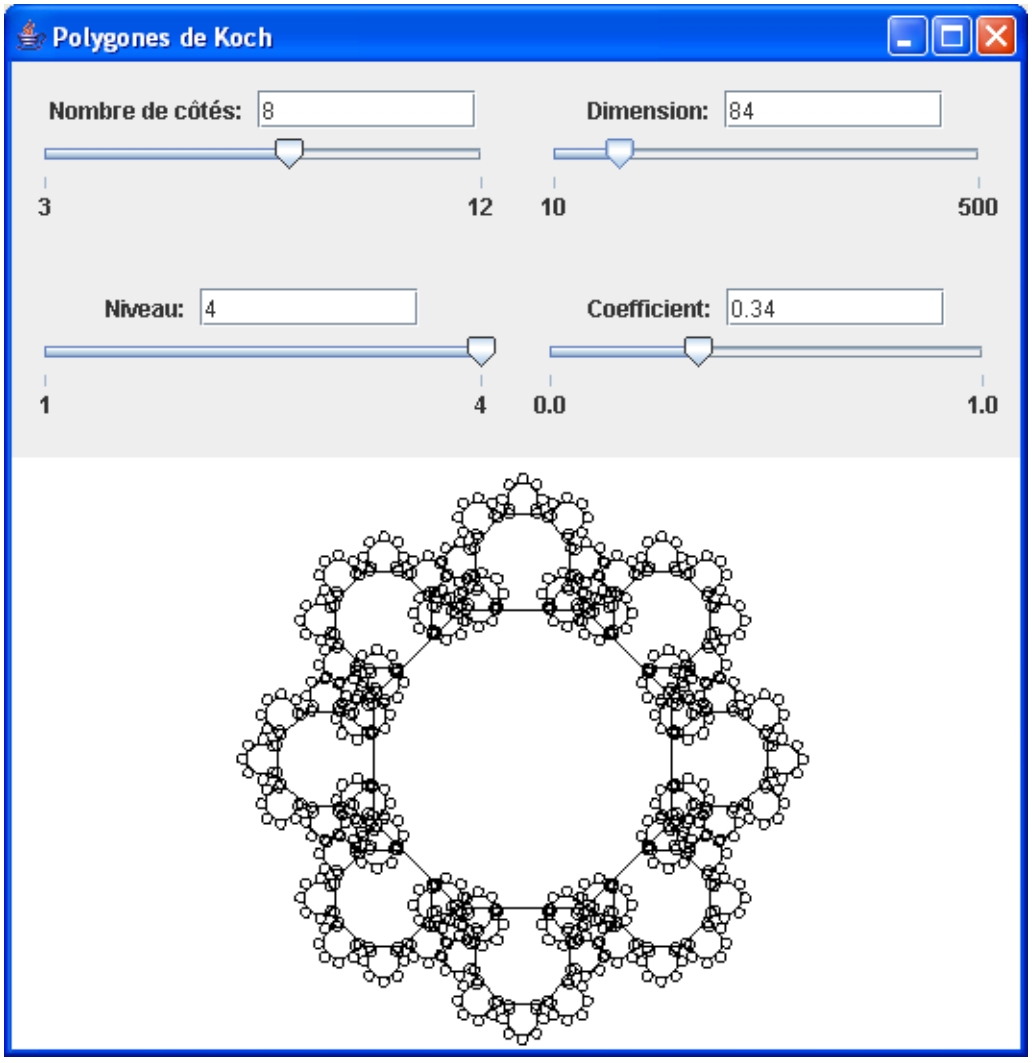
Expresso
Manuel de l'utilisateur

Version Préliminaire partielle (17/01/2006)



André Boileau
boileau.andre@uqam.ca

Maurice Garançon
garanson@math.uqam.ca

Table des matières

1. Introduction

Expresso a vu le jour un peu par hasard. L'un des auteurs cherchait un langage de programmation pour initier ses étudiants, futurs enseignants en mathématiques, à la programmation. Ses exigences étaient simples, il fallait un langage puissant, multi-plateforme, facile à apprendre et gratuit. Pendant ce temps l'autre auteur découvrait Java et l'environnement de programmation BlueJ.
D'un côté Java fournissait le langage puissant, multi-plateforme, gratuit, en plus compatible avec le Web mais malheureusement trop complexe pour les besoins envisagés.
D'un autre côté BlueJ fournit un environnement de programmation Java multi-plateforme, simple à utiliser et doté de capacités largement suffisante pour nos besoins.
Si proches du but nous ne pouvions pas abandonner et c'est ainsi que nous avons décidé de créer Expresso qui est un langage basé sur Java qui permet cependant de programmer sans objets.

Expresso met à la disposition de L'utilisateur un interface configurable composé minimalement d'une page graphique dans laquelle on peut dessiner à l'aide de commandes de style « géométrie de la tortue » et aussi de commandes de style « géométrie analytique ». On peut aussi compléter cet interface, à l'aide de commandes élémentaires, en y ajoutant des menus, des boutons, des glissières, une zone de texte, en utilisant des fenêtres de dialogue ou en réagissant aux clics ou aux « glisser » de souris.
Ça donne un interface limité mais facile à construire et qui, pour l'instant, nous semble assez complet pour nos besoins.

Nous avons dû faire des choix et le temps dira si ces choix étaient bons. De toute façon nous sommes ouverts à toute suggestion et aux critiques .. constructives.

2.L'environnement de programmation

La programmation avec Expresso se fait dans l'environnement gratuit BlueJ. Dans les paragraphes qui suivent nous expliquons comment se procurer BlueJ et comment le configurer pour rendre la programmation Expresso possible.

2.1. Comment se procurer « BlueJ »

BlueJ peut être télécharger gratuitement à partir du site suivant:

http://www.bluej.org/download/download.html

On conseille d'utiliser la version 2.0.2 ou une version plus récente.

Pour Windows :

Il est nécessaire que Java soit installé. Il faut la version J2SE 1.4.2 ou une version plus récente (il faut une version appelée JDK ou SDK pas JRE ou la version pour Netbeans).

Pour Macintosh :

Une version récente de Java est installée par défaut.

2.2. Comment installer « BlueJ »

Pour Windows il suffit de double cliquer l'installateur « bluejsetup.exe » et de suivre les instructions.

Pour Mac ..

2.3. Comment franciser « BlueJ »

Une fois BlueJ installé on peut le franciser, pour cela il faut trouver le fichier « **bluej.defs** ».

Pour Windows, si lors de l'installation de BlueJ vous avez accepté les paramètres proposés par défaut il doit se trouver dans « **C:/Program Files/BlueJ/lib/** ».

Pour Mac, il faut d'abord trouver l'icône de **BlueJ** qui normalement se trouve dans le dossier « **Applications** ». Une fois l'icône de **BlueJ** localisée, il faut la cliquer avec le bouton droit (Ou contrôle-clic du bouton gauche) et dans le menu contextuel qui apparaît choisir « **Afficher le contenu du paquet** ». On obtient alors un dossier « **Contents** » qui contient un dossier « **Ressources** » qui contient lui-même un dossier « **Java** ». Le fichier « **bluej.defs** » doit se trouver dans ce dernier dossier.

Le fichier « **bluej.defs** » doit maintenant être modifié. Pour cela il faut l'ouvrir dans un éditeur de texte et trouver la ligne

```
#bluej.langage=french
```

sur cette ligne enlever le symbole # pour quelle devienne

```
bluej.langage=french
```

et d'autre part ajouter le symbole # au début de toute autre ligne « bluej.langage=... » qui n'en possède pas.

2.4. Configurer le mode de compilation de BlueJ

Pour rendre les applications et applets produits avec **Expresso** compatibles avec le plus grand

nombre de plateformes et versions de Java il convient de modifier le mode de compilation de **BlueJ**. Ce n'est pas plus compliqué que pour franciser **BlueJ**. Il faut, comme ci-dessus, éditer le fichier « **bluej.defs** » et dans ce fichier trouver la ligne

```
bluej.compiler.options=-source 1.4
```

(Noter que le nombre qui suit « **source** » peut être différent suivant la version de BlueJ que vous avez installé.)

Commentez cette ligne en ajoutant # au début

```
#bluej.compiler.options=-source 1.4
```

puis ajoutez au dessous la ligne suivante

```
bluej.compiler.options=-source 1.3 -target 1.3
```

Enregistrez et c'est terminé.

2.5. Comment installer le paquetage « Expresso »

En téléchargeant « **Expresso** » on obtient plusieurs fichiers dont « **Expresso.ja**r ». La localisation de ce fichier sur le disque dur n'a pas d'importance mais nous conseillons de le placer dans le même dossier que **BlueJ** de façon à pouvoir le retrouver facilement.

Pour rendre « Expresso » disponible dans BlueJ il faut:

- ouvrir BlueJ

- sélectionner le menu Outil/Préférences

- dans la fenêtre qui apparaît cliquer sur le tab « Bibliothèques »

- dans le nouvel affichage cliquer sur le bouton « Ajouter »

- dans la fenêtre de sélection qui s'ouvre naviguer pour localiser « **Expresso.jar** » et le sélectionner. Cliquer sur « Ouvrir » (ou « Choisir » selon le système utilisé).

C'est fait. Au prochain démarrage de **BlueJ**, **Expresso** sera disponible.

2.6. Comment installer les « templates »: « Expresso » et « AppletExpresso »

En téléchargeant « **Expresso** » on obtient plusieurs fichiers dont « **Expresso.tmpl** » et « **AppletExpresso.tmpl** », il reste à placer ces fichiers.

Localiser d'abord le fichier « **bluej.defs** » comme en 2.3 dépendant de la plateforme utilisée. Dans le même dossier que « **bluej.defs** » se trouve le dossier « french » (si vous utilisez une autre langue que le français choisissez le dossier correspondant). Dans ce dossier « **french** » il y a un dossier « **templates** » et dans ce dernier un dossier « **newclass** ». Il faut copier les fichier « **Expresso.tmpl** » et « **AppletExpresso.tmpl** » dans ce dossier « newclass ».

Une fois toutes ces étapes parcourues vous êtres prêt à programmer en « **Expresso** » à l'aide de **BlueJ.**

3. La programmation issue de Java

À venir.

En attendant on peut consulter par exemple le chapitre 1 du livre « Java et la programmation objet » de Divay aux éditions Dunod.

3.1. Les opérateurs

3.2. Les variables et les types

3.2.1. Déclaration

3.2.2. Affectation

3.3. Les tableaux

3.4. Les conditionnelles

3.4.1. If

3.4.2. If...else...

3.4.3. Switch...case...

3.5. Les boucles

3.5.1. For

3.5.2. While

4. Le langage Expresso

Le langage « Expresso » est construit au dessus de Java dans le but de fournir un langage simplifié. Il permet tout d'abord de construire facilement un interface graphique comprenant une zone de dessin, une zone de texte, des menus, des boutons et des glissières. Cette interface est configurable et seule la zone de dessin est toujours présente, la présence et le comportement des autres composants sont laissés au gré de l'utilisateur à travers des instructions d'utilisation élémentaires.
Le langage lui même comprend, en particulier, des instructions de dessin et de gestion de liste inspirées de la géométrie de la tortue du langage Logo. Nous passons ici en revue ces différentes instructions en précisant leurs paramètres, leur syntaxe et leur effet.

4.1.Le langage de la tortue

On peut dessiner à l'écran en imaginant qu'on commande à un animal électronique, la tortue, qui est caractérisé par:

- sa position donnée par des coordonnées x et y,
- sa direction (ou son cap) donnée par un angle avec l'axe vertical et mesuré positivement dans le sens horaire.

La tortue transporte avec elle un crayon qui peut être levé ou baissé et qui, s'il est baissé, laissera une trace de couleur sur le parcours de la tortue.
Voici donc les commandes qui permettent de commander la tortue et par le fait même de dessiner dans la zone de dessin.

Déplacements .

avance(d) *abréviation* **av(d)**

Le paramètre d est un nombre entier ou décimal et cette instruction fait avancer la tortue de d pixels dans sa direction courante. Si le crayon est baissé un segment sera tracé entre les positions initiale et finale de la tortue.

recule(d) *abréviation* **re(d)**

Le paramètre d est un nombre entier ou décimal et cette instruction fait reculer la tortue de d pixels par rapport à sa direction courante. Si le crayon est baissé un segment sera tracé entre les positions initiale et finale de la tortue.

Position

fixePos(x, y) *abréviation* **fPos(x, y)**

Les paramètres x et y sont des nombres entiers ou décimaux. L'instruction modifie la position de la tortue pour qu'elle prenne les coordonnées x et y. La tortue trace un segment entre les positions initiale et finale si le crayon est baissé.

posX()

Retourne un nombre décimal représentant l'abscisse de la position de la tortue.

posY()

Retourne un nombre décimal représentant l'ordonnée de la position de la tortue.

Orientation

droite(angle) *abréviation* **dr(angle)**

Le paramètre angle est un nombre entier ou décimal et cette instruction change la direction de la tortue en lui ajoutant la valeur angle. Donc si angle est positif la tortue tourne vers sa droite.

cap()

Retourne un nombre décimal (type double) qui représente la direction courante de la tortue, c'est à dire l'angle qu'elle fait avec la verticale mesuré positivement dans le sens horaire.

fixe Cap(d) *abréviation* **fCap(d)**

Le paramètre d est un nombre entier ou décimal et cette instruction modifie la direction de la tortue pour qu'elle fasse un angle de d avec la verticale.

gauche(angle) *abréviation* **ga(angle)**

Le paramètre angle est un nombre entier ou décimal et cette instruction change la direction de la tortue en lui retranchant la valeur angle. Donc si angle est positif la tortue tourne vers sa gauche.

vers(x, y)

Les paramètres x et y sont des nombres entiers ou décimaux et cette instruction retourne un nombre décimal représentant l'angle que doit faire la tortue avec la verticale pour pointer en direction du point de coordonnées (x, y).

Le crayon

baisseCrayon() *abréviation* **bc()**

Met la tortue dans un état ou ses déplacements laisseront une trace.

couleurCrayon(c) *abréviation* **cc(c)**

couleurCrayon(r, v, b) *abréviation* **cc(r, g, b)**

Cette instruction a deux syntaxes, elle accepte pour paramètres une couleur c prédéfinie (noir, blanc, jaune, vert, rouge, bleu, etc..) ou trois entiers compris entre 0 et 255 définissant une couleur RGB. Les paramètres r, v, b donnant les quantités de rouge, vert et bleu respectivement. Comme son nom l'indique, son effet est de changer la couleur avec laquelle la tortue trace.

leveCrayon() *abréviation* **lc()**

Met la tortue dans un état ou ses déplacements ne laisseront aucune trace.

retourneCouleurCrayon()

Retourne une couleur (type Color) représentant la couleur courante du crayon.

RetourneCouleurCrayonR() *abréviation* **retCCr()**

Retourne un entier représentant la quantité de rouge dans la couleur courante du crayon.

retourneCouleurCrayonV() *abréviation* **retCCv()**

Retourne un entier représentant la quantité de vert dans la couleur courante du crayon.

retourneCouleurCrayonB() *abréviation* **retCCb()**

Retourne un entier représentant la quantité de bleu dans la couleur courante du crayon.

retourneModeInverse()

Retourne un booléen, « true » si le crayon est en mode de tracé inverse et « false » autrement.

retourneTailleCrayon() *abréviation* **retTC()**

Retourne un nombre décimal (type double) représentant la taille actuelle du crayon.

tailleCrayon(ep) *abréviation* **tc(ep)**

Le crayon est de forme carré, cette instruction change la taille de ce carré pour pour que la longueur de son côté devienne ep, où ep est un nombre entier ou décimal.

traceInverse()

Met le crayon dans un mode où lorsqu'il trace deux fois un pixel avec la même couleur, ce pixel reprend la couleur qu'il avait avant les deux tracés. Autrement dit le second tracé annule l'effet du premier.

traceNormal()

Annule le mode de tracé inverse.

4.2.La zone graphique

centreTortue()

Cette instruction place l'origine des coordonnées de la tortue au centre de la zone graphique.

ecrisGraphique(x, y, texte)

Les paramètres x et y sont des nombres entiers ou décimaux et le paramètre texte est une chaîne de caractères. Cette instruction écris le contenu du paramètre « texte » dans la zone graphique à partir du point (x, y) et en utilisant la couleur du crayon.

fixeOrigineTortue(x, y)

Les paramètres x et y sont des nombres entiers ou décimaux qui représentent un point de la zone graphique en coordonnées de cette zone (c'est à dire origine au coin en haut à gauche, axe horizontal pointant à droite et axe vertical pointant vers le bas). Cette instruction place l'origine des coordonnées de la tortue au point (x, y) de la zone graphique.

hauteurZoneGraphique()

Retourne un nombre entier qui représente la hauteur actuelle de la zone graphique (en pixels).

largeurZoneGraphique()

Retourne un nombre entier qui représente la largeur actuelle de la zone graphique (en pixels).

miseAJourGraphique()

Force un rafraîchissement de la zone graphique.

origineTortueX()

Retourne un nombre décimal (type double) qui représente l'abcisse de l'origine des coordonnées de la tortue considérée comme un point de la zone graphique.

origineTortueY()

Retourne un nombre décimal (type double) qui représente l'ordonnée de l'origine des coordonnées de la tortue considérée comme un point de la zone graphique.

videGraphique()

Efface le contenu de la zone graphique.

Pour tracer des formes pleines avec la tortue

couleurRemplissage(c)

couleurRemplissage(r, v, b)

Cette instruction a deux syntaxes, elle accepte pour paramètres une couleur **c** prédéfinie (noir, blanc, jaune, vert, rouge, bleu, etc..) ou trois entiers compris entre 0 et 255 définissant une couleur RGB. Les paramètres **r**, **v**, **b** donnant les quantités de rouge, vert et bleu respectivement. Comme son nom l'indique, son effet est de changer la couleur avec laquelle les figures seront remplies.

debutRemplir()

finRemplir()

Toutes les instructions de déplacement de la tortue rencontrées entre debutRemplir() et finRemplir() contribuent à créer un polygone qui sera rempli avec la couleur de remplissage courante.

retourneCouleurRemplissage()

Retourne la couleur (type Color) actuelle de remplissage.

RetourneCouleurRemplissageR() *abréviation* **retCRr()**

Retourne un entier représentant la quantité de rouge dans la couleur de remplissage actuelle.

retourneCouleurRemplissageV() *abréviation* **retCRv()**

Retourne un entier représentant la quantité de vert dans la couleur de remplissage actuelle.

retourneCouleurRemplissageB() *abréviation* **retCRb()**

Retourne un entier représentant la quantité de bleu dans la couleur de remplissage actuelle.

Dessiner sans la tortue

Toutes les instructions ci-dessous tracent avec la couleur du crayon de la tortue, à l'exception de celles qui tracent des formes pleines qui utilisent la couleur de remplissage. De plus tous les paramètres représentant des coordonnées sont tous relatifs aux coordonnées de la tortue.

arc(cx, cy, r, angleDeb, angleFin)

Les paramètres sont tous des nombres entiers ou décimaux. Cette instruction trace un arc de cercle centré en (cx, cy) de rayon r débutant à l'angle angleDeb et finissant à l'angle angleFin. Le centre et les angles sont tous relatifs aux coordonnées de la tortue.

cercle(cx, cy, r)

Les paramètres sont tous les trois des nombres entiers ou décimaux. Cette instruction trace un cercle de rayon r centré en (cx, cy).

disque(cx, cy, r)

Les paramètres sont tous les trois des nombres entiers ou décimaux. Cette instruction trace un disque de rayon r centré en (cx, cy) en utilisant la couleur de remplissage.

ellipse(x1, y1, x2, y2)

Les paramètres sont tous les quatre des nombres entiers ou décimaux. Cette instruction trace une ellipse inscrite dans le rectangle dont une diagonale va du point (x1, y1) au point (x2, y2).

rectangle(x1, y1, x2, y2)

Les paramètres sont tous les quatre des nombres entiers ou décimaux. Cette instruction trace un rectangle dont une diagonale va du point (x1, y1) au point (x2, y2).

rectanglePlein(x1, y1, x2, y2)

Les paramètres sont tous des nombres entiers ou décimaux. Cette instruction trace un rectangle plein dont une diagonale va du point (x1, y1) au point (x2, y2) et en utilisant la couleur de remplissage.

segment(x1, y1, x2, y2)

Les paramètres sont tous les quatre des nombres entiers ou décimaux. Cette instruction trace un segment allant du point (x1, y1) au point (x2, y2). Il faut noter que lorsque cette instruction se retrouve entre debutRemplir() et finRemplir() elle contribue à définir les polygones à remplir.

4.3.La zone de texte

ecris(texte)

Le paramètre texte peut être une chaîne de caractères, un entier, un décimal, un booléen ou une liste. Cette instruction écrit le contenu du paramètre texte dans la zone graphique.

ecrisRC(texte)

Le paramètre texte peut être une chaîne de caractères, un entier, un décimal, un booléen ou une liste. Cette instruction écrit le contenu du paramètre texte dans la zone graphique et le fait suivre d'un retour à la ligne.

fixerProportionZoneTexte(d)

Le paramètre d est un nombre entier ou décimal entre 0 et 1 (0 <= d <= 1). Cette instruction fixe la proportion de la fenêtre occupée par la zone de texte à (d * 100)%.

tailleTexte(t)

Le paramètre t est un nombre entier et cette instruction fixe la taille du texte de la zone texte à t points.

videTexte()

Efface tout le texte présent dans la zone de texte.

4.4.Le langage des Listes

Création

listeVide()

Retourne une liste vide.

vecteurVersListe(tableau)

Le paramètre *tableau* est un tableau d'entiers, de décimaux, de booléens ou de chaînes de caractères. Cette instruction retourne une liste contenant les éléments du tableau.

Ajout d'éléments

insereItem(elem, index, liste)

Le paramètre *elem* peut être un entier, un décimal, une chaîne de caractères, un booléen ou une liste, le paramètre *liste* doit être une liste. Cette instruction retourne une copie de la liste *liste* avec l'élément *elem* ajouté a la position *index*. *(version destructive: itemPut(elem, index, liste))*

metsPremier(elem, liste) *abréviation* **mp(elem, liste)**

Le paramètre *elem* peut être un entier, un décimal, une chaîne de caractères, un booléen ou une liste, le paramètre *liste* doit être une liste. Cette instruction retourne une copie de la liste *liste* avec l'élément *elem* ajouté au début. *(version destructive: fPut(elem, liste))*

metsDernier(elem, liste) *abréviation* **md(elem, liste)**

Le paramètre *elem* peut être un entier, un décimal, une chaîne de caractères, un booléen ou une liste, le paramètre *liste* doit être une liste. Cette instruction retourne une copie de la liste *liste* avec l'élément *elem* ajouté a la fin. *(version destructive: lPut(elem, liste))*

Remplacement d'éléments

remplaceItem(elem, index, liste)

Le paramètre *elem* peut être un entier, un décimal, une chaîne de caractères, un booléen ou une liste, le paramètre *liste* doit être une liste. Cette instruction retourne une copie de la liste *liste* avec l'élément *elem* a la position *index* à la place de l'élément qui s'y trouvait précédemment. *(version destructive: replaceItem(elem, index, liste))*

Lecture d'éléments

dernier(liste) *abréviation* **de(liste)**

Le paramètre *liste* doit être une liste. Cette instruction retourne le dernier élément de la liste *liste*. La liste *liste* n'est pas modifiée.

item(index, liste) *abréviation* **pr(liste)**

Le paramètre *liste* doit être une liste. Cette instruction retourne l'élément de la liste *liste* qui se trouve à la position *index*. La liste *liste* n'est pas modifiée.

premier(liste) *abréviation* **pr(liste)**

Le paramètre *liste* doit être une liste. Cette instruction retourne le premier élément de la liste *liste*. La liste *liste* n'est pas modifiée.

Suppression d'éléments

saufDernier(liste) *abréviation* **sd(liste)**

Le paramètre *liste* doit être une liste. Cette instruction retourne une copie de la liste *liste* sans son dernier élément. *(version destructive: butFirst(liste))*

saufPremier(liste) *abréviation* **sp(liste)**

Le paramètre *liste* doit être une liste. Cette instruction retourne une copie de la liste *liste* sans son premier élément. *(version destructive: butLast(liste))*

supprimerItem(index, liste)

Le paramètre *liste* doit être une liste. Cette instruction retourne une copie de la liste *liste* sans l'élément qui se trouvait à la position index. *(version destructive: butItem(index, liste))*

Les prédicats

booleenP(elem)

Le paramètre *elem* doit être de type Liste. Cette instruction retourne un booléen : *true* si *elem* est un booléen et *false* autrement. Si *elem* est un booléen on récupère sa valeur avec l'instruction valBool(elem).

chaineP(elem)

Le paramètre *elem* doit être de type Liste. Cette instruction retourne un booléen : *true* si *elem* est une chaîne de caractères et *false* autrement. Si *elem* est une chaîne de caractères on récupère sa valeur avec l'instruction valChaine(elem).

decimalP(elem)

Le paramètre *elem* doit être de type Liste. Cette instruction retourne un booléen : *true* si *elem* est un décimal et *false* autrement. Si *elem* est un décimal on récupère sa valeur avec l'instruction valDec(elem).

elementP(elem, liste)

Le paramètre *elem* peut être un entier, un décimal, une chaîne de caractères ou une liste. Le paramètre *liste* doit être une liste. Cette instruction retourne un booléen : *true* si *elem* est un élément de la liste *liste* et *false* autrement.

entierP(elem)

Le paramètre *elem* doit être de type Liste. Cette instruction retourne un booléen : *true* si *elem* est un entier et *false* autrement. Si *elem* est un entier on récupère sa valeur avec l'instruction valEnt(elem).

listeP(elem)

Le paramètre *elem* doit être de type Liste. Cette instruction retourne un booléen : *true* si *elem* est une liste et *false* autrement.

nombreP(elem)

Le paramètre *elem* doit être de type Liste. Cette instruction retourne un booléen : *true* si *elem* est un nombre et *false* autrement. Si *elem* est un nombre on peut récupérer sa valeur avec l'instruction valDec(elem).

videP(elem)

Le paramètre *elem* doit être une liste. Cette instruction retourne un booléen : *true* si *elem* est une liste vide et *false* autrement.

Divers

clone(liste)

Le paramètre *liste* doit être une liste. Cette instruction retourne une copie de la liste *liste*.

compte(liste)

Le paramètre *liste* doit être une liste. Cette instruction retourne un le nombre d'élément dans *liste*.

phrase(liste1, liste2)

Les paramètres *liste1 et liste2* doivent être des listes. Cette instruction retourne une copie de la concaténation des deux listes. *(version destructive sentence(liste1, liste2))*

type(elem)

Le paramètre *elem* doit être de type Liste. Cette instruction retourne un entier représentant le type de *elem* (1 pour entier, 2 pour décimal, 3 pour chaîne, 4 pour booléen, 5 pour liste).

valBool(elem)

Le paramètre *elem* doit être de type Liste et représenter un booléen. Cette instruction retourne le booléen représenté par *elem*.

valChaine(elem)

Le paramètre *elem* doit être de type Liste et représenter une chaîne. Cette instruction retourne la chaîne représentée par *elem*.

valDec(elem)

Le paramètre *elem* doit être de type Liste et représenter un décimal. Cette instruction retourne la valeur décimale de *elem*.

valEnt(elem)

Le paramètre *elem* doit être de type Liste et représenter un entier. Cette instruction retourne la valeur entière de *elem*.

4.5.Divers

appletP

Cette instruction retourne *« true »* si l'exécution à lieu dans un applet et *« false »* autrement.

attendre(n)

n est un entier et cette instruction laisse passer n millisecondes.

bip()

Instruction qui produit un son « bip ».

couleurRVB(r, v, b)

Les paramètres r, v, b sont des entiers entre 0 et 255. Cette instruction retourne une couleur (type Color) dont r, v, b représentent les quantités de rouge, vert et bleu.

egalesP(ch1, ch2)

Retourne vrai si les chaînes ch1 et ch2 sont égales et faux autrement.

hasard()

Cette instruction retourne un nombre décimal d (type double) aléatoire entre 0 et 1 (0 <= d <1).

hasard(min, max)

Les paramètres min et max sont des entiers (min <= max). Cette instruction retourne un entier n aléatoire entre min et max(min <= n <= max).

interruption()

Retourne « true » si une interruption a été demandée (via un bouton d'interruption) et « false » autrement.

interruptionPermise()

Retourne « true » si le programme est configuré de façon que les interruptions soient permises.

permettreInterruption(b)

Cette instruction configure l'environnement pour que le bouton d'interruption soit actif s'il existe.

quitter()

Ferme l'application.

unicode(ch)

Le paramètre ch est une chaîne de caractères dans laquelle les caractères accentués ont été représentés par un code particulier. Cette fonction retourne une chaîne dans laquelle les caractères accentués ont été remplacés par leur représentation unicode.

Le code:

suivi de la lettre sans accent, suivi des 3 premiers caractères du nom de l'accent ou du symbole (gra pour grave, aig pour aigu, cir pour circonflexe, ced pour cédille, tre pour trema).

Par exemple: à ---> #agra (# a gra).

valEnt(d)

Le paramètre d est un nombre décimal (type double). Cette instruction retourne l'entier le plus proche de d.

4.6.Les boutons

activerBouton(int i)

Rend actif le bouton numéro *i*.

changerNomBouton(i, nom)

i est un entier et nom une chaîne de caractères. L'effet est de changer le nom du bouton numéro i pour qu'il prenne la valeur du paramètre nom.*(Voir la façon de numéroter les boutons chapitre ...)*

desactiverBouton(int i)

Désactive le bouton numéro *i*.

4.7.Les glissières

On peut ajouter jusqu'à 12 glissières sur deux lignes à raison d'un maximum de six par ligne. Ces glissières présentent une légende, un champ de texte indiquant la valeur courante de la glissière et, au dessous, la glissière proprement dite avec ses valeur minimale et maximale affichées.

ajouterGlissiereLigne1(légende, min, max, posDebut, nbDec)

ajouterGlissiereLigne1(légende, min, max)

Le paramètre « légende » est une chaîne de caractères qui représente le texte affiché au dessus de la glissière. Les paramètres min, max et posDebut sont des nombres décimaux représentant respectivement les valeurs maximale, minimale et initiale de la glissière. Les paramètre nbDec est un entier indiquant le nombre de décimales désirées. Cette instruction ajoute sur la ligne 1 une glissière configurée a l'aide des paramètres fournis. Lorsque les paramètres posDebut et nbDec ne sont pas présents ils prennent les valeurs par défaut posDebut = min et nbDec = 0.

ajouterGlissiereLigne2(légende, min, max, posDebut, nbDec)

ajouterGlissiereLigne2(légende, min, max)

Identique à ci-dessus mais la glissière est ajoutée sur la deuxième ligne.

fixeValeurGlissiere(*i*, *v*)

i est un entier et *v* un décimal. L'instruction donne à la glissière numéro *i* la valeur *v*.

valeurGlissiere(i)

Le paramètre i est un entier. L'instruction retourne un nombre décimal qui est la valeur actuelle de la glissière numéro i.

4.8.Dialogues

choixMultiple(titre, message, bouton1)

choixMultiple(titre, message, bouton1, bouton2)

choixMultiple(titre, message, bouton1, bouton2, bouton3)

choixMultiple(titre, message, bouton1, bouton2, bouton3, bouton4)

Cette instruction ouvre une fenêtre de dialogue avec un titre, un message et de 1 à 4 boutons. Elle retourne le numéro du bouton cliqué.

Les paramètres titre et message sont des chaînes de caractères donnant le titre de la fenêtre et le message à afficher. Les paramètres bouton1, bouton2, bouton3, bouton4 sont des chaînes de caractères donnant les noms des boutons à afficher.

demanderBooleen(titre, message)

demanderBooleen(titre, message, valInit)

Cette instruction ouvre une fenêtre de dialogue avec un titre, un message, un champ de texte et un bouton « OK ». Si le paramètre valInit est présent, ce doit être un booléen et il apparaît dans le champ de texte lors de l'ouverture de la fenêtre.

Lorsqu'on clique le bouton « OK » l'instruction retourne le booléen écrit dans le champ de texte.

demanderChaine(titre, message)

demanderChaine(titre, message, valInit)

Cette instruction ouvre une fenêtre de dialogue avec un titre, un message, un champ de texte et un bouton « OK ». Si le paramètre valInit est présent, ce doit être une chaîne et il apparaît dans le champ de texte lors de l'ouverture de la fenêtre.

Lorsqu'on clique le bouton « OK » l'instruction retourne la chaîne écrite dans le champ de texte.

demanderEntier(titre, message)

demanderEntier(titre, message, valInit)

Cette instruction ouvre une fenêtre de dialogue avec un titre, un message, un champ de texte et un bouton « OK ». Si le paramètre valInit est présent, ce doit être un entier et il apparaît dans le champ de texte lors de l'ouverture de la fenêtre.

Lorsqu'on clique le bouton « OK » l'instruction retourne l'entier écrit dans le champ de texte.

demanderNombre(titre, message)

demanderNombre(titre, message, valInit)

Cette instruction ouvre une fenêtre de dialogue avec un titre, un message, un champ de texte et un bouton « OK ». Si le paramètre valInit est présent, ce doit être un entier ou un décimal et il apparaît dans le champ de texte lors de l'ouverture de la fenêtre.

Lorsqu'on clique le bouton « OK » l'instruction retourne le décimal écrit dans le champ de texte.

message(mess)

Cette instruction ouvre une fenêtre de dialogue avec un titre, un message et un bouton « OK ».

Lorsqu'on clique le bouton « OK » la fenêtre disparaît sans rien retourner.

4.9. Fichiers

ajouterDonnee(donnee)

Le paramètre donnee peut être un nombre entier ou décimal, un booléen ou une chaîne de caractères. Cette instruction ajoute la valeur du paramètre donnee à l'ensemble des données destinées à être sauvegardées dans un fichier.

lireDonnee(i)

i est un entier qui désigne le numéro de ligne de la donnée à lire, parmi les données qui ont été lues dans un fichier. Cette méthode retourne, sous forme de chaîne de caractères, la donnée qui était à la ligne i dans le fichier.

lireDonneeSuivante()

Cette méthode retourne, sous forme de chaîne de caractères, la donnée qui suit la dernière donnée lue. Si une telle donnée n'existe pas il y aura une erreur.

lireFichier()

Cette méthode permet de donner le nom, ou de sélectionner, un fichier à ramener en mémoire. Une fois le fichier ramené en mémoire on accède aux données ligne par ligne avec la méthode lireDonnee(*numéro de ligne*).

rembobiner()

Replace la prochaine donnée à lire au début du fichier lu.

sauverDonnees()

Cette méthode permet de donner un nom de fichier et de sélectionner un emplacement où sera sauvegardé un fichier texte portant le nom choisi. Dans ce fichier seront sauvegardées les données mise en réserve à l'aide de la méthode ajouterDonnee(*donnee*).

tailleFichier()

retourne le nombre de données présentes dans le fichier lu.

valEnt(donnee)

Le paramètre donnee est une chaîne de caractère. Si donnee est une chaîne représentant un entier, cette méthode transforme la chaîne en entier et retourne cet entier.

valDec(donnee)

Le paramètre donnee est une chaîne de caractère. Si donnee est une chaîne représentant un nombre décimal, cette méthode transforme la chaîne en décimal et retourne ce décimal.

valBool(donnee)

Le paramètre donnee est une chaîne de caractère. Si donnee est une chaîne représentant un booléen (*true* ou *false*), cette méthode transforme la chaîne en booléen et retourne ce booléen.

viderDonneesASauver()

Cette méthode réinitialise à vide l'ensemble des données à sauvegarder.

viderDonneesRamenees()

Cette méthode réinitialise à vide l'ensemble des données lues à partir d'un fichier.

4.10.Les mathématiques

Les constantes

E la constante d'Euler, base du logarithme népérien.

Pi le rapport de la circonférence du cercle à son diamètre

Les fonctions

abs(x)

Le paramètre x peut être un entier ou un décimal. Cette fonction retourne la valeur absolue de **x**. Le type de la valeur de retour est le même que celui de x.

acos(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne l'arc (*en radians*) dont le cosinus est **x.**

acosD(x)

Le paramètre x peut être un entier ou un décimal. Cette fonction retourne l'arc (*en degrés*) dont le cosinus est **x.**

arrondis(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne l'entier le plus proche de **x.**

asin(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne l'arc (*en radians*) dont le sinus est **x.**

asinD(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne l'arc (*en degrés*) dont le sinus est **x.**

atan(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne l'arc (*en radians entre -Pi et Pi*) dont la tangente est **x.**

atanD(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne l'arc (*en degrés entre -90 et 90*) dont la tangente est **x.**

atan2(x, y)

Les paramètres **x** et **y** peuvent être entiers ou décimaux. Cette fonction retourne l'arc (*en radians entre 0 et 2Pi*) dont la tangente est y/x **.**

atan2D(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne l'arc (*en degrés entre 0 et 360*) dont la tangente est y/x **.**

cos(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne le cosinus de x radians**.**

cosD(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne le cosinus de x degrés**.**

divEnt(a, b)

a et b sont des entiers et l'instruction retourne un entier qui est le quotient entier de a par b.

exp(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne $\mathbf{E^x}$**.**

log(x)

Le paramètre **x** peut être un entier ou un décimal positif. Cette fonction retourne le logarithme de x.

max(x, y)

Les paramètres **x** et **y** peuvent être entiers ou un décimaux. Cette fonction retourne le plus grand parmi **x** et **y**. Si **x** et **y** sont entiers la valeur retournée est de type **int** sinon elle est de type **double**.

min(x, y)

Les paramètres **x** et **y** peuvent être entiers ou un décimaux. Cette fonction retourne le plus petit parmi **x** et **y**. Si **x** et **y** sont entiers la valeur retournée est de type **int** sinon elle est de type **double**.

mod(a, b)

a et b sont des entiers et l'instruction retourne un entier qui est le reste de la division entière de a par b (a modulo b)

plafond(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne le plus petit entier plus grand ou egal à **x**.

plancher(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne le plus grand entier plus petit ou egal à **x**.

pow(x, y)

Les paramètres **x** et **y** peuvent être entiers ou décimaux. Cette fonction retourne **x^y**.

puissance(x, y)

Les paramètres **x** et **y** peuvent être entiers ou décimaux. Cette fonction retourne **x^y**.

racine(x)

Le paramètre **x** peut être un entier ou un décimal positif ou nul. Cette fonction retourne la racine carré positive de x.

sin(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne le sinus de **x** radians.

sinD(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne le sinus de **x** degrés.

sqrt(x)

Le paramètre **x** peut être un entier ou un décimal positif ou nul. Cette fonction retourne la racine carré positive de x.

tan(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne la tangente de **x** radians.

tanD(x)

Le paramètre **x** peut être un entier ou un décimal. Cette fonction retourne la tangente de **x** degrés.

5. La programmation dans l'environnement Expresso

Pour programmer dans l'environnement Expresso il faut d'abord ouvrir l'environnement BlueJ en double cliquant son icône, et dans le menu « Projet » choisir l'item « Nouveau Projet ». Dans la fenêtre de dialogue qui s'ouvre il faut choisir un emplacement pour notre projet, lui donner un nom et cliquer sur le bouton « Enregistrer ».
Ceci étant fait, on programme dans un fichier « Expresso » obtenu en cliquant sur le bouton « Nouvelle classe », en sélectionnant « Expresso », dans la fenêtre de choix qui s'ouvre et en donnant le nom « Expresso » dans le champ de texte « Nom ».
Le fichier obtenu est divisé en plusieurs zones et au début de chacune de ces zones on trouve des indications de ce qu'on peut et ne peut pas y faire (*Pour une description en grands détails, consulter le tutoriel*).

5.1. Configuration de l'interface

5.1.1. Les variables de paramétrage de l'interface

Dans la zone « Paramètres de l'interface » on trouve plusieurs variables dont les valeurs peuvent êtres modifiées et qui permettent de paramétrer les différents éléments de l'interface.

La fenêtre

baseFenetre = ... ; cette variable entière (**int**) fixe la largeur de la fenêtre en pixels. On peut modifier sa valeur, en inscrivant un entier après le signe =.

hauteurFenetre = ... ; cette variable entière (**int**) fixe la hauteur de la fenêtre en pixels. On peut modifier sa valeur, en inscrivant un entier après le signe =.

titreFenetre = ... ; cette variable chaîne de caractères (**String**) fixe le titre de la fenêtre. On peut modifier sa valeur, en inscrivant une phrase entre guillemets ("...") après le signe =.

zoneTexte = true (ou **false**); cette variable booléenne (**boolean**) indique si on veut avoir une zone de texte (valeur **true**) ou si on ne veut pas en avoir (valeur **false**).

Les boutons

On peut également avoir, dans l'interface, des boutons (maximum 16) disposés sur une ou deux lignes:

La variable **nomsBoutons1** reçoit les noms des boutons de la première ligne et la variable **nomsBoutons2** reçoit les noms des boutons de la deuxième ligne. Les noms de boutons sont des chaînes de caractères entre guillemets et sont séparés par des virgules. Par exemples
nomsBoutons1 = {"Dessiner", "Quitter"};
fera apparaître deux boutons nommés **Dessiner** et **Quitter** sur une ligne. Si on ajoute
nomsBoutons2 = {"Effacer"};
un bouton nommé **Effacer** apparaîtra sur une deuxième ligne au dessous des deux autres boutons.
Lorsque une de ces variables est laissée vide la ligne de boutons correspondante n'apparaît pas. Si les deux variables sont vides il n'y a pas de boutons. Cependant, même vides, ces variables doivent être présentes, sans quoi il y aura une erreur de compilation.

Le bouton d'interruption

On peut aussi créer un bouton d'interruption pour permettre à un utilisateur éventuel d'arrêter en cours de route une action qu'il trouve trop longue à s'exécuter.

Pour créer un tel bouton on précède son nom des caractères « Interruption_ » dans la variable nomBouton1 ou 2 où il est déclaré.

Pour l'utiliser, il faut d'abord déclarer que les interruptions seront permises avec la procédure « permettreIntrruption() » que l'on place dans la procédure initialisation. Ensuite, dans les actions

que l'on veut pouvoir interrompre, il faut vérifier périodiquement si une interruption est demandée à l'aide de la procédure « interruption() » (qui retourne « true » si une interruption a été demandée et « false » autrement) et insérer les instructions nécessaires pour arrêter l'action en cours si l'interruption a été demandée.

Les menus

Pour créer des menus (maximum 8) il suffit, comme pour les boutons, d'écrire les noms des menus et de leurs items dans les variables **nomsMenus**. Par exemples
nomsMenus1 = {"Graphique", "Dessiner", "Effacer", "Quitter"};
créera un premier menu nommé **Graphique** avec pour items de menu, dans l'ordre, **Dessiner**, **Effacer**, **Quitter**.
Il y a huit variables **nomsMenus1**, **nomsMenus2**, **...**, **nomsMenus8** permettant de créer jusqu'à huit menus. Ces variables doivent toujours être présentes, elles peuvent cependant être vides. Lorsqu'une de ces variables est vide elle est ignorée **ainsi que toutes celles qui suivent**. Par exemple dans l'exemple ci-dessus si la variable **nomsMenus2** est vide
nomsMenus2 = {};
il n'y aura qu'un seul menu (le menu Graphique) que les autres variables soient vides ou pas.
De plus chaque menu ne doit contenir qu'un maximum de 8 items, les items supplémentaires s'il y en a seront ignorés.

Les menus spéciaux

On peut créer des menus dont les items sont des cases à cocher ou des boutons poussoirs. Pour cela il suffit de précéder le nom du menu par les caractères « s_ » pour un menu « cases à cocher » et par « c_ » pour un menu « boutons poussoirs ». Un menu « cases à cocher » permet de faire des sélections multiples c'est à dire que plusieurs cases peuvent être cochées simultanément. Par contre un menu « boutons poussoirs » ne permet qu'une sélection unique, lorsqu'on sélectionne un item son bouton poussoir devient sélectionné et les autres deviennent désélectionnés.
Par programmation on peut vérifier l'état des items de ces menus à l'aide des procédures

- etatMenuItem(numeroMenu, numeroItem) qui retourne « true » si l'item de menu correspondant est sélectionné et « false » autrement.
- selectionMenuItem(numeroMenu, numeroItem, b) qui sélectionne l'item de menu correspondant si « b » est « true » et le désélectionne si « b » est « false ».

À part ces particularités, ces menus se comportent exactement comme les autres.

La procédure initialisation

Toujours dans la zone « Paramètres de l'interface » on trouve une procédure « initialisation() ». C'est dans cette procédure qu'on doit placer les instructions qu'on veut voir exécutées dès l'ouverture du programme.

5.1.2.La création des glissières

On peut ajouter jusqu'à 12 glissières réparties sur deux lignes (avec un maximum de 6 par ligne) en invoquant les méthodes **ajouterGlissiereLigne1(...)** et **ajouterGlissiereLigne2(...)** dans la méthode **ajoutDeGlissieres()**. Les méthodes **ajouterGlissiereLigne..** servent à configurer les glissières mais c'est la méthode **ajoutDeGlissieres()** qui les ajoute réellement à l'interface. C'est pourquoi l'invocation des méthodes **ajouterGlissiereLigne..** doit toujours se faire dans la méthode **ajoutDeGlissieres()**.
Les méthodes **ajouterGlissiereLigne1 et ajouterGlissiereLigne2** possèdent 3, 4 ou 5 paramètres entiers, les trois premiers sont obligatoires et donnent la légende, les valeurs minimale et maximale de la glissière. Le quatrième paramètre indique la position initiale du curseur. S'il est absent le curseur sera sur la valeur minimale. Le cinquième paramètre indique le nombre de

décimales voulues. S'il est absent il est considéré comme nul, c'est à dire 0 décimales. Il est a noter que la valeur retournée par la méthode **valeurGlissiere(numéro)** est toujours un nombre décimal (type double), ceci même lorsque la glissière est configurée avec 0 décimales.

5.2.Les actions associées aux éléments d'interface.

5.2.1.Les actions des boutons

Les boutons sont numérotés dans l'ordre ou on les rencontre dans les variables **nomsBoutons1** puis **nomsBoutons2**. À chaque bouton est associé une méthode dont le nom est composé du mot **actionBouton** suivi du numéro du bouton cette méthode sera automatiquement invoquée lorsqu'on cliquera sur le bouton et provoquera une action correspondant au programme qu'elle contient. Par exemple pour associer une action au bouton numéro 1 on écrira

```
public void actionBouton1(){ // sera déclenchée par un clic sur le bouton 1
instructions;
}
```

Les méthodes **actionBouton*i*** inutiles peuvent être détruites sans danger.

5.2.2.Les actions des menus

Les menus sont numérotés comme les variables qui les contiennent, donc le menu 1 est le menu contenu dans la variable **nomsMenus1**, le menu 2 est celui contenu dans la variable **nomsMenus2**, etc.
À l'intérieur d'un menu les items sont numérotés dans l'ordre ou ils sont rencontrés dans la variable **nomsMenus** correspondante. Par exemple si on a déclaré
nomsMenus3 = {"Graphique", "Dessiner", "Effacer", "Quitter"};
Graphique sera le menu 3, **Dessiner** sera l'item 1 du menu 3, **Effacer** sera l'item 2 du menu 3, **Quitter** sera l'item 3 du menu 3.
A chaque item de menu est associée une méthode qui sera invoquée lorsqu'on sélectionnera cet item. Cette méthode a un nom composé comme suit:
le mot **actionMenu** suivi du numéro du menu, suivi du mot **Item**, suivi du numéro de l'item de menu.
Par exemple pour associer une action à l'item de menu **Effacer** de l'exemple ci-dessus on programmera :

```
public void actionMenu3Item2(){ // puisque Effacer est l'item 2 du menu 3
instructions;
}
```

5.2.3.Les actions des glissières

Les glissières sont numérotées de gauche à droite dans l'ordre ou elles apparaissent d'abord sur la ligne 1 puis sur la ligne 2.
La valeur d'une glissière correspond à la position de son curseur et est toujours indiquée dans le champ de texte situé au dessus de la glissière. Par programmation on peut consulter la valeur de la glissière numéro i à l'aide de la méthode **valeurGlissiere(i).** Il est important de noter que cette valeur est toujours un nombre décimal. On peut modifier la valeur d'une glissière en déplaçant son curseur ou par programmation à l'aide de la méthode **fixeValeurGlissiere(i, v)**
qui donnera à la glissière numéro i la valeur v.
On peut associer une action à chaque glissière, action qui sera déclenchée lors du déplacement du

curseur. Pour cela on place les instructions qu 'on désire voir exécutées lors de la manipulation de la glissière numéro *i* dans la méthode **actionGlissiere*i*(d)**:

public void actionGlissiere*i*(d){
...instructions
}

Cette méthode sera invoquée lors du déplacement du curseur de la glissière #i.
Il faut noter que le paramètre d de cette méthode prendra, lors de l'exécution, la valeur de la glissière on peut donc l'utiliser dans *instructions* à la place de **valeurGlissiere(i)**.

5.2.4.Les actions associées à la souris

Il y a quatre actions associées à la souris:

- Le « clic », qui correspond à peser sur le bouton et le relâcher sans déplacer la souris.
- Le « glisser » qui se décompose en 3 actions: peser sur le bouton, déplacer la souris et relâcher le bouton.

On peut faire exécuter des instructions pour répondre à chacun de ces 4 événements.
Par exemple les instructions à exécuter pour répondre à un clic doivent être placées dans la méthode clicSouris(x, y) ou x et y représentent les coordonnées du point cliqué en coordonnées tortue. On écrira donc

public void clicSouris(double x, double y){
instructions pouvant utiliser x et y
}

Les autres méthodes associées à la souris sont
debutGlisser(x, y), finGlisser(x, y), glisserEnCours(x, y)
elles sont aussi invoquées lors de l'événement correspondant, debutGlisser(x, y) lorsqu'on pèse sur le bouton au début d'un glissement, glisserEnCours(x, y) pendant le déplacement de la souris avec le bouton appuyé et finGlisser(x, y) lorsqu'on relâche le bouton de la souris après un déplacement. Dans tous les cas les variables x et y sont des nombres décimaux (type double) qui représentent les coordonnées du point ou se trouve la souris lors de l'événement. Ces coordonnées sont toujours des coordonnées tortue.

6. Pour continuer

Dans ce manuel nous venons de faire le tour de l'environnement et du langage d'Expresso.
Pour continuer on peut également trouver sur le site

http://www.math.uqam.ca/_boileau/AMQ2005.html

un tutoriel qui expose en détail la programmation de quelques exemples.
Sur ce même site on peut visionner quelques applets dont les sources sont disponibles.
Nous espérons que l'ensemble de cette documentation permettra à toutes les personnes intéressées d'utiliser Expresso avec autant de plaisir que nous en avons.