tandis que le second détermine l'ordonnée.

Exemple :

FIXEPOS [200 100] → -

¹⁶ Dans les versions ultérieures, il est prévu que les deux se comporteront comme Delphi : pour le moment, le tracé en transparence de la tortue sur une surface elle-même transparente pose avec Lazarus un problème que je n'ai pas su résoudre !

- **FIXEXY** (raccourci : **FXY**) : attend deux entiers en entrée – ne renvoie rien – la tortue est envoyée à la position indiquée par les paramètres : le premier élément fixe l'abscisse tandis que le second détermine l'ordonnée.

Exemple :

FIXEXY 200 100 → -

- **FIXEX** (raccourci : **FX**) : attend un entier en entrée – ne renvoie rien – l'abscisse de la tortue est fixée à l'entier fourni en entrée.

Exemple :

FIXEX 200 → -

- **FIXEY** (raccourci : **FY**) : attend un entier en entrée – ne renvoie rien – l'ordonnée de la tortue est fixée à l'entier fourni en entrée.

Exemple :

FIXEY 100 → -

- **POS** : n'attend rien en entrée – renvoie une liste de deux entiers – la liste renvoyée contient un premier entier qui indique l'abscisse de la tortue et un second entier qui indique son ordonnée.

Exemple :

ECRIS POS → 200 100

- **XCOOR** : n'attend rien en entrée – renvoie un entier – l'entier renvoyé indique l'abscisse de la tortue.

Exemple :

ECRIS XCOOR → 200

- **YCOOR** : n'attend rien en entrée – renvoie un entier – l'entier renvoyé indique l'ordonnée de la tortue.

Exemple :

ECRIS YCOOR → 100

- **FIXEVITESSE** : attend un entier en entrée – ne renvoie rien – la vitesse de la tortue est fixée à l'entier fourni en entrée.

Exemple :

FIXEY 100 → -



La vitesse est relative aux performances de l'ordinateur sur lequel est installé **GVLOGO**. Elle est automatiquement limitée à 100.

- **VITESSE** : n'attend rien en entrée – renvoie un entier – l'entier renvoyé indique la vitesse de dessin de la tortue.

Exemple :

ECRIS VITESSE → 100

CAP DE LA TORTUE

- **FIXECAP** (raccourci : **FCAP**) : attend un réel en entrée – ne renvoie rien – le cap de la tortue est fixé au nombre fourni en entrée. Le cap est compris entre 0 et 360.

Exemple :

FIXECAP 90 → -

- **CAP** : n'attend rien en entrée – renvoie un entier – l'entier renvoyé indique le cap de la tortue.

Exemple :

ECRIS CAP → 90

- **VERS** : attend une liste de deux entiers en entrée – renvoie un entier – L'entier renvoyé indique le cap que devrait avoir la tortue pour pointer vers le point dont les coordonnées sont fournies en entrée.

Exemple :

VERS [0 0] → 214



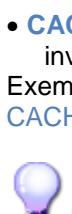
Pour rappel : le cap 90 est celui qui pointe vers le haut de l'écran. Le cap 0 pointe vers sa droite. En fait, le cap augmente en tournant dans le sens inverse des aiguilles d'une montre.

ETAT DE LA TORTUE

- **MONTRETORTUE** (raccourci : **MT**) : n'attend rien en entrée – ne renvoie rien – la tortue devient visible.

Exemple :

MONTRETORTUE → -



La tortue dessine plus rapidement lorsqu'elle est invisible.

- **CACHETORTUE** (raccourci : **CT**) : n'attend rien en entrée – ne renvoie rien – la tortue devient invisible.

Exemple :

CACHETORTUE → -

La tortue dessine plus rapidement lorsqu'elle est invisible.

- **VISIBLE?** : n'attend rien en entrée – renvoie un booléen – la primitive renvoie "VRAI si la tortue est visible, "FAUX sinon.

Exemple :

ECRIS VISIBLE? → VRAI

- **ETATTORTUE** : n'attend rien en entrée – renvoie une liste – la liste renvoyée fournit les données relatives à la tortue : son abscisse, son ordonnée, son cap, sa taille, sa visibilité et si elle est figurée par un triangle.

Exemple :

ECRIS ETATTORTUE → 300 300 90 8 VRAI VRAI

- **FIXEETATTORTUE** : attend une liste en entrée – ne renvoie rien – l'état de la tortue est fixé selon le contenu de la liste : abscisse, ordonnée, orientation, taille, visibilité et type de tortue. La tortue dessine durant le changement si le crayon est baissé.

Exemple :

FIXEETATTORTUE [100 100 0 8 VRAI VRAI] → -

- **TORTUENORMALE** : n'attend rien en entrée – ne renvoie rien – quand elle est visible, la tortue est symbolisée par un triangle. C'est le mode par défaut.

Exemple :

TORTUENORMALE → -

- **TORTUEVERTE** : n'attend rien en entrée – ne renvoie rien – quand elle est visible, la tortue est dessinée à partir d'images png.

Exemple :

TORTUEVERTE → -



L'image est plus sympathique, mais le dessin est un peu plus lent et la précision pour les angles moindre (5°).

- **FIXETAILLETORTUE** : attend un entier en entrée – ne renvoie rien – la taille de la tortue est fixée au nombre fourni en entrée. La taille ne peut excéder 20 et ne concerne que le type par défaut, à savoir le triangle. La valeur par défaut est de 8.

Exemple :

FIXETAILLETORTUE 10 → -



La taille ne peut excéder 20 et ne concerne que le type par défaut, à savoir le triangle. La valeur par défaut est de 8.

- **TAILLETORTUE** : n'attend rien en entrée – renvoie un entier – l'entier renvoyé indique la taille actuelle de la tortue.

Exemple :

ECRIS TAILLETORTUE → 8



La taille renvoyée si la tortue n'est pas la tortue par défaut sera toujours de 8.

LE CRAYON

- **BAISSECRAYON** (raccourci : **BC**) : n'attend rien en entrée – ne renvoie rien – le crayon est baissé et donc dessine avec les attributs qui lui ont été affectés. C'est la position par défaut du crayon.

Exemple :

BAISSECRAYON → -

- **LEVECRAYON** (raccourci : **LC**) : n'attend rien en entrée – ne renvoie rien – le crayon est levé et donc ne dessine pas lorsque la tortue est déplacée.

Exemple :

LEVECRAYON → -

- **BAISSE?** : n'attend rien en entrée – renvoie un booléen – la primitive renvoie "VRAI si le crayon est baissé, "FAUX sinon.

Exemple :

ECRIS BAISSE? → VRAI

- **FIXECOULEURCRAYON** (raccourci : **FCC**) : attend un entier en entrée – ne renvoie rien – la primitive détermine la couleur d'écriture du crayon de la tortue¹⁷.

Exemple :

FIXECOULEURCRAYON 5 → -



Avec Delphi, la couleur fuchsia est à éviter pour le crayon, car elle sert de couleur de transparence pour pouvoir dessiner indépendamment le fond. Écrire avec cette couleur sera donc... invisible !

- **COULEURCRAYON** (raccourci : **CC**) : n'attend rien en entrée – renvoie un entier – l'entier renvoyé correspond à la couleur du crayon de la tortue.

Exemple :

ECRIS COULEURCRAYON → 5

- **FIXEPAISSEURCRAYON** (raccourci : **FEC**) : attend un entier en entrée – ne renvoie rien – l'épaisseur du trait du crayon de la tortue est fixée au nombre fourni en entrée.

Exemple :

FIXEPAISSEURCRAYON 2 → -

¹⁷ Pour les couleurs admises, se référer au tableau n°1 fourni avec **FIXECOULEURFOND**.

- **EPAISSEURCRAYON** (raccourci : **EC**) : n'attend rien en entrée – renvoie un entier – l'entier renvoyé correspond à l'épaisseur de trait du crayon de la tortue.

Exemple :

ECRIS EPAISSEURCRAYON → 2

- **INVERSE** : n'attend rien en entrée – ne renvoie rien – le crayon écrit dans la couleur complémentaire de la couleur en cours.

Exemple :

INVERSE → -

- **GOMME** : n'attend rien en entrée – ne renvoie rien – le crayon efface en avançant.

Exemple :

GOMME → -



Avec Delphi, si la couleur de l'écran est modifiée, les traits effacés réapparaîtront.

- **NORMAL** : n'attend rien en entrée – ne renvoie rien – le crayon n'inverse plus et n'efface plus.

Exemple :

NORMAL → -

- **ETATCRAYON** : n'attend rien en entrée – renvoie une liste – la liste renvoyée fournit les données relatives au crayon de la tortue : sa couleur, son épaisseur, s'il écrit ou non, si l'écriture est inversée ou non.

Exemple :

ECRIS ETATCRAYON → 1 1 VRAI FAUX

- **FIXEETATCRAYON** (raccourci : **FEC**) : attend une liste en entrée – ne renvoie rien – l'état du crayon est fixé selon le contenu de la liste : couleur, épaisseur, écriture ou non, inversion ou non.

Exemple :

FIXEETATTORTUE [1 2 VRAI FAUX] → -

FORMES PREDEFINIES

GVLOGO fournit plusieurs formes prédefinies qui seront dessinées soit à une position indiquée par l'utilisateur soit par rapport à la position de la tortue. Ces formes ne sont pas orientables suivant un angle (par exemple, celui de l'orientation de la tortue), mais de telles formes sont facilement programmables en LOGO lui-même.



Il faudra permettre aux primitives d'être modifiables pour définir son propre carré ou son propre rectangle avec le même identifiant. Une autre solution consiste en définissant des procédures avec des noms adaptés : **MonCarre**, **MonRectangle**...

- **RECTANGLE** : attend une liste en entrée – ne renvoie rien – un rectangle est dessiné avec les coordonnées de deux points fournies en entrée¹⁸.

Exemple :

RECTANGLE [100 100 200 200] → -

¹⁸ Le premier point est celui situé en haut à gauche tandis que le second est celui dessiné en bas à droite.

- **RECTANGLEARRONDI** : attend une liste en entrée – ne renvoie rien – un rectangle arrondi est dessiné avec les coordonnées de deux points fournies en entrée.

Exemple :

RECTANGLEARRONDI [50 100 150 200] → -

- **CARRE** : attend une liste en entrée – ne renvoie rien – un carré est dessiné avec les coordonnées du point supérieur gauche suivies de la longueur de son côté.

Exemple :

CARRE [20 20 50] → -

- **ELLIPSE** : attend une liste en entrée – ne renvoie rien – une ellipse est dessinée avec les coordonnées de deux points fournies en entrée.

Exemple :

ELLIPSE [100 100 200 200] → -

- **CERCLE** : attend une liste en entrée – ne renvoie rien – un cercle est dessiné avec les coordonnées du premier point du cercle suivies de la longueur de son rayon.

Exemple :

CERCLE [20 20 50] → -

- **REmplis** : n'attend rien en entrée – ne renvoie rien – les figures seront pleines et par conséquent recouvriront celles qui seront placées dans leur espace de dessin.

Exemple :

REmplis → -

- **LAISSEVOIR** : n'attend rien en entrée – ne renvoie rien – les figures seront transparentes et par conséquent ne recouvriront pas celles qui seront placées dans leur espace de dessin.

Exemple :

LAISSEVOIR → -

IMPLEMENTATION DE LA TORTUE

L'implémentation de la tortue pose trois types de problèmes :

- ✓ mathématiques : il faut se plonger dans quelques formules de trigonométrie afin de tenir compte de l'orientation de la tortue pendant ses déplacements ;
- ✓ graphiques : il faut gérer un fond indépendant du tracé et une tortue qui doit laisser une trace en se déplaçant à la manière d'un *sprite*¹⁹ ;
- ✓ de communication par événements : une tortue personnalisée sera tributaire des dessins envoyés par l'application tandis que le fond dépendra du contrôle sous-jacent à la surface de la tortue (généralement un panneau). Il est aussi intéressant de notifier tout changement intervenu en rapport avec la tortue, par exemple pour afficher son état dans la barre de statut.

CONSTANTES

L'unité n'est modifiée que modestement par l'ajout de quelques constantes :

```
{ tortue }

DgToRad = Pi / 180; // pour les conversions en radians
RadToDg = 180 / Pi; // pour les conversions en degrés
DefaultScale = 100; // échelle par défaut
DefaultHeading = 90; // cap par défaut
TurtleDefaultSize = 8; // taille d'une tortue par défaut
TurtleMaxSize = 20; // taille maximale de la tortue
TurtleMaxSpeed = 100 ; // vitesse maximale de la tortue
```

¹⁹ Un *sprite* (parfois appelé lutin en français) est un objet qui se déplace sur une surface. On doit avoir l'impression qu'il glisse sur cette surface.

Quelques types viennent compléter le tout afin de définir le type d'écran et de tortue :

```
{ tortue }

// type d'écrans : enroule, fenêtre illimitée ou champ clos
TScreenTurtle = (teWin, teGate, teRoll);
// types de tortue
TTurtleKind = (tkTriangle, tkPng, tkOwner);
```

UN PEU DE MATHEMATIQUES

Afin de déterminer un déplacement de la tortue sur la surface de travail, il faut connaître ses coordonnées pas à pas en fonction de son cap. Pour cela, on utilisera les fonctions sinus et cosinus.

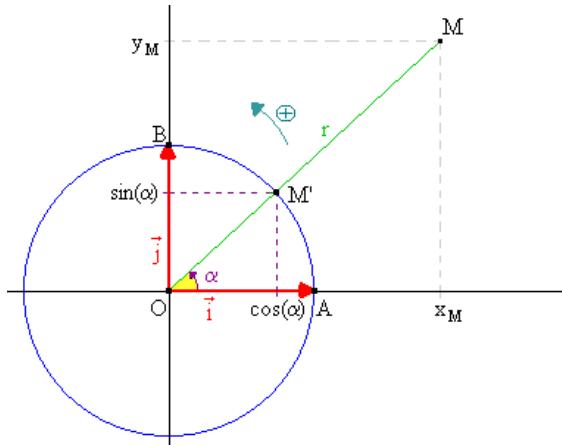


Figure 9 - Point avec sinus et cosinus

Aussi bien Lazarus que Delphi fournissent une procédure bien pratique²⁰ pour les déterminer : **SinCos** définie pour un réel étendu de la manière suivante :

```
procedure SinCos(const Theta: Extended; var Sin, Cos: Extended);
```

Theta est l'angle exprimé en radians et les sinus et cosinus sont renvoyés dans les paramètres appropriés.

Voici par exemple comment elle est utilisée dans la procédure **Move** qui déplace la tortue :

```
procedure TGVTurtle.Move(const Value: Real);
// *** la tortue se déplace ***
var
  SinT, CosT: Extended;
  TX, TY: Real;
begin
  // calcul du cosinus et du sinus du cap
  SinCos((fHeading - 90) * DgToRad, SinT, CosT);
  // calcul des nouvelles coordonnées
  TX := fX - Value * SinT * (fScaleX / DefaultScale);
  TY := fY + Value * CosT * (fScaleY / DefaultScale);
  SetPos(Round(TX), Round(TY)); // déplacement si possible
end;
```

fX et **fY** sont les coordonnées avant le déplacement. On remarquera la transformation des degrés en radians pour l'orientation ainsi que l'utilisation des échelles pour déterminer le nombre réel de pixels à franchir. La position est arrondie puisque on ne peut dessiner que des pixels à des positions entières.

²⁰ ... et deux fois plus rapide qu'un appel successif aux fonctions voulues !

De la même manière, lorsqu'on voudra dessiner le triangle représentant la tortue, des formules similaires seront utilisées :

```
procedure TGV Turtle.ToggleTurtleTriangle;
// *** montre/cache la tortue triangle ***
var
  CosT, SinT: Extended;
  X1, X2, X3, Y1, Y2, Y3: Integer;
  PenSave: TPen;
begin
  // calcul des coordonnées des points de la tortue
  SinCos((90 + Heading) * DgToRad, SinT, CosT);
  X1 := Round(CoordX + Size * CosT - SinT);
  Y1 := Round(CoordY + Size * SinT + CosT);
  X2 := Round(CoordX - Size * CosT - SinT);
  Y2 := Round(CoordY - Size * SinT + CosT);
  X3 := Round(CoordX - CosT + (Size shl 1) * SinT);
  Y3 := Round(CoordY - SinT - (Size shl 1) * CosT);
  [...]
```



L'utilisation de `shl` au lieu d'une multiplication par deux traditionnelle est très efficace en termes de vitesse d'exécution.

Une autre fonction complexe est `Towards` qui permet de rendre l'orientation nécessaire de la tortue pour qu'elle pointe vers un point. Elle utilise la fonction mathématique `ArcTan` qui permet de déterminer un angle dont la tangente est donnée.

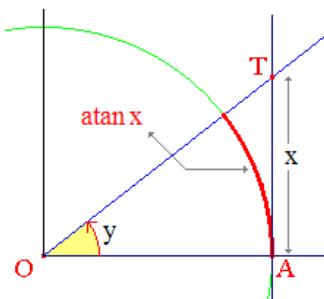


Figure 10 - Arc Tangente

En voici le listing²¹ :

```
function TGV Turtle.Towards(const X, Y: Integer): Real;
// *** renvoie le cap vers un point ***
var
  PX, PY: Integer;
begin
  PX := CoordX - X; // calcul des différences entre les points
  PY := Y - CoordY;
  Result := 0; // suppose 0
  // évalue suivant les calculs
  if ((PX = 0) and (PY < 0)) then
    Result := 270
  else if ((PX = 0) and (PY > 0)) then
    Result := 90
  else if ((PX > 0) and (PY >= 0)) then
    Result := 180 - ArcTan(PY / PX) * RadToDg
  else if ((PX < 0) and (PY > 0)) then
    Result := (ArcTan(PY / Abs(PX)) * RadToDg)
  else if ((PX < 0) and (PY <= 0)) then
    Result := 360 - (ArcTan(PY / PX) * RadToDg)
  else if ((PX > 0) and (PY < 0)) then
```

²¹ La détermination des cas a été établie dans une ancienne FAQ concernant Delphi.

```
Result := 180 + (ArcTan(Abs(PY) / PX) * RadToDg);
end;
```

Enfin, la fonction **Distance** permet de calculer la distance entre la tortue et un point donné. Elle fait appel à de simples calculs à partir de triangles rectangles, donc en appliquant le théorème de Pythagore.

La formule bien connue est :

$$\sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}.$$

avec x_a, y_a les coordonnées du premier point et x_b, y_b celles du second.

La fonction en Pascal donne²² :

```
function TGV Turtle.Distance(const X, Y: Integer): Real;
// *** renvoie la distance de la tortue à un point donné ***
begin
  Result := Sqr(Sqr(X - CoordX) + Sqr(Y - CoordY));
end;
```

SE DEPLACER SUR L'ECRAN

En dehors de ces quelques notions de mathématiques, le déplacement de la tortue à l'écran pose des problèmes d'affichage. Suivant le type de tortue, deux solutions ont été adoptées :

- ✓ un affichage en mode ou exclusif pour le triangle ;
- ✓ un affichage par transparence pour la tortue personnalisée.

Rappelons que le mode ou exclusif est particulièrement intéressant pour le dessin. En effet, selon l'algèbre booléenne, il fournit la table de vérité suivante :

0 sur 0 → 0	, puis en recommençant : 0 sur 0 → 0
0 sur 1 → 1	, puis en recommençant : 0 sur 1 → 1
1 sur 0 → 1	, puis en recommençant : 1 sur 1 → 0
1 sur 1 → 0	, puis en recommençant : 1 sur 0 → 1

Le principe est de valoir 1 si une seule des deux valeurs de départ est 1. Si on applique deux fois ce ou exclusif, on revient à la valeur initiale. En transposant ce principe à l'affichage, on remarque que les pixels d'origine sont conservés.

C'est exactement ce que fait la procédure qui dessine le triangle de la tortue :

```
try
  PenSave.Assign(Canvas.Pen);
  with Canvas.Pen do
    begin
      Style := psSolid;
      Mode := pmXor; // tortue visible en mode ou exclusif
      MoveTo(X1, Y1); // dessin de la tortue en mode ou exclusif
      Width := 2;
      LineTo(X2, Y2);
      Width := 1;
      LineTo(X3, Y3);
      LineTo(X1, Y1);
      MoveTo(CoordX, CoordY);
      Assign(PenSave); // récupération du crayon
    end;
```

²² **CoordX** et **CoordY** sont les coordonnées actuelles de la tortue.

```

finally
  PenSave.Free; // libération du crayon provisoire
end;

```

Après avoir été sauvegardé, le crayon est modifié pour écrire en mode ou exclusif. La procédure **MoveTo** déplace le curseur de l'écriture sur le canevas sans écrire tandis que **LineTo** écrit. La taille du trait est modifiée pour dessiner l'arrière de la tortue grâce à la propriété **Width**.

La plupart des routines utilisées ont le même squelette qui consiste à cacher provisoirement la tortue, d'effectuer la tâche puis de rétablir la tortue dans son état d'origine. Par exemple, **DoGo** qui centralise tous les déplacements procède ainsi :

```

procedure TGV Turtle.DoGo(const X, Y: Integer);
// *** effectue un déplacement de la tortue ***
var
  TV: Boolean;
begin
  TV := TurtleVisible; // sauvegarde la tortue
  try
    TurtleVisible := False; // la cache
    // si champ clos et hors limites => erreur
    if (Screen <> teGate) or IsWithinLimits(X, Y) then
      begin
        fX := X;
        fY := Y;
        // ralentit le dessin
        Sleep(2 * Speed);
        // dessine
        if PenDown then
          LineTo(X, Y)
        else
          MoveTo(X, Y);
      end;
    finally
      TurtleVisible := TV; // restaure la tortue
    end;
  end;

```



Bien que ce soit une erreur que de vouloir faire sortir la tortue de son espace alors qu'elle est en mode champ clos, **GVLOGO** se contente de ne pas déplacer la tortue dans ce cas. On pourrait aussi déclencher une exception, mais, d'un point de vue pédagogique, il a semblé plus intéressant de laisser à l'utilisateur le soin de comprendre ce qu'il se passe dans ce cas.

On notera aussi que les procédures **LineTo** et **MoveTo** adaptent les coordonnées fournies afin de les rendre cohérentes avec un repère dont l'origine est située en bas à gauche de la surface de dessin et non en haut à gauche comme en Pascal.

En ce qui concerne la tortue personnalisée, les données sont tout à fait différentes. Le choix a été fait de partir d'une série de « clichés » de la tortue au format « png », car il supporte très bien la transparence. Malheureusement, les algorithmes qui permettent les rotations sont complexes : c'est pourquoi la solution d'images préenregistrées a été choisie. Elle est par ailleurs assez rapide.

Le principe de son dessin est relativement simple :

- ✓ on conserve une vue de la surface de dessin sans la tortue ;
- ✓ on récupère la bonne image de la tortue ;
- ✓ on dessine la tortue à l'écran ;
- ✓ on restitue la surface sauvegardée en cas de déplacement.

Voici ce que donne la procédure en charge de ce travail :

```
procedure TGV Turtle.ToggleTurtlePNG;
// *** montre/cache la tortue png ***
var
  CosT, SinT: Extended;
  X, Y: Integer;
begin
  if fTurtleVisible then // visible ?
  begin
    BeforeChange;
    // calcul des coordonnées de la tortue
    SinCos((90 + Heading) * DgToRad, SinT, CosT);
    X := Round(CoordX + CosT - SinT);
    Y := Round(CoordY + SinT + CosT);
    // copie de l'écran
    fTemplImg.Assign(Picture.Bitmap);
    // sauvegarde de l'ancienne image
    fOldImg.Assign(Picture.Bitmap);
    {$IFDEF Delphi}
    fOldImg.TransparentColor := clWhite;
    {$ENDIF}
    // copie de la tortue .png
    with fTemplImg do
    begin
      {$IFDEF Delphi}
      TransparentColor := clWhite;
      {$ENDIF}
      Canvas.Draw(X - (fTurtleImg.Width shr 1),
                  cY(Y) - (fTurtleImg.Height shr 1), fTurtleImg);
    end;
    // copie vers l'écran
    Canvas.Draw(0, 0, fTemplImg);
  end
  else
    // on rétablit l'image sans la tortue
    Canvas.Draw(0, 0, fOldImg);
end;
```



Le comportement de la tortue avec Lazarus était erratique : j'ai préféré garder une structure proche de celle mise en œuvre pour Delphi tout en ne l'exploitant pas vraiment en ce qui concerne le fond de la surface de dessin. Ainsi, au lieu de se servir de la surface du contrôle de support comme fond, la version Lazarus peint directement sur son propre canevas. En effet, la transparence fonctionne avec Lazarus dès qu'il s'agit de dessiner une forme simple sur le canevas ; mais si l'on désire déplacer la tortue sur une surface transparente qui prend sa couleur de fond sur un contrôle, le dessin en entier disparaît... Il résulte de la solution provisoire adoptée qu'un changement de couleur de fond réinitialise complètement le dessin. On trouverait sans doute une solution en utilisant des bibliothèques telles que BGRABitmap ou Graphics32.

Par ailleurs, on assiste, en exploitant la GDI de Windows, à d'étonnantes phénomènes concernant les couleurs et/ou les formes dessinées. Il serait sans doute préférable de travailler avec des bibliothèques GDI+, OPENGL, SDL, DirectX... On garantirait ainsi une certaine stabilité d'affichage. Avec la même unité de départ, on constatera ainsi que la tortue triangulaire de Delphi a une couleur inversée du trait qu'elle va dessiner alors qu'elle est de la même couleur que le trait avec Lazarus.



En Delphi, afin de rendre transparente la surface de dessin et d'afficher un fond indépendant, une couleur doit être « sacrifiée » : le choix pour **GVLOGO** est celui communément opéré, à savoir la couleur **clFuchsia**.

LES EVENEMENTS

La récupération de l'image désirée de la tortue est effectuée par le programme qui exploite l'unité **TGVTurtles**. Même si les multiples liens entre les procédures et les unités rendent l'ensemble plutôt complexe, le principe de fonctionnement est simple : quand l'unité **TGVTurtles** a besoin de dessiner la tortue, elle exécute la procédure **BeforeChange** qui envoie en paramètre essentiel l'orientation de la tortue. En retour, elle s'attend à ce que l'unité appelée ou le programme ait affecté la bonne image à sa propriété **TurtleImg**.

Autrement dit, il s'agit d'une sorte d'aller-retour entre les deux unités : la première envoie un pointeur vers elle-même et l'orientation de la tortue si elle trouve une procédure affectée à son champ privé **fOnBeforeChange** ; la seconde récupère l'orientation de la tortue, cherche l'image correspondante et l'affecte en retour à la propriété **TurtleImg** de l'unité génératrice de l'événement.

La méthode **BeforeChange** est déclarée ainsi :

```
// gestion de l'action avant le changement
procedure BeforeChange; dynamic;
```

Et voici sa définition :

```
procedure TGVTurtle.BeforeChange;
// *** gestion avant le changement ***
// (permet de mettre à jour une image pour la tortue avant de la dessiner)
begin
  if Assigned(fOnBeforeChange) then
    fOnBeforeChange(Self, Round(Heading));
end;
```

On voit que la procédure fait elle-même référence au champ privé **fOnBeforeChange** défini ainsi :

```
fOnBeforeChange: TTurtleBeforeEvent; // notification avant cap changé
```

Quant au type **TTurtleBeforeEvent**, il fournit un squelette de la procédure appelante :

```
TTurtleBeforeEvent = procedure(Sender: TObject; cHeading: Integer) of object;
```

Ce squelette est essentiel puisqu'il permet aux unités de communiquer entre elles.

Voici à présent comment le programme de test opère pour répondre à la requête de l'unité **TGVTurtles** :

```
procedure TMainFormGVVTurtles.TurtleBeforePaint(Sender: TObject;
  cHeading: Integer);
// image de la tortue avant son affichage
var
  BitM: TBitmap;
begin
  // charge l'image de la tortue
  BitM := TBitmap.Create;
  try
    // les images de la tortue sont proposées tous les 5 degrés
    iTurtle.GetBitmap(Round(GVTurtle.Heading) div 5, BitM);
    // celle qui correspond est assignée au bitmap
    GVTurtle.TurtleImg.Assign(BitM);
  finally
    BitM.Free;
  end;
end;
```

On remarquera que **fOnBeforeChange** de l'unité **TGVTurtles** et **TurtleBeforePaint** du programme de test ont exactement le même en-tête calqué sur **TTurtleBeforeEvent**. Peu importe que

les noms soient différents : ce qui compte, ce sont les paramètres absolument identiques dans leur nombre, leur agencement et leurs types.

Pour que l'ensemble fonctionne, il faut évidemment affecter la procédure appelée au gestionnaire d'événements approprié avec une ligne du genre :

```
GVTurtle.OnBeforeChange := TurtleBeforePaint;
```

Ici, **GVTurtle** est une instance de la classe **TGVTurtle**. Le programme « sait » à présent quelle procédure il doit appeler lorsque l'événement est déclenché.

Un second événement plus simple est déclenché à chaque modification de la tortue : il s'agit de **TTurtleEvent**.

```
// changement de la tortue
TTurtleEvent = procedure(Sender: TObject; cX, cY, cHeading: Integer;
cVisible, cDown: Boolean; cColor: TColor) of object;
```

Cet événement transmet les coordonnées de la tortue, son cap, l'état de sa visibilité, celui de sa capacité à écrire et la couleur d'écriture du crayon. Son intérêt est, par exemple, de pouvoir renseigner en temps réel une barre de statut qui affichera en clair ces données pour informer l'utilisateur.

Du côté de l'unité **TGVTurtles**, on retrouve :

- ✓ un champ privé qui conservera la procédure affectée si elle existe :

```
fOnchange: TTurtleEvent; // changement notifié
```

- ✓ une procédure appelée quand nécessaire et en charge d'appeler elle-même la procédure de l'unité qui exploite la tortue :

```
// gestion du changement
procedure Change; dynamic;
```

- ✓ la définition de cette procédure :

```
procedure TGVTurtle.Change;
// *** gestion du changement ***
begin
// le crayon inverse-t-il ?
fPenInverse := (Canvas.Pen.Mode = pmNot);
// le crayon efface-t-il ?
fPenRubber := (Canvas.Pen.Color = Canvas.Brush.Color);
if Assigned(fOnchange) then // on exécute le gestionnaire s'il existe
  fOnchange(Self, CoordX, CoordY, Round(Heading), TurtleVisible, PenDown,
  Canvas.Pen.Color);
```

En dehors des premières lignes qui profitent de cet appel pour mettre à jours des marqueurs concernant l'inversion du crayon et son mode gomme, il ressemble tout à fait au gestionnaire vu précédemment : si le champ privé **fOnChange** est affecté, on envoie les éléments nécessaires au fonctionnement de la procédure visée.

Du côté du programme de test, on retrouve :

- ✓ la déclaration de la procédure qui sera appelée en se conformant au squelette fourni par **TTurtleEvent** :

```
// message de la tortue
procedure TurtleState(Sender: TObject; cX, cY, cHeading: Integer;
cVisible, cDown: Boolean; cColor: TColor);
```

- ✓ sa définition qui affiche les données fournies sur la barre de statut :

```
procedure TMainFormGVTurtles.TurtleState(Sender: TObject;
  cX, cY, cHeading: Integer; cVisible, cDown: Boolean; cColor: TColor);
// état de la tortue
begin
  // données de la tortue
  statusbar.Panels[1].Text := 'X: ' + IntToStr(cX) + ' Y: ' + IntToStr(cY) +
    ' Cap: ' + IntToStr(cHeading) + ' Visible: ' + IfThen(cVisible, P_True,
    P_False) + ' Baissé: ' + IfThen(cDown, P_True, P_False);
end;
```

- ✓ afin de créer le lien entre les deux unités, son affectation lors de la création de la fiche :

```
GVTurtle.OnChange := TurtleState; // gestionnaire de changement
```

LA CLASSE TGVTURTLE

Voici l'interface de la copieuse classe **TGVTurtle** :

```
{ TGVTurtle - la tortue }
TGVTurtle = class(TImage)
private
  fx: Real; // abscisse de la tortue
  fy: Real; // ordonnée de la tortue
  fTurtleKind: TTurtleKind; // type de tortue
  fTurtleVisible: Boolean; // visibilité de la tortue
  fHeading: Real; // cap de la tortue
  fSize: Integer; // taille de la tortue
  fPenDown: Boolean; // crayon levé/baissé
  fScreen: TScreenTurtle; // écran de la tortue
  fScaleX: Integer; // échelle des X
  fScaleY: Integer; // échelle des Y
  fPenRubber: Boolean; // gomme du crayon
  fPenReverse: Boolean; // inversion du crayon
  fOnchange: TTurtleEvent; // changement notifié
  fSavedTurtle: TTurtle; // sauvegarde d'une tortue
  fScreenColor: TColor; // couleur de l'écran
  fFilled: Boolean; // drapeau de remplissage
  fTempColor: TColor; // sauvegarde provisoire de la couleur (PenRubber)
{$IFDEF Delphi}
  fTurtleImg: TPngImage; // image de la tortue
  fTempImg: TBitmap; // image temporaire
  fOldImg: TBitmap; // image conservée
{$ELSE}
  fTurtleImg: TCustomBitmap;
  fTempImg: TCustomBitmap; // image temporaire
  fOldImg: TCustomBitmap; // image conservée
{$ENDIF}
  fOnBeforeChange: TTurtleBeforeEvent; // notification avant cap changé
  fOnBackGroundChange: TTurtleBackGroundEvent; // changement de fond
  fPenColor: TColor; // couleur du crayon
  fSpeed: Integer; // vitesse de la tortue
  function GetCoordX: Integer; // abscisse de la tortue
  function GetCoordY: Integer; // ordonnée de la tortue
  procedure SetCoordX(const Value: Integer); // abscisse de la tortue
  procedure SetCoordY(const Value: Integer); // ordonnée de la tortue
  procedure SetTurtleKind(const Value: TTurtleKind); // type de tortue
  procedure SetTurtleVisible(const Value: Boolean); // visibilité de la tortue
  procedure SetHeading(const Value: Real); // cap
  procedure SetSize(const Value: Integer); // taille de la tortue
  procedure SetPenReverse(const Value: Boolean); // inversion de l'écriture
  procedure SetRubberPen(const Value: Boolean); // la tortue efface
  procedure SetScreenColor(const Value: TColor); // couleur d'écran
  procedure SetPenDown(const Value: Boolean); // crayon baissé ou levé
  procedure SetFilled(const Value: Boolean); // remplissage
```

```

procedure SetPenColor(const Value: TColor); // couleur du crayon
procedure SetSpeed(const Value: Integer); // vitesse de dessin de la tortue
protected
  // change l'ordonnée pour le nouveau repère
  function cY(Y: Integer): Integer;
  // effectue un déplacement
  procedure DoGo(const X, Y: Integer);
  // coordonnées dans limites ?
  function IsWithinLimits(const X, Y: Integer): Boolean;
  // montre/cache la tortue png
  procedure ToggleTurtlePNG;
  // montre/cache la tortue triangle
  procedure ToggleTurtleTriangle;
  // gestion du changement
  procedure Change; dynamic;
  // gestion de l'action avant le changement
  procedure BeforeChange; dynamic;
  // gestion du changement de fond
  procedure BackGroundChange; dynamic;
public
  // création
  constructor Create(AOwner: Tcomponent); override;
  // destruction
  destructor Destroy; override;
  // déplacement en écrivant
  procedure LineTo(X, Y: Integer);
  // déplacement sans écrire
  procedure MoveTo(X, Y: Integer);
  // réinitialisation de la tortue
  procedure Reinit;
  // tortue à l'origine
  procedure Home;
  // nettoyage de l'écran
  procedure Wipe;
  // la tortue se déplace
  procedure Move(const Value: Real);

  // la tortue tourne
  procedure Turn(const Value: Real);
  // fixe les coordonnées de la tortue
  procedure SetPos(const X, Y: Integer);
  // renvoie le cap vers un point
  function Towards(const X, Y: Integer): Real;
  // renvoie la distance de la tortue à un point donné
  function Distance(const X,Y: Integer): Real;
  // dessine un rectangle
  procedure Rectangle(const X1, Y1, X2, Y2: Integer); overload;
  // dessine un rectangle à l'emplacement de la tortue
  procedure Rectangle(const X2, Y2: Integer); overload;
  // dessine un carré
  procedure Square(const X1, Y1, L: Integer); overload;
  // dessine un carré à l'emplacement de la tortue
  procedure Square(const L: Integer); overload;
  // dessine un rectangle arrondi
  procedure RoundRect(const X1, Y1, X2, Y2: Integer); overload;
  // dessine un rectangle arrondi à l'emplacement de la tortue
  procedure RoundRect const (X2, Y2: Integer); overload;
  // dessine une ellipse
  procedure Ellipse(const X1, Y1, X2, Y2: Integer); overload;
  // dessine une ellipse à l'emplacement de la tortue
  procedure Ellipse(const X2, Y2: Integer); overload;
  // dessine un cercle
  procedure Circle(const X1, Y1, R: Integer); overload;
  // dessine un cercle à l'emplacement de la tortue
  procedure Circle(const R: Integer); overload;
  // dessine un arc de cercle
  procedure Arc(const X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer); overload;

```

```

// dessine un arc de cercle à l'emplacement de la tortue
procedure Arc(const X2, Y2, X3, Y3, X4, Y4: Integer); overload;
// dessine une corde
procedure Chord(const X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer); overload;
// dessine une corde à l'emplacement de la tortue
procedure Chord(const X2, Y2, X3, Y3, X4, Y4: Integer); overload;
// dessine une section d'ellipse
procedure Pie(const X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer); overload;
// dessine une section d'ellipse à l'emplacement de la tortue
procedure Pie(const X2, Y2, X3, Y3, X4, Y4: Integer); overload;
// dessine un polygone
procedure Polygon(Points: array of TPoint);
// dessine un polygone non couvrant
procedure PolyLine(Points: array of TPoint);
// sauvegarde la tortue
procedure SaveTurtle;
// récupère une tortue sauvée
procedure ReloadTurtle(const Clean: Boolean);
published
  // abscisse de la tortue
  property CoordX: Integer read GetCoordX write SetCoordX;
  // ordonnée de la tortue
  property CoordY: Integer read GetCoordY write SetCoordY;
  // type de tortue
  property Kind: TTurtleKind read fTurtleKind write SetTurtleKind
    default tkTriangle;
  // dessin alternatif de la tortue
{$IFDEF Delphi} // image en cours de la tortue PNG
  property TurtleImg: TPngImage read fTurtleImg write fTurtleImg;
{$ELSE}
  property TurtleImg: TCustomBitmap read fTurtleImg write fTurtleImg;
{$ENDIF}
  // visibilité de la tortue
  property TurtleVisible: Boolean read fTurtleVisible write SetTurtleVisible
    default True;
  // direction de la tortue
  property Heading: Real read fHeading write SetHeading;
  // taille de la tortue
  property Size: Integer read fSize write SetSize default TurtleDefaultSize;
  // drapeau d'écriture
  property PenDown: Boolean read fPenDown write SetPenDown default True;
  // type de zone de déplacement
  property Screen: TScreenTurtle read fScreen write fScreen default teWin;
  // échelle des X
  property ScaleX: Integer read fScaleX write fScaleX default DefaultScale;
  // échelle des Y
  property ScaleY: Integer read fScaleY write fScaleY default DefaultScale;
  // état de la gomme
  property PenRubber: Boolean read fPenRubber write SetRubberPen
    default False;
  // état de l'inversion d'écriture
  property PenReverse: Boolean read fPenReverse write SetPenReverse
    default False;
  // couleur du crayon
  property PenColor: TColor read fPenColor write SetPenColor default clRed;
  // état du remplissage
  property Filled: Boolean read fFilled write SetFilled default True;
  // vitesse de dessin de la tortue
  property Speed: Integer read fSpeed write SetSpeed default TurtleMaxSpeed div 2;
  // couleur du fond d'écran
  property ScreenColor: TColor read fScreenColor write SetScreenColor;
  // événement après le changement de la tortue
  property OnChange: TTurtleEvent read fOnchange write fOnchange;
  // événement avant le changement de la tortue
  property OnBeforeChange: TTurtleBeforeEvent read fOnBeforeChange
    write fOnBeforeChange;

```

```
// événement de changement du fond
property OnBackGroundChange: TTurtleBackGroundEvent read fOnBackGroundChange
  write fOnBackGroundChange;
end;
```

En dehors des aspects déjà étudiés, la classe qui descend de `TImage`²³ ne présente pas de problèmes majeurs. La tortue fait bien sûr appel à de nombreuses propriétés afin que l'utilisateur puisse en maîtriser l'aspect et le comportement. L'interface est aussi alourdie par de nombreuses fonctions de dessins : une première version dessine en se référant à des coordonnées traditionnelles tandis qu'une seconde version surchargée grâce à `overload` utilise les coordonnées de la tortue pour amorcer son dessin.



On notera que les coordonnées sont des réels alors que la position finale est un entier : cela permet de calculer au plus près la position de la tortue avant de l'arrondir, évitant ainsi des approximations cumulées.

TEST DE L'UNITE TGVTURTLES

Comme toujours, le programme de test est décliné en quatre versions :

- ✓ Lazarus pour Windows 32 ;
- ✓ Lazarus pour Linux ;
- ✓ Delphi pour Windows 32 ;
- ✓ Delphi pour Windows 64.

L'utilisateur pourra tester la plupart des fonctions et propriétés définies pour manipuler la tortue. Le programmeur sera plus intéressé par le mécanisme des notifications d'événements tel que décrit plus haut.

Pour avoir une idée des possibilités de dessin de la tortue, un bouton marqué « dessin » dessine une série de 36 carrés en rosace. Pour se rendre compte de la rapidité de la tortue triangulaire par rapport à celle graphique, il est judicieux d'utiliser ce bouton spécial dans trois configurations : graphique, triangle et tortue cachée.

On pourra aussi se servir des versions Delphi et Lazarus pour comparer leurs performances et noter les différences !

Dans le dossier Lazarus, on trouvera un dossier nommé « transparence » : il contient un programme rudimentaire permettant de traiter les problèmes simples de transparence.

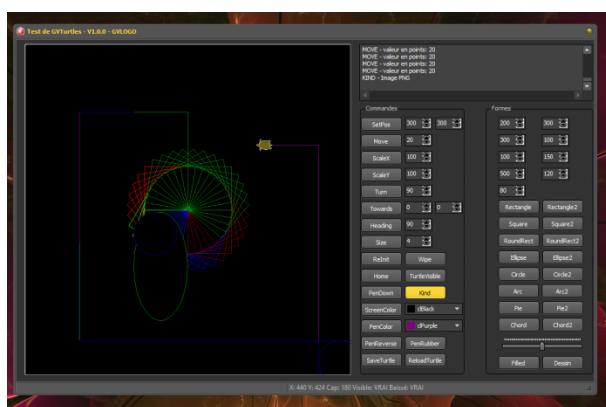


Figure 11 - Test de la tortue avec Delphi

²³ J'étudie d'autres possibilités de descendance afin de régler ces problèmes de transparence avec Lazarus.

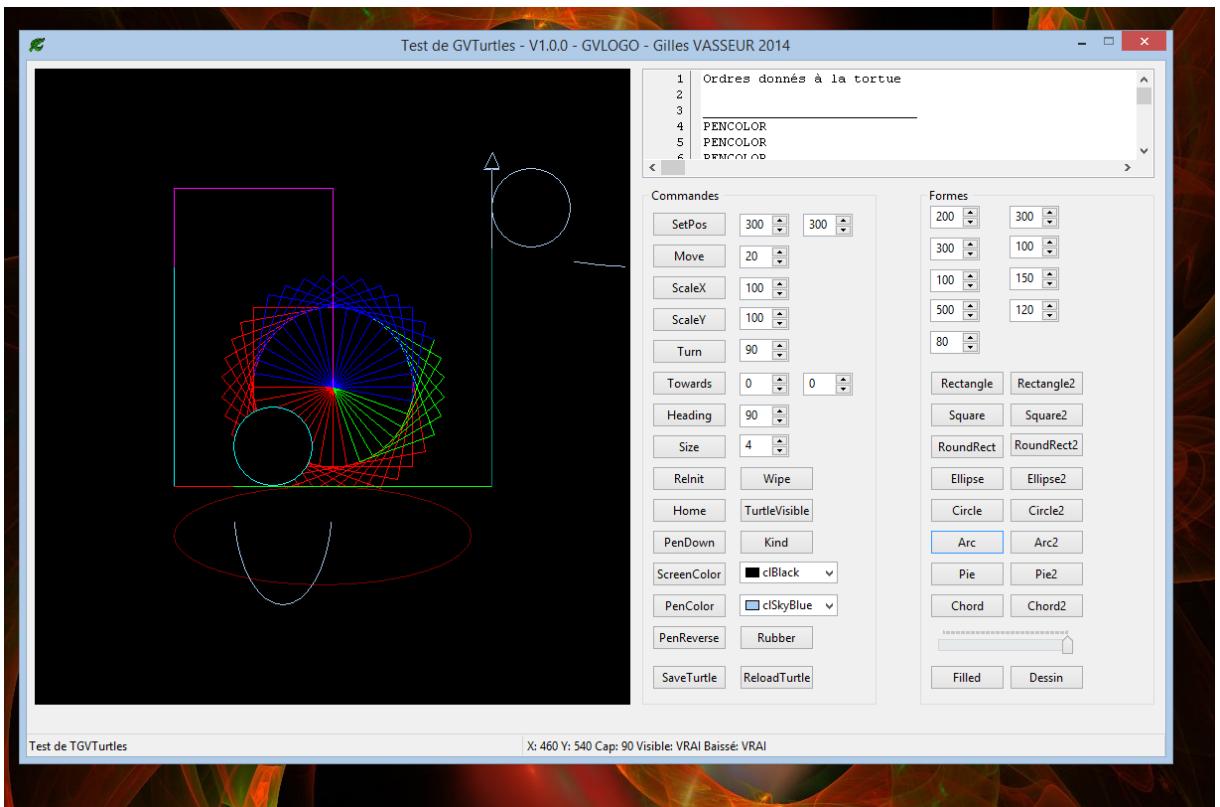


Figure 12 - Test de la tortue avec Lazarus

RECREATION : EASYTURTLE (LOGICIEL DE DESSIN)

LE PROJET EASYTURTLE

Avant d'aborder les outils de programmation et de rentrer dans le cœur de l'interpréteur, une récréation s'impose avec la création d'un petit logiciel de dessin baptisé **EasyTurtle**. Il s'agit de mettre en œuvre l'unité **GVTurtles** en permettant à un enfant de dessiner avec la tortue, de rejouer l'ensemble des ordres qu'il lui aura donnés, ou encore de charger et de sauvegarder ses réalisations.

Pour le programmeur, il s'agira d'utiliser quelques outils particulièrement utiles, en particulier les listes d'actions.



Le projet n'est proposé que dans sa version Lazarus (Linux et Windows), mais pourrait tout aussi bien être écrit avec Delphi.

MODE D'EMPLOI RAPIDE

L'ECRAN D'ACCUEIL

L'écran d'accueil se présente comme ceci :

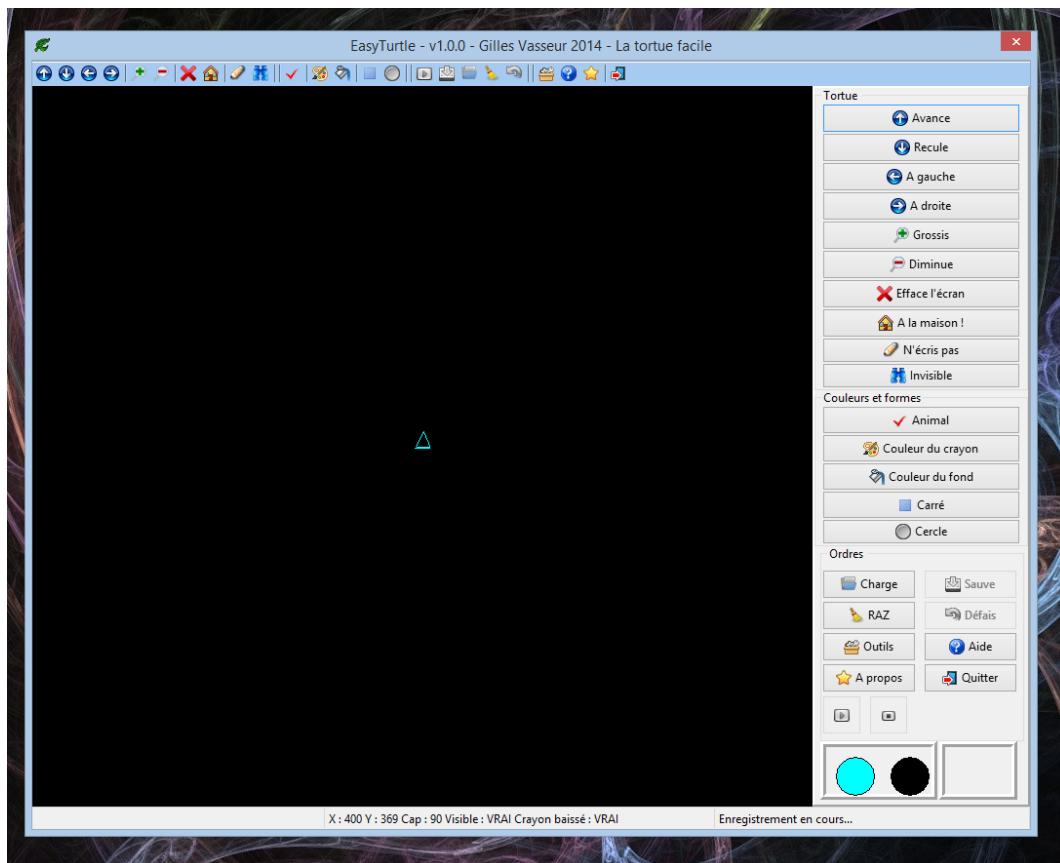


Figure 13 - Ecran principal de EasyTurtle (Windows)

Sur l'essentiel de la fenêtre se situe la zone d'affichage de la tortue : c'est ici que seront réalisés les dessins. Afin de faciliter sa manipulation, la zone de dessin est en mode « clos », ce qui signifie que la tortue ne peut pas être perdue de vue, car tout ordre tendant à la faire disparaître conduira à la faire buter contre le bord de son champ comme s'il y avait une barrière.

À droite de cette zone d'affichage, on aperçoit toute une série de boutons : ce sont les actions mises à disposition de l'utilisateur.

LA TORTUE

On distingue ainsi les actions concernant la tortue :



Figure 14 - Actions de la tortue (EasyTurtle)

La tortue peut donc avancer, reculer, tourner à gauche et à droite. Chacune de ces opérations se fait selon une valeur par défaut modifiable grâce à une fenêtre de réglage des préférences.

Si elle est de forme triangulaire, la tortue peut aussi grossir et rapetisser. Dans le cas contraire, ces boutons sont grisés, et par conséquent inactifs.

L'utilisateur peut effacer l'écran, renvoyer la tortue à son origine, l'autoriser ou lui interdire d'écrire, la rendre visible ou invisible.



Un bouton indique toujours l'action à réaliser, et non l'état de la tortue. Ainsi, si le bouton comporte le message « N'écris plus », c'est que la tortue laisse actuellement une trace et qu'une pression sur le bouton lui demandera de ne plus écrire.

COULEURS ET FORMES

Le panneau suivant s'occupe des couleurs et des formes :

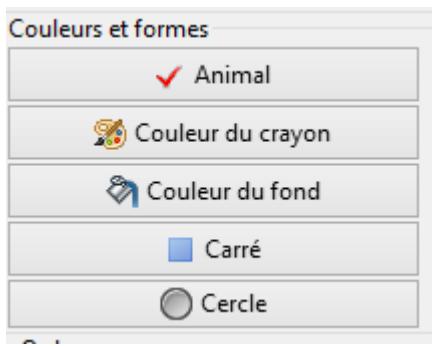


Figure 15 - Couleurs et formes (EasyTurtle)

Le premier bouton permute l'apparence de la tortue, entre le triangle et le dessin au format png. Le second ouvre une fenêtre de choix de la couleur du crayon, tout comme le suivant pour la couleur de fond. Les deux derniers dessinent respectivement un carré et un cercle à l'emplacement de la tortue.



Avec Lazarus, lorsque la couleur de fond de la surface de dessin de la tortue change, l'écran est entièrement réinitialisé.

ORDRES GENERAUX

Le panneau suivant regroupe les ordres généraux concernant le travail effectué par la tortue :

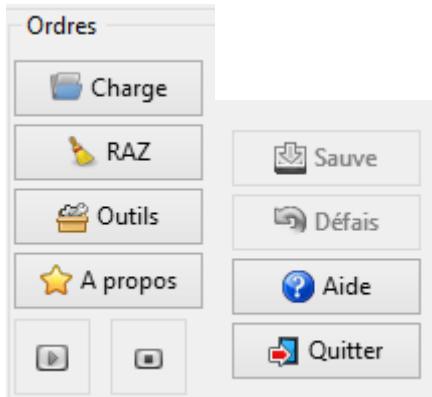


Figure 16 - Ordres (EasyTurtle)

EasyTurtle propose un enregistrement des ordres donnés à la tortue. C'est à partir de cet enregistrement qu'opéreront les boutons, « Charge », « Sauve », « RAZ » et « Défais ».

L'utilisateur peut sauvegarder et charger son travail. Il peut ouvrir des fenêtres spécialisées : Outils, À Propos et Aide. Il peut aussi rejouer une séquence enregistrée et interrompre cette répétition : ce sont les deux boutons sans légende qui apparaissent en bas à gauche de la copie d'écran. Il peut encore annuler le dernier ordre donné à la tortue (bouton « Défais ») ou remettre à zéro toute la séquence (bouton « RAZ »). Enfin, il peut quitter le logiciel.



Le fait de charger une suite d'ordres l'exécute immédiatement après le chargement.



Suivant l'état du logiciel, certains boutons seront désactivés : par exemple, les boutons « Défais », « Sauve » et « Rejoue » ne seront activés que si des ordres ont été enregistrés. Le bouton « Stop » ne sera activé que si le bouton « Rejoue » a été pressé et seulement le temps de la répétition. Lorsque le bouton « Rejoue » a été pressé, tous les boutons, sauf « Quitter » et « Stop », sont désactivés.



Pour réinitialiser la séquence d'ordres, plusieurs possibilités sont offertes : chargement d'un nouveau fichier d'ordres, pression sur le bouton « RAZ » et modification du fond de l'écran. Contrairement au bouton « Efface l'écran », le bouton « RAZ » conserve la couleur d'écriture et celle du fond de l'écran.

L'AIDE

Une aide peut être demandée :

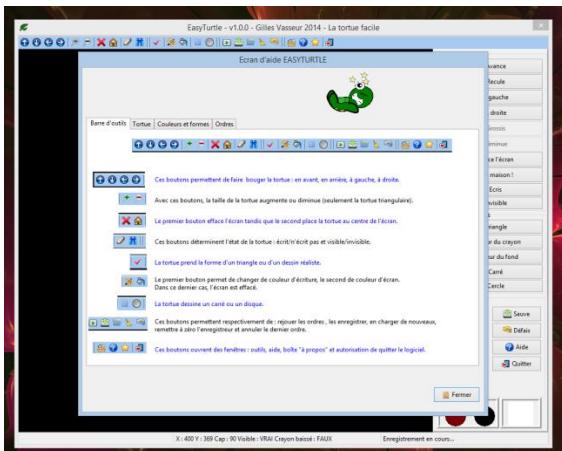


Figure 17 - Aide (EasyTurtle)

BOITE « À PROPOS »

De même, on peut accéder à une boîte « À propos » :

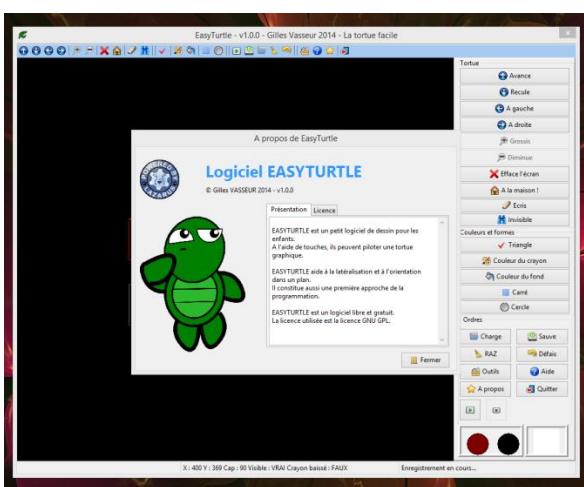


Figure 18 - A propos (EasyTurtle)

BOITE DES PREFERENCES

Enfin, une boîte d'outils permet de modifier les valeurs par défaut de certains ordres concernant le dessin effectué par la tortue :

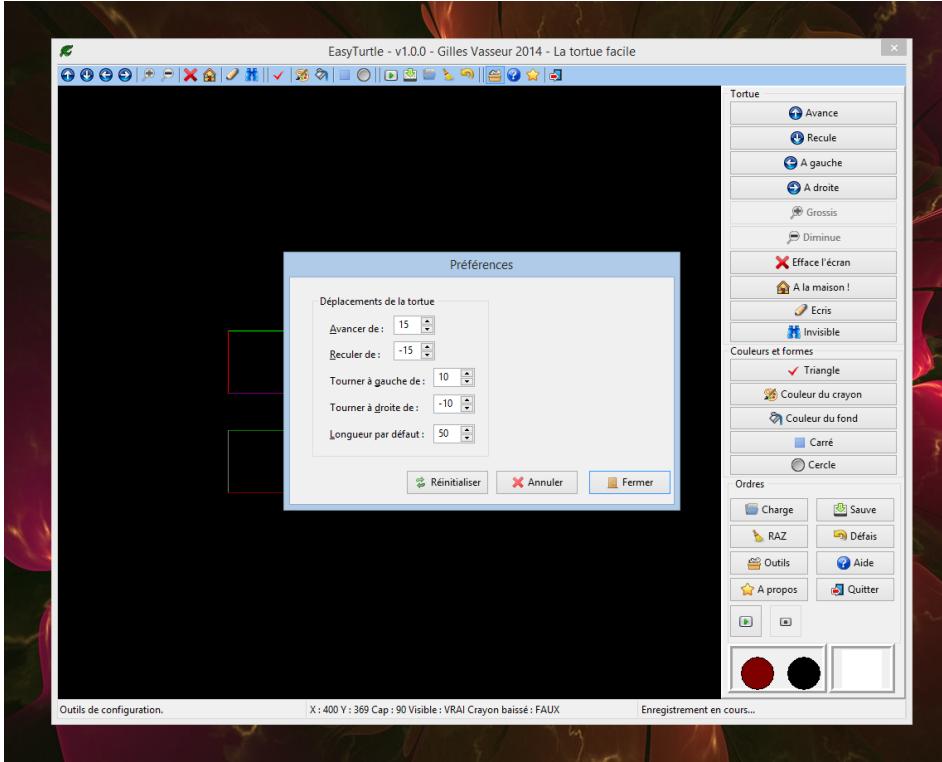


Figure 19 - Outils (EasyTurtle)



Les modifications apportées grâce à la boîte des préférences sont enregistrées avec le fichier des ordres. Les valeurs sont données en pixels ou en degrés.

AUTRES ELEMENTS

Tous les boutons décrits ci-dessus ont leur double dans la barre d'outils située en haut de la fenêtre principale :



Les pictogrammes sont évidemment les mêmes afin de renforcer la cohérence du logiciel. De plus, la plupart des ordres peuvent être donnés par une combinaison de touches indiquée en aide ponctuelle près du bouton avant de le presser et dans la barre de statut.

Pour terminer cette présentation rapide, il faut noter des indicateurs fournis par la barre de statut et par deux disques situés en bas à droite de la fenêtre principale :

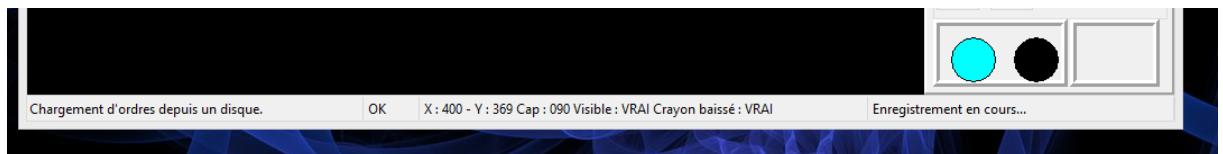


Figure 20 - Barre de statut (EasyTurtle)

On prend ainsi connaissance d'une aide succincte concernant le bouton survolé par la souris, des principales valeurs associées à la tortue et des valeurs attribuées à la couleur du crayon de la tortue (premier disque) et au fond de l'écran (second disque), ainsi que l'état du logiciel : « enregistrement en cours », « sauvegarde en cours », « chargement en cours » et « répétition des ordres ».

LA PROGRAMMATION

La suite de cette partie décrit le fonctionnement d'**EasyTurtle**. Contrairement aux autres logiciels d'exemples, celui-ci fonctionne à partir de plusieurs fiches :

	Nom	Modifié le	Type	Taille
éments récem	Enregistrements	02/08/2014 16:08	Dossier de fichiers	
ts	Linux	02/08/2014 12:44	Dossier de fichiers	
ts	Win32	02/08/2014 16:24	Dossier de fichiers	
ts	EasyTurtle.ico	02/08/2014 12:44	Icône	134 Ko
ts	EasyTurtle.lpi	02/08/2014 16:19	Lazarus Project Inf...	4 Ko
ts	EasyTurtle.lpr	02/08/2014 12:44	Lazarus Project M...	2 Ko
ts	EasyTurtle.lps	02/08/2014 16:19	Fichier LPS	8 Ko
ts	EasyTurtle.res	02/08/2014 16:19	Fichier RES	7 Ko
ts	gvabout.lfm	02/08/2014 12:44	Lazarus Form	8 724 Ko
ts	gvabout.pas	02/08/2014 12:44	Pascal Source Code	3 Ko
ts	GVConsts.pas	02/08/2014 12:44	Pascal Source Code	5 Ko
ts	GVLOGO_Icon.ico	02/08/2014 12:44	Icône	5 Ko
ts	gvtools.lfm	02/08/2014 12:49	Lazarus Form	4 Ko
ts	gvtools.pas	02/08/2014 12:49	Pascal Source Code	4 Ko
ts	GVTurtles.pas	02/08/2014 12:44	Pascal Source Code	32 Ko
ts	help.lfm	02/08/2014 12:44	Lazarus Form	364 Ko
ts	help.pas	02/08/2014 12:44	Pascal Source Code	7 Ko
ts	main.lfm	02/08/2014 16:18	Lazarus Form	1 494 Ko
ts	main.pas	02/08/2014 16:18	Pascal Source Code	30 Ko

Figure 21 - Répertoire de EasyTurtle.

La fiche principale s'appelle « Main ». « GVAbout » contient la boîte « À propos », « Help » l'aide et « GVTools » la boîte des préférences.

L'appel d'une fiche externe se fait selon un modèle courant depuis la fiche « Main ». Voici, par exemple, l'appel de la boîte « À propos » :

```

24 |   |
25 |procedure TMainForm.ActionAboutExecute(Sender: TObject);
26 |  // *** boîte à propos ***
27 |begin
28 |  GVAabout.AboutForm := TAboutForm.Create(Self); // on crée la fiche
29 |  try
30 |    GVAabout.AboutForm.ShowModal; // on l'affiche
31 |  finally
32 |    GVAabout.AboutForm.Free; // on la libère
33 |  end;
34 |end;
35 |

```

Figure 22 - Appel d'une autre fiche (EasyTurtle).

LA FICHE PRINCIPALE

La fiche « Main » contient toutes les méthodes nécessaires au fonctionnement d'**EasyTurtle**. Les éléments les plus complexes sont ceux relatifs à la mémorisation des ordres donnés à la tortue : le logiciel utilise à cette fin un tableau ouvert géré par une méthode nommée **Memorize** :

```

1 |  |
2 |procedure TMainForm.Memorize(const Value: Integer);
3 |  // *** mémorisation d'une action ***
4 |begin
5 |  SetLength(MemoInt, Length(MemoInt) + 1); // on augmente la taille du tableau
6 |  MemoInt[Length(MemoInt) - 1] := Value; // on enregistre
7 |  fSaved := False; // séquence non enregistrée
8 |end;
9 |

```

Figure 23 - Mémorisation des ordres (EasyTurtle).

Cette méthode ajuste la taille du tableau avant d'enregistrer la nouvelle donnée. Elle indique aussi que la séquence a été modifiée en vue d'un futur enregistrement.

La plupart des ordres sont gérés d'une manière identique. Voici par exemple l'ordre **ActionForwardExecute** qui fait avancer la tortue :

```

285 |  |
286 |procedure TMainForm.ActionForwardExecute(Sender: TObject);
287 |  // *** la tortue avance ***
288 |begin
289 |  GVTurtle.Move(pForward); // la tortue bouge
290 |  Memorize(CT_Forward); // mémorisation
291 |end;
292 |

```

Figure 24 - La tortue avance (EasyTurtle).

On exécute l'ordre et on l'enregistre, rien de plus facile ! Simplement, afin de permettre aux boutons, à la barre d'outils et aux combinaisons de touches de fonctionner de la même manière sans dupliquer le code, on utilise un composant **TActionList** qui centralise les actions.

En plus de l'exécution, on a prévu une mise à jour (disponibilité, visibilité, affichage) de chaque fonction suivant l'état du logiciel :

```

procedure TMainForm.ActionForwardUpdate(Sender: TObject);
// actions actives/inactives
begin
  // seulement si enregistrement
  (Sender as TAction).Enabled := (pState = stRecording);
end;

```

Figure 25 - Mise à jour (EasyTurtle).

Ici, l'objet appelant (qui doit être une action) n'est activé que si le mode est celui de l'enregistrement. En effet, il ne faut pas continuer à dessiner au cours de la sauvegarde, du chargement ou si l'on est en train de rejouer toute la séquence.



On aurait pu indiquer directement **ActionForward** dans la méthode, mais le transtypage (**Sender as TAction**) permet de partager le même gestionnaire avec d'autres actions au comportement identique (**ActionBackward**, par exemple).

Rejouer la séquence exige de dispatcher les ordres en fonction de leur enregistrement :

```

procedure TMainForm.Replay;
// *** rejoue les actions ***
begin
  GVTurtle.ReInit; // réinitialisation de la tortue
  GVTurtle.Screen := teGate; // écran clos
  fCmd := C_MinCmds; // on pointe sur le premier élément du tableau hors entête
  pForward := MemoInt[1]; // on récupère les données de l'entête
  pBackward := MemoInt[2];
  pLeft := MemoInt[3];
  pRight := MemoInt[4];
  pLength := MemoInt[5];
  GVTurtle.ScreenColor := MemoInt[6];
  // on boucle tant qu'il y a des ordres et qu'un arrêt n'a pas été demandé
  while (fCmd < Length(MemoInt)) and (pState = stPlaying) do // on balaie le tableau
    begin
      //images adaptées pour les roues
      tbReplay.ImageIndex := (fCmd mod 31) + 15;
      ImageListBigWait.GetBitmap(fCmd mod 31, ImgRound.Picture.Bitmap);
      // on permet aux messages d'être traités
      Application.ProcessMessages;
      // on répartit le travail suivant les ordres enregistrés
      case MemoInt[fCmd] of
        CT_Forward : GVTurtle.Move(pForward); // avance
        CT_Backward : GVTurtle.Move(pBackward); // recule
        CT_Left : GVTurtle.Turn(pLeft); // à gauche
        CT_Right : GVTurtle.Turn(pRight); // à droite
        CT_Bigger : GVTurtle.Size := GVTurtle.Size + 2; // taille + 2
        CT_Minus : GVTurtle.Size := GVTurtle.Size - 2; // taille - 2
        CT_Home : begin
          GVTurtle.Home; // maison
          Inc(fCmd, 3); // ordre suivant
        end;
        CT_UpDown : GVTurtle.PenDown := not GVTurtle.PenDown; // crayon baissé
        // visibilité
        CT_SeeSaw : GVTurtle.TurtleVisible := not GVTurtle.TurtleVisible;
        CT_Kind : if GVTurtle.Kind <> tkTriangle then // type
          GVTurtle.Kind := tkTriangle
        else
      end;
    end;
  end;

```

Figure 26 - Rejouer une séquence (EasyTurtle).

Le champ privé `fCmd` est le pointeur utilisé sur l'ordre en cours. Avant de rejouer les ordres, on s'occupe de l'en-tête qui contient les valeurs modifiables depuis la fenêtre des paramètres. Ces valeurs sont enregistrées avec l'éventuel fichier de sauvegarde.



On remarquera la présence de `Application.ProcessMessages` qui permet à l'application de réagir aux événements tels que l'appui sur le bouton « Stop ». On profite aussi de cette boucle pour animer certains boutons : ainsi, deux roues seront animées. Certaines données ne sont utiles qu'en cas d'action de correction, en remontant dans le temps : pour rejouer la séquence, on les ignore simplement (voir le traitement de `CT_Home`, par exemple).

La partie la plus complexe de cette unité est celle relative à la fonction « Défaire » :

```

575 procedure TMainForm.ActionUndoExecute(Sender: TObject);
576   . . . // *** défaire la dernière action ***
577 begin
578   if (Length(MemoInt) > (C_MinCmds + 4)) and // ordre le plus long = HOME
      (MemoInt[Length(MemoInt) - 4] = CT_Home) then
579     begin
580       GVTurtle.PenRubber := GVTurtle.PenDown; // on efface
       GVTurtle.SetPos(Round(MemoInt[Length(MemoInt) - 2]),
                      Round(MemoInt[Length(MemoInt) - 1])); // on se repositionne
       GVTurtle.Heading := MemoInt[Length(MemoInt) - 3]; // direction de la tortue
       GVTurtle.PenRubber := False; // on écrit normalement
       SetLength(MemoInt, Length(MemoInt) - 4); // on réajuste la mémorisation
     end
     else
     if (Length(MemoInt) > (C_MinCmds + 3)) and (MemoInt[Length(MemoInt) - 3]
       = CT_Pen) then // couleur de crayon
     begin
       GVTurtle.PenColor := MemoInt[Length(MemoInt) - 2]; // couleur récupérée
       SetLength(MemoInt, Length(MemoInt) - 3); // on ajuste
     end
     else
     begin
       case MemoInt[Length(MemoInt) - 1] of
         CT_Forward: begin // on a avancé
           GVTurtle.PenRubber := GVTurtle.PenDown; // on efface
           GVTurtle.Move(-pForward); // donc on recule
           GVTurtle.PenRubber := False; // on écrit normalement
         end;
         CT_Backward: begin // on a reculé
           GVTurtle.PenRubber := GVTurtle.PenDown; // on efface
           GVTurtle.Move(-pBackward); // donc on avance
           GVTurtle.PenRubber := False; // on écrit normalement
         end;
         CT_Left: GVTurtle.Turn(-pLeft); // gauche devient droite
         CT_Right: GVTurtle.Turn(-pRight); // droite devient gauche
         CT_SeeSaw: GVTurtle.TurtleVisible := not GVTurtle.TurtleVisible; // visibilité
         CTUpDown: GVTurtle.PenDown := not GVTurtle.PenDown; // écriture ou non
         CT_Kind: if GVTurtle.Kind = tkTriangle then // type de tortue
                   GVTurtle.Kind := tkPNG
     end;
   end;
end;

```

Figure 27 - Annuler la dernière action (EasyTurtle).

En effet, pour être annulées, certaines fonctions exigent que l'état de la tortue soit enregistré au préalable : ainsi, pour annuler un retour à l'origine, il faut se souvenir de l'ancien emplacement de la tortue, mais aussi de son orientation. En amont, il faut donc que chaque ordre mémorise ce qu'attendra une éventuelle correction.



La solution adoptée ici est de repasser sur le dernier trait en l'effaçant. Cette méthode est très rapide, mais elle peut effacer un trait qui devrait être conservé parce qu'il est recouvrant.

La méthode `ActionSaveExecute` d'enregistrement de la séquence doit tenir compte des remarques précédentes :

```

430
.   procedure TMainForm.ActionSaveExecute(Sender: TObject);
.   // *** sauvegarde des ordres de la tortue ***
.   var
.     F: TextFile;
.     OK: Boolean;
.     I: Integer;
.   begin
438     OK := False; // abandon par défaut
.     repeat
440       Ok := SaveDialog.Execute; // dialogue de sauvegarde
.       if Ok then
.         begin
.           // confirmation si le fichier existe
.           if FileExists(SaveDialog.FileName) then // boîte de dialogue si existe
445             // demande de remplacement si le fichier existe
.             case MessageDlg(Format(ME_Replace, [ExtractFileName(SaveDialog.FileName)]),
.                           mtConfirmation, mbYesNoCancel, 0) of
.               mrYes: Ok := True; // on écrase l'ancien fichier
.               mrNo: Ok := False; // on recommence
.               mrCancel: Exit; // abandon si le fichier existe
.             end;
.           else
.             Exit; // abandon dès la boîte de dialogue
455         until Ok;
.       try
.         AssignFile(F, SaveDialog.FileName); // fichier assigné
.         pState := stSaving; // mode sauvegarde indiqué
.         try
460           Rewrite(F); // on remet à zéro le fichier
.           for I := 0 to Length(MemoInt) - 1 do // on balaie les ordres enregistrés
.             begin
.               // images qui tournent pendant le chargement
.               tbReplay.ImageIndex := (I mod 31) + 15;
.               ImageListBigWait.GetBitmap(I mod 31, ImgRound.Picture.Bitmap);
.               Application.ProcessMessages; // on traite les messages
.               writeln(F, MemoInt[I]); // on écrit dans le fichier
.               fSaved := True; // sauvegarde effectuée
.             end;
470         except
.           // erreur de sauvegarde
.           MessageDlg(Format(ME_SaveError, [ExtractFileName(SaveDialog.FileName)]),
.                      mtError, [mbOk], 0);
.         end;
475       finally
.         CloseFile(F); // fermeture du fichier
.         pState := stRecording;
.         Refresh; // remet à jour les voyants
.       end;
.     end;
480   end;

```

Figure 28 - Sauvegarde (EasyTurtle).

Cette méthode vérifie l'existence du fichier avant d'éventuellement l'écraser puis enregistre les données dans l'ordre du tableau des commandes. Elle anime aussi deux roues en permettant à l'application de gérer les événements grâce à la méthode `Application.ProcessMessages`.

Le chargement d'un fichier est un brin plus complexe :

```

340  procedure TMainForm.ActionLoadExecute(Sender: TObject);
.    // ouverture d'un fichier de commandes
.    var
.        F: TextFile;
.        Num, I: Integer;
345  begin
.    if OpenDialog.Execute then // boîte de dialogue d'ouverture de fichier
begin
try
.    AssignFile(F,OpenDialog.FileName); // fichier assigné
pState := stLoading; // statut signifié
try
.        Reset(F); // fichier réinitialisé
readln(F, Num);
if Num <> CT_Version then
begin
.            MessageDlg(Format(ME_VersionError,[ExtractFileName(SaveDialog.FileName)]),
mtError, [mbOk], 0); // signale une erreur de version
Exit; // sortie
end;
readln(F, Num); // récupère les données de l'entête
pForward := Num;
readln(F, Num);
pBackward := Num;
readln(F, Num);
pLeft := Num;
readln(F, Num);
pRight := Num;
readln(F, Num);
pLength := Num;
readln(F, Num);
GVTurtle.ScreenColor := Num;
ActionRAZExecute(Sender); // initialisation
I := -1; // pointeur de travail pour l'affichage
repeat
.        Inc(I); // élément suivant
// images des roues mises à jour
tbReplay.ImageIndex := (I mod 31) + 15;
ImageListBigWait.GetBitmap(I mod 31 ,ImgRound.Picture.Bitmap);
Application.ProcessMessages; // on permet les messages
readln(F, Num); // on lit une donnée
Memorize(Num); // qu'on enregistre
until EOF(F) : // jusqu'à la fin du fichier
except
.        MessageDlg(Format(ME_LoadError,[ExtractFileName(SaveDialog.FileName)]),
mtError, [mbOk], 0); // signale une erreur de lecture
ActionEraseExecute(Sender); // on remet à zéro
end;
finally
CloseFile(F); // ferme le fichier
pState := stRecording; // on enregistre de nouveau
fSaved := True; // enregistrement déjà fait
Refresh; // remet à jour les voyants
end;
ActionReplyExecute(Sender); // rejoue immédiatement la séquence
end;
end;

```

Figure 29 - Chargement d'un fichier (EasyTurtle).

Le premier élément du fichier doit contenir le numéro de version actuelle, soit 100. Il est suivi de l'en-tête puis des données proprement dites. Au fur et à mesure de la lecture, on fait tourner les roues habituelles et on enregistre les ordres grâce à la méthode **Memorize**. À la fin de la méthode, on lance l'exécution de la séquence afin d'avoir un écran à jour : sans ce dernier point, la fonction « Défaire » serait erronée puisqu'elle corrigerait un écran non dessiné !

On peut enfin mentionner la méthode **CloseQuery** :

```

655  .  procedure TMainForm.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
.  .  // *** demande de fermeture ***
.  begin
659  |  // fermeture à confirmer
660  |  if not fSaved then // si séquence non enregistrée
.  begin // demande d'enregistrement
.  .  case MessageDlg(ME_NotSaved, mtConfirmation, mbYesNoCancel, 0) of
.  .  .  mrYes: begin // oui
.  .  .  .  ActionSaveExecute(Sender); // sauvegarde
.  .  .  .  CanClose := fSaved; // on sort si c'est fait
.  .  .  end;
.  .  .  mrNo: CanClose := True; // on sort
.  .  .  mrCancel: CanClose := False; // on ne sort pas
.  .  .  end;
.  .  end
.  .  else // cas où il n'y a rien à enregistrer
.  .  .  CanClose := (MessageDlg(ME_Close, mtConfirmation, mbYesNo, 0) = mrYes);
.  .  .  // on arrête le dessin si nécessaire
.  .  |  if CanClose then
.  .  .  pState := stRecording;
.  .  end;
.  end;

```

Figure 30 - Demande de fermeture (EasyTurtle).

Lors d'une demande de fermeture du logiciel, on vérifie préalablement si la séquence en cours, lorsqu'elle a été modifiée, doit être enregistrée. Les messages diffèrent suivant les cas.



Le test de **CanClose** qui modifie si nécessaire **pState** s'explique par le fait que l'utilisateur peut avoir demandé la fermeture du logiciel alors qu'on rejoue une séquence. On interrompt alors cette répétition et l'on ferme le logiciel.

LES AUTRES FICHES

La boîte des préférences comprise dans l'unité **GVTools** est d'une simplicité enfantine : elle modifie ou non les données qu'elle doit gérer suivant le bouton pressé.

L'unité **Help** permet d'introduire quelques animations très simples : une image suit le déplacement de la souris en modifiant son champ **Left**, les composants **TLabel** ont leur texte en gras ou non suivant l'entrée/la sortie de la souris de leur surface...

L'unité **GVAbout** comprend aussi une animation aléatoire de la tortue.

LES OUTILS DE PROGRAMMATION

LES PILES

DEFINITION

Une pile informatique peut être figurée par une pile d'assiettes posées sur une table : lorsque j'empile une donnée (*push* en anglais), je pose une assiette et lorsque je dépile une donnée (*pull* en anglais), je retire une assiette de la pile. La dernière donnée est donc la première à pouvoir être extraite : on parle de pile LIFO (*Last In First Out* = dernier entré, premier sorti). Une pile permet en fait de mémoriser les données manipulées dans l'ordre où elles seront à utiliser.

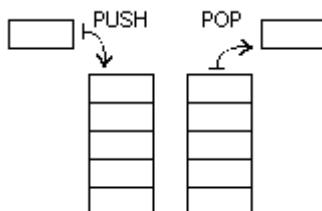


Figure 31 - Notion de pile LIFO

Les piles sont très utilisées par **GVLOGO**. Elles permettent tout à la fois de faire fonctionner l'interpréteur en stockant des données à récupérer plus tard dans un certain ordre, et d'effectuer des calculs complexes, en particulier pour l'évaluation d'expressions arithmétiques telles que $(2 * 4,5) / 8$.

Comme la lecture d'une ligne de programme s'effectue par convention de la gauche vers la droite, si l'on rencontre une opération rien n'indique a priori que les arguments nécessaires sont fournis : mentalement, nous découvrons les éléments au fur et à mesure de notre lecture. Un programme fait de même !

EXEMPLES

Les exemples qui suivent seront très détaillés. En effet, le fonctionnement d'une pile est au cœur de celui de l'interpréteur lui-même.

UN EXEMPLE SIMPLE

On souhaite effectuer l'opération **SOMME 24 35** qui additionne les deux nombres fournis en paramètres. Pour cet exemple, on utilisera deux piles, la première contenant les données, la seconde les opérations à effectuer et le nombre de paramètres attendus.

Tout d'abord, on rencontre une opération à effectuer : ici, une addition. On sait qu'elle nécessite deux données : on empile cette opération et le nombre de paramètres attendus.

<pile vide>	2
	SOMME

Ensuite, on rencontre une donnée. Afin de la mémoriser, on empile **24**. **SOMME** n'attend plus qu'un paramètre : on décrémente le sommet de la seconde pile :

24	1
SOMME	

On rencontre encore une donnée. On empile **35** et on décrémente le sommet de l'autre pile :

35	0
SOMME	

Le **0** au sommet de la seconde pile indique que les données nécessaires sont acquises. On retire le sommet de la seconde pile pour en extraire l'opération qui peut à présent être effectuée.

35	SOMME
24	

On effectue alors l'addition sur la pile :

59	<pile vide>
----	-------------

On récupère le résultat (**59**), par exemple pour l'afficher, et la première pile est vide.

Cette technique permet par conséquent de mémoriser des résultats intermédiaires : on peut imaginer que **35** est le résultat d'une autre opération et sa présence sur la pile fournira toujours le second élément nécessaire à l'addition initiale.

UN EXEMPLE PLUS COMPLEXE

On souhaite à présent effectuer l'opération **SOMME 24 PRODUIT 5 7**, sachant que **PRODUIT** multiplie deux données sur la pile.

Tout d'abord, on rencontre l'opération à effectuer : ici, une addition. On sait qu'elle nécessite deux données :

<pile vide>	2
SOMME	

Ensuite, on rencontre une donnée. Afin de la mémoriser, on empile **24**.

SOMME n'attend plus qu'un paramètre.

On décrémente le sommet de la seconde pile :

24	1
SOMME	

On rencontre une seconde opération qui est **PRODUIT**, elle-même nécessitant deux données. Il faut empiler ces informations :

24	2
	PRODUIT
	1
	SOMME

La donnée suivante est **5** qu'on empile en décrémentant par ailleurs le sommet de l'autre pile :

5	1
24	PRODUIT
	1
	SOMME

On a donc une opération **SOMME** pendante qui attend encore une donnée et une opération **PRODUIT** qui elle aussi attend une donnée.

La donnée suivante est **7** qu'on empile :

7	0
	PRODUIT
24	1
	SOMME

Le **0** au sommet de la seconde pile indique que l'opération pendante qui la suit a les paramètres qu'elle attend à sa disposition. On dépile ce **0** devenu inutile.

7	PRODUIT
5	1
24	SOMME

On récupère alors **PRODUIT**. On peut donc effectuer cette opération et déposer son résultat sur le sommet de la première pile en décrémentant le sommet de la seconde pile.

35	0
24	SOMME

On en a terminé avec la multiplication. La pile contient les deux données nécessaires à **SOMME** : on peut donc l'effectuer comme dans l'exemple 1.

Le **0** au sommet de la seconde pile indique que les données nécessaires sont acquises. On retire le sommet de la seconde pile pour en extraire l'opération qui peut à présent être effectuée.

35	SOMME
24	

On effectue alors l'addition sur la pile :

59	<pile vide>
----	-------------

On récupère le résultat (59), par exemple pour l'afficher, et la première pile est vide.



Il est bien entendu que cet enchaînement d'opérations exige des éléments non encore réalisés :

- ✓ le flux du texte à exécuter doit être acquis et analysé pour en définir les éléments de base ;
- ✓ il faut pouvoir distinguer une opération d'une donnée ;
- ✓ le nombre de paramètres d'une opération doit être retrouvé pour être empilé ;
- ✓ une pile fournit au minimum les moyens d'empiler et de dépiler une donnée. D'autres opérations élémentaires sont souvent utiles : duplication de la dernière donnée, test de la profondeur de la pile, incrémentation et décrémentation du sommet... Les piles d'entiers et de réels fournissent d'autre part l'essentiel des opérations qu'on peut effectuer sur les données numériques qu'elles gèrent : addition, soustraction...

OPERATIONS SUR LES PILES (ET LES QUEUES)

GVLOGO n'a pas pour vocation de traiter des opérations de bas niveau comme les piles, mais il en fournit cependant une structure minimale. De plus, il implémente une structure de queue : les éléments sont alors stockés les uns après les autres, et c'est le plus ancien qui est déstocké le premier.

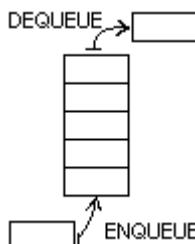


Figure 32 - Notion de queue



Les primitives ci-après sont décrites dans ce chapitre, mais relèvent du traitement des listes. Elles simulent le fonctionnement d'une pile ou d'une queue à partir d'une liste.

- **EMPILE** : attend le nom d'une variable puis un mot ou une liste – ne renvoie rien – la primitive place le second paramètre au début de la variable qui doit être une liste.

Exemples :

EMPILE "MAPILE 124 → -
EMPILE "MAPILE 568 → -

- **SOMMET** : attend le nom d'une variable en entrée – renvoie une liste ou un mot – la primitive renvoie le dernier élément stocké dans la pile indiquée par le paramètre qui doit renvoyé à une liste sans le retirer de cette dernière.

Exemple :

ECRIS SOMMET "MAPILE → 568

- **DEPILE** : attend le nom d'une variable en entrée – renvoie une liste ou un mot – la primitive renvoie le dernier élément stocké dans la pile indiquée par le paramètre qui doit renvoyé à une liste en le retirant de cette dernière.

Exemples :

ECRIS DEPILE "MAPILE → 568
ECRIS DEPILE "MAPILE → 124

- **QUEUE** : attend une variable puis une liste ou un mot en entrée – ne renvoie rien – la primitive stocke le second paramètre dans la première liste pour former une queue.

Exemple :

QUEUE "MAQUEUE [ceci est stocké] → -

- **DEQUEUE** : attend une variable en entrée – renvoie une liste ou un mot – la primitive renvoie le premier élément stocké dans une queue en le retirant de cette dernière.

Exemple :

ECRIS DEQUEUE "MAQUEUE → un ancien élément stocké



Les notions de queues et de piles sont très souples pour **GVLOGO** si bien qu'il est possible de traiter n'importe quelle variable pourvu qu'elle renvoie une liste. De plus, les primitives **ENQUEUE** et **EMPILE** sont strictement équivalentes : on peut donc empiler une donnée et traiter la liste comme une pile, une queue ou une liste ordinaire, et réciproquement !

IMPLEMENTATION DES PILES

L'unité **TGVStacks** offre une classe de pile générique et les classes nécessaires pour manipuler des entiers, des réels et des chaînes de caractères.



On aura tout intérêt, si l'on utilise Delphi, à utiliser les piles fournies dans **System.Generics.Collections**²⁴. Comme Lazarus ne sait (pas encore) traiter les classes génériques de manière aussi étendue que Delphi, la solution adoptée pour le projet est de travailler à partir d'une classe générique commune. Elle est plus limitée dans la mesure où elle ne connaît pas d'énumération. D'autre part, la notification est moins complète aussi, Lazarus n'acceptant que les classes comme support des génériques. En revanche, plusieurs fonctionnalités intéressantes ont été ajoutées.

CONSTANTES

L'unité **TGVConsts** a été complétée. On a fixé la taille minimale de la pile à 8, ce qui permet de ne pas réallouer en permanence de la mémoire pour l'accroissement de la pile. Trois nouvelles erreurs possibles ont été par ailleurs prévues.

```
{ piles }

CMinStack = 8; // minimum d'espace pour une pile

type
  { erreurs }
  TGVError = (
    C_None, // pas d'erreur
    [...]
    C_EmptyStack, // pile interne vide
    C_OutOfMemory, // mémoire insuffisante pour la pile
    C_LowStack // pile insuffisante
  );
```

²⁴ Voir le dossier « EasyStack » qui contient un exemple de programme avec Delphi : on remarquera qu'en quelques lignes les extensions désirées sont implémentées.

On retrouve ces erreurs dans les chaînes de ressources :

```
ME_EmptyStack = 'La pile interne est vide.';
ME_OutOfMemory = 'La mémoire est insuffisante pour la pile.';
ME_LowStack = 'Pas assez d"éléments dans la pile (%d pour %d).';
```

Enfin, des constantes énumérées ont été définies pour les notifications :

```
TGVStackNotification = (stAdded, stRemoved, stChanged, stCleared);
```

La première indique qu'un élément a été ajouté à la pile, la deuxième qu'un élément a été retiré, la troisième qu'un changement autre est intervenu, la dernière que la pile a été remise à zéro.

LA CLASSE TGVSTACK

Les piles sont implémentées en utilisant les classes génériques. Pour rappel, ces classes fournissent un modèle sur lequel viendront s'appuyer des classes spécialisées. Ainsi, à partir d'une seule classe de pile, on peut créer toutes les piles imaginables simplement en précisant le type à utiliser.

Les avantages évidents de cette méthode sur celle plus classique de définitions de classes au cas par cas sont un gain de temps, une souplesse largement accrue, une sécurité renforcée puisque le code n'a pas à être ressaisi pour chacune des classes et la possibilité de modifier le comportement de toutes les classes spécialisées à partir de la seule modification de la classe générique :

```
{$IFNDEF Delphi}generic{$ENDIF}TGVStack<T> = class
[...]
end;

// piles spécialisées
TGVIntegerStack = {$IFNDEF Delphi}specialize{$ENDIF} TGVStack<Integer>;
TGVRealStack = {$IFNDEF Delphi}specialize{$ENDIF} TGVStack<Real>;
TGVStringStack = {$IFNDEF Delphi}specialize{$ENDIF} TGVStack<string>;
```

En ce qui concerne l'interface de la classe **TGVStack**, en voici le listing :

```
{ TGVStack }

{$IFNDEF Delphi}generic{$ENDIF}TGVStack<T> = class
private
  fItems: array of T;
  fCount: Integer; // nombre d'éléments
  fCapacity: Integer; // capacité actuelle
  fOnNotify: TGVStackEvent; // notification
  procedure Expand; // expansion si nécessaire
  function GetCapacity: Integer; // capacité actuelle
  function GetItem(N: Integer): T; // accès à un élément
  procedure SetCapacity(const Value: Integer); // fixe la capacité
protected
  procedure Notify(Action: TGVStackNotification); virtual; // notification
  procedure DoPush(const Value: T); // empilement
  function DoPop: T; // dépilement
public
  constructor Create; overload; // création
  destructor Destroy; override; // destruction
  procedure Clear; // nettoyage
  procedure Push(const Value: T); // empilement avec notification
  function Pop: T; // dépilement avec notification
  function Peek: T; // sommet de la pile
  procedure Drop; // sommet de la pile éjecté
  procedure Dup; // duplication au sommet de la pile
  procedure Swap; // inversion au sommet de la pile
  procedure Over; // duplication de l'avant-dernier
```

```

procedure Rot; // rotation au sommet de la pile
procedure Shrink; // contraction de la pile
property Count: Integer read fCount default 0; // compte des éléments
// capacité de la pile
property Capacity: Integer read GetCapacity write SetCapacity
  default CMinStack;
// notification d'un changement
property OnNotify: TGVStackEvent read fOnNotify write fOnNotify;
// accès direct à un élément
property Item[N: Integer]: T read GetItem; default;
end;

```

L'interface appelle les remarques suivantes :

- ✓ le paramètre **T** qui apparaît souvent sera celui remplacé par le type réel mis en œuvre par la classe spécialisée ;
- ✓ le fondement de la pile est le tableau ouvert **fItems** qui verra son dernier élément pointé par le champ privé **fCount** ;
- ✓ la notion de capacité (à travers le champ privé **fCapacity**) complique un peu la classe, mais permet d'accélérer les empilements en ne procédant pas sans cesse à des allocations de mémoire ;
- ✓ la propriété **Item** est définie par défaut, ce qui signifie par conséquent que l'utilisateur n'a pas besoin de la spécifier (les deux écritures sont équivalentes : **MaPile.Item[2]** et **Mapile[2]**) ;
- ✓ la plupart des méthodes définies sont destinées à manipuler la pile : **Push**, **Peek**, **Pop**, **Drop**, **Dup**, **Swap**, **Over** et **Rot**. Les programmeurs en Forth reconnaîtront là des outils très familiers ;
- ✓ la présence des méthodes **DoPush** et **DoPop** se justifie par le fait que ces opérations de base sur la pile sont employées par les autres méthodes qui ne doivent déclencher l'événement **OnNotify** qu'une seule fois : ce sont donc deux méthodes qui ne notifient rien, contrairement à **Push** et à **Pop**.

TEST DE L'UNITE TGVSTACKS

Pour ne pas changer, le programme de test est décliné en quatre versions :

- ✓ Lazarus pour Windows 32 ;
- ✓ Lazarus pour Linux ;
- ✓ Delphi pour Windows 32 ;
- ✓ Delphi pour Windows 64.

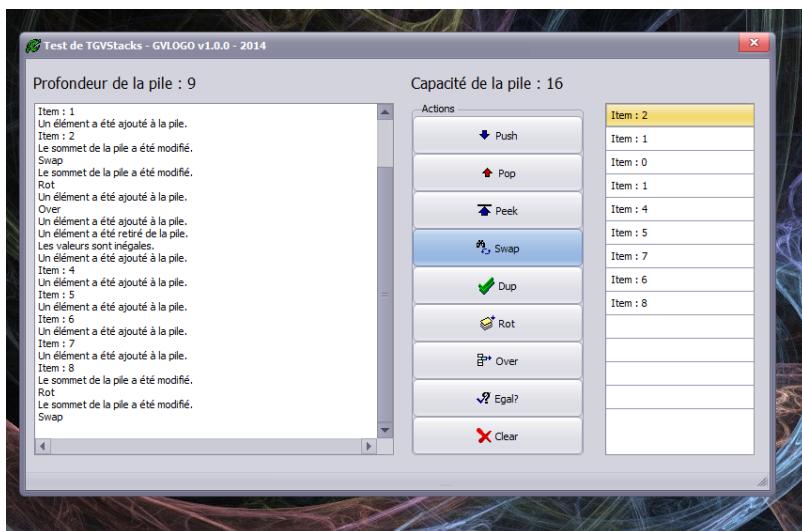


Figure 33 - Test des piles avec Delphi

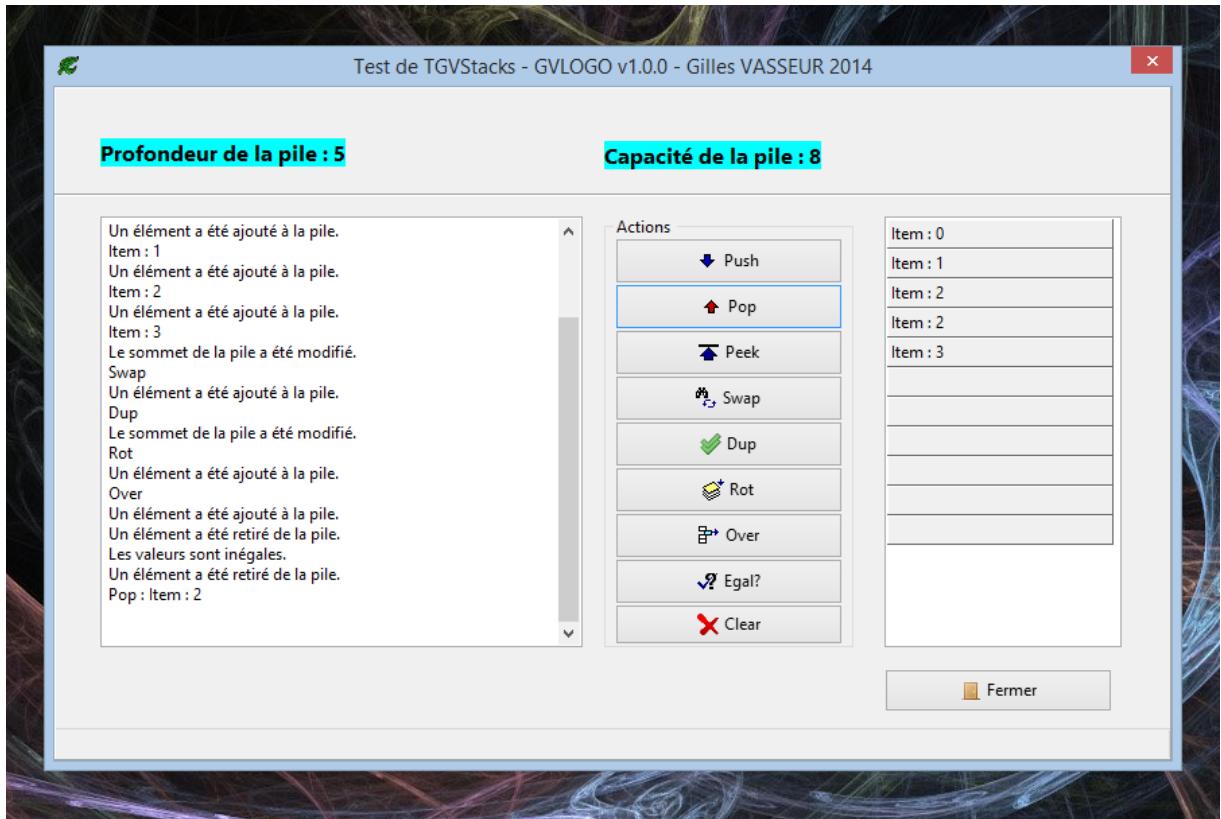


Figure 34 - Test des piles avec Lazarus (Windows)

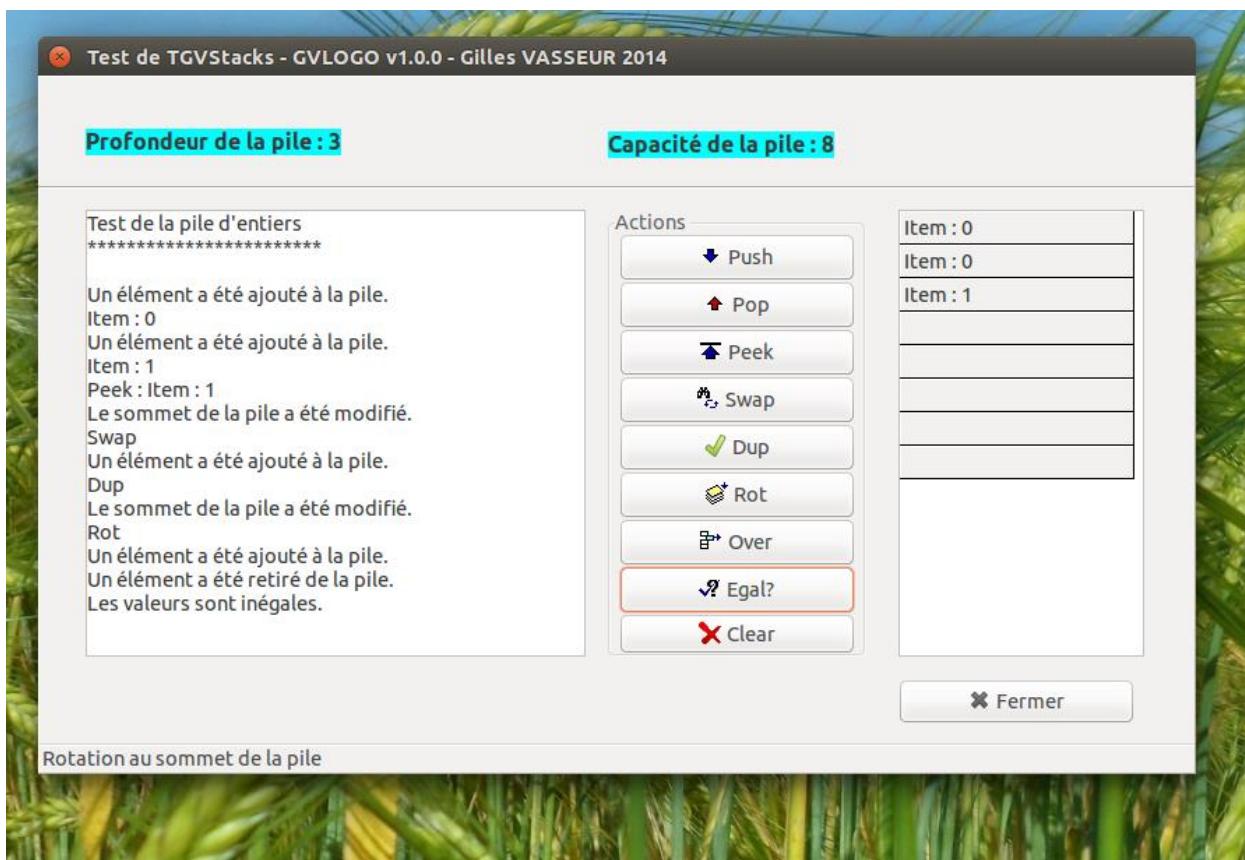


Figure 35 - Test des piles avec Lazarus (Linux)

Une méthode exige quelques commentaires :

```
procedure TMainForm.StackChanged(Sender: TObject; Act: TGVStackNotification);
// changement de la pile
var
  I: Integer;
begin
  for I := 0 to FStackStr.Count - 1 do
    sgStack.Cells[0,I] := FStackStr[I];
  case Act of
    stAdded : begin
      mmoActions.Lines.Add('Un élément a été ajouté à la pile.');
      sgStack.RowCount := FStackStr.Count + 4;
    end;
    stRemoved : begin
      mmoActions.Lines.Add('Un élément a été retiré de la pile.');
      sgStack.Cells[0,FStackStr.Count] := EmptyStr;
    end;
    stChanged : mmoActions.Lines.Add('Le sommet de la pile a été modifié.');
    stCleared: begin
      mmoActions.Lines.Add('La pile est vide.');
      for I := 0 to sgStack.RowCount -1 do
        sgStack.Cells[0,I] := EmptyStr;
    end;
  end;
  UpdateButtons;
end;
```

Cette méthode a été affectée à la propriété **OnNotify** de la pile. Elle remplit le composant **StringGrid** avec les éléments de la pile, répartit son travail suivant l'action notifiée et met à jour les boutons qui seront actifs ou non selon la profondeur de la pile. On remarquera la mise à jour permanente de la **StringGrid** afin de ne pas laisser de chaînes visibles après un dépilement, et l'augmentation si nécessaire du nombre de lignes disponibles sans quoi une exception serait déclenchée pour un index hors limites.

Une autre méthode peut poser problème : il s'agit de celle appliquée lorsque l'utilisateur presse un bouton. Cette méthode est partagée par tous les boutons de la fenêtre. Afin de déterminer quel bouton a été pressé, on se sert de la propriété **TabOrder** qui renvoie l'ordre du composant si l'utilisateur utilise la touche de tabulation.



On utilise plutôt la propriété particulière **Tag** qui a le mérite de ne pas dépendre de l'ordre des composants, mais qui ne vérifie pas qu'elle est unique. Avec **TabOrder**, on devra revoir le fichier source si l'on modifie l'ordre des composants ou si l'on en ajoute/supprime un. Ici, le choix de **TabOrder** ne se justifie que par le choix pédagogique de montrer que plusieurs solutions existent quand il s'agit de résoudre un problème !

```
procedure TMainForm.btnPushClick(Sender: TObject);
var
  St: string;
begin
  case (Sender as TBitBtn).TabOrder of
    0:
      begin
        St := 'Item : ' + IntToStr(FStackStr.Count);
        FStackStr.Push(St);
      end;
    1:
      St := 'Pop : ' + FStackStr.Pop;
    2:
      St := 'Peek : ' + FStackStr.Peek;
    3:
      begin
        St := 'Swap';
        FStackStr.Swap;
      end;
  end;
end;
```

```

end;
4:
begin
  St := 'Dup';
  FStackStr.Dup;
end;
5:
begin
  St := 'Rot';
  FStackStr.Rot;
end;
6:
begin
  St := 'Over';
  FStackStr.Over;
end;
7:
if IsEqual then
  St := 'Les valeurs sont égales.'
else
  St := 'Les valeurs sont inégales.';
8:
begin
  FStackStr.Clear;
  FStackStr.Shrink;
  St := '*** CLEAR ***';
end;
end;
lblCapacity.Caption := 'Capacité de la pile : ' +
  IntToStr(FStackStr.Capacity);
lblDepth.Caption := 'Profondeur de la pile : ' + IntToStr(FStackStr.Count);
mmoActions.Lines.Append(St);
end;

```

Une chaîne **St** est préparée avant d'être ajoutée au composant **TMemo**. Comme la taille de la pile peut être modifiée, les composants **TLabel** correspondants sont ajustés.



Il pouvait être plus judicieux de placer cette dernière mise à jour dans la méthode **StackChanged** qui couvre automatiquement tous les changements de la pile. Le lecteur pourra s'amuser à réaliser ce petit changement.

L'EVALUATION D'UNE EXPRESSION MATHEMATIQUE

DEFINITION

OPERATIONS DANS L'EVALUATEUR

IMPLEMENTATION DE L'EVALUATEUR

LES COMPOSANTS DU LANGAGE

LES VARIABLES

LES PROCEDURES

LES PAQUETS

LE NOYAU

LES PRIMITIVES

LE PROGRAMME FINAL

L'INTERPRETEUR

L'INTERFACE UTILISATEUR

MODE D'EMPLOI DE GVLOGO

INSTALLER GVLOGO

INTERFACE ET MENUS

CREER UN PROGRAMME

EXECUTER UN PROGRAMME

SAUVEGARDER ET RECUPERER UN PROGRAMME

MODIFIER UN PROGRAMME

LES MESSAGES D'ERREUR

LE DEBOGAGE

METTRE A JOUR LE LOGICIEL

PROGRAMMES EXEMPLES

TRAVAILLER AVEC LA SOURIS

TRAVAILLER AVEC LES LISTES

TRAVAILLER AVEC LES LISTES DE PROPRIETES

MANIPULER L'ESPACE DE TRAVAIL

LICENCE GNU

La licence GNU GPL protégeant et accompagnant tous les logiciels décrits dans ce livre est fournie avec les sources des programmes et unités. Elle garantit que ces programmes resteront dans le cadre du logiciel libre et gratuit.

Voici le texte en français de la licence GNU :

LICENCE PUBLIQUE GÉNÉRALE GNU

Version 3, du 29 juin 2007.

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>

Chacun est autorisé à copier et distribuer des copies conformes de ce document de licence, mais toute modification en est proscrite.

Traduction française par Philippe Verdy
<verdy_p (à) wanadoo (point) fr>, le 30 juin 2007.

Avertissement important au sujet de cette traduction française.

Ceci est une traduction en français de la licence “GNU General Public License” (GPL). Cette traduction est fournie ici dans l'espoir qu'elle facilitera sa compréhension, mais elle ne constitue pas une traduction officielle ou approuvée d'un point de vue juridique.

La Free Software Foundation (FSF) ne publie pas cette traduction et ne l'a pas approuvée en tant que substitut valide au plan légal pour la licence authentique “GNU General Public Licence”. Cette traduction n'a pas encore été passée en revue attentivement par un juriste et donc le traducteur ne peut garantir avec certitude qu'elle représente avec exactitude la signification légale des termes de la licence authentique “GNU General Public License” publiée en anglais. Cette traduction n'établit donc légalement aucun des termes et conditions d'utilisation d'un logiciel sous licence GNU GPL — seul le texte original en anglais le fait. Si vous souhaitez être sûr que les activités que vous projetez seront autorisées par la GNU General Public License, veuillez vous référer à sa seule version anglaise authentique.

La FSF vous recommande fermement de ne pas utiliser cette traduction en tant que termes officiels pour vos propres programmes ; veuillez plutôt utiliser la version anglaise authentique telle que publiée par la FSF. Si vous choisissez d'acheminer cette traduction en même temps qu'un Programme sous licence GNU GPL, cela ne vous dispense pas de l'obligation d'acheminer en même temps une copie de la licence authentique en anglais, et de conserver dans la traduction cet avertissement important en français et son équivalent en anglais ci-dessous.

Important Warning About This French Translation.

This is a translation of the GNU General Public License (GPL) into French. This translation is distributed in the hope that it will facilitate understanding, but it is not an official or legally approved translation.

The Free Software Foundation (FSF) is not the publisher of this translation and has not approved it as a legal substitute for the authentic GNU General Public License. The translation has not been reviewed carefully by lawyers, and therefore the translator cannot be sure that it exactly represents the legal meaning of the authentic GNU

General Public License published in English. This translation does not legally state the terms and conditions of use of any Program licenced under GNU GPL — only the original English text of the GNU LGPL does that. If you wish to be sure whether your planned activities are permitted by the GNU General Public License, please refer to its sole authentic English version.

The FSF strongly urges you not to use this translation as the official distribution terms for your programs; instead, please use the authentic English version published by the FSF. If you choose to convey this translation along with a Program covered by the GPL Licence, this does not remove your obligation to convey at the same time a copy of the authentic GNU GPL License in English, and you must keep in this translation this important warning in English and its equivalent in French above.

Préambule

La Licence Publique Générale GNU (“GNU General Public License”) est une licence libre, en “copyleft”, destinée aux œuvres logicielles et d’autres types de travaux.

Les licences de la plupart des œuvres logicielles et autres travaux de pratique sont conçues pour ôter votre liberté de partager et modifier ces travaux. En contraste, la Licence Publique Générale GNU a pour but de garantir votre liberté de partager et changer toutes les versions d’un programme — afin d’assurer qu’il restera libre pour tous les utilisateurs. Nous, la Free Software Foundation, utilisons la Licence Publique Générale GNU pour la plupart de nos logiciels ; cela s’applique aussi à tout autre travail édité de cette façon par ses auteurs. Vous pouvez, vous aussi, l’appliquer à vos propres programmes.

Quand nous parlons de logiciel libre (“free”), nous nous référons à la liberté (“freedom”), pas au prix. Nos Licences Publiques Générales sont conçues pour assurer que vous ayez la liberté de distribuer des copies de logiciel libre (et le facturer si vous le souhaitez), que vous receviez le code source ou pourriez l’obtenir si vous le voulez, que vous pourriez modifier le logiciel ou en utiliser toute partie dans de nouveaux logiciels libres, et que vous sachiez que vous avez le droit de faire tout ceci.

Pour protéger vos droits, nous avons besoin d’empêcher que d’autres vous restreignent ces droits ou vous demande de leur abandonner ces droits. En conséquence, vous avez certaines responsabilités si vous distribuez des copies d’un tel programme ou si vous le modifiez : les responsabilités de respecter la liberté des autres.

Par exemple, si vous distribuez des copies d’un tel programme, que ce soit gratuit ou contre un paiement, vous devez accorder aux destinataires les mêmes libertés que vous avez reçues. Vous devez aussi vous assurer qu’eux aussi reçoivent ou peuvent recevoir son code source. Et vous devez leur montrer les termes de cette licence afin qu’ils connaissent leurs droits.

Les développeurs qui utilisent la GPL GNU protègent vos droits en deux étapes : (1) ils affirment leur droits d'auteur (“copyright”) sur le logiciel, et (2) vous accordent cette Licence qui vous donne la permission légale de le copier, le distribuer et/ou le modifier.

Pour la protection des développeurs et auteurs, la GPL stipule clairement qu'il n'y a pas de garantie pour ce logiciel libre. Aux fins à la fois des utilisateurs et auteurs, la GPL requiert que les versions modifiées soient marquées comme changées, afin que leurs problèmes ne soient pas attribués de façon erronée aux auteurs des versions précédentes.

Certains dispositifs sont conçus pour empêcher l'accès des utilisateurs à l'installation ou l'exécution de versions modifiées du logiciel à

l'intérieur de ces dispositifs, alors que les fabricants le peuvent. Ceci est fondamentalement incompatible avec le but de protéger la liberté des utilisateurs de modifier le logiciel. L'aspect systématique de tels abus se produit dans le secteur des produits destinés aux utilisateurs individuels, ce qui est précédemment ce qui est le plus inacceptable. Aussi, nous avons conçu cette version de la GPL pour prohiber cette pratique pour ces produits. Si de tels problèmes surviennent dans d'autres domaines, nous nous tenons prêt à étendre cette restriction à ces domaines dans de futures versions de la GPL, autant qu'il sera nécessaire pour protéger la liberté des utilisateurs.

Finalement, chaque programme est constamment menacé par les brevets logiciels. Les États ne devraient pas autoriser de tels brevets à restreindre le développement et l'utilisation de logiciels libres sur des ordinateurs d'usage général ; mais dans ceux qui le font, nous voulons spécialement éviter le danger que les brevets appliqués à un programme libre puisse le rendre effectivement propriétaire. Pour empêcher ceci, la GPL assure que les brevets ne peuvent être utilisés pour rendre le programme non-libre.

Les termes précis et conditions concernant la copie, la distribution et la modification suivent.

TERMES ET CONDITIONS

Article 0. Définitions.

« Cette Licence » se réfère à la version 3 de la “GNU General Public License” (le texte original en anglais).

« Droit d'Auteur » signifie aussi les droits du “copyright” ou voisins qui s'appliquent à d'autres types de travaux, tels que ceux sur les masques de semi-conducteurs.

« Le Programme » se réfère à tout travail qui peut être sujet au Droit d'Auteur (“copyright”) et dont les droits d'utilisation sont concédés en vertu de cette Licence. Chacun des Licenciés, à qui cette Licence est concédée, est désigné par « vous. » Les « Licenciés » et les « Destinataires » peuvent être des personnes physiques ou morales (individus ou organisations).

« Modifier » un travail signifie en obtenir une copie et adapter tout ou partie du travail d'une façon nécessitant une autorisation d'un titulaire de Droit d'Auteur, autre que celle permettant d'en produire une copie conforme. Le travail résultant est appelé une « version modifiée » du précédent travail, ou un travail « basé sur » le précédent travail.

Un « Travail Couvert » signifie soit le Programme non modifié soit un travail basé sur le Programme.

« Propager » un travail signifie faire quoi que ce soit avec lui qui, sans permission, vous rendrait directement ou indirectement responsable d'un délit de contrefaçon suivant les lois relatives au Droit d'Auteur, à l'exception de son exécution sur un ordinateur ou de la modification d'une copie privée. La propagation inclue la copie, la distribution (avec ou sans modification), la mise à disposition envers le public, et aussi d'autres activités dans certains pays.

« Acheminer » un travail signifie tout moyen de propagation de celui-ci qui permet à d'autres parties de réaliser ou recevoir des copies. La simple interaction d'un utilisateur à travers un réseau informatique, sans transfert effectif d'une copie, ne constitue pas un acheminement.

Une interface utilisateur interactive affiche des « Notices Légales Appropriées » quand elle comprend un dispositif convenable, bien visible et évident qui (1) affiche une notice appropriée sur les droits d'auteur et (2) informe l'utilisateur qu'il n'y a pas de garantie pour le travail (sauf si des garanties ont été fournies hors du cadre de

cette Licence), que les licenciés peuvent acheminer le travail sous cette Licence, et comment voir une copie de cette Licence. Si l'interface présente une liste de commandes utilisateur ou d'options, tel qu'un menu, un élément évident dans la liste présentée remplit ce critère.

Article 1. Code source.

Le « code source » d'un travail signifie la forme préférée du travail permettant ou facilitant les modifications de celui-ci. Le « code objet » d'un travail signifie toute forme du travail qui n'en est pas le code source.

Une « Interface Standard » signifie une interface qui est soit celle d'une norme officielle définie par un organisme de normalisation reconnu ou, dans le cas des interfaces spécifiées pour un langage de programmation particulier, une interface largement utilisée parmi les développeurs travaillant dans ce langage.

Les « Bibliothèques Système » d'un travail exécutable incluent tout ce qui, en dehors du travail dans son ensemble, (a) est inclus dans la forme usuelle de paquetage d'un Composant Majeur mais ne fait pas partie de ce Composant Majeur et (b) sert seulement à permettre l'utilisation du travail avec ce Composant Majeur ou à implémenter une Interface Standard pour laquelle une implémentation est disponible au public sous forme de code source ; un « Composant Majeur » signifie, dans ce contexte, un composant majeur essentiel (noyau, système de fenêtrage, etc.) du système d'exploitation (le cas échéant) d'un système sur lequel le travail exécutable fonctionne, ou bien un compilateur utilisé pour produire le code objet du travail, ou un interprète de code objet utilisé pour exécuter celui-ci.

Le « Source Correspondant » d'un travail sous forme de code objet signifie l'ensemble des codes sources nécessaires pour générer, installer et (dans le cas d'un travail exécutable) exécuter le code objet et modifier le travail, y compris les scripts pour contrôler ces activités. Cependant, cela n'inclue pas les Bibliothèques Système du travail, ni les outils d'usage général ou les programmes libres généralement disponibles qui peuvent être utilisés sans modification pour achever ces activités mais ne sont pas partie de ce travail. Par exemple le Source Correspondant inclut les fichiers de définition d'interfaces associés aux fichiers sources du travail, et le code source des bibliothèques partagées et des sous-routines liées dynamiquement, pour lesquelles le travail est spécifiquement conçu pour les requérir via, par exemple, des communications de données ou contrôles de flux internes entre ces sous-programmes et d'autres parties du travail.

Le Source Correspondant n'a pas besoin d'inclure tout ce que les utilisateurs peuvent regénérer automatiquement à partir d'autres parties du Source Correspondant.

Le Source Correspondant pour un travail sous forme de code source est ce même travail.

Article 2. Permissions de base.

Tous les droits accordés suivant cette Licence le sont jusqu'au terme des Droits d'Auteur ("copyright") sur le Programme, et sont irrévocables pourvu que les conditions établies soient remplies. Cette Licence affirme explicitement votre permission illimitée d'exécuter le Programme non modifié. La sortie produite par l'exécution d'un Travail Couvert n'est couverte par cette Licence que si cette sortie, étant donné leur contenu, constitue un Travail Couvert. Cette Licence reconnaît vos propres droits d'usage raisonnable ("fair use" en législation des États-Unis d'Amérique) ou autres équivalents, tels qu'ils sont pourvus par la loi applicable sur le Droit d'Auteur ("copyright").

Vous pouvez créer, exécuter et propager sans condition des Travaux Couverts que vous n'acheminez pas, aussi longtemps que votre licence demeure en vigueur. Vous pouvez acheminer des Travaux Couverts à d'autres personnes dans le seul but de leur faire réaliser des modifications à votre usage exclusif, ou pour qu'ils vous fournissent des facilités vous permettant d'exécuter ces travaux, pourvu que vous vous conformiez aux termes de cette Licence lors de l'acheminement de tout matériel dont vous ne contrôlez pas le Droit d'Auteur ("copyright"). Ceux qui, dès lors, réalisent ou exécutent pour vous les Travaux Couverts ne doivent alors le faire qu'exclusivement pour votre propre compte, sous votre direction et votre contrôle, suivant des termes qui leur interdisent de réaliser, en dehors de leurs relations avec vous, toute copie de votre matériel soumis au Droit d'Auteur.

L'acheminement dans toutes les autres circonstances n'est permis que selon les conditions établies ci-dessous. La concession de sous-licences n'est pas autorisé ; l'article 10 rend cet usage non nécessaire.

Article 3. Protection des droits légaux des utilisateurs envers les lois anti-contournement.

Aucun Travail Couvert ne doit être vu comme faisant partie d'une mesure technologique effective selon toute loi applicable remplissant les obligations prévues à l'article 11 du traité international sur le droit d'auteur adopté à l'OMPI le 20 décembre 1996, ou toutes lois similaires qui prohibent ou restreignent le contournement de telles mesures.

Si vous acheminez un Travail Couvert, vous renoncez à tout pouvoir légal d'interdire le contournement des mesures technologiques dans tous les cas où un tel contournement serait effectué en exerçant les droits prévus dans cette Licence pour ce Travail Couvert, et vous déclarez rejeter toute intention de limiter l'opération ou la modification du Travail, en tant que moyens de renforcer, à l'encontre des utilisateurs de ce Travail, vos droits légaux ou ceux de tierces parties d'interdire le contournement des mesures technologiques.

Article 4. Acheminement des copies conformes.

Vous pouvez acheminer des copies conformes du code source du Programme tel que vous l'avez reçu, sur n'importe quel support, pourvu que vous publiez scrupuleusement et de façon appropriée sur chaque copie une notice de Droit d'Auteur appropriée ; gardez intactes toutes les notices établissant que cette Licence et tous les termes additionnels non permis ajoutés en accord avec l'article 7 s'appliquent à ce code ; et donnez à chacun des Destinataires une copie de cette Licence en même temps que le Programme.

Vous pouvez facturer à un prix quelconque, y compris gratuit, chacune des copies que vous acheminez, et vous pouvez offrir une protection additionnelle de support ou de garantie en échange d'un paiement.

Article 5. Acheminement des versions sources modifiées.

Vous pouvez acheminer un travail basé sur le Programme, ou bien les modifications pour le produire à partir du Programme, sous la forme de code source suivant les termes de l'article 4, pourvu que vous satisfassiez aussi à chacune des conditions requises suivantes :

- a) Le travail doit comporter des notices évidentes établissant que vous l'avez modifié et donnant la date correspondante.
- b) Le travail doit comporter des notices évidentes établissant qu'il est édité selon cette Licence et les conditions ajoutées d'après l'article 7. Cette obligation vient modifier l'obligation de l'article 4 de « garder intactes toutes les notices. »
- c) Vous devez licencier le travail entier, comme un tout, suivant

cette Licence à quiconque entre en possession d'une copie. Cette Licence s'appliquera en conséquence, avec les termes additionnels applicables prévus par l'article 7, à la totalité du travail et chacune de ses parties, indépendamment de la façon dont ils sont empaquetés. Cette licence ne donne aucune permission de licencier le travail d'une autre façon, mais elle n'invalidise pas une telle permission si vous l'avez reçue séparément.

- d) Si le travail a des interfaces utilisateurs interactives, chacune doit afficher les Notices Légales Appropriées ; cependant si le Programme a des interfaces qui n'affichent pas les Notices Légales Appropriées, votre travail n'a pas à les modifier pour qu'elles les affichent.

Une compilation d'un Travail Couvert avec d'autres travaux séparés et indépendants, qui ne sont pas par leur nature des extensions du Travail Couvert, et qui ne sont pas combinés avec lui de façon à former un programme plus large, dans ou sur un volume de stockage ou un support de distribution, est appelé un « agrégat » si la compilation et son Droit d'Auteur résultant ne sont pas utilisés pour limiter l'accès ou les droits légaux des utilisateurs de la compilation en deça de ce que permettent les travaux individuels. L'inclusion d'un Travail Couvert dans un agrégat ne cause pas l'application de cette Licence aux autres parties de l'agrégrat.

Article 6. Acheminement des formes non sources.

Vous pouvez acheminer sous forme de code objet un Travail Couvert suivant les termes des articles 4 et 5, pourvu que vous acheminiez également suivant les termes de cette Licence le Source Correspondant lisible par une machine, d'une des façons suivantes :

- a) Acheminer le code objet sur, ou inclus dans, un produit physique (y compris un support de distribution physique), accompagné par le Source Correspondant fixé sur un support physique durable habituellement utilisé pour les échanges de logiciels.
- b) Acheminer le code objet sur, ou inclus dans, un produit physique (y compris un support de distribution physique), accompagné d'une offre écrite, valide pour au moins trois années et valide pour aussi longtemps que vous fournissez des pièces de rechange ou un support client pour ce modèle de produit, afin de donner à quiconque possède le code objet soit (1) une copie du Source Correspondant à tout logiciel dans ce produit qui est couvert par cette Licence, sur un support physique durable habituellement utilisé pour les échanges de logiciels, pour un prix non supérieur au coût raisonnable de la réalisation physique de l'acheminement de la source, ou soit (2) un accès permettant de copier le Source Correspondant depuis un serveur réseau sans frais.
- c) Acheminer des copies individuelles du code objet avec une copie de l'offre écrite de fournir le Source Correspondant. Cette alternative est permise seulement occasionnellement et non commercialement, et seulement si vous avez reçu le code objet avec une telle offre, en accord avec l'article 6 alinéa b.
- d) Acheminer le code objet en offrant un accès depuis un emplacement désigné (gratuit ou contre facturation) et offrir un accès équivalent au Source Correspondant de la même façon via le même emplacement et sans facturation supplémentaire. Vous n'avez pas besoin d'obliger les Destinataires à copier le Source Correspondant en même temps que le code objet. Si l'emplacement pour copier le code objet est un serveur réseau, le Source Correspondant peut être sur un serveur différent (opéré par vous ou par un tiers) qui supporte des facilités équivalentes de copie, pourvu que vous mainteniez des directions claires à proximité du code objet indiquant où trouver le Source Correspondant. Indépendamment de quel serveur héberge le Source Correspondant, vous restez obligé de vous assurer qu'il reste disponible aussi longtemps que nécessaire pour satisfaire à ces

obligations.

e) Acheminer le code objet en utilisant une transmission d'égal-à-égal, pourvu que vous informez les autres participants sur où le code objet et le Source Correspondant du travail sont offerts sans frais au public général suivant l'article 6 alinéa d. Une portion séparable du code objet, dont le code source est exclu du Source Correspondant en tant que Bibliothèque Système, n'a pas besoin d'être inclus dans l'acheminement du travail sous forme de code objet.

Un « Produit Utilisateur » est soit (1) un « Produit de Consommation », ce qui signifie toute propriété personnelle tangible normalement utilisée à des fins personnelles, familiales ou relatives au foyer, soit (2) toute chose conçue ou vendue pour l'incorporation dans un lieu d'habitation. Pour déterminer si un produit constitue un Produit de Consommation, les cas ambigus sont résolus en fonction de la couverture. Pour un produit particulier reçu par un utilisateur particulier, l'expression « normalement utilisée » ci-dessus se réfère à une utilisation typique ou l'usage commun de produits de même catégorie, indépendamment du statut de cet utilisateur particulier ou de la façon spécifique dont cet utilisateur particulier utilise effectivement ou s'attend lui-même ou est attendu à utiliser ce produit. Un produit est un Produit de Consommation indépendamment du fait que ce produit a ou n'a pas d'utilisations substantielles commerciales, industrielles ou hors Consommation, à moins que de telles utilisations représentent le seul mode significatif d'utilisation du produit.

Les « Informations d'Installation » d'un Produit Utilisateur signifient toutes les méthodes, procédures, clés d'autorisation ou autres informations requises pour installer et exécuter des versions modifiées d'un Travail Couvert dans ce Produit Utilisateur à partir d'une version modifiée de son Source Correspondant. Les informations qui suffisent à assurer la continuité de fonctionnement du code objet modifié ne doivent en aucun cas être empêchées ou interférées du seul fait qu'une modification a été effectuée.

Si vous acheminez le code objet d'un Travail Couvert dans, ou avec, ou spécifiquement pour l'utilisation dans, un Produit Utilisateur et l'acheminement se produit en tant qu'élément d'une transaction dans laquelle le droit de possession et d'utilisation du Produit Utilisateur est transféré au Destinataire définitivement ou pour un terme fixé (indépendamment de la façon dont la transaction est caractérisée), le Source Correspondant acheminé selon cet article-ci doit être accompagné des Informations d'Installation. Mais cette obligation ne s'applique pas si ni vous ni aucune tierce partie ne détient la possibilité d'installer un code objet modifié sur le Produit Utilisateur (par exemple, le travail a été installé en mémoire morte).

L'obligation de fournir les Informations d'Installation n'inclue pas celle de continuer à fournir un service de support, une garantie ou des mises à jour pour un travail qui a été modifié ou installé par le Destinataire, ou pour le Produit Utilisateur dans lequel il a été modifié ou installé. L'accès à un réseau peut être rejeté quand la modification elle-même affecte matériellement et défavorablement les opérations du réseau ou viole les règles et protocoles de communication au travers du réseau.

Le Source Correspondant acheminé et les Informations d'Installation fournies, en accord avec cet article, doivent être dans un format publiquement documenté (et dont une implémentation est disponible auprès du public sous forme de code source) et ne doit nécessiter aucune clé ou mot de passe spécial pour le dépaquetage, la lecture ou la copie.

Article 7. Termes additionnels.

Les « permissions additionnelles » désignent les termes qui supplémentent ceux de cette Licence en émettant des exceptions à l'une

ou plusieurs de ses conditions. Les permissions additionnelles qui sont applicables au Programme entier doivent être traitées comme si elles étaient incluent dans cette Licence, dans les limites de leur validité suivant la loi applicable. Si des permissions additionnelles s'appliquent seulement à une partie du Programme, cette partie peut être utilisée séparément suivant ces permissions, mais le Programme tout entier reste gouverné par cette Licence sans regard aux permissions additionnelles.

Quand vous acheminez une copie d'un Travail Couvert, vous pouvez à votre convenance ôter toute permission additionnelle de cette copie, ou de n'importe quelle partie de celui-ci. (Des permissions additionnelles peuvent être rédigées de façon à requérir leur propre suppression dans certains cas où vous modifiez le travail.) Vous pouvez placer les permissions additionnelles sur le matériel acheminé, ajoutées par vous à un Travail Couvert pour lequel vous avez ou pouvez donner les permissions de Droit d'Auteur ("copyright") appropriées.

Nonobstant toute autre clause de cette Licence, pour tout constituant que vous ajoutez à un Travail Couvert, vous pouvez (si autorisé par les titulaires de Droit d'Auteur pour ce constituant) compléter les termes de cette Licence avec des termes :

- a) qui rejettent la garantie ou limitent la responsabilité de façon différente des termes des articles 15 et 16 de cette Licence ; ou
- b) qui requièrent la préservation de notices légales raisonnables spécifiées ou les attributions d'auteur dans ce constituant ou dans les Notices Légales Appropriées affichées par les travaux qui le contiennent ; ou
- c) qui prohibent la représentation incorrecte de l'origine de ce constituant, ou qui requièrent que les versions modifiées d'un tel constituant soit marquées par des moyens raisonnables comme différentes de la version originale ; ou
- d) qui limitent l'usage à but publicitaire des noms des concédants de licence et des auteurs du constituant ; ou
- e) qui refusent à accorder des droits selon la législation relative aux marques commerciales, pour l'utilisation dans des noms commerciaux, marques commerciales ou marques de services ; ou
- f) qui requièrent l'indemnisation des concédants de licences et auteurs du constituant par quiconque achemine ce constituant (ou des versions modifiées de celui-ci) en assumant contractuellement la responsabilité envers le Destinataire, pour toute responsabilité que ces engagements contractuels imposent directement à ces octroyants de licences et auteurs.

Tous les autres termes additionnels non permissifs sont considérés comme des « restrictions avancées » dans le sens de l'article 10. Si le Programme tel que vous l'avez reçu, ou toute partie de celui-ci, contient une notice établissant qu'il est gouverné par cette Licence en même temps qu'un terme qui est une restriction avancée, vous pouvez ôter ce terme. Si un document de licence contient une restriction avancée mais permet la reconcession de licence ou l'acheminement suivant cette Licence, vous pouvez ajouter un Travail Couvert constituant gouverné par les termes de ce document de licence, pourvu que la restriction avancée ne survit pas à une telle cession de licence ou acheminement.

Si vous ajoutez des termes à un Travail Couvert en accord avec cet article, vous devez placer, dans les fichiers sources appropriés, une déclaration des termes additionnels qui s'appliquent à ces fichiers, ou une notice indiquant où trouver les termes applicables.

Les termes additionnels, qu'ils soient permissifs ou non permissifs, peuvent être établis sous la forme d'une licence écrite séparément, ou établis comme des exceptions ; les obligations ci-dessus s'appliquent dans chacun de ces cas.

Article 8. Terminaison.

Vous ne pouvez ni propager ni modifier un Travail Couvert autrement que suivant les termes de cette Licence. Toute autre tentative de le propager ou le modifier est nulle et terminera automatiquement vos droits selon cette Licence (y compris toute licence de brevet accordée selon le troisième paragraphe de l'article 11).

Cependant, si vous cessez toute violation de cette Licence, alors votre licence depuis un titulaire de Droit d'Auteur ("copyright") est réinstaurée (a) à titre provisoire à moins que et jusqu'à ce que le titulaire de Droit d'Auteur termine finalement et explicitement votre licence, et (b) de façon permanente si le titulaire de Droit d'Auteur ne parvient pas à vous notifier de la violation par quelque moyen raisonnable dans les soixante (60) jours après la cessation.

De plus, votre licence depuis un titulaire particulier de Droit d'Auteur est réinstaurée de façon permanente si ce titulaire vous notifie de la violation par quelque moyen raisonnable, c'est la première fois que vous avez reçu une notification deviolation de cette Licence (pour un travail quelconque) depuis ce titulaire de Droit d'Auteur, et vous résolvez la violation dans les trente (30) jours qui suivent votre réception de la notification.

La terminaison de vos droits suivant cette section ne terminera pas les licences des parties qui ont reçu des copies ou droits de votre part suivant cette Licence. Si vos droits ont été terminés et non réinstaurés de façon permanente, vous n'êtes plus qualifié à recevoir de nouvelles licences pour les mêmes constituants selon l'article 10.

Article 9. Acceptation non requise pour obtenir des copies.

Vous n'êtes pas obligé d'accepter cette licence afin de recevoir ou exécuter une copie du Programme. La propagation asservie d'un Travail Couvert qui se produit simplement en conséquence d'une transmission d'égal-à-égal pour recevoir une copie ne nécessite pas l'acceptation. Cependant, rien d'autre que cette Licence ne vous accorde la permission de propager ou modifier un quelconque Travail Couvert. Ces actions enfreignent le Droit d'Auteur si vous n'acceptez pas cette Licence. Par conséquent, en modifiant ou propageant un Travail Couvert, vous indiquez votre acceptation de cette Licence pour agir ainsi.

Article 10. Cession automatique de Licence aux Destinataires et intermédiaires.

Chaque fois que vous acheminez un Travail Couvert, le destinataire reçoit automatiquement une licence depuis les concédants originaux, pour exécuter, modifier et propager ce travail, suivant les termes de cette Licence. Vous n'êtes pas responsable du renforcement de la conformation des tierces parties avec cette Licence.

Une « transaction d'entité » désigne une transaction qui transfère le contrôle d'une organisation, ou de substantiellement tous ses actifs, ou la subdivision d'une organisation, ou la fusion de plusieurs organisations. Si la propagation d'un Travail Couvert résulte d'une transaction d'entité, chaque partie à cette transaction qui reçoit une copie du travail reçoit aussi les licences pour le travail que le prédecesseur intéressé à cette partie avait ou pourrait donner selon le paragraphe précédent, plus un droit de possession du Source Correspondant de ce travail depuis le prédecesseur intéressé si ce prédecesseur en dispose ou peut l'obtenir par des efforts raisonnables.

Vous ne pouvez imposer aucune restriction avancée dans l'exercice des droits accordés ou affirmés selon cette Licence. Par exemple, vous ne pouvez imposer aucun paiement pour la licence, aucune royaltie, ni aucune autre charge pour l'exercice des droits accordés selon cette Licence ; et vous ne pouvez amorcer aucun litige judiciaire (y compris

une réclamation croisée ou contre-réclamation dans un procès) sur l'allégation qu'une revendication de brevet est enfreinte par la réalisation, l'utilisation, la vente, l'offre de vente, ou l'importation du Programme ou d'une quelconque portion de celui-ci.

Article 11. Brevets.

Un « contributeur » est un titulaire de Droit d'Auteur ("copyright") qui autorise l'utilisation selon cette Licence du Programme ou du travail sur lequel le Programme est basé. Le travail ainsi soumis à licence est appelé la « version contributive » de ce contributeur.

Les « revendications de brevet essentielles » sont toutes les revendications de brevets détenues ou contrôlées par le contributeur, qu'elles soient déjà acquises par lui ou acquises subséquemment, qui pourraient être enfreintes de quelque manière, permises par cette Licence, sur la réalisation, l'utilisation ou la vente de la version contributive de celui-ci. Aux fins de cette définition, le « contrôle » inclue le droit de concéder des sous-licences de brevets d'une manière consistante, nécessaire et suffisante, avec les obligations de cette Licence.

Chaque contributeur vous accorde une licence de brevet non exclusive, mondiale et libre de toute royaltie, selon les revendications de brevet essentielles, pour réaliser, utiliser, vendre, offrir à la vente, importer et autrement exécuter, modifier et propager les contenus de sa version contributive.

Dans les trois paragraphes suivants, une « licence de brevet » désigne tous les accords ou engagements exprimés, quel que soit le nom que vous lui donnez, de ne pas mettre en vigueur un brevet (telle qu'une permission explicite pour mettre en pratique un brevet, ou un accord pour ne pas poursuivre un Destinataire pour cause de violation de brevet). « Accorder » une telle licence de brevet à une partie signifie conclure un tel accord ou engagement à ne pas faire appliquer le brevet à cette partie.

Si vous acheminez un Travail Couvert, dépendant en connaissance d'une licence de brevet, et si le Source Correspondant du travail n'est pas disponible à quiconque copie, sans frais et suivant les termes de cette Licence, à travers un serveur réseau publiquement accessible ou tout autre moyen immédiatement accessible, alors vous devez soit (1) rendre la Source Correspondante ainsi disponible, soit (2) vous engager à vous priver pour vous-même du bénéfice de la licence de brevet pour ce travail particulier, soit (3) vous engager, d'une façon consistante avec les obligations de cette Licence, à étendre la licence de brevet aux Destinataires de ce travail. « Dépendant en connaissance » signifie que vous avez effectivement connaissance que, selon la licence de brevet, votre acheminement du Travail Couvert dans un pays, ou l'utilisation du Travail Couvert par votre Destinataire dans un pays, infreindrait un ou plusieurs brevets identifiables dans ce pays où vous avez des raisons de penser qu'ils sont valides.

Si, conformément à ou en liaison avec une même transaction ou un même arrangement, vous acheminez, ou propagez en procurant un acheminement de, un Travail Couvert et accordez une licence de brevet à l'une des parties recevant le Travail Couvert pour lui permettre d'utiliser, propager, modifier ou acheminer une copie spécifique du Travail Couvert, alors votre accord est automatiquement étendu à tous les Destinataires du Travail Couvert et des travaux basés sur celui-ci.

Une licence de brevet est « discriminatoire » si, dans le champ de sa couverture, elle n'inclut pas un ou plusieurs des droits qui sont spécifiquement accordés selon cette Licence, ou en prohibe l'exercice, ou est conditionnée par le non-exercice d'un ou plusieurs de ces droits. Vous ne pouvez pas acheminer un Travail Couvert si vous êtes partie à un arrangement selon lequel une partie tierce exerçant son activité dans la distribution de logiciels et à laquelle vous effectuez un paiement fondé sur l'étendue de votre activité d'acheminement du travail, et selon lequel la partie tierce accorde, à une quelconque

partie qui recevrait depuis vous le Travail Couvert, une licence de brevet discriminatoire (a) en relation avec les copies du Travail Couvert acheminées par vous (ou les copies réalisées à partir de ces copies), ou (b) avant tout destinée et en relation avec des produits spécifiques ou compilations contenant le Travail Couvert, à moins que vous ayez conclu cet arrangement ou que la licence de brevet ait été accordée avant le 28 mars 2007.

Rien dans cette Licence ne devrait être interprété comme devant exclure ou limiter toute licence implicite ou d'autres moyens de défense à une infraction qui vous seraient autrement disponible selon la loi applicable relative aux brevets.

Article 12. Non abandon de la liberté des autres.

Si des conditions vous sont imposées (que ce soit par décision judiciaire, par un accord ou autrement) qui contredisent les conditions de cette Licence, elles ne vous excusent pas des conditions de cette Licence. Si vous ne pouvez pas acheminer un Travail Couvert de façon à satisfaire simultanément vos obligations suivant cette Licence et toutes autres obligations pertinentes, alors en conséquence vous ne pouvez pas du tout l'acheminer. Par exemple, si vous avez un accord sur des termes qui vous obligent à collecter pour le réacheminement des royalties depuis ceux à qui vous acheminez le Programme, la seule façon qui puisse vous permettre de satisfaire à la fois à ces termes et ceux de cette Licence sera de vous abstenir entièrement d'acheminer le Programme.

Article 13. Utilisation avec la Licence Générale Publique Affero GNU.

Nonobstant toute autre clause de cette Licence, vous avez la permission de lier ou combiner tout Travail Couvert avec un travail placé sous la version 3 de la Licence Générale Publique GNU Affero ("GNU Affero General Public License") en un seul travail combiné, et d'acheminer le travail résultant. Les termes de cette Licence continueront à s'appliquer à la partie formant un Travail Couvert, mais les obligations spéciales de la Licence Générale Publique GNU Affero, article 13, concernant l'interaction à travers un réseau s'appliqueront à la combinaison en tant que telle.

Article 14. Versions révisées de cette License.

La Free Software Foundation peut publier des versions révisées et/ou nouvelles de la Licence Publique Générale GNU ("GNU General Public License") de temps en temps. De telles version nouvelles resteront similaires dans l'esprit avec la présente version, mais peuvent différer dans le détail afin de traiter de nouveaux problèmes ou préoccupations.

Chaque version reçoit un numéro de version distinctif. Si le Programme indique qu'une version spécifique de la Licence Publique Générale GNU « ou toute version ultérieure » ("or any later version") s'applique à celui-ci, vous avez le choix de suivre soit les termes et conditions de cette version numérotée, soit ceux de n'importe quelle version publiée ultérieurement par la Free Software Foundation. Si le Programme n'indique pas une version spécifique de la Licence Publique Générale GNU, vous pouvez choisir l'une quelconque des versions qui ont été publiées par la Free Software Foundation.

Si le Programme spécifie qu'un intermédiaire peut décider quelles versions futures de la Licence Générale Publique GNU peut être utilisée, la déclaration publique d'acceptation d'une version par cet intermédiaire vous autorise à choisir cette version pour le Programme.

Des versions ultérieures de la licence peuvent vous donner des permissions additionnelles ou différentes. Cependant aucune obligation additionnelle n'est imposée à l'un des auteurs ou titulaires de Droit d'Auteur du fait de votre choix de suivre une version ultérieure.

Article 15. Déclaration d'absence de garantie.

IL N'Y A AUCUNE GARANTIE POUR LE PROGRAMME, DANS LES LIMITES PERMISES PAR LA LOI APPLICABLE. À MOINS QUE CELA NE SOIT ÉTABLI DIFFÉREMENT PAR ÉCRIT, LES PROPRIÉTAIRES DE DROITS ET/OU LES AUTRES PARTIES FOURNISSENT LE PROGRAMME « EN L'ÉTAT » SANS GARANTIE D'AUCUNE SORTE, QU'ELLE SOIT EXPRIMÉE OU IMPLICITE, CECI COMPRENNANT, SANS SE LIMITER À CELLES-CI, LES GARANTIES IMPLICITES DE COMMERCIALISABILITÉ ET D'ADÉQUATION À UN OBJECTIF PARTICULIER. VOUS ASSUMEZ LE RISQUE ENTIER CONCERNANT LA QUALITÉ ET LES PERFORMANCES DU PROGRAMME. DANS L'ÉVENTUALITÉ OÙ LE PROGRAMME S'AVÉRERAIT DÉFECTUEUX, VOUS ASSUMEZ LES COÛTS DE TOUS LES SERVICES, RÉPARATIONS OU CORRECTIONS NÉCESSAIRES.

Article 16. Limitation de responsabilité.

EN AUCUNE AUTRE CIRCONSTANCE QUE CELLES REQUISES PAR LA LOI APPLICABLE OU ACCORDÉES PAR ÉCRIT, UN TITULAIRE DE DROITS SUR LE PROGRAMME, OU TOUT AUTRE PARTIE QUI MODIFIE OU ACHEMINE LE PROGRAMME COMME PERMIS CI-DESSUS, NE PEUT ÊTRE TENU POUR RESPONSABLE ENVERS VOUS POUR LES DOMMAGES, INCLUANT TOUT DOMMAGE GÉNÉRAL, SPÉCIAL, ACCIDENTEL OU INDUIT SURVENANT PAR SUITE DE L'UTILISATION OU DE L'INCAPACITÉ D'UTILISER LE PROGRAMME (Y COMPRIS, SANS SE LIMITER À CELLES-CI, LA PERTE DE DONNÉES OU L'INEXACTITUDE DES DONNÉES RETOURNÉES OU LES PERTES SUBIES PAR VOUS OU DES PARTIES TIERCES OU L'INCAPACITÉ DU PROGRAMME À FONCTIONNER AVEC TOUT AUTRE PROGRAMME), MÊME SI UN TEL TITULAIRE OU TOUTE AUTRE PARTIE A ÉTÉ AVISÉ DE LA POSSIBILITÉ DE TELS DOMMAGES.

Article 17. Interprétation des sections 15 et 16.

Si la déclaration d'absence de garantie et la limitation de responsabilité fournies ci-dessus ne peuvent prendre effet localement selon leurs termes, les cours de justice qui les examinent doivent appliquer la législation locale qui approche au plus près possible une levée absolue de toute responsabilité civile liée au Programme, à moins qu'une garantie ou assumption de responsabilité accompagne une copie du Programme en échange d'un paiement.

FIN DES TERMES ET CONDITIONS.

Comment appliquer ces termes à vos nouveaux programmes

Si vous développez un nouveau programme et voulez qu'il soit le plus possible utilisable par le public, la meilleure façon d'y parvenir et d'en faire un logiciel libre que chacun peut redistribuer et changer suivant ces termes-ci.

Pour appliquer ces termes, attachez les notices suivantes au programme. Il est plus sûr de les attacher au début de chacun des fichiers sources afin de transporter de façon la plus effective possible l'exclusion de garantie ; et chaque fichier devrait comporter au moins la ligne de réservation de droit ("copyright") et une indication permettant de savoir où la notice complète peut être trouvée :

<une ligne donnant le nom du programme et une brève idée de ce qu'il fait.>
Copyright (C) <année> <nom de l'auteur> — Tous droits réservés.

Ce programme est un logiciel libre ; vous pouvez le redistribuer ou le modifier suivant les termes de la "GNU General Public License" telle que publiée par la Free Software Foundation : soit la version 3 de cette licence, soit (à votre gré) toute version ultérieure.

Ce programme est distribué dans l'espoir qu'il vous sera utile, mais SANS AUCUNE GARANTIE : sans même la garantie implicite de COMMERCIALISABILITÉ

ni d'ADÉQUATION À UN OBJECTIF PARTICULIER. Consultez la Licence Générale Publique GNU pour plus de détails.

Vous devriez avoir reçu une copie de la Licence Générale Publique GNU avec ce programme ; si ce n'est pas le cas, consultez :
<http://www.gnu.org/licenses/>.

Ajoutez également les informations permettant de vous contacter par courrier électronique ou postal.

Si le programme produit une interaction sur un terminal, faites lui afficher une courte notice comme celle-ci lors de son démarrage en mode interactif :

```
<programme> Copyright (C) <année> <nom de l'auteur>
Ce programme vient SANS ABSOLUMENT AUCUNE GARANTIE ; taper "affiche g" pour
les détails. Ceci est un logiciel libre et vous êtes invité à le redistribuer
suivant certaines conditions ; taper "affiche c" pour les détails.
```

Les commandes hypothétiques "affiche g" and "affiche c" devrait afficher les parties appropriées de la Licence Générale Publique. Bien sûr, les commandes de votre programme peuvent être différentes ; pour une interface graphique, vous pourriez utiliser une « boîte À propos. »

Vous devriez également obtenir de votre employeur (si vous travaillez en tant que programmeur) ou de votre école un « renoncement aux droits de propriété » pour ce programme, si nécessaire. Pour plus d'informations à ce sujet, et comment appliquer la GPL GNU, consultez
<http://www.gnu.org/licenses/>.

La Licence Générale Publique GNU ne permet pas d'incorporer votre programme dans des programmes propriétaires. Si votre programme est une bibliothèque de sous-routines, vous pourriez considérer qu'il serait plus utile de permettre de lier des applications propriétaires avec la bibliothèque. Si c'est ce que vous voulez faire, utilisez la Licence Générale Publique Limitée GNU au lieu de cette Licence ; mais d'abord, veuillez lire <http://www.gnu.org/philosophy/why-not-lgpl.html>.

TABLE DES FIGURES

<i>Figure 1 – Test de TGVWords avec Delphi</i>	16
<i>Figure 2 – Test de TGVWords avec Lazarus (Windows)</i>	17
<i>Figure 3 - Test de TGVLists avec Delphi</i>	26
<i>Figure 4 - Test de TGVLists avec Lazarus</i>	26
<i>Figure 5 - Test des chaînes avec Lazarus</i>	27
<i>Figure 6 - Test des listes de propriétés avec Delphi</i>	32
<i>Figure 7 - Test des listes de propriétés avec Lazarus</i>	33
<i>Figure 8 - Tortue avec cap 90 Tortue 180 Tortue 90 png</i>	33
<i>Figure 9 - Point avec sinus et cosinus</i>	42
<i>Figure 10 - Arc Tangente</i>	43
<i>Figure 11 - Test de la tortue avec Delphi</i>	52
<i>Figure 12 - Test de la tortue avec Lazarus</i>	53
<i>Figure 13 - Ecran principal de EasyTurtle (Windows)</i>	54
<i>Figure 14 - Actions de la tortue (EasyTurtle)</i>	55
<i>Figure 15 - Couleurs et formes (EasyTurtle)</i>	56
<i>Figure 16 - Ordres (EasyTurtle)</i>	56
<i>Figure 17 - Aide (EasyTurtle)</i>	57
<i>Figure 18 - A propos (EasyTurtle)</i>	57
<i>Figure 19 - Outils (EasyTurtle)</i>	58
<i>Figure 20 - Barre de statut (EasyTurtle)</i>	59
<i>Figure 21 - Répertoire de EasyTurtle.</i>	59
<i>Figure 22 - Appel d'une autre fiche (EasyTurtle).</i>	60
<i>Figure 23 - Mémorisation des ordres (EasyTurtle).</i>	60
<i>Figure 24 - La tortue avance (EasyTurtle).</i>	60
<i>Figure 25 - Mise à jour (EasyTurtle).</i>	61
<i>Figure 26 - Rejouer une séquence (EasyTurtle).</i>	61
<i>Figure 27 - Annuler la dernière action (EasyTurtle).</i>	62
<i>Figure 28 - Sauvegarde (EasyTurtle).</i>	63
<i>Figure 29 - Chargement d'un fichier (EasyTurtle).</i>	64
<i>Figure 30 - Demande de fermeture (EasyTurtle).</i>	65
<i>Figure 31 - Notion de pile LIFO</i>	66
<i>Figure 32 - Notion de queue</i>	69
<i>Figure 33 - Test des piles avec Delphi</i>	72
<i>Figure 34 - Test des piles avec Lazarus (Windows)</i>	73
<i>Figure 35 - Test des piles avec Lazarus (Linux)</i>	73