# RAPPORT DE PROJET

Projet de trimestre 1

**GtkSpider**

Jérémy **Berthet** & Reynald **Borer**

*EIVD*, 5 août 2005

# Table des matières

# 1. Introduction

Dans le cadre de nos cours à l'École d'Ingénieurs du canton de Vaud (EIVD), orientation Informatique Logiciel 1ère année, nous avons un projet de trimestre à effectuer. Ce projet doit être fait par groupe de 2 personnes, et le but est premièrement de se rendre compte de ce que le développement en petit groupe implique, mais aussi comment planifier au mieux le développement pour respecter le cahier des charges et les délais fixés.

Dans le cadre de ce projet, nous avons décidé de développer un clone de la libraire graphique *Spider*. Le but de ce « clone » est simple: pouvoir utiliser les fonctionnalités de *Spider* sur un autre environnement que Microsoft Windows™.

Dans la suite de ce rapport nous allons tout d'abord rappeler le cahier des charges du projet, puis nous résumerons le travail effectué et nous développerons plus en détails les choix techniques concernant le développement. En annexes, ce document comprend la planification ainsi que le journal de travail, la documentation utilisateur et la documentation technique du projet, ainsi que le listage complet du code source.

# 2. Cahier des charges

But:
Le but de ce projet est de créer un paquetage similaire à *Spider* mais qui soit multi-plate forme. Spider est en effet utilisé dans plusieurs cours, et le fait qu'il ne fonctionne que sous Microsoft Windows™ pose des problèmes à certains étudiants qui n'utilisent pas ce système d'exploitation. Nous désirons donc créer un paquetage le plus proche possible de *Spider* pour faciliter la migration.

Détails techniques:

- Nous pensons utiliser la librairie *GtkAda* pour développer ce paquetage. En effet, celle-ci fonctionne sur diverse plate formes et permet entre autre d'afficher des fenêtres graphiques.
- Le paquetage devra mettre à disposition des procédures et fonctions le plus proche possible de celles fournies par le paquetage *Spider*. Dans l'idéal, les spécifications seraient totalement identiques, mais il est envisageable qu'au vu de l'architecture de *GtkAda* cela ne soit pas possible. Après une première évaluation, nous pensons devoir ajouter une procédure permettant de garder la fenêtre de dessin ouverte, ce qui n'est pas fait automatiquement avec *GtkAda*.
- Dans un premier temps nous nous focaliserons sur la gestion basique de la fenêtre graphique, en codant les procédures des paquetage *Spider* et *Spider-Draw*. Si nous avons assez de temps, nous coderons ensuite les procédures contenues dans *Spider-User*, et nous évaluerons la possibilité de coder un paquetage pour faire des menus et des entrées utilisateur.

Délais:

- D'ici au **24 mai**, nous devrons avoir étudié les possibilités graphiques de *GtkAda* et déterminé les procédures qui devront être modifiées par rapport à la spécification de Spider.
- Le délai pour rendre le code ainsi que le rapport est fixé au **5 août.**

Membres: Le groupe est constitué des personnes suivantes:

- Reynald Borer
- Jérémy Berthet

Licence:
Dans la mesure du possible, et si la politique de l'école le permet, nous aimerions distribuer notre travail sous la licence libre GPL version 2. En effet, nous pensons que ce projet pourrais intéresser beaucoup de personnes, autant dans l'école qu'à l'extérieur, et nous aimerions donc faciliter la suite du développement du paquetage.

# 3. Résumé

Dans le cadre des cours de programmation de l'EIVD, les étudiants sont amenés à utiliser une librairie, *Spider*, qui permet d'afficher des fenêtres graphiques et qui est très simple d'utilisation. Néanmoins, cette librairie ne fonctionne que sous le système d'exploitation Microsoft Windows™, ce qui peut poser problème à certains étudiants n'utilisant pas cette plateforme.

Le but de ce projet est donc de fournir une librairie graphique équivalente à *Spider*, mettant à disposition exactement les même fonctionnalités, mais étant multi-plate-forme.

A cet effet, nous avons décidé d'utiliser la librairie **GtkAda**, qui fournis des fonctions d'accès à la librairie *Gtk+*. *Gtk+* est un ensemble de librairies qui permettent de créer des interfaces graphiques, mais aussi d'accéder à des fonctions de bas-niveau tel que des aires de dessin. L'avantage de cette ensemble de librairies est qu'il fonctionne sur plusieurs plate-formes, dont entre autre Microsoft Windows™ et Linux, ce qui est un point essentiel pour notre projet.

La conformité de notre paquetage avec la librairie *Spider* est totale. En effet, tous les sous-programmes respectent exactement la même syntaxe, et fournissent les même fonctionnalités. Nous n'avons pas réussi à implémenter 3 fonctions, mais un rapide sondage nous à permis de nous rendre compte que ces fonctions étaient finalement très peu utilisées, ce qui n'est pas trop dommageable à notre projet.

# 4. Documentation de développement

## *4.1. Introduction*

De nos jours, le développeur de logiciels doit impérativement savoir compter avec un choix multiple de plate-formes informatiques. Tant au niveau matériel que logiciel, les entreprises sont toutes équipées différemment et l'inter-opérabilité entre les programmes devient un paramètre essentiel. La possibilité de développer un logiciel qui puisse tourner sur plusieurs plate-formes est donc un plus pour le développeur, et cela se passe souvent par l'intermédiaire de librairies qui supportent plusieurs de ces plate-formes.

Dans le cadre de la formation à l'EIVD, les étudiants en informatique logiciel se voient proposer une libraire graphique simple (*Spider*) pour la réalisation d'applications exploitant du graphisme. Le but de cette librairie est de pouvoir représenter graphiquement des objets sans avoir besoin de passer beaucoup de temps à comprendre le fonctionnement de cette librairie. Elle reste donc très basique, permettant simplement de dessiner des traits, des points, des cercles, etc.

Le problème qui à été constaté avec cette librairie est qu'elle ne fonctionne que sous le système d'exploitation Microsoft Windows™. Cela contraint donc les étudiants à utiliser et développer

sous cette plate-forme. Certains étudiants, dont nous faisons partie, utilisent une autre plate-forme et se sont donc trouvé dans l'impossibilité d'utiliser cette librairie.

C'est dans le but de résoudre ce problème que le projet *GtkSpider* a été lancé. Son but est de fournir les même fonctionnalités que la librairie Spider, mais en utilisant une librairie multi-plate-forme. Une fois ce dernier abouti, chaque étudiant sera théoriquement libre d'utiliser la plate-forme de son choix pour créer des logiciels graphiques en Ada durant sa première année de formation à l'EIVD.

## *4.2. Analyse*

### 4.2.1. Fonctionnement de *Spider*

Le fonctionnement du paquetage *Spider* pour Ada s'est trouvé être relativement simple. En effet, ce paquetage fait appel à une librairie système, elle aussi appelée *Spider*, donc le code du paquetage ne contient que des appels à cette librairie. Il apparaît donc que cette librairie est développée pour plusieurs langages de programmation, dont entre autre le langage C.

Nous avons pu obtenir le code source de cette librairie système qui est développée en langage C. Elle effectue directement des appels au système d'exploitation Microsoft Windows™, c'est pourquoi elle est difficilement portable.

La meilleure solution pour écrire le paquetage Ada aurait été de créer pour chaque système d'exploitation voulu une librairie système mettant à disposition exactement les même fonctionnalités que la librairie système *Spider*. Cela nous aurais permis de réutiliser directement le paquetage Ada *Spider* sans avoir besoin de le modifier.

Néanmoins cette solution n'était pas envisageable pour notre projet. En effet, il s'agit d'un projet uniquement développé en Ada. De plus, nous n'avons à l'heure actuelle pas les connaissances nécessaires pour développer une telle librairie système. C'est pourquoi nous avons orienté notre choix vers **GtkAda**.

### 4.2.2. Choix de *GtkAda*

La librairie **GtkAda** est en fait un ensemble de paquetages permettant d'utiliser les fonctionnalités des librairies *Gtk+*. Ces librairies permettent d'afficher toute sorte d'éléments à l'écran, que cela soit des fenêtres, des menus, des boutons ou des aires de dessin. L'avantage de ces librairies est qu'elles fonctionnent sur plusieurs plate-formes, entre autre Unix, Microsoft Windows™, Mac OS X, etc.

Les librairies *Gtk+* sont très évoluées et se révèlent parfois assez complexe à maîtriser et utiliser. C'est une des raisons qui font qu'elles ne sont en elle-même pas adaptées au développement de petites applications dans leur version Ada 95.

Mais au vu du caractère multi-plate-forme de ces librairies, **GtkAda** devient ainsi la librairie idéale pour développer un clone de *Spider* sans avoir à se soucier de la plate-forme sur laquelle le programme tourne. Ce travail est assuré par **GtkAda** et *Gtk+.*

### 4.2.3. Choix effectués

Peu de choix ont été effectués, la plupart nous étant imposés pour garder la compatibilité avec la librairie *Spider*.

### 4.2.4. État actuel du travail

La librairie *Spider* a été complètement clonée en s'appuyant sur *GtkAda*. Un des enfants les plus importants, *Spider.Draw,* est lui aussi cloné à environ 90%. Trois sous programmes n'ont malheureusement pas pu être implémenté dans ce dernier.

Deux d'entres eux concernent le traitement du texte, elles permettent d'obtenir la hauteur et la largeur maximale des caractères pour la police utilisée. Le troisième permet d'obtenir la couleur d'un pixel à un endroit donné de la fenêtre de dessin.

Des explications sont fournies dans l'annexe de documentation technique quand à ces problèmes.

## *4.3. Méthode de travail*

Notre premier besoin a été de maîtriser au mieux l'outil principal qui permet de réaliser notre projet. **GtkAda** est un ensemble important de librairies dans lesquelles il devient très vite difficile de s'y retrouver.

Bien que l'ayant déjà utilisé dans un autre projet de développement pour concevoir une interface graphique, notre expérience ne nous à été que peu utile dans ce projet. En effet, il s'agit d'utiliser les procédures d'un niveau inférieur permettant de dessiner à proprement dit, tandis que les procédures que nous connaissions déjà nous permettaient de mettre en place des interfaces graphiques (boutons, menu, etc.).

Nous nous sommes donc en premier lieu fixé un objectif qui nous permettrait de déclarer "Maintenant nous comprenons **GtkAda** dans le sens de nos besoins et nous pouvons réaliser pleinement notre projet". L'objectif fixé à cet effet était simplement de pouvoir afficher une fenêtre contenant une ligne noire.

Nous sommes parti d'un exemple de code **GtkAda** qui permettait de dessiner un petit cercle lorsque l'utilisateur cliquait avec sa souris dans la fenêtre. Nous avons essayé à partir de cet exemple de comprendre le fonctionnement et nous avons effectué plusieurs essais infructueux.

La documentation de **GtkAda** ne nous a pas été d'un grand secours. Néanmoins, après quelques e-mails échangés sur la liste de diffusion de la librairie et la consultation d'autres exemples, nous avons finalement compris le fonctionnement des divers éléments tels que l'aire de dessin, la fenêtre *Gdk* et la matrice de dessin. Nous avons finalement pu écrire un programme de test que nous avons transformé en paquetage par la suite.

Une fois cette étape atteinte, nous avons remis les spécifications de *Spider* sous nos yeux et nous nous sommes partagé le travail pour reproduire au mieux les sous programmes présent dans la dite spécification. Nous avons tout d'abord commencé par le paquetage parent *Gtkspider*, puis une fois celui-ci terminé et intensivement testé, nous nous sommes attaqué au paquetage *Gtkspider.Draw*.

## *4.4. Critique des outils utilisés*

Vous trouverez ici une courte critique des outils externe que nous avons utilisé pendant notre projet.

### 4.4.1. GtkAda

Sans cette librairie, il nous aurait été impossible de mener à terme notre projet. C'est en effet la seule librairie que nous connaissons qui permet d'effectuer du graphisme et qui fonctionne sur plusieurs plate-formes informatiques. Néanmoins, si nous avons une critique à faire, c'est bien concernant le manque de documentation. Il n'est en effet pas rare de trouver des paquetages qui ne sont pas documentés, surtout dans notre cas. Nous avons en effet besoin de fonctions d'assez bas niveau par rapport à *Gtk*, et nous avons découvert que certains paquetages que nous utilisons ne sont même pas indiqué dans la documentation. Cela nous à forcé à avancer par essai dans le code, et à demander de l'aide sur la liste de diffusion du projet.

Néanmoins les développeurs de cette librairie font un travail exceptionnel, car au vu de la complexité de la librairie *Gtk+* nous nous sommes rendu compte que tout ce dont nous avions besoin pour effectuer notre projet était implémenté dans cette librairie.

### 4.4.2. CVS

Afin de faciliter le développement en groupe mais aussi de garder une trace de toutes les modifications, nous avons décidé de mettre en place un serveur CVS (*Concurrent Versions System*). CVS est un système centralisé de gestion de fichiers, qui garde trace de toutes les modifications effectuées. Ce serveur nous a permis d'éviter de devoir nous envoyer tout le temps les fichiers par e-mail ou par un autre moyen, et éviter de devoir demander à son collègue si il a modifié quelque chose avant de commencer à travailler.

## *4.5. Performance de GtkSpider*

Afin de comparer les performances entre *Spider* et *GtkSpider*, nous avons utilisé le programme "Rayons". Pour rappel, ce programme permet de dessiner un nombre donné de rayons dans un cercle dont on connaît la longueur du rayon et la position du centre.

Le test consiste à faire dessiner un très grand nombre de rayon au programme (10'000 en l'occurrence) et de chronométrer le temps que mettent les deux versions pour terminer le processus.

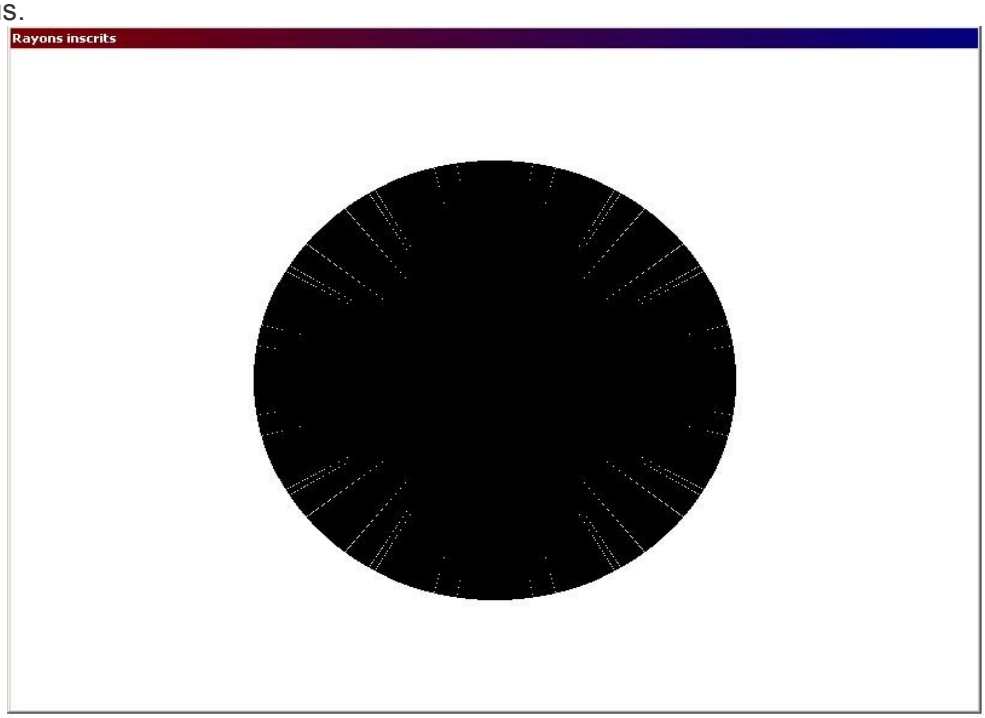


*Illustration 1: Exemple d'exécution du programme "Rayons" avec 10'000 rayons*

Bien que la méthode de mesure ne soit pas des plus fiable en général, la différence est telle qu'il n'y a pas à s'inquiéter des erreurs de chronométrage.

Il a fallut environ 12 secondes à la version *GtkSpider* du programme pour exécuter la tâche, alors qu'il a fallut moins d'une seule seconde à la version *Spider.*

La seule explication que nous ayons pu trouver à cette immense différence se situe au niveau du fonctionnement de **GtkAda.** Effectivement, comme expliqué dans la documentation technique, nous dessinons d'abord les éléments dans une "Matrice de dessin" (« pixmap ») avant d'en faire le rendu à l'écran. Mais, ce rendu ne peut s'effectuer que sur des zones rectangulaires, ce qui signifie qu'à chaque rayons dessiné, il y a beaucoup plus de point qui sont re-dessiné inutilement.

Un moyen d'empêcher cela sera de laisser la liberté à l'utilisateur d'effectuer le rendu à l'écran quand bon lui semble, mais cela modifierait le fonctionnement du paquetage qui ne serait ainsi plus suffisamment proche de *Spider* et donc hors du cadre du projet.

## *4.6. Conclusion*

Nous estimons que le but de ce projet à été atteint. En effet, les étudiants peuvent maintenant avoir à disposition une librairie graphique simple multi-plate-forme. La simplicité d'approche de cette librairie fait qu'elle est très simple d'accès, c'est pourquoi il nous semblait important de ne pas la limiter à une seule plate-forme.

Ce projet nous a permis de mieux comprendre le fonctionnement de *Gtk+* ainsi que **GtkAda**, et nous espérons que cela nous sera utile pour la suite de nos études et de nos carrières.

Néanmoins ce paquetage est loin d'être terminé. En effet, il manque certains fonctionnalités du paquetage original *Spider*. De même, nous pensons qu'il serait intéressant d'effectuer un sondage des utilisateurs afin de savoir si certaines fonctionnalités pourraient être rajoutées.

Les différences entre les performances étant aussi importante, il serait intéressant de ré-écrire cette librairie en se passant *Gtk+* et en faisant directement des appels au système graphique du système d'exploitation. C'est comme cela que fonctionne *Spider* sous Microsoft Windows™, mais cela enlève tout de suite le caractère multi-plate-forme de la librairie. De plus, elle devrait être écrite en langage C, ce qui sort du cadre de ce projet.

Berthet Jérémy & Borer Reynald

## *4.7. Bibliographie / Webographie*

[1] GtkAda Reference Manual version 2.4
http://libre.act-europe.fr/GtkAda/docs/gtkada_rm_toc.html

[2] Gtk+ Reference Manual
http://developer.gnome.org/doc/API/2.0/gtk/

[3] Gdk Reference Manual
http://developer.gnome.org/doc/API/2.0/gdk/index.html

[4] Pango Reference Manual
http://developer.gnome.org/doc/API/2.0/pango/index.html

[5] Glib Reference Manual
http://developer.gnome.org/doc/API/2.0/glib/index.html

# Annexe A. Planification du projet

La planification du projet s'est révélée très simple. Nous savions dès le départ comment les paquetages allaient être découpés, nous avons donc juste fixé quelques dates pour lesquels nous avions des objectifs clairs.

La première étape fut de faire valider le projet et son cahier des charges. Pour cela, les 2 premiers cours étaient prévu (**10** et **12 mai 2005**).

Une fois le projet validé, une première date butoir était fixée par les professeurs. En date du **24 mai** nous devions présenter l'approche du travail choisie, qui devait être validée par les professeurs.

Nous avons ensuite fixé le 26 mai pour avoir compris et appréhendé le fonctionnement de la librairie **GtkAda**, suite de quoi nous devions nous lancer dans le développement du paquetage principal *Spider* et de son enfant *Spider.Draw*. Nous avons décidé de ne développer le paquetage *Spider.User* que si nous avions assez de temps, ce qui ne fut pas le cas.

A la fin des cours (le 17 juin), nous nous sommes rendu compte que nous n'avions pas encore développé toutes les procédures du paquetage. Nous avons alors continué pendant les vacances d'été avec comme date butoir la fin du mois de juillet. Parallèlement à cela nous avons aussi décidé d'écrire la documentation, que nous n'avions jusqu'à présent pas fait.

La date de rendu du projet était quand à elle fixée au 5 août au plus tard.

# Annexe B. Journal de travail

10.05.2005: Premier contact.
Présentation des sujets.
Offre au client pour notre sujet .
Préparation de l'environnement de développement (installation de GtkAda sur les portables, etc ...).

12.05.2005: Compréhension des concepts de base de l'environnement *Gtk+*.
Premières réflexions sur le design du paquetage de base *GtkSpider* équivalent à *Spider*.
Mise en place d'un serveur CVS pour faciliter le développement en commun.

17.05.2005: Essais de code pour la gestion avancée d'une fenêtre (qu'on ne puisse pas changer ses dimensions).
Essais de code pour tracer une ligne sur un canevas. Le canevas a pu être affiché avec son pixmap derrière, mais pas encore la ligne.

19.05.2005: Affichage d'une ligne réussie lors du chargement de la fenêtre. Orientation vers l'idée de créer nos propres événements pour l'interfaçage sur notre librairie Spider.
Recherche diverses et améliorations des connaissances de GtkAda.

24.05.2005: Divers essais pour afficher une ligne dynamiquement (pas uniquement au chargement de la fenêtre) soldé par des échecs.
Décision de demander un peu d'aide via la mailing-list de GtkAda.

26.05.2005: Abandon de la tentative de créer nos événements. Découverte des problématiques liées à la queue d'événements.

31.05.2005: Premier programme dessinant une ligne (ou n'importe quelle forme paramétrée) ailleurs que dans le chargement de celui-ci. Il est enfin possible de penser le codage d'un paquetage.

02.06.2005: L'application de ce qui a été utilisé dans un programme ne semble pas fonctionner correctement dans un paquetage. Décision de refaire une recherche de documentation sur internet.
Déblocage de la situation. Après divers essais, il apparaît que ce n'étais pas la partie événements qui posait problème, mais de mauvaises coordonnées lors de l'affichage de la matrice de dessin dans la fenêtre (source d'erreur dans la documentation **GtkAda**). Le paquetage Spider a été porté avec succès pour **GtkAda** aujourd'hui même. Il reste encore des tests routinier et du nettoyage de code pour fournir le paquetage *Gtkspider* de base complet et en ordre.

07.06.2005: Les fonctions de dessins traites les événements Gtk en attente à chaque appel ceci afin de re-dessiner les zones masquées de la fenêtre. Elles lèvent aussi l'exception `Fenetre_Non_Init` si l'utilisateur essaie de dessiner sans avoir initialisé la fenêtre.
Correction d'un bug dans le dessin des lignes si l'utilisateur donne un point de départ plus grand que le point d'arrivée.

09.06.2005: Certaines variables utilisées dans le corps de *Gtkspider* sont mise dans la partie privée de la spécification afin de pouvoir les utiliser dans les paquetages enfants.
Création de 2 procédures interne à *Gtkspider.Draw* pour convertir les couleurs du type `tColor` en `Guint16` utilisé par **GtkAda**.
Quelques tests pour la définition des couleurs.

14.06.2005: D'autres variables dans la partie privée du parent.
La définition des couleurs fonctionnent, de même que leur lecture.
Codage + test de la procédure permettant de dessiner un rectangle.

16.06.2005: Le paquetage Spider fonctionne presque correctement, il y a juste un message d'erreur avec la procédure `Close_Window`.
Correction d'un bug lors de la conversion des couleurs dans le paquetage.
La procédure permettant le dessin du cercle est codée, de même que la procédure `Put_Pixel`.
Trouvé une fonction pour afficher du texte. Est déclarée obsolète dans la documentation Gtk+ et affiche des messages d'erreurs dans la console.
nous continuons les recherches pour les procédures pour le texte ainsi que `Get_Pixel`.

Pour le `Display_Text` Reynald regarde du côté de *Pango*, mais les paquetages **GtkAda** permettant d'interagir avec *Pango* ne sont pas bien documentés (*Pango* est une librairie de *Gtk+* qui permet de gérer tous les textes possibles).

Décision de ne pas implémenter le paquetage *Spider.User* par manque de temps, mais aussi par manque de connaissances et de documentation sur le sujet.

25.07.05: Afin d'afficher du texte, il nous faut un pointeur sur un article étendant l'article de base d'une fenêtre. Regroupement de toutes les variables définissant des éléments de la fenêtre dans un article, et utilisation d'une variable pointeur sur ce type article dans le reste du code.

29.07.05: La procédure `Display_Text` du paquetage *Gtkspider.Draw* fonctionne, de même que la procédure `Close_Window` du paquetage parent *Gtkspider.*
Pour la procédure Get_Pixel (qui indique la couleur à la position courante), nous n'avons pas trouvé de solution avec ce qui est disponible dans **GtkAda**. Une idée possible aurait été d'utiliser une variable tableau de la taille de la fenêtre de dessin et comportant des champs pour la couleur, mais cela nous pose problème lorsque nous dessinons des lignes en diagonales car nous n'avons aucun moyen de savoir quels pixels seront coloriés).

# ANNEXE C

# DOCUMENTATION UTILISATEUR

Projet de trimestre 1

**GtkSpider**

Jérémy **Berthet** & Reynald **Borer**
*EIVD*, 5 août 2005

# Table des matières

# 1. Informations générales

*GtkSpider* est un paquetage permettant la création de petite applications en ADA nécessitant un rendu "graphique". S'appuyant sur les librairies **GtkAda**, il a l'avantage de fonctionner de manière similaire sur une grande majorité des systèmes d'exploitation existants.

Ce document vous offre un aperçu de l'utilisation de *GtkSpider* par des exemples et des explications généralisées. Si vous comptez étendre les fonctionnalités offertes ou garder sous la main un référencement de l'ensemble des sous-programmes, vous pouvez exploiter la documentation technique qui est un excellent complément à ce document.

## *1.1. Configuration requise pour l'utilisation*

La configuration requise pour l'utilisation du paquetage nécessite les éléments suivants:
- Compilateur Ada (le paquetage n'a été testé qu'avec le compilateur GNAT)
- GtkAda installé et correctement configuré (site web de **GtkAda**: voir page 24)

Les paquetages *GtkSpider* peuvent soit être copié dans le répertoire courant, soit installés dans un répertoire séparé. Pour la deuxième solution, il faut ajouter un paramètre au compilateur GNAT afin qu'il trouve les paquetages:

```
gnatmake mon_fichier.adb -I/chemin/complet/vers/gtkspider/
```

## *1.2. Version courante*

Les développeurs de *GtkSpider* n'ayant pas l'intention de poursuivre dans l'immédiat le développement du paquetage, ce dernier est à considérer comme **version finale 1.0**.

Ils restent néanmoins ouverts à toute proposition de reprise du projet et d'intégration notamment d'un clone de *Spider.User*.

# 2. Mode d'emploi

## *2.1. Architecture générale du paquetage*

*GtkSpider* se compose d'un paquetage ADA parent (g*tkspider)* et d'un enfant (g*tkspider.draw).*

Le premier offre les fonctionnalités nécessaires à la création de la fenêtre de dessin, au positionnement du pointeur de dessin ainsi qu'au dessin de lignes. De plus, il met à disposition une procédure particulière nécessaire à **GtkAda**.

Le second, comme un paquetage enfant se doit de le faire, étend les fonctionnalités du premier. Il offre les procédures permettant de dessiner directement des formes géométriques, il permet la gestion des couleurs ainsi que l'écriture de texte dans la fenêtre de dessin.

## *2.2. Utilisation des fonctionnalités*

### 2.2.1. Migration depuis *Spider*

Si vous désirez simplement convertir un programme utilisant *Spider* en un programme utilisant *GtkSpider*, il faut en premier lieu modifier les clauses de contextes pour avoir :

```
with GtkSpider;
with GtkSpider.Draw    -- Seulement au besoin
```

Ensuite, il faut ajouter, pour une question de confort principalement, des appels à la procédure "`Boucle_Dessin`". Effectivement, un problème posé par l'utilisation de **GtkAda** est le fait que le dessin affiché n'est pas rafraîchit automatiquement. Ainsi, lors de l'attente d'une entrée de l'utilisateur, par exemple, il se peut que le dit utilisateur glisse sa fenêtre de console par dessus la fenêtre de dessin. Lorsque il retire sa console, la partie de la fenêtre de dessin qui était masquée ne sera pas ré-affichée sans un appel à "`Boucle_Dessin`" ou un appel à une procédure dessinant quelque chose dans la zone concernée.

Il est conseillé de se rapporter aux exemples ci-dessous pour mieux saisir l'utilisation de la procédure "`Boucle_Dessin`". Un exemple de ce genre de problèmes est sur l'image ci-dessous. Notez la zone grise en bas sur la gauche, qui correspond à la zone ou une fenêtre à été mise en avant-plan:

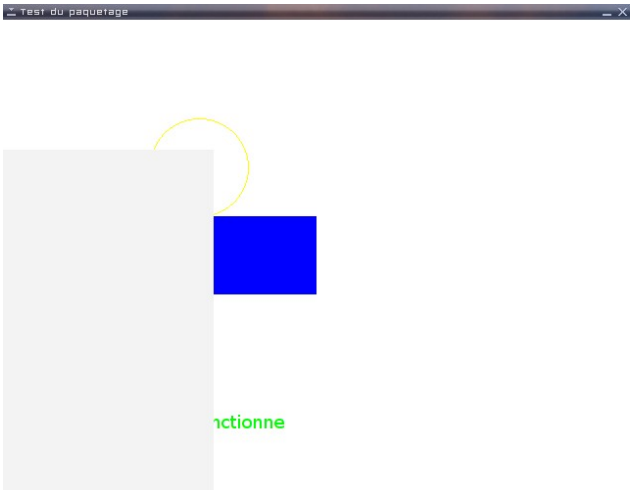


*Illustration 1: Problème du rafraîchissement de l'affichage*

### 2.2.2. Première application

Si vous n'avez jamais utilisé *Spider* (ou une autre librairie graphique) auparavant, alors il faut d'abord comprendre comment se compose la fenêtre de dessin.

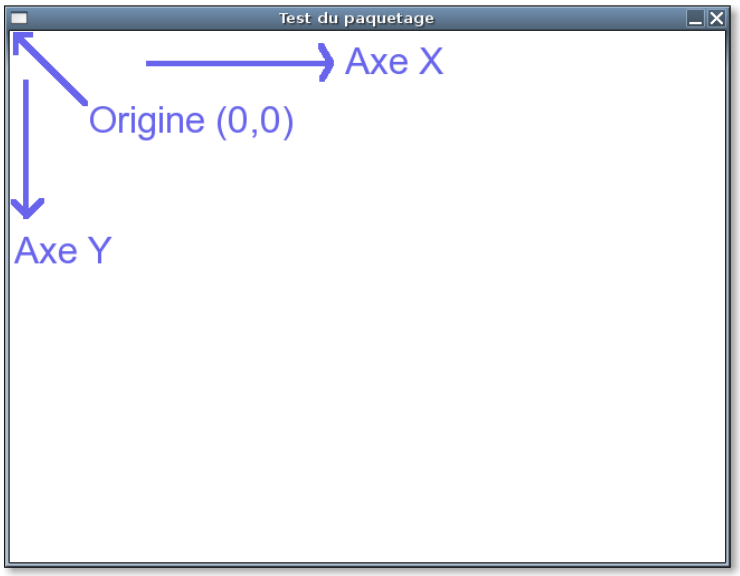


*Illustration 2: La fenêtre de dessin*

Comme on peut le constater sur la figure, l'origine se situe en haut à gauche de la fenêtre de dessin. Cela a pour effet direct d'inverser la direction de l'axe des ordonnées par rapport au système d'axe cartésien auquel la plupart des gens sont habitués.

Il faut aussi noter qu'il n'y a qu'un seul cadran de représenté et que les coordonnées absolues ne sont jamais négatives. Des coordonnées relatives, signifiant un déplacement, pourront elles être négatives.

A tout moment durant l'exécution du programme, il existe un point de repère que nous nommerons "pointeur". Celui-ci est situé au point (0, 0) à la création de la fenêtre de dessin et peut être déplacé à l'aide des procédures `Move_To(Coordonnées absolues)` et `Move(Coordonnées relatives)`.

Afin de pouvoir utiliser ces procédures ainsi que les procédures qui vous permettront de dessiner, il faut en premier lieu créer la fenêtre de dessin. Ceci se fait grâce à la procédure `Init_Window("Titre de la fenêtre")` qui créera automatiquement une fenêtre de 800x600 pixels. Si cette taille ne vous convient pas, il faudra alors utiliser `Init_Size_Window()` qui prend comme paramètres supplémentaires la taille désirée.

### 2.2.3. Dessiner et colorier

Le paquetage de base *GtkSpider* ne met à disposition que deux procédures de dessin qui sont `Line_To()` et `Line()`. La première permet de dessiner une ligne en spécifiant les coordonnées de départ et les coordonnées d'arrivée. La seconde se base sur le "pointeur" pour les coordonnées de départ et demande le déplacement relatif en paramètre.

Le paquetage enfant *GtkSpider.Draw* offre trois procédures supplémentaires permettant de dessiner des formes rectangulaires (`Box()`), circulaires (`Circle()`) ou simplement un unique point (`Put_Pixel()`).

Il est possible de spécifier la couleur dans laquelle on désire dessiner à l'aider de la procédure `Set_Color_Pen()`. Des constantes ont été déclarées pour des couleurs fréquemment utilisées, sinon vous pouvez spécifier un code *RGB* à l'aide d'une variable de type `tColor`.

### 2.2.4. Ajout de texte

*GtkSpider.Draw* permet aussi d'écrire du texte directement dans la fenêtre de dessin à l'aide de la procédure `Display_Text()`. Cette dernière, très basique, n'offre aucune possibilité de paramétrage de la police ou de la taille qui peut varier suivant le système d'exploitation. A utiliser avec précaution donc.

Au même titre que pour le dessin, il est possible de spécifier la couleur du texte désiré à l'aide de `Set_Color_Text()`.

### 2.2.5. Notes

Même si vous n'avez jamais utilisé *Spider*, il est recommandé de lire maintenant le chapitre 2.2.1.

Des exemples d'utilisation sont disponibles au chapitre 3 et l'utilisation de chaque procédure est décrite dans son en-tête et dans la documentation technique.

## *2.3. Les exceptions*

Il y a trois exceptions définies dans le paquetage *GtkSpider.*

*Fenetre_Deja_Init* : Dans un programme, il ne peut y avoir qu'une fenêtre de dessin et cette dernière ne peut être modifiée en cours d'exécution. Cette exception est levée lorsque l'utilisateur fait appel aux procédures `Init_Window()` ou `Init_Size_Window()` alors qu'une d'entre elle a déjà été appelée auparavant.

*Fenetre_Non_Init* : Il est nécessaire d'avoir une fenêtre de dessin avant de vouloir dessiner. Cette exception est levée si l'utilisateur fait appel à une procédure de dessin avant un appel à `Init_Window()` ou `Init_Size_Window()`.

*Erreur_Inconnue* : Comme son nom l'indique, cette exception est levée lorsqu'il n'est pas possible de déterminer la cause exacte d'une erreur d'exécution. Généralement, il s'agit d'exceptions qui ont été levées au sein de sous-programmes propres à **GtkAda**. Une telle erreur peut survenir s'il y a tentative de dépassement de l'air de dessin ou encore suite à l'entrée de codes de couleurs incorrects.

# 3. Exemples complets

## *3.1. demo1.adb*

### 3.1.1. Introduction

Ce programme basique permet de tester la très grande majorité des fonctionnalités de *Gtkspider*.

Il donne un exemple en complément avec le chapitre 2.2 et montre une proposition d'utilisation de la procédure "`Boucle_Event`".

### *3.1.2. Code source*

```
-- ------------------------------------------------------------------------------
-- Fichier       : demo1.adb
-- Auteur        : Berthet Jérémy & Borer Reynald
-- Date          : 04.08.2005
--
-- But           : Faire un test de l'ensemble des fonctions offertes par
--                 gtkspider
--
-- Remarque(s)   : Utilise GtkSpider
--
-- Modifications : Date / Auteur / Raison
--
-- Compilateur   : GNAT 3.15p
-- Licence       : Gnu General Public License v2
-- ------------------------------------------------------------------------------

with Ada.Text_IO;         use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;

with GtkSpider;           use GtkSpider;
with GtkSpider.Draw;      use GtkSpider.Draw;

procedure Demo1 is

   -- Pour la saisie utilisateur
   Touche : Boolean := False;
   Caractere : Character;

begin -- Demo1

   --Ouverture de la fenêtre graphique
   Init_Size_Window ( "Test du paquetage", 640, 480 );

   Put_Line ( "Debut de la demonstration ..." );
   New_Line ( 2 );

   Put_Line ( "Taille de la fenetre: " );
   Put ( "- Horizontale: " );
   Put ( Get_Max_X, 0 );
   New_Line;
   Put ( "Verticale: " );
   Put ( Get_Max_Y, 0 );
   New_Line ( 2 );

   Put_Line ( "Deplacement du curseur en (10, 10)" );
   Put_Line( "Dessin d'une ligne sur un deplacement de (50, 150)" );

   --Déplacement du curseur
   Move_To(10, 10);
```

```
--Ligne de longueur 50 sur X et 150 sur Y
Line(50, 150);

Put_Line("Pressez ENTER pour continuer");
-- boucle pour traiter les événements en attendant une touche clavier
while not Touche loop
   Get_Immediate ( Caractere, Touche );
   Boucle_Dessin;
end loop;
--Supprime le retour à la ligne
Get_Immediate ( Caractere, Touche );
New_Line;

Put_Line("Dessin d'une ligne rouge  de (640, 480) vers (120, 320)");
Put_Line("Deplacement du curseur de (100, 100). Dessin d'un pixel rouge");

-- Changement de la couleur du trait
Set_Color_Pen(Red);

-- Dessin d'une ligne entre 2 points
Line_To(640, 480, 120, 320);

-- Déplacement du curseur
Move(100, 100);

-- Dessin d'un pixel
Put_Pixel;

Put_Line("Pressez ENTER pour continuer");
-- boucle pour traiter les événements en attendant une touche clavier
while not Touche loop
   Get_Immediate ( Caractere, Touche );
   Boucle_Dessin;
end loop;
--Supprime le retour à la ligne
Get_Immediate ( Caractere, Touche );

Put_Line ( "Effacement de la fenetre" );

-- Effacement de la fenêtre
Clear_Window;

Put_Line( "Dessin d'un rectangle au fond bleu" );

-- Déplacement du curseur
Move_To(200, 200);

-- Couleur bleu
Set_Color_Pen ( blue );

-- Dessin du rectangle
Box ( 120, 80, Fill );

Put_Line ( "Dessin d'un cercle jaune" );

-- Changement de couleur
Set_Color_Pen ( yellow );

-- Déplacement du curseur
Move_To ( 200, 150 );

-- Cercle de rayon 50 vide
Circle ( 50, NoFill );

Put_Line("Pressez ENTER pour continuer");
-- boucle pour traiter les événements en attendant une touche clavier
while not Touche loop
   Get_Immediate ( Caractere, Touche );
```

```
      Boucle_Dessin;
   end loop;
   --Supprime le retour à la ligne
   Get_Immediate ( Caractere, Touche );

   -- Déplacement du curseur
   Move_To(100, 400);

   --Couleur en vert
   Set_Color_Text ( green );

   -- Affichage du texte
   Display_Text ( "GtkSpider fonctionne" );

   Put_Line("Pressez ENTER pour terminer");
   -- boucle pour traiter les événements en attendant une touche clavier
   while not Touche loop
      Get_Immediate ( Caractere, Touche );
      Boucle_Dessin;
   end loop;
   --Supprime le retour à la ligne
   Get_Immediate ( Caractere, Touche );

   -- Fermeture de la fenêtre
   Close_Window;

end Demo1;
```

## *3.2. demo2.adb*

### 3.2.1. Introduction

Ce programme a été écrit à l'origine avec *Spider*. Il démontre bien la facilité de migration d'une librairie à l'autre.

En effet, à part modifier les clauses de contextes et rajouter à des endroits clés la procédure "Boucle_Dessin" (pas obligatoire d'ailleurs mais fortement conseillée), le code est exactement le même entre les deux versions.

Ce programme permet de dessiner les rayons d'un cercle, en demandant à l'utilisateur la position du centre du cercle, le nombre et la longueur des rayons. La couleur de chaque rayon est générée de manière aléatoire.

### 3.2.2. Code source

```
-- ------------------------------------------------------------------------
-- Fichier       : demo2.adb
-- Auteur        : Berthet Jérémy & Borer Reynald
-- Date          : 04.08.2004
--
-- But           : Ce programme permet de dessiner les rayons d'un cercle.
--                 Il demande à l'utilisateur la position du centre du cercle,
--                 la longueur ainsi que le nombre de rayons, et les affiche
--                 ensuite dans la fenêtre de dessin.
--                 Le nombre de rayon minimal à dessiner est de 3
--                 L'utilisateur peut effectuer le dessin plusieurs fois sans
--                 avoir besoin de relancer le programme à chaque fois
--
-- Remarque(s)   : hypothèse: l'utilisateur entre des valeurs correctes (pas de texte)
--
--
-- Modifications : Date  /  Auteur/  Raison
--
-- Compilateur   : GNAT 3.15p
-- Licence       : Gnu General Public License v2
-- ------------------------------------------------------------------------

with Ada.Text_IO;                       use Ada.Text_IO;
with Ada.Integer_Text_IO;               use Ada.Integer_Text_IO;
with Gtkspider;                         use Gtkspider;
with GtkSpider.Draw;                    use GtkSpider.Draw;
with Ada.Numerics.Elementary_Functions; use Ada.Numerics.Elementary_Functions;
with Ada.Numerics.Discrete_Random;

procedure Demo2 is

   -- Couleurs aléatoires
   subtype Rgb_Range is Integer range 0 .. 255;
   package Rgb_Aleatoire is new Ada.Numerics.Discrete_Random(Rgb_Range);
   use Rgb_Aleatoire;
   Generateur : Generator;
   Couleur : TColor;

   Centre_Cercle_X  : Natural;  -- position X du centre du cercle
   Centre_Cercle_Y  : Natural;  -- position Y du centre du cercle
   Grandeur_Rayon   : Positive; -- grandeur du rayon
   Nbre_Rayons      : Positive; -- nombre de rayons à dessiner
   Angle            : Float;    -- utilisé pour le calcul de l'angle entre les rayons
   X                : Float;    -- extrémité du rayon sur l'axe X
   Y                : Float;    -- extrémité du rayon sur l'axe Y
   Continue         : Character;-- Stocke si l'utilisateur veux continuer le programme
   Touche           : Boolean := False;
```

```
   PERIMETRE_CERCLE : constant Float   := 360.0; -- perimetre d'un cercle en degré
   NB_RAYON_MIN     : constant Integer := 3;     -- nombre de rayons minimum

begin -- Dessin_Rayons

   Reset ( Generateur );

   -- présentation du programme
   Put_Line("*** DESSIN DES RAYONS DU CERCLE ***");
   New_Line;
   Put_Line("Ce programme permet de dessiner les rayons d'un cercle en lui ");
   Put_Line("fournissant la position du centre du cercle, la longueur du rayon ");
   Put_Line("et le nombre de rayons désirés.");
   New_Line;
   Put_Line("ATTENTION: après l'ouverture de la fenêtre graphique veuillez ");
   Put_Line("revenir sur cette fenêtre pour entrer les valeurs.");
   New_Line(2);

   -- On initialise la fenêtre graphique
   Init_Window ( "Fenetre de dessin des rayons" );

   -- On boucle sur l'entier du programme et on demande à la fin de la boucle
   -- si l'utilisateur veux quitter le programme ou continuer
   Boucle_Globale: loop

      -- Efface la fenêtre
      Clear_Window;

      -- entrée des données utilisateur
      -- la position est à entrer sur une seule ligne
      Put_Line("Introduisez la position du centre du cercle (coordonées X Y sur la même ligne");
      Put("Position [X Y] : ");
      Get(Centre_Cercle_X);
      Get(Centre_Cercle_Y);
      Skip_Line;
      New_Line(2);

      Put_Line("Grandeur du rayon désirée ?");
      Put("rayon : ");
      Get(Grandeur_Rayon);
      Skip_Line;
      New_Line(2);

      -- Boucle tant que le nombre de rayons entré est plus petit que NB_RAYON_MIN
      Get_Rayon: loop
         Put("Nombre de rayons désirés (minimum ");
         Put(NB_RAYON_MIN, 0);
         Put_Line(") ?");
         Put("Nombre de rayons : ");
         Get(Nbre_Rayons);
         Skip_Line;
         exit Get_Rayon when Nbre_Rayons >= NB_RAYON_MIN;
      end loop Get_Rayon;

      -- Calcul de l'angle entre les rayons
      Angle := PERIMETRE_CERCLE / Float(Nbre_Rayons);

      -- On boucle sur le nombre de rayons à dessiner
      for I in 0 .. Nbre_Rayons - 1 loop

         -- couleur aléatoire
         Couleur := ( Random(Generateur),
                      Random(Generateur),
                      Random(Generateur));
      Set_Color_Pen(Couleur);

         -- calcul des positions des extrémités du rayon (en degré)
         X := cos( Float(I) * Angle, PERIMETRE_CERCLE) * Float(Grandeur_Rayon);
         Y := sin( Float(I) * Angle, PERIMETRE_CERCLE) * Float(Grandeur_Rayon);
```

```
         -- On ajoute la position du centre pour obtenir la position relativement
         -- au centre du cercle
         X := Float(Centre_Cercle_X) + X;
         Y := Float(Centre_Cercle_Y) + Y;

         --On dessine une droite depuis le centre à l'extrémité du rayon
         Line_To(Centre_Cercle_X,
                 Centre_Cercle_Y,
                 Integer(X),
                 Integer(Y) );

      end loop; --for

      Put_Line("Voulez-vous continuer ? [o / n]");
      Put("continuer : ");

      while not Touche loop
         Get_Immediate ( Continue, Touche );
         Boucle_Dessin;
      end loop;

      Skip_Line;

      exit Boucle_Globale when Continue = 'n'  or Continue = 'N';

      Touche := False;
      New_Line;

   end loop Boucle_Globale;

   Close_Window;
   New_Line;
   Put_Line("Fin du programme, pressez <enter> pour fermer la fenêtre");
   Skip_Line;

end Demo2;
```

# 4. Liens

[1] Site web de GtkAda: http://libre.act-europe.fr/GtkAda/

# ANNEXE D

# DOCUMENTATION TECHNIQUE

Projet de trimestre 1

**GtkSpider**

Jérémy **Berthet** & Reynald **Borer**
*EIVD*, 5 août 2005

# Table des matières

# 1. Informations générales

Dans la suite de ce document vous trouverez la documentation technique du projet de trimestre **GtkSpider**. Ce document contient tout d'abord les configurations requises pour utiliser le paquetage, pour continuer à le développer et pour le compiler. Ensuite vous trouverez une brève explication sur le découpage du code, puis la description des paquetages, avec pour chacun son but, son niveau de conformité avec le paquetage *Spider* original, ainsi que des explications sur les clauses de contexte, les déclarations et les sous-programmes.

## *1.1. Configuration requise pour l'exécution*

La configuration requise pour l'utilisation du paquetage nécessite les éléments suivants:
- Compilateur Ada (le paquetage n'a été testé qu'avec le compilateur GNAT)
- GtkAda installé et correctement configuré (site web de **GtkAda**: voir page 41)

Les paquetages *GtkSpider* peuvent soit être copié dans le répertoire courant, soit installés dans un répertoire séparé. Pour la deuxième solution, il faut ajouter un paramètre au compilateur GNAT afin qu'il trouve les paquetages:

```
gnatmake mon_fichier.adb -I/chemin/complet/vers/gtkspider/
```

## *1.2. Configuration requise pour le développement*

La configuration requise pour le développement est identique à celle pour l'exécution.

## *1.3. Informations pour la compilation*

Afin de compiler le paquetage, celui-ci nécessite **GtkAda** correctement installé et configuré. Référez-vous à la documentation utilisateur de **GtkAda** pour l'installation de la librairie.

# 2. Découpage du code

## 2.1. Architecture générale de GtkAda

La librairie *Gtk+* à été développée de manière à être le plus portable possible. Elle se décompose en 3 parties: *Gtk, Gdk* et *Glib.* Ci-dessous un schéma de la structure de ces parties et de leur interaction avec le système d'exploitation:

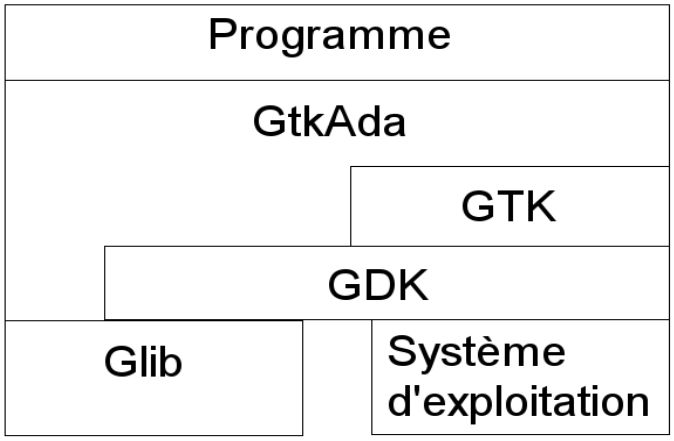


*Illustration 1: schéma de GtkAda (tiré de la documentation)*

## 2.2. Architecture du paquetage GtkSpider

Le découpage en paquetage est identique au découpage de la librairie *Spider*. Le paquetage parent *GtkSpider* fournis les fonctionnalités du paquetage *Spider*, tandis que le paquetage *GtkSpider.Draw* fournis les fonctionnalités du paquetage *Spider.Draw*. Ces fonctionnalités sont décrites dans les paragraphes concernant chaque paquetage.

La fonctionnement de la fenêtre est le suivant:

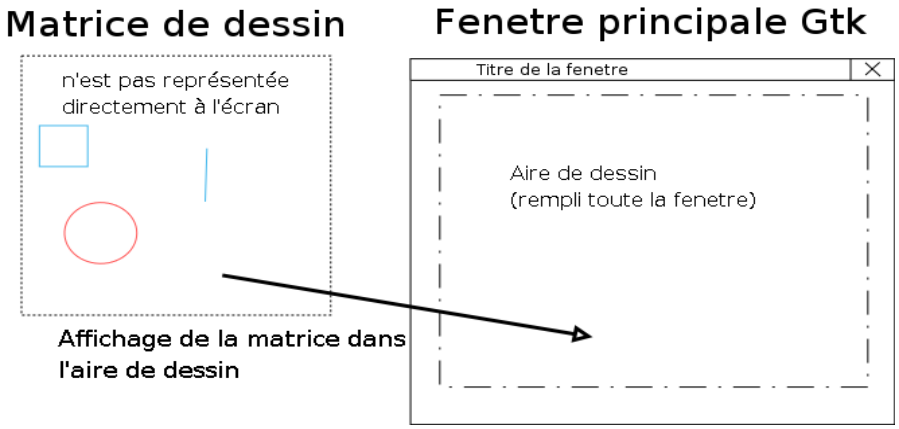


*Illustration 2: Fonctionnement de la fenêtre GtkSpider*

Les dessins sont effectués dans une zone mémoire qui n'est pas directement affichée (un « pixmap »). Une fois la forme dessinée, la procédure nous indique quelle zone du « pixmap » à été modifiée, et on re-dessine cette zone dans l'aire de dessin de la fenêtre. L'avantage de cette méthode est que l'on peux positionner ce « pixmap » n'importe ou dans l'aire de dessin, et cela évite aussi que la fenêtre « clignote » pendant le dessin.

# 3. Description des paquetages

## *3.1. Gtkspider*

### 3.1.1. Introduction

Ce paquetage, qui est le paquetage parent, met à disposition les fonctions de base pour le dessin. Ces fonctions sont entre autre la gestion de la fenêtre (ouverture, fermeture, effacement), les déplacements du curseur dans la fenêtre (déplacements relatifs et absolus), ainsi que le dessin de lignes noires.

### 3.1.2. Conformité avec *Spider*

Ce paquetage est totalement conforme avec la spécification du paquetage *Spider*. Tous les sous-programmes prennent exactement les même paramètres, et leurs fonctionnements sont identiques.

Une différence est quand même à indiquer: nous avons du rajouter une procédure *Boucle_Dessin* qui permet de traiter les événements *Gtk.* En effet, le fonctionnement de *Gtk* est un fonctionnement événementiel. A chaque fois que quelque chose se passe dans la fenêtre, un événement est levé et il faut le traiter. C'est le cas par exemple lorsque la fenêtre est déplacée ou cachée par un autre programme. Le Ré-affichage du contenu de la fenêtre ne se fait pas automatiquement, mais *Gtk* envoie un événement indiquant qu'il faut re-dessiner une partie de la fenêtre, puis c'est à nous de le traiter. Pour automatiser ce traitement, **GtkAda** fourni une boucle principale (`Gtk.Main.Main`) qui, une fois lancée, traite les événements lorsqu'ils se produisent. Mais cette boucle est bloquante, une fois lancée il est impossible de l'arrêter à moins d'appeler une procédure pour la terminer, par l'intermédiaire d'un événement qui s'est produit.

Comme il nous est impossible d'utiliser cette boucle dans notre paquetage, nous avons contourné le problème en fournissant à l'utilisateur une procédure permettant de traiter tous les événements en attente, et qu'il faut dans l'idéal appeler à chaque fois que le programme attend une entrée utilisateur. Un exemple d'utilisation de cette procédure est disponible dans la documentation utilisateur.

### 3.1.3. Clauses de contexte de la spécification

| **Clause:** | `with Gtk.Window; use Gtk.Window;` |
|---|---|
| **Description:** | Permet de gérer la fenêtre principale (ouverture, fermeture, etc.) |

| **Clause:** | `with Gdk.GC; use Gdk.GC;` |
|---|---|
| **Description:** | Pour définir un contexte graphique (gestion des couleurs) |

| **Clause:** | `with Gtk.Drawing_Area; use Gtk.Drawing_Area;` |
|---|---|
| **Description:** | Fournis un espace vide dans lequel il est possible de dessiner n'importe quel élément |

| **Clause:** | `with Gdk.Window; use Gdk.Window;` |
|---|---|
| **Description:** | Fenêtre de bas niveau représentant une zone rectangulaire à l'écran |

| **Clause:** | `with Gdk.Pixmap; use Gdk.Pixmap;` |
|---|---|
| **Description:** | Zone de dessin qui n'est pas affichée à l'écran |

| **Clause:** | `with Pango.Font; use Pango.Font;` |
|---|---|
| **Description:** | Fourni une structure représentant une police de caractère qui peut être affichée indépendamment du système d'exploitation utilisé |

### 3.1.4. Clauses de contexte du corps

| **Clause:** | `with Glib; use Glib;` |
|---|---|
| **Description:** | Fournis des fonctions utilitaires d'abstraction par rapport au langage C; fournis entre autre des types qui sont compatibles sur plusieurs systèmes d'exploitations |

| **Clause:** | `with Gdk.Drawable; use Gdk.Drawable;` |
|---|---|
| **Description:** | Fournis des fonctions pour dessiner des traits, points et autres sur des éléments de type `Gdk.Window` ou `Gdk.Pixmap` |

| **Clause:** | `with Gdk.Event; use Gdk.Event;` |
|---|---|
| **Description:** | Fournis des fonctions pour traiter les événements de la fenêtre |

| **Clause:** | `with Gdk.Color; use Gdk.Color;` |
|---|---|
| **Description:** | Gestion des couleurs |

| **Clause:** | `with Gdk.Rectangle; use Gdk.Rectangle;` |
|---|---|
| **Description:** | Permet de dessiner des rectangles |

| **Clause:** | `with Gtk.Enums; use Gtk.Enums;` |
|---|---|
| **Description:** | Types énumérés utilisés dans *Gtk+* |

| **Clause:** | `with Gtk.Main; use Gtk.Main;` |
|---|---|
| **Description:** | Fournis des fonctions pour traiter les événements en attente |

| **Clause:** | `with Gtk.Handlers; use Gtk.Handlers;` |
|---|---|
| **Description:** | Permet de "connecter" des sous-programmes à des événements émis par les divers éléments d'une fenêtre *Gtk+* |

| **Clause:** | `with Gtk.Style; use Gtk.Style;` |
|---|---|
| **Description:** | Permet de gérer le style de la fenêtre (couleurs, police de caractères, etc.) |

| **Clause:** | `with Gtk.Widget; use Gtk.Widget;` |
|---|---|
| **Description:** | Fournis des fonctions de base pour tous les éléments affichables |

| **Clause:** | `with Pango.Enums; use Pango.Enums;` |
|---|---|
| **Description:** | Fournis quelques constantes pour le traitement des polices de caractères |

| | |
|---|---|
| **Clause:** | `package Destroyed is new Gtk.Handlers.Callback (Widget_Type => Gtk_Window_Record);` |
| **Description:** | Instance de `Gtk.Handlers.Callback` permettant de connecter des sous-programmes aux événements sur la fenêtre principale |
| **Visibilité:** | Corps du paquetage |

| | |
|---|---|
| **Clause:** | `package Drawing_Area_Handlers is new Gtk.Handlers.Return_Callback (Widget_Type => Gtk_Drawing_Area_Record, Return_Type => Boolean);` |
| **Description:** | Instance de `Gtk.Handlers.Return_Callback` permettant de connecter des sous-programmes aux événements sur l'aire de dessin |
| **Visibilité:** | Corps du paquetage |

## 3.1.5. Structures de données

### *3.1.5.1. Types*

| | |
|---|---|
| **Nom:** | `Fenetre_Spider_Record` |
| **Description:** | Article étendant `Gtk_Window_Record` avec des champs vers les éléments contenus dans la fenêtre de dessin |
| **Visibilité:** | Type privé |

```
type Fenetre_Spider_Record is new Gtk_Window_Record with record

   Fenetre_Gdk : Gdk.Window.Gdk_Window;             -- Fenêtre Gdk
   Aire_Dessin : Gtk.Drawing_Area.Gtk_Drawing_Area; -- Aire de dessin
   Pixmap : Gdk.Pixmap.Gdk_Pixmap;                  -- Matrice de dessin
   Defaut_Gc : Gdk.GC.Gdk_GC;                       -- Contexte graphique
   Police : Pango.Font.Pango_Font_Description;      -- Police de caractères

end record;
```

| | |
|---|---|
| **Nom:** | `Fenetre_Spider_Access` |
| **Description:** | Type pointeur sur le type article défini ci-dessus |
| **Visibilité:** | Type privé |

```
type Fenetre_Spider_Access is access all Fenetre_Spider_Record'Class;
```

### *3.1.5.2. Variables*

| | |
|---|---|
| **Nom:** | `Fenetre` |
| **Type:** | `Fenetre_Spider_Access` |
| **Visibilité:** | Variable privée |
| **Description:** | Variable pointeur sur la structure de la fenêtre |

| | |
|---|---|
| **Nom:** | `Fenetre_Initialisee` |
| **Type:** | `Boolean` |
| **Val. par défaut:** | `False` |
| **Visibilité:** | Variable privée |
| **Description:** | Indique si la fenêtre a été précédemment initialisée |

| | |
|---|---|
| **Nom:** | `Global_Taille_X` |
| **Type:** | `Natural` |
| **Val. par défaut:** | `800` |
| **Visibilité:** | Variable privée |
| **Description:** | Taille dans l'axe des X par défaut de la fenêtre, stocke aussi la taille courante |

| | |
|---|---|
| **Nom:** | `Global_Taille_Y` |
| **Type:** | `Natural` |
| **Val. par défaut:** | `600` |
| **Visibilité:** | Variable privée |
| **Description:** | Taille dans l'axe des Y par défaut de la fenêtre, stocke aussi la taille courante |

| | |
|---|---|
| **Nom:** | `Cur_X` |
| **Type:** | `Natural` |
| **Val. par défaut:** | `0` |
| **Visibilité:** | Variable privée |
| **Description:** | Position courante du curseur dans l'axe des X |

| | |
|---|---|
| **Nom:** | `Cur_Y` |
| **Type:** | `Natural` |
| **Val. par défaut:** | `0` |
| **Visibilité:** | Variable privée |
| **Description:** | Position courante du curseur dans l'axe des Y |

#### *3.1.5.3. Exceptions*

Les exceptions utilisées dans ce paquetage sont strictement identiques à celles du paquetage *Spider*.

| | |
|---|---|
| **Nom:** | `Fenetre_Deja_Init` |
| **Description:** | L'utilisateur tente d'initialiser la fenêtre alors que celle-ci est déjà ouverte |

| | |
|---|---|
| **Nom:** | `Fenetre_Non_Init` |
| **Description:** | L'utilisateur tente dessiner dans la fenêtre alors que celle-ci n'est pas initialisée |

| | |
|---|---|
| **Nom:** | `Erreur_Inconnue` |
| **Description:** | Une autre erreur s'est produite lors de l'appel à certaines fonctions de GtkAda |

### 3.1.6. Sous-programmes

| | | | |
|---|---|---|---|
| **Nom:** | `Get_Max_X` | | |
| **Type:** | fonction | | |
| **Retour:** | `Natural` | | |
| **Visibilité:** | Externe | | |
| **Description:** | Retourne la taille maximale de l'écran graphique en X | | |

| | | | |
|---|---|---|---|
| **Nom:** | `Get_Max_Y` | | |
| **Type:** | fonction | | |
| **Retour:** | `Natural` | | |
| **Visibilité:** | Externe | | |
| **Description:** | Retourne la taille maximale de l'écran graphique en Y | | |

| | | | |
|---|---|---|---|
| **Nom:** | `Move_To` | | |
| **Type:** | procédure | | |
| **Paramètre(s):** | `X` | `in` | `Natural` |
| | `Y` | `in` | `Natural` |
| **Visibilité:** | Externe | | |
| **Description:** | Place le curseur à une position absolue dans la fenêtre de dessin | | |

| | | | |
|---|---|---|---|
| **Nom:** | `Move` | | |
| **Type:** | procédure | | |
| **Paramètre(s):** | `DX` | `in` | `Natural` |
| | `DY` | `in` | `Natural` |
| **Visibilité:** | Externe | | |
| **Description:** | Déplace le curseur relativement à la position actuelle | | |

| | | | |
|---|---|---|---|
| **Nom:** | `Line_To` | | |
| **Type:** | procédure | | |
| **Paramètre(s):** | `X1` | `in` | `Natural` |
| | `Y1` | `in` | `Natural` |
| | `X2` | `in` | `Natural` |
| | `Y2` | `in` | `Natural` |
| **Visibilité:** | Externe | | |
| **Exception(s):** | `Fenetre_Non_Init` | La fenêtre de dessin n'est pas initialisée, dessin impossible | |
| **Description:** | Dessine une ligne entre 2 points représentés par leurs coordonnées (X1, Y1) et (X2, Y2) | | |

| **Nom:** | `Line` | | |
|---|---|---|---|
| **Type:** | procédure | | |
| **Paramètre(s):** | `DX` | `in` | `Natural` |
| | `DY` | `in` | `Natural` |
| **Visibilité:** | Externe | | |
| **Exception(s):** | `Fenetre_Non_Init` | La fenêtre de dessin n'est pas initialisée, dessin impossible | |
| **Description:** | Dessine une ligne en partant du point courant et en effectuant un déplacement de DX sur l'axe X et DY sur l'axe Y | | |

| **Nom:** | `Init_Window` | | |
|---|---|---|---|
| **Type:** | procédure | | |
| **Paramètre(s):** | `Title` | `in` | `String` |
| **Visibilité:** | Externe | | |
| **Exception(s):** | `Fenetre_Deja_Init` | La fenêtre de dessin est déjà initialisée | |
| | `Erreur_Inconnue` | Erreur lors de l'initialisation de la fenêtre | |
| **Description:** | Ouvre la fenêtre de dessin en utilisant le paramètre `Title` pour le titre de la fenêtre ainsi que le contenu des variables `Global_Taille_X` et `Global_Taille_Y` pour la taille | | |

| **Nom:** | `Init_Size_Window` | | |
|---|---|---|---|
| **Type:** | procédure | | |
| **Paramètre(s):** | `Title` | `in` | `String` |
| | `Taille_X` | `in` | `Natural` |
| | `Taille_Y` | `in` | `Natural` |
| **Visibilité:** | Externe | | |
| **Exception(s):** | `Fenetre_Deja_Init` | La fenêtre de dessin est déjà initialisée | |
| | `Erreur_Inconnue` | Erreur lors de l'initialisation de la fenêtre | |
| **Description:** | Ouvre la fenêtre de dessin en utilisant le paramètre `Title` pour le titre de la fenêtre ainsi que le contenu des paramètres `Taille_X` et `Taille_Y` pour la taille | | |

| **Nom:** | `Close_Window` | |
|---|---|---|
| **Type:** | procédure | |
| **Visibilité:** | Externe | |
| **Exception(s):** | `Fenetre_Non_Init` | La fenêtre de dessin n'est pas initialisée |
| **Description:** | Ferme la fenêtre graphique | |

| | | |
|---|---|---|
| **Nom:** | `Clear_Window` | |
| **Type:** | procédure | |
| **Visibilité:** | Externe | |
| **Exception(s):** | `Fenetre_Non_Init` | La fenêtre de dessin n'est pas initialisée |
| **Description:** | Efface le contenu de la fenêtre graphique (dessine un carré blanc) | |

| | |
|---|---|
| **Nom:** | `Boucle_Dessin` |
| **Type:** | procédure |
| **Visibilité:** | Externe |
| **Description:** | Permet de traiter les événements *Gtk* en attente |
| **Remarques:** | Cette procédure est nécessaire à cause du fonctionnement interne de *Gtk*. En effet, la programmation en *Gtk* est de la programmation événementielle, une fois la fenêtre ouverte, le programme attend des événements de la part de l'utilisateur (mouvement ou clic de souris, déplacement de la fenêtre, etc.). Le fonctionnement interne de *Gtk* fait qu'il ne dessine que les parties visibles de la fenêtre. Donc si pendant le dessin l'utilisateur met un autre programme en avant plan, la zone couverte ne sera plus dessinée, et un événement *expose_event* sera généré pour que *Gtk* re-dessine la zone. Il faut donc traiter ces événements, ce que fait cette procédure. |

| | | | |
|---|---|---|---|
| **Nom:** | `Destroy_Event` | | |
| **Type:** | procédure | | |
| **Paramètre(s):** | `Window` | `access` | `Gtk.Window.Gtk_Window_Record'Class` |
| **Visibilité:** | Interne | | |
| **Description:** | Procédure appelée lorsque l'utilisateur clique sur la croix pour fermer la fenêtre principale | | |

| | | | |
|---|---|---|---|
| **Nom:** | `Expose_Event` | | |
| **Type:** | fonction | | |
| **Retour:** | `Boolean` | | |
| **Paramètre(s):** | `Dessin` | `access` | `Gtk_Drawing_Area_Record'Class` |
| | `Event` | `in` | `Gdk.Event.Gdk_Event` |
| **Visibilité:** | Interne | | |
| **Description:** | Fonction appelée à chaque fois que la fenêtre à besoin d'être re-dessinée (par exemple car masquée par une autre fenêtre). | | |

## *3.2. GtkSpider.Draw*

### 3.2.1. Introduction

Ce paquetage enfant met à disposition les sous-programmes du paquetage *Spider.Draw*. Ces sous-programmes permettent de changer la couleur du texte, de l'arrière-plan d'une figure, d'afficher du texte, de dessiner des cercles et des rectangles.

### 3.2.2. Conformité avec Spider

Tous les sous-programmes implémentés dans ce paquetage respectent la même spécification que ceux du paquetage *Spider.Draw*. 3 de ces sous-programmes n'ont pas été implémentés. Il s'agit des fonctions `Get_Width_Char` et `Get_Height_Char` qui permettent de connaître la largeur et la hauteur maximum de la police de caractères, ainsi que la fonction `Get_Pixel` qui indique la couleur du pixel à la position courante. Nous n'avons pas trouvé de moyen dans **GtkAda** pour obtenir ces informations, c'est pourquoi elles ne sont pas implémentées. Néanmoins, un rapide sondage nous à permis de nous rendre compte qu'elles n'étaient presque pas utilisées, ce qui ne devrait pas être dommageable à notre paquetage.

### 3.2.3. Clauses de contexte de la spécification

La spécification ne contient aucune clause de contexte.

### 3.2.4. Clauses de contexte du corps

| **Clause:** | `with Glib; use Glib;` |
|---|---|
| **Description:** | Fournis des fonctions utilitaires d'abstraction par rapport au langage C; fournis entre autre des types qui sont compatibles sur plusieurs systèmes d'exploitations |

| **Clause:** | `with Gdk.Drawable; use Gdk.Drawable;` |
|---|---|
| **Description:** | Fournis des fonctions pour dessiner des traits, points et autres sur des éléments de type `Gdk.Window` ou `Gdk.Pixmap` |

| **Clause:** | `with Gdk.Color; use Gdk.Color;` |
|---|---|
| **Description:** | Gestion des couleurs |

| **Clause:** | `with Gtk.Widget; use Gtk.Widget;` |
|---|---|
| **Description:** | Fournis des fonctions de base pour tous les éléments affichables |

| **Clause:** | `with Pango.Layout; use Pango.Layout;` |
|---|---|
| **Description:** | Permet de mettre en forme une grande quantité de caractères de manière correcte (paragraphes, lignes, etc.) |

## 3.2.5. Structures de données

### *3.2.5.1. Types*

| **Nom:** | `tFill` |
|---|---|
| **Description:** | Type énumératif pour indiquer l'état d'une forme géométrique (remplie ou dessin uniquement de la bordure) |
| **Visibilité:** | Externe |

```
type tFill is ( fill, noFill );
```

| **Nom:** | `tColor` |
|---|---|
| **Description:** | Article représentant une couleur au format RGB |
| **Visibilité:** | Externe |

```
   type tColor is record
      R : Integer;
      G : Integer;
      B : Integer;
   end record;
```

### *3.2.5.2. Variables*

Plusieurs variables de type `tColor` représentant les couleurs usuelles sont définies pour l'utilisateur du paquetage. La liste complète avec les valeurs des couleurs au format *RGB* est la suivante:

```
black        : constant tColor := (  0,  0,  0);
blue         : constant tColor := (  0,  0,255);
green        : constant tColor := (  0,255,  0);
cyan         : constant tColor := (  0,255,255);
red          : constant tColor := (255,  0,  0);
magenta      : constant tColor := (255,  0,255);
brown        : constant tColor := (128, 64,  0);
lightGray    : constant tColor := (192,192,192);
darkGray     : constant tColor := (128,128,128);
lightBlue    : constant tColor := (128,128,255);
lightGreen   : constant tColor := (128,255,128);
lightCyan    : constant tColor := (128,255,255);
lightRed     : constant tColor := (255,128,128);
lightMagenta : constant tColor := (255,128,255);
yellow       : constant tColor := (255,255,  0);
white        : constant tColor := (255,255,255);
```

| **Nom:** | `Rgb_Max` |
|---|---|
| **Type:** | `Constante universelle` |
| **Val. par défaut:** | `255` |
| **Visibilité:** | Interne |
| **Description:** | Valeur maximum possible pour les champs de l'article `tColor` |

| | |
|---|---|
| **Nom:** | `Val_Conversion` |
| **Type:** | `constant Guint16` |
| **Val. par défaut:** | `Guint16'Last / Guint16 ( Rgb_Max )` |
| **Visibilité:** | Interne |
| **Description:** | La représentation des couleurs au format *RGB* dans **GtkAda** est au format `Guint16` défini dans le paquetage *Glib.* Cette constante permet d'effectuer les conversions avec les valeurs du paquetage *GtkSpider-Draw* |

| | |
|---|---|
| **Nom:** | `Pen_Color` |
| **Type:** | `Gdk_Color` |
| **Val. par défaut:** | `Gdk.Color.Parse ( "black" );` |
| **Visibilité:** | Interne |
| **Description:** | Stocke la couleur courante du trait (valeur par défaut: noir) |

| | |
|---|---|
| **Nom:** | `Background_Color` |
| **Type:** | `Gdk_Color` |
| **Val. par défaut:** | `Gdk.Color.Parse ( "white" );` |
| **Visibilité:** | Interne |
| **Description:** | Stocke la couleur courante de l'arrière-plan du texte (valeur par défaut: blanc) |

| | |
|---|---|
| **Nom:** | `Text_Color` |
| **Type:** | `Gdk_Color` |
| **Val. par défaut:** | `Gdk.Color.Parse ( "black" );` |
| **Visibilité:** | Interne |
| **Description:** | Stocke la couleur courante du texte (valeur par défaut: noir) |

#### *3.2.5.3. Exceptions*

Aucune exception n'est définie dans le paquetage.

### 3.2.6. Sous-programmes

| **Nom:** | `Rgb_Guint16` | | |
|---|---|---|---|
| **Type:** | fonction | | |
| **Paramètre(s):** | `Color` | `in` | `Integer` |
| **Retour:** | `Guint16` | | |
| **Visibilité:** | Interne | | |
| **Description:** | Converti une composante d'une couleur *RGB* (valeur maximum définie dans `Rgb_Max`) dans le type `Guint16` pour les fonctions de couleur dans **GtkAda** | | |

| **Nom:** | `Guint16_Rgb` | | |
|---|---|---|---|
| **Type:** | fonction | | |
| **Paramètre(s):** | `Color` | `in` | `Guint16` |
| **Retour:** | `Integer` | | |
| **Visibilité:** | Interne | | |
| **Description:** | Converti une composante d'une couleur **GtkAda** (`Guint16`) dans le format utilisé pour `tColor` (valeur maximum définie dans `Rgb_Max`) | | |

| **Nom:** | `Set_Color_Pen` | | |
|---|---|---|---|
| **Type:** | procédure | | |
| **Paramètre(s):** | `Color` | `in` | `tColor` |
| **Visibilité:** | Externe | | |
| **Exception(s):** | `Erreur_Inconnue` | Problème lors de la création de la couleur | |
| **Description:** | Défini la couleur utilisée pour les traits | | |

| **Nom:** | `Set_Color_Background` | | |
|---|---|---|---|
| **Type:** | procédure | | |
| **Paramètre(s):** | `Color` | `in` | `tColor` |
| **Visibilité:** | Externe | | |
| **Exception(s):** | `Erreur_Inconnue` | Problème lors de la création de la couleur | |
| **Description:** | Défini la couleur utilisée pour l'arrière-plan du texte | | |

| **Nom:** | `Set_Color_Text` | | |
|---|---|---|---|
| **Type:** | procédure | | |
| **Paramètre(s):** | `Color` | `in` | `tColor` |
| **Visibilité:** | Externe | | |
| **Exception(s):** | `Erreur_Inconnue` | Problème lors de la création de la couleur | |
| **Description:** | Défini la couleur utilisée pour le texte | | |

| | |
|---|---|
| **Nom:** | `Get_Color_Pen` |
| **Type:** | fonction |
| **Retour:** | `tColor` |
| **Visibilité:** | Externe |
| **Description:** | Retourne la couleur utilisée pour les traits |

| | |
|---|---|
| **Nom:** | `Get_Color_Background` |
| **Type:** | fonction |
| **Retour:** | `tColor` |
| **Visibilité:** | Externe |
| **Description:** | Retourne la couleur utilisée pour l'arrière-plan du texte |

| | |
|---|---|
| **Nom:** | `Get_Color_Text` |
| **Type:** | fonction |
| **Retour:** | `tColor` |
| **Visibilité:** | Externe |
| **Description:** | Retourne la couleur utilisée pour le texte |

| | | | |
|---|---|---|---|
| **Nom:** | `Circle` | | |
| **Type:** | procédure | | |
| **Paramètre(s):** | `Radius` | `in` | `Natural` |
| | `Filled` | `in` | `tFill` |
| **Visibilité:** | Externe | | |
| **Exception(s):** | `Fenetre_Non_Init` | La fenêtre de dessin n'est pas initialisée | |
| **Description:** | Dessine un cercle dont le centre est à la position courante. `Radius` donne le rayon du cercle, `Filled` indique si il doit être rempli ou si uniquement le bord doit être dessiné | | |

| | | | |
|---|---|---|---|
| **Nom:** | `Box` | | |
| **Type:** | procédure | | |
| **Paramètre(s):** | `Width` | `in` | `Integer` |
| | `Height` | `in` | `Integer` |
| | `Filled` | `in` | `tFill` |
| **Visibilité:** | Externe | | |
| **Exception(s):** | `Fenetre_Non_Init` | La fenêtre de dessin n'est pas initialisée | |
| **Description:** | Dessine un rectangle dont le coin supérieur gauche est à la position courante. `Width` et `Height` définissent la longueur et la largeur du rectangle, `Filled` indique si le rectangle est plein ou si uniquement la bordure est dessinée. | | |

| **Nom:** | `Put_Pixel` | |
|---|---|---|
| **Type:** | procédure | |
| **Visibilité:** | Externe | |
| **Exception(s):** | `Fenetre_Non_Init` | La fenêtre de dessin n'est pas initialisée |
| **Description:** | Dessine un pixel à la position courante | |

| **Nom:** | `Display_Text` | | |
|---|---|---|---|
| **Type:** | procédure | | |
| **Paramètre(s):** | `Text` | `in` | `String` |
| **Visibilité:** | Externe | | |
| **Exception(s):** | `Fenetre_Non_Init` | La fenêtre de dessin n'est pas initialisée | |
| **Description:** | Affiche le texte passé dans le paramètre `Text` à la position courante | | |

# 4. Liens

Site web de GtkAda: http://libre.act-europe.fr/GtkAda/
GtkAda User's Guide: http://libre.act-europe.fr/GtkAda/docs/gtkada_ug.html

# Annexe E. Code source

## *E.1. Gtkspider.ads*

```
-- -----------------------------------------------------------------------------
-- Fichier       : gtkspider.ads
-- Auteur        : Berthet Jeremy & Borer Reynald
-- Date          : 29.05.2005
--
-- But           : Fournis des fonctions graphiques minimales en s'inspirant
--                 de celles fournies par Spider
--
-- Remarque(s)   : Utilise GtkAda
--
-- Modifications : Date / Auteur / Raison
--
-- Compilateur   : GNAT 3.15p
-- Licence       : Gnu General Public License v2
-- -----------------------------------------------------------------------------

with Gtk.Window;       use Gtk.Window;
with Gdk.GC;           use Gdk.GC;
with Gtk.Drawing_Area; use Gtk.Drawing_Area;
with Gdk.Window;       use Gdk.Window;
with Gdk.Pixmap;       use Gdk.Pixmap;
with Pango.Font;       use Pango.Font;

package Gtkspider is

   ---------------
   -- Get_Max_X --
   ---------------
   -- But : Retourne la taille maximale de l'écran graphique en X
   --
   -- Paramètre(s) : -
   --
   -- Résultat : Natural
   --
   -- Exception(s) : -
   -- --
   function Get_Max_X return Natural;


   ---------------
   -- Get_Max_Y --
   ---------------
   -- But : Retourne la taille maximale de l'écran graphique en Y
   --
   -- Paramètre(s) : -
   --
   -- Résultat : Natural
   --
   -- Exception(s) : -
   -- --
   function Get_Max_Y return Natural;


   -------------
   -- Move_To --
   -------------
   -- But : Placer le curseur a une position (absolue) de l'écran.
   --
   -- Paramètre(s) : X => Nouvelle position du point sur l'axe X
   --                Y => Nouvelle position du point sur l'axe Y
   --
   -- Exception(s) : -
   -- --
```

```
   procedure Move_To ( X : in Natural;
                       Y : in Natural );


   ----------
   -- Move --
   ----------
   -- But : Placer le curseur a une position en spécifiant le déplacement
   --
   -- Paramètre(s) : DX => Déplacement relatif du point sur l'axe X
   --                DY => Déplacement relatif du point sur l'axe Y
   --
   -- Exception(s) : -
   -- --
   procedure Move ( DX : in Integer;
                    DY : in Integer );


   -------------
   -- Line_To --
   -------------
   -- But : Dessine une ligne entre 2 points
   --
   -- Paramètre(s) : X1, Y1 => Coordonnées du point de départ
   --                X2, Y2 => Coordonnées du point d'arrivée
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Line_To ( X1 : in Natural;
                       Y1 : in Natural;
                       X2 : in Natural;
                       Y2 : in Natural );


   ----------
   -- Line --
   ----------
   -- But : Dessine une ligne en partant du point courant
   --
   -- Paramètre(s) : DX => Déplacement sur l'axe des X pour le point arrivée
   --                DY => Déplacement sur l'axe des Y pour le point arrivée
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Line ( DX : in Integer;
                    DY : in Integer );


   -----------------
   -- Init_Window --
   -----------------
   -- But        : Ouvre la fenêtre graphique de l'application avec une taille
   --              par défaut de 800 x 600 pixels
   --
   -- Paramètre(s) : Title => Titre de la fenêtre
   --
   -- Exception(s) : Fenetre_Deja_Init => Fenêtre déjà initialisée
   --                Erreur_Inconnue => Erreur lors de l'initialisation de
   --                                   la fenêtre
   -- --
   procedure Init_Window ( Title : in String );


   ----------------------
   -- Init_Size_Window --
   ----------------------
   -- But        : Ouvre la fenêtre graphique de l'application avec la taille
   --              demandée
   --
```

```
   -- Paramètre(s) : Title => Titre de la fenêtre
   --              Taille_X, Taille_Y => taille de la fenêtre
   --
   -- Exception(s) : Fenetre_Deja_Init => Fenêtre déjà initialisée
   --                Erreur_Inconnue => Erreur lors de l'initialisation de
   --                                   la fenêtre
   -- --
   procedure Init_Size_Window ( Title :    in String;
                                Taille_X : in Natural;
                                Taille_Y : in Natural);


   ------------------
   -- Close_Window --
   ------------------
   -- But        : Ferme la fenêtre graphique courante
   --
   -- Paramètre(s) : -
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Close_Window;


   ------------------
   -- Clear_Window --
   ------------------
   -- But        : Efface la fenêtre graphique
   --
   -- Paramètre(s) : -
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Clear_Window;


   --------------------
   -- Boucle_Dessin  --
   --------------------
   -- But        : Cette boucle permet à GtkAda de traiter tous les événements
   --              en attente, par exemple le réaffichage d'une zone qui était
   --              masquée par une fenêtre, etc.
   --
   -- Paramètre(s) : -
   --
   -- Exception(s) : -
   -- --
   procedure Boucle_Dessin;


   -- Exceptions
   Fenetre_Deja_Init : exception; -- L'utilisateur tente d'initialiser la
                                  -- fenêtre déjà ouverte
   Fenetre_Non_Init  : exception; -- La fenêtre n'est pas initialisée
   Erreur_Inconnue   : exception; -- Erreur inconnue


private

   -- Type contenant tous les éléments de la fenêtre de dessin
   type Fenetre_Spider_Record is new Gtk_Window_Record with record

      Fenetre_Gdk : Gdk.Window.Gdk_Window;             -- Fenetre Gdk
      Aire_Dessin : Gtk.Drawing_Area.Gtk_Drawing_Area; -- Aire de dessin
      Pixmap : Gdk.Pixmap.Gdk_Pixmap;                  -- Matrice de dessin
      Defaut_Gc : Gdk.GC.Gdk_GC;                       -- Contexte graphique
      Police : Pango.Font.Pango_Font_Description;      -- Police de caractères

   end record;
```

```
   -- Type pointeur sur l'article contenant les éléments de la fenêtre
   type Fenetre_Spider_Access is access all Fenetre_Spider_Record'Class;

   -- Fenêtre principale
   Fenetre : Fenetre_Spider_Access;

   --Stocke l'état de l'initialisation de la fenêtre
   Fenetre_Initialisee : Boolean := False;

   -- Taille de la fenêtre, avec valeurs par défaut
   Global_Taille_X : Natural := 800;
   Global_Taille_Y : Natural := 600;

   -- Conserver la position courante du pointeur dans l'espace de dessin
   Cur_X : Natural := 0;
   Cur_Y : Natural := 0;

end Gtkspider;
```

## *E.2. Gtkspider.adb*

```
-- -----------------------------------------------------------------------------
-- Fichier       : gtkspider.adb
-- Auteur        : Berthet Jeremy & Borer Reynald
-- Date          : 29.05.2005
--
-- But           : Fournis des fonctions graphiques minimales en s'inspirant
--                 de celles fournies par Spider
--
-- Remarque(s)   : Utilise GtkAda
--
-- Modifications : Date / Auteur / Raison
--
-- Compilateur   : GNAT 3.15p
-- Licence       : Gnu General Public License v2
-- -----------------------------------------------------------------------------
with Glib;             use Glib;

with Gdk.Drawable;     use Gdk.Drawable;
with Gdk.Event;        use Gdk.Event;
with Gdk.Color;        use Gdk.Color;
with Gdk.Rectangle;    use Gdk.Rectangle;

with Gtk.Enums;        use Gtk.Enums;
with Gtk.Main;         use Gtk.Main;
with Gtk.Handlers;     use Gtk.Handlers;
with Gtk.Style;        use Gtk.Style;
with Gtk.Widget;       use Gtk.Widget;

with Pango.Enums;      use Pango.Enums;


package body Gtkspider is

   -- Pour la gestion des événements sur la fenêtre
   package Destroyed is new Gtk.Handlers.Callback
     ( Widget_Type => Gtk_Window_Record );

   -- Pour la gestion des événements sur l'aire de dessin
   package Drawing_Area_Handlers is new Gtk.Handlers.Return_Callback
     ( Widget_Type => Gtk_Drawing_Area_Record,
       Return_Type => Boolean );


   -------------------
   -- Destroy_Event --
   -------------------
   -- But : procédure appelée lorsque l'utilisateur ferme la fenêtre en cliquant
   --       sur la croix
   --
   -- Paramètre(s) : Window => Pointeur sur la fenêtre
   --
   -- Exception(s) : -
   -- --
   procedure Destroy_Event (Window : access Gtk.Window.Gtk_Window_Record'Class) is

      -- Variable inutilisée => message du compilateur
      pragma Warnings (Off, Window);

   begin

      -- Ferme la fenêtre
      Close_Window;

   end Destroy_Event;


   ------------------
```

```
   -- Expose_Event --
   ------------------
   -- But : fonction appelée à chaque fois que la fenêtre à besoin d'être
   --       redessinée (par exemple car elle était masquée par une autre fenêtre)
   --
   -- Paramètre(s) : Dessin => Pointeur sur l'aire de dessin
   --                Event => événement Gdk, contient entre autre la partie
   --                         exacte à redessiner
   --
   -- Résultat : Natural
   --
   -- Exception(s) : -
   -- --
   function Expose_Event
            ( Dessin : access Gtk_Drawing_Area_Record'Class;
              Event : in Gdk.Event.Gdk_Event) return Boolean is

      Area : constant Gdk_Rectangle := Get_Area ( Event );
      Window : constant Gdk_Window := Get_Window ( Dessin );
      Gc : constant Gdk_Gc := Gtk.Style.Get_White_Gc (Get_Style ( Dessin ));

   begin -- Expose_Event

      -- Redessine la zone donnée en paramètre
      Gdk.Drawable.Draw_Drawable ( Drawable => Window,
                                   Gc => Gc,
                                   Src => Fenetre.Pixmap,
                                   Xsrc => Area.X, Ysrc => Area.Y,
                                   Xdest => Area.X, Ydest => Area.Y,
                                   Width => Gint ( Area.Width ),
                                   Height => Gint ( Area.Height ) );

      return True;

   end Expose_Event;


   --------------------
   -- Boucle_Dessin  --
   --------------------
   -- But        : Cette boucle permet à GtkAda de traiter tous les événements
   --              en attente, par exemple le réaffichage d'une zone qui était
   --              masquée par une fenêtre, etc.
   --
   -- Paramètre(s) : -
   --
   -- Exception(s) : -
   -- --
   procedure Boucle_Dessin is

      Main_Loop : Boolean;

      -- Variable jamais lue => message du compilateur
      pragma Warnings ( Off, Main_Loop );

   begin -- Boucle_Dessin

      -- Traite tous les événements en attente, puis termine le sous-programme
      while Gtk.Main.Events_Pending loop

         Main_Loop := Main_Iteration ( False );

      end loop;

   end Boucle_Dessin;


   -----------------
   -- Init_Window --
```

```
   ----------------
   -- But        : Ouvre la fenêtre graphique de l'application avec une taille
   --              par défaut de 800 x 600 pixels
   --
   -- Paramètre(s) : Title => Titre de la fenêtre
   --
   -- Exception(s) : Fenetre_Deja_Init => Fenêtre déjà initialisée
   --                Erreur_Inconnue => Erreur lors de l'initialisation de
   --                                   la fenêtre
   -- --
   procedure Init_Window ( Title : in String ) is

   begin -- Init_Window

      Init_Size_Window ( Title, Global_Taille_X, Global_Taille_Y );

   end Init_Window;


   ----------------------
   -- Init_Size_Window --
   ----------------------
   -- But        : Ouvre la fenêtre graphique de l'application avec la taille
   --              demandée
   --
   -- Paramètre(s) : Title => Titre de la fenêtre
   --              Taille_X, Taille_Y => taille de la fenêtre
   --
   -- Exception(s) : Fenetre_Deja_Init => Fenêtre déjà initialisée
   --                Erreur_Inconnue => Erreur lors de l'initialisation de
   --                                   la fenêtre
   -- --
   procedure Init_Size_Window ( Title :    in String;
                                Taille_X : in Natural;
                                Taille_Y : in Natural) is

      Startup_Error : Boolean; --Problème à l'initialisation de GtkAda

      Taille_Police : constant := 10; --Taille de la police de caractères

   begin -- Init_Size_Window

      -- Reproduit le fonctionnement de Spider
      if Fenetre_Initialisee then
         raise Fenetre_Deja_Init;
      else
         Fenetre_Initialisee := True;
      end if;

      -- Nouvel objet pour la fenêtre
      Fenetre := new Fenetre_Spider_Record;

      -- Met à jour les variables globales
      Global_Taille_X := Taille_X;
      Global_Taille_Y := Taille_Y;

      -- Initialise la structure interne de GtkAda, retourne False si
      -- il y a une erreur (pas d'accès à l'écran, etc.)
      Startup_Error := Gtk.Main.Init_Check;

      -- Informe l'utilisateur du problème
      if not Startup_Error then
         raise Erreur_Inconnue;
      end if;

      --Crée une nouvelle fenêtre Gtk
      Gtk.Window.Initialize ( Fenetre, Window_Toplevel );

      --Événement destroy sur la fenêtre principale: quand l'utilisateur
```

```
      -- ferme le fenêtre avec la croix
      Destroyed.Connect ( Fenetre, "destroy",
                          Destroyed.To_Marshaller (Destroy_Event'Access) );

      --Titre de la fenêtre
      Set_Title ( Fenetre, Title );

      --Taille par défaut de la fenêtre
      Set_Default_Size ( Fenetre, Gint ( Taille_X ), Gint ( Taille_Y ) );

      --L'utilisateur ne peux pas redimensionner la fenêtre
      Set_Resizable ( Fenetre, False );

      --Crée une zone de dessin
      Gtk_New ( Fenetre.Aire_Dessin );

      --Taille de l'aire de dessin
      Set_USize ( Fenetre.Aire_Dessin, Gint ( Taille_X ), Gint ( Taille_Y ) );

      -- Ajoute l'aire de dessin à la fenêtre
      Add ( Fenetre, Fenetre.Aire_Dessin );

      --On décide à quels événements l'aire de dessin doit répondre
      Set_Events ( Fenetre.Aire_Dessin, Exposure_Mask );

      --On "connecte" l'événement expose_event à la fonction qui le traite
      Drawing_Area_Handlers.Connect(Widget => Fenetre.Aire_Dessin,
                                    Name   => "expose_event",
                                    Marsh  => Drawing_Area_Handlers.To_Marshaller
                                      (Expose_Event'Access));

      -- Affiche la fenêtre afin de pouvoir dessiner dedans
      Show_All ( Fenetre );

      -- Récupére la fenêtre Gdk (bas niveau) de l'aire de dessin
      Fenetre.Fenetre_Gdk := Get_Window ( Fenetre.Aire_Dessin );

      -- Crée un nouveau pixmap dans lequel on va dessiner
      Gdk.Pixmap.Gdk_New ( Fenetre.Pixmap, Fenetre.Fenetre_Gdk,
                           Gint (Taille_X), Gint (Taille_Y), -1);

      -- Crée le contexte graphique avec des couleurs par défaut
      Gdk.GC.Gdk_New ( Fenetre.Defaut_Gc, Fenetre.Fenetre_Gdk );
      Set_Foreground ( Fenetre.Defaut_Gc,
                       Gdk.Color.Black (Gtk.Widget.Get_Default_Colormap) );
      Set_Background ( Fenetre.Defaut_Gc,
                       Gdk.Color.White (Gtk.Widget.Get_Default_Colormap) );

      -- Efface la fenêtre courante
      Clear_Window;

      -- Police de caractères, avec sa taille
      Fenetre.Police := Get_Font_Description ( Get_Style ( Fenetre.Aire_Dessin));
      Pango.Font.Set_Size ( Fenetre.Police, Taille_Police * Pango_Scale );

      -- Traite tous les événements en attente
      Boucle_Dessin;

   end Init_Size_Window;


   ------------------
   -- Clear_Window --
   ------------------
   -- But        : Efface la fenêtre graphique
   --
   -- Paramètre(s) : -
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
```

```
   -- --
   procedure Clear_Window is

   begin -- Clear_Window

      -- effacement d'une fenêtre pas initialisée
      if not Fenetre_Initialisee then
         raise Fenetre_Non_Init;
      end if;

      --Dessine en premier lieu un rectangle blanc pour "effacer" le pixmap
      Draw_Rectangle ( Drawable => Fenetre.Pixmap,
                       Gc => Gtk.Style.Get_White_Gc ( Get_Style ( Fenetre ) ),
                       Filled => True,
                       X => 0, Y => 0,
                       Width => Gint ( Global_Taille_X ),
                       Height => Gint ( Global_Taille_Y ) );

      -- Affichage sur l'aire de dessin de l'espace affecté dans le pixmap
      Draw_Drawable ( Drawable => Fenetre.Fenetre_Gdk,
                      Gc => Gtk.Style.Get_White_Gc ( Get_Style ( Fenetre ) ),
                      Src => Fenetre.Pixmap,
                      Xsrc => 0, Ysrc => 0,
                      Xdest => 0, Ydest => 0,
                      Width => Gint ( Global_Taille_X ),
                      Height => Gint ( Global_Taille_Y ) );

      -- Traite les événements en attente
      Boucle_Dessin;

   end Clear_Window;


   ------------------
   -- Close_Window --
   ------------------
   -- But        : Ferme la fenêtre graphique courante
   --
   -- Paramètre(s) : -
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Close_Window is

   begin -- Close_Window

      if Fenetre_Initialisee then

         -- Traite les événements en attente
         Boucle_Dessin;

         -- Ferme la fenêtre
         Destroy ( Fenetre );

         Fenetre_Initialisee := False;


      else
         -- impossible de fermer une fenêtre inexistante
         raise Fenetre_Non_Init;

      end if;

   end Close_Window;


   ---------------
   -- Get_Max_X --
   ---------------
```

```
   -- But : Retourne la taille maximale de l'écran graphique en X
   --
   -- Paramètre(s) : -
   --
   -- Résultat : Natural
   --
   -- Exception(s) : -
   -- --
   function Get_Max_X return Natural is

   begin -- Get_Max_X

      return Global_Taille_X;

   end Get_Max_X;


   ---------------
   -- Get_Max_Y --
   ---------------
   -- But : Retourne la taille maximale de l'écran graphique en Y
   --
   -- Paramètre(s) : -
   --
   -- Résultat : Natural
   --
   -- Exception(s) : -
   -- --
   function Get_Max_Y return Natural is

   begin -- Get_Max_Y

      return Global_Taille_Y;

   end Get_Max_Y;


   -------------
   -- Move_To --
   -------------
   -- But : Placer le curseur a une position (absolue) de l'écran.
   --
   -- Paramètre(s) : X => Nouvelle position du point sur l'axe X
   --                Y => Nouvelle position du point sur l'axe Y
   --
   -- Exception(s) : -
   -- --
   procedure Move_To(X : in Natural;
                     Y : in Natural) is

   begin -- Move_To

      Cur_X := X;
      Cur_Y := Y;

   end Move_To;


   ----------
   -- Move --
   ----------
   -- But : Placer le curseur a une position en spécifiant le déplacement
   --
   -- Paramètre(s) : DX => Déplacement relatif du point sur l'axe X
   --                DY => Déplacement relatif du point sur l'axe Y
   --
   -- Exception(s) : -
   -- --
   procedure Move(DX : in Integer;
```

```
                DY : in Integer) is

   begin -- Move

      Cur_X := Cur_X + DX;
      Cur_Y := Cur_Y + DY;

   end Move;


   -------------
   -- Line_To --
   -------------
   -- But : Dessine une ligne entre 2 points
   --
   -- Paramètre(s) : X1, Y1 => Coordonnées du point de départ
   --                X2, Y2 => Coordonnées du point d'arrivée
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Line_To(X1 : in Natural;
                     Y1 : in Natural;
                     X2 : in Natural;
                     Y2 : in Natural) is

      Origine_X : Natural := X1;
      Origine_Y : Natural := Y1;

      Destination_X : Natural := X2;
      Destination_Y : Natural := Y2;

   begin -- Line_To

      -- impossible de dessiner dans une fenêtre inexistante
      if not Fenetre_Initialisee then
         raise Fenetre_Non_Init;
      end if;

      -- Dessin de la ligne dans le décors sur le pixmap
      Draw_Line ( Fenetre.Pixmap, Fenetre.Defaut_Gc, Gint(X1), Gint(Y1),
                  Gint(X2), Gint(Y2));

      -- Calcul de la zone affectée par la ligne
      if X2 - X1 < 0 then
         Origine_X := X2;
         Destination_X := X1;
      end if;

      if Y2 - Y1 < 0 then
         Origine_Y := Y2;
         Destination_Y := Y1;
      end if;

      -- Affichage sur l'aire de dessin de l'espace affecté dans le pixmap
      Draw_Drawable (Drawable => Fenetre.Fenetre_Gdk,
                     Gc => Fenetre.Defaut_Gc,
                     Src => Fenetre.Pixmap,
                     Xsrc => Gint(Origine_X), Ysrc => Gint(Origine_Y),
                     Xdest => Gint(Origine_X), Ydest => Gint(Origine_Y),
                     Width => Gint (Destination_X),
                     Height => Gint (Destination_Y));

      -- Mise à jour des coordonnées courantes
      Cur_X := X2;
      Cur_Y := Y2;

      -- Traite les événements en attente
      Boucle_Dessin;
```

```
   end Line_To;


   ----------
   -- Line --
   ----------
   -- But : Dessine une ligne en partant du point courant
   --
   -- Paramètre(s) : DX => Déplacement sur l'axe des X pour le point arrivée
   --                DY => Déplacement sur l'axe des Y pour le point arrivée
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Line(DX : in Integer;
                  DY : in Integer) is
   begin

      Line_To ( Cur_X, Cur_Y, Cur_X + DX, Cur_Y + DY );

   end Line;

end Gtkspider;
```

## *E.3. Gtkspider-Draw.ads*

```
-- -----------------------------------------------------------------------------
-- Fichier       : gtkspider-draw.ads
-- Auteur        : Berthet Jeremy & Borer Reynald
-- Date          : 29.05.2005
--
-- But           : Fournis des fonctions graphiques supplémentaires à GtkSpider
--
-- Remarque(s)   : Utilise GtkAda
--
-- Modifications : Date / Auteur / Raison
--
-- Compilateur   : GNAT 3.15p
-- Licence       : Gnu General Public License v2
-- -----------------------------------------------------------------------------

package Gtkspider.Draw is

   -- Type de remplissage pour les formes.
   type tFill is ( fill, noFill );

   -- Type pour les couleur
   type tColor is record
      R : Integer;
      G : Integer;
      B : Integer;
   end record;

   -- définition de quelques couleurs
   black        : constant tColor := (  0,  0,  0);
   blue         : constant tColor := (  0,  0,255);
   green        : constant tColor := (  0,255,  0);
   cyan         : constant tColor := (  0,255,255);
   red          : constant tColor := (255,  0,  0);
   magenta      : constant tColor := (255,  0,255);
   brown        : constant tColor := (128, 64,  0);
   lightGray    : constant tColor := (192,192,192);
   darkGray     : constant tColor := (128,128,128);
   lightBlue    : constant tColor := (128,128,255);
   lightGreen   : constant tColor := (128,255,128);
   lightCyan    : constant tColor := (128,255,255);
   lightRed     : constant tColor := (255,128,128);
   lightMagenta : constant tColor := (255,128,255);
   yellow       : constant tColor := (255,255,  0);
   white        : constant tColor := (255,255,255);


   -------------------
   -- Get_Color_Pen --
   -------------------
   -- But : Indique la couleur utilisée pour les dessins
   --
   -- Paramètre(s) : -
   --
   -- Résultat : tColor
   --
   -- Exception(s) : -
   -- --
   function Get_Color_Pen return tColor;


   --------------------------
   -- Get_Color_Background --
   --------------------------
   -- But : Indique la couleur utilisée pour l'arrière plan du texte
   --
   -- Paramètre(s) : -
   --
```

```
   -- Résultat : tColor
   --
   -- Exception(s) : -
   -- --
   function Get_Color_Background return tColor;


   --------------------
   -- Get_Color_Text --
   --------------------
   -- But : Indique la couleur utilisée pour le texte
   --
   -- Paramètre(s) : -
   --
   -- Résultat : tColor
   --
   -- Exception(s) : -
   -- --
   function Get_Color_Text return tColor;


   -------------------
   -- Set_Color_Pen --
   -------------------
   -- But : Défini la couleur utilisée pour les dessins
   --
   -- Paramètre(s) : Color => la couleur voulue au format tColor
   --
   -- Exception(s) : Erreur_Inconnue => Problème lors de la création
   --                                   de la couleur
   -- --
   procedure Set_Color_Pen ( Color : in tColor );


   --------------------------
   -- Set_Color_Background --
   --------------------------
   -- But : Défini la couleur utilisée pour l'arrière plan du texte
   --
   -- Paramètre(s) : Color => la couleur voulue au format tColor
   --
   -- Exception(s) : Erreur_Inconnue => Problème lors de la création
   --                                   de la couleur
   -- --
   procedure Set_Color_Background ( Color : in tColor );


   --------------------
   -- Set_Color_Text --
   --------------------
   -- But : Défini la couleur utilisée pour le texte
   --
   -- Paramètre(s) : Color => la couleur voulue au format tColor
   --
   -- Exception(s) : Erreur_Inconnue => Problème lors de la création
   --                                   de la couleur
   -- --
   procedure Set_Color_Text ( Color : in tColor );


   ------------------
   -- Display_Text --
   ------------------
   -- But : Affiche un texte dans la fenêtre graphique à partir de la position
   --       courante
   --
   -- Paramètre(s) : Text => chaîne de caractères à afficher
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
```

```
   -- --
   procedure Display_Text ( Text : in String );


   ------------
   -- Circle --
   ------------
   -- But : Dessine un cercle dont le centre est à la position courante
   --
   -- Paramètre(s) : Radius => Rayon du cercle
   --                Filled => Indique si le cercle est plein ou si il n'y a
   --                          que le contour
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Circle ( Radius : in Natural;
                      Filled : in tFill);


   ---------
   -- Box --
   ---------
   -- But : Dessine un rectangle dont le coin supérieur gauche est à la position
   --       courante
   --
   -- Paramètre(s) : Width => Longueur du rectangle
   --                Height => Largeur du rectangle
   --                Filled => Indique si le rectangle est plein ou si il n'y a
   --                          que le contour
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Box ( Width  : in Integer;
                   Height : in Integer;
                   Filled : in tFill );


   ---------------
   -- Put_Pixel --
   ---------------
   -- But : Dessine un point (pixel) à la position courante
   --
   -- Paramètre(s) : -
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Put_Pixel;

end Gtkspider.Draw;
```

## *E.4. Gtkspider-Draw.adb*

```
-- ------------------------------------------------------------------------------
-- Fichier       : gtkspider-draw.adb
-- Auteur        : Berthet Jeremy & Borer Reynald
-- Date          : 29.05.2005
--
-- But           : Fournis des fonctions graphiques supplémentaires à GtkSpider
--
-- Remarque(s)   : Utilise GtkAda
--
-- Modifications : Date / Auteur / Raison
--
-- Compilateur   : GNAT 3.15p
-- Licence       : Gnu General Public License v2
-- ------------------------------------------------------------------------------
with Glib;             use Glib;

with Gdk.Drawable;     use Gdk.Drawable;
with Gdk.Color;        use Gdk.Color;
with Gtk.Widget;       use Gtk.Widget;

with Pango.Layout;     use Pango.Layout;


package body Gtkspider.Draw is

   -- Valeur max pour les valeurs RGB du type color
   Rgb_Max : constant := 255;
   Val_Conversion : constant Guint16 := Guint16'Last / Guint16 ( Rgb_Max );

   -- Variables stockant les couleurs
   Pen_Color        : Gdk_Color := Gdk.Color.Parse ( "black" );
   Background_Color : Gdk_Color := Gdk.Color.Parse ( "white" );
   Text_Color       : Gdk_Color := Gdk.Color.Parse ( "black" );


   -----------------
   -- Rgb_Guint16 --
   -----------------
   -- But : Converti une valeur RGB Integer (type tColor) dans le type Guint16
   --       pour les fonctions de couleurs dans GtkAda
   --
   -- Paramètre(s) : color => La couleur à convertir
   --
   -- Exception(s) : -
   -- --
   function Rgb_Guint16 ( Color : Integer ) return Guint16 is

   begin -- Rgb_Guint16

      if Color = 0 then
         return 0;

      elsif abs ( Color ) >= Rgb_Max then
         return Guint16'Last;

      else
         return Guint16 ( abs ( Color ) ) * Val_Conversion ;
      end if;

   end Rgb_Guint16;


   -----------------
   -- Guint16_Rgb --
   -----------------
   -- But : Converti une valeur Guint16 représentant une couleur RGB dans
   --       le type Integer pour utiliser le type tColor
```

```
   --
   -- Paramètre(s) : color => La couleur à convertir
   --
   -- Exception(s) : -
   -- --
   function Guint16_Rgb ( Color : Guint16 ) return Integer is

   begin -- Guint16_Rgb
      return Integer ( Color / Val_Conversion ) ;
   end Guint16_Rgb;


   -------------------
   -- Set_Color_Pen --
   -------------------
   -- But : Défini la couleur utilisée pour les dessins
   --
   -- Paramètre(s) : Color => la couleur voulue au format tColor
   --
   -- Exception(s) : Erreur_Inconnue => Problème lors de la création
   --                                   de la couleur
   -- --
   procedure Set_Color_Pen ( Color : in tColor ) is

   begin -- Set_Color_Pen

      -- Stocke la couleur dans la variable
      Set_Rgb ( Pen_Color, Rgb_Guint16 ( Color.R ),
                           Rgb_Guint16 ( Color.G ),
                           Rgb_Guint16 ( Color.B ) );

      -- Alloue la couleur
      Gdk.Color.Alloc ( Get_Default_Colormap, Pen_Color );

      -- Change la couleur définie dans le contexte graphique
      Set_Foreground ( Fenetre.Defaut_Gc, Pen_Color );

   exception
      when Wrong_Color =>
         raise Erreur_Inconnue;

   end Set_Color_Pen;


   --------------------------
   -- Set_Color_Background --
   --------------------------
   -- But : Défini la couleur utilisée pour l'arrière plan du texte
   --
   -- Paramètre(s) : Color => la couleur voulue au format tColor
   --
   -- Exception(s) : Erreur_Inconnue => Problème lors de la création
   --                                   de la couleur
   -- --
   procedure Set_Color_Background ( Color : in tColor ) is

   begin -- Set_Color_Background

      -- Stocke la couleur dans la variable
      Set_Rgb ( Background_Color, Rgb_Guint16 ( Color.R ),
                                  Rgb_Guint16 ( Color.G ),
                                  Rgb_Guint16 ( Color.B ) );

      -- Alloue la couleur
      Gdk.Color.Alloc ( Get_Default_Colormap, Background_Color );

      -- Change la couleur définie dans le contexte graphique
      Set_Background ( Fenetre.Defaut_Gc, Background_Color );
```

```
   exception
      when Wrong_Color =>
         raise Erreur_Inconnue;

   end Set_Color_Background;


   --------------------
   -- Set_Color_Text --
   --------------------
   -- But : Défini la couleur utilisée pour le texte
   --
   -- Paramètre(s) : Color => la couleur voulue au format tColor
   --
   -- Exception(s) : Erreur_Inconnue => Problème lors de la création
   --                                   de la couleur
   -- --
   procedure Set_Color_Text ( Color : in tColor ) is

   begin -- Set_Color_Background

      -- Stocke la couleur dans la variable
      Set_Rgb ( Text_Color, Rgb_Guint16 ( Color.R ),
                            Rgb_Guint16 ( Color.G ),
                            Rgb_Guint16 ( Color.B ) );

      -- Alloue la couleur
      Gdk.Color.Alloc ( Get_Default_Colormap, Text_Color );

   exception
      when Wrong_Color =>
         raise Erreur_Inconnue;

   end Set_Color_Text;


   -------------------
   -- Get_Color_Pen --
   -------------------
   -- But : Indique la couleur utilisée pour les dessins
   --
   -- Paramètre(s) : -
   --
   -- Résultat : tColor
   --
   -- Exception(s) : -
   -- --
   function Get_Color_Pen return tColor is

   begin -- Get_Color_Pen

      return ( Guint16_Rgb ( Gdk.Color.Red   ( Pen_Color ) ),
               Guint16_Rgb ( Gdk.Color.Green ( Pen_Color ) ),
               Guint16_Rgb ( Gdk.Color.Blue  ( Pen_Color ) ) );

   end Get_Color_Pen;


   --------------------------
   -- Get_Color_Background --
   --------------------------
   -- But : Indique la couleur utilisée pour l'arrière plan du texte
   --
   -- Paramètre(s) : -
   --
   -- Résultat : tColor
   --
   -- Exception(s) : -
   -- --
```

```
   function Get_Color_Background return tColor is

   begin -- Get_Color_Background

      return ( Guint16_Rgb ( Gdk.Color.Red   ( Background_Color ) ),
               Guint16_Rgb ( Gdk.Color.Green ( Background_Color ) ),
               Guint16_Rgb ( Gdk.Color.Blue  ( Background_Color ) ) );

   end Get_Color_Background;


   --------------------
   -- Get_Color_Text --
   --------------------
   -- But : Indique la couleur utilisée pour le texte
   --
   -- Paramètre(s) : -
   --
   -- Résultat : tColor
   --
   -- Exception(s) : -
   -- --
   function Get_Color_Text return tColor is

   begin -- Get_Color_Text

      return ( Guint16_Rgb ( Gdk.Color.Red   ( Text_Color ) ),
               Guint16_Rgb ( Gdk.Color.Green ( Text_Color ) ),
               Guint16_Rgb ( Gdk.Color.Blue  ( Text_Color ) ) );

   end Get_Color_Text;


   ------------
   -- Circle --
   ------------
   -- But : Dessine un cercle dont le centre est à la position courante
   --
   -- Paramètre(s) : Radius => Rayon du cercle
   --                Filled => Indique si le cercle est plein ou si il n'y a
   --                          que le contour
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Circle ( Radius : in Natural;
                      Filled : in tFill ) is

   begin -- Circle

      -- impossible de dessiner dans une fenêtre inexistante
      if not Fenetre_Initialisee then
         raise Fenetre_Non_Init;
      end if;

      Draw_Arc ( Drawable => Fenetre.Pixmap,
                 Gc => Fenetre.Defaut_Gc,
                 Filled => Boolean'Val ( abs ( tFill'Pos ( Filled ) - 1 ) ),
                 X => Gint ( Cur_X - Radius ),
                 Y => Gint ( Cur_Y - Radius ),
                 Width => Gint ( 2*Radius ),
                 Height => Gint ( 2*Radius ),
                 Angle1 => Gint ( 0 ),
                 Angle2 => Gint ( 23040 ) ); -- Angle en 1/64 de degrés

      Draw_Drawable ( Drawable => Fenetre.Fenetre_Gdk,
                      Gc => Fenetre.Defaut_Gc,
                      Src => Fenetre.Pixmap,
                      Xsrc => Gint ( Cur_X - Radius ),
                      Ysrc => Gint ( Cur_Y - Radius ),
```

```
                    Xdest => Gint ( Cur_X - Radius ),
                    Ydest => Gint ( Cur_Y - Radius ),
                    Width => Gint ( 2* ( Radius + 1 ) ),
                    Height => Gint (2* ( Radius + 1 ) ) );

      --Traite les événements en attente
      Boucle_Dessin;

   end Circle;


   ---------
   -- Box --
   ---------
   -- But : Dessine un rectangle dont le coin supérieur gauche est à la position
   --       courante
   --
   -- Paramètre(s) : Width => Longueur du rectangle
   --                Height => Largeur du rectangle
   --                Filled => Indique si le rectangle est plein ou si il n'y a
   --                          que le contour
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Box ( Width  : in Integer;
                   Height : in Integer;
                   Filled : in tFill ) is

   begin -- Box

      -- impossible de dessiner dans une fenêtre inexistante
      if not Fenetre_Initialisee then
         raise Fenetre_Non_Init;
      end if;

      if Width < 0 then
         Cur_X := Cur_X - abs Width;
      end if;

      if Height < 0 then
         Cur_Y := Cur_Y - abs Height;
      end if;

      Draw_Rectangle ( Drawable => Fenetre.Pixmap,
                       Gc => Fenetre.Defaut_Gc,
                       Filled => Boolean'Val ( abs ( tFill'Pos ( Filled ) - 1 ) ),
                       X => Gint ( Cur_X ), Y => Gint ( Cur_Y ),
                       Width => Gint ( abs Width ),
                       Height => Gint ( abs Height ) );

      Draw_Drawable (Drawable => Fenetre.Fenetre_Gdk,
                     Gc => Fenetre.Defaut_Gc,
                     Src => Fenetre.Pixmap,
                     Xsrc => Gint ( Cur_X ),
                     Ysrc => Gint ( Cur_Y ),
                     Xdest => Gint ( Cur_X ),
                     Ydest => Gint ( Cur_Y ),
                     Width => Gint ( abs Width + 1 ),
                     Height => Gint ( abs Height + 1 ) );

      --Traite les événements en attente
      Boucle_Dessin;

   end Box;


   ---------------
   -- Put_Pixel --
   ---------------
```

```
   -- But : Dessine un point (pixel) à la position courante
   --
   -- Paramètre(s) : -
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Put_Pixel is

   begin -- Put_Pixel

      -- impossible de dessiner dans une fenêtre inexistante
      if not Fenetre_Initialisee then
         raise Fenetre_Non_Init;
      end if;

      Draw_Point ( Drawable => Fenetre.Pixmap,
                   Gc => Fenetre.Defaut_Gc,
                   X => Gint ( Cur_X ),
                   Y =>  Gint ( Cur_Y ) );

      Draw_Drawable ( Drawable => Fenetre.Fenetre_Gdk,
                      Gc => Fenetre.Defaut_Gc,
                      Src => Fenetre.Pixmap,
                      Xsrc => Gint ( Cur_X ),
                      Ysrc => Gint ( Cur_Y ),
                      Xdest => Gint ( Cur_X ),
                      Ydest => Gint ( Cur_Y ),
                      Width => 1,
                      Height => 1);

      --Traite les événements en attente
      Boucle_Dessin;

   end Put_Pixel;


   ------------------
   -- Display_Text --
   ------------------
   -- But : Affiche un texte dans la fenêtre graphique à partir de la position
   --       courante
   --
   -- Paramètre(s) : Text => chaîne de caractères à afficher
   --
   -- Exception(s) : Fenetre_Non_Init => la fenêtre n'est pas initialisée
   -- --
   procedure Display_Text ( Text : in String ) is

         Layout : Pango.Layout.Pango_Layout;

   begin -- Display_Text

      -- impossible de dessiner dans une fenêtre inexistante
      if not Fenetre_Initialisee then
         raise Fenetre_Non_Init;
      end if;

      -- Utilise la couleur pour la police
      Set_Foreground ( Fenetre.Defaut_Gc, Text_Color );

      -- Layout pour pango
      Layout := Create_Pango_Layout ( Fenetre );

      -- Police de caractère
      Set_Font_Description ( Layout, Fenetre.Police );

      -- Texte à écrire
      Set_Text ( Layout, Text );
```

```
      -- Dessine dans la fenêtre
      Draw_Layout ( Drawable => Fenetre.Pixmap,
                    Gc => Fenetre.Defaut_Gc,
                    X => Gint ( Cur_X ),
                    Y => Gint ( Cur_Y ),
                    Layout => Layout );

      -- Affiche le pixmap dans la fenêtre
      Draw_Drawable ( Drawable => Fenetre.Fenetre_Gdk,
                      Gc => Fenetre.Defaut_Gc,
                      Src => Fenetre.Pixmap,
                      Xsrc => Gint ( Cur_X ),
                      Ysrc => Gint ( Cur_Y ),
                      Xdest => Gint ( Cur_X ),
                      Ydest => Gint ( Cur_Y ),
                      Width => -1,
                      Height => -1);

      -- Remet la couleur d'avant-plan
      Set_Foreground ( Fenetre.Defaut_Gc, Pen_Color );

      --Traite les événements en attente
      Boucle_Dessin;

   end Display_Text;

end GtkSpider.Draw;
```

# Annexe F. Tests effectués

Nous avons testé les sous-programmes des paquetages tout au long du développement de ceux-ci, et nous avons assez fréquemment comparé le résultat affiché avec celui de la librairie *Spider*.

Pour avoir un aperçu des tests que nous avons effectués, nous avons un code source dans la documentation utilisateur qui teste toutes les fonctionnalités de *GtkSpider.*