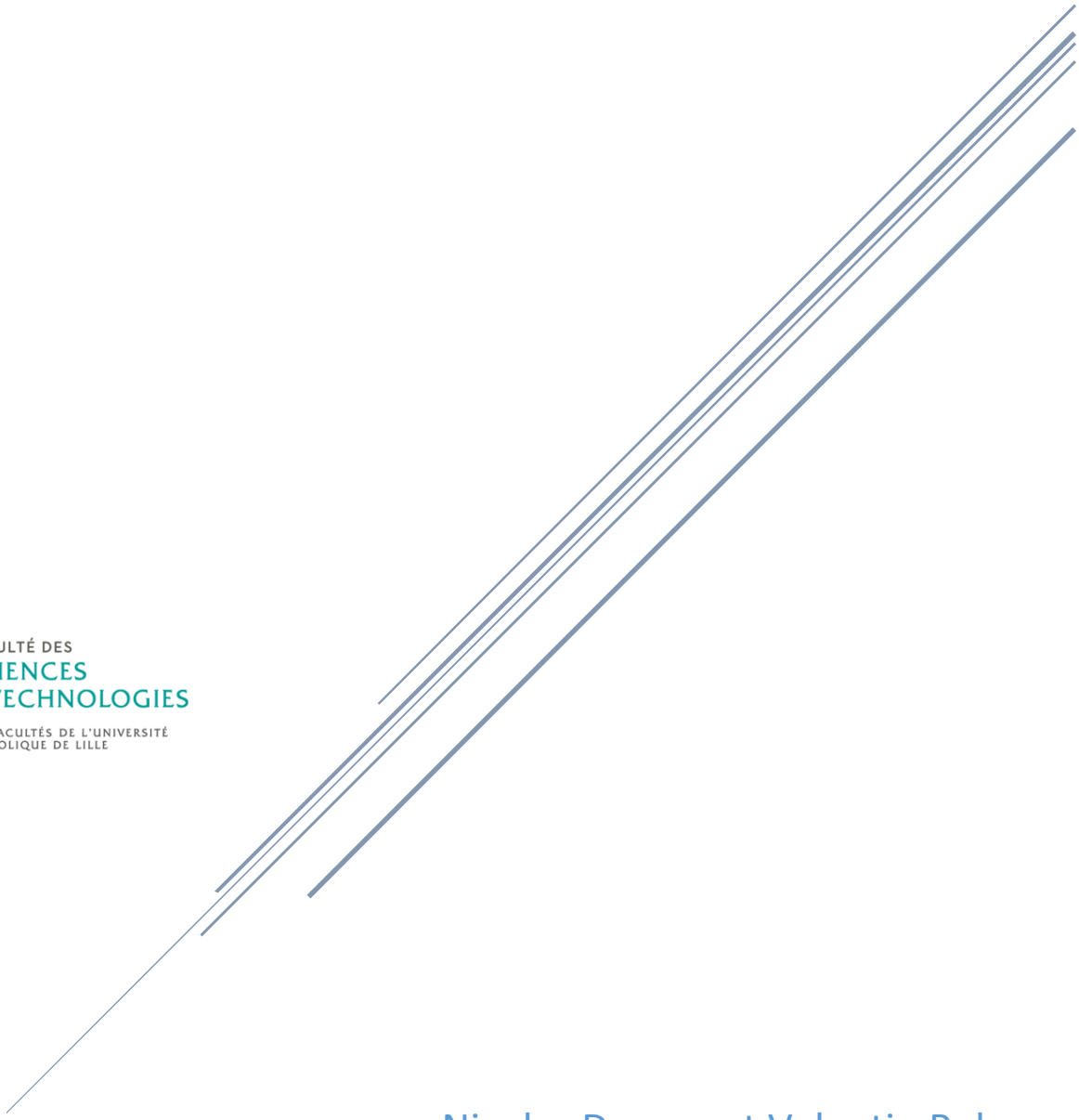
PROJET DE COMPLEXITE ET STRUCTURES DE DONNEES:

Développer un Interpréteur Logo avec l'aide d'Adagraph



FACULTÉ DES
SCIENCES
& TECHNOLOGIES
LES FACULTÉS DE L'UNIVERSITÉ CATHOLIQUE DE LILLE

Nicolas Ducom et Valentin Polo
Licence 3 Informatique

Sommaire

Consignes

Développer un interpréteur Logo à l'aide du langage Ada et du moteur graphique Adagraph. Le choix des fonctions à développer est libre.

Cheminement

Le Logo étant un langage mal définie, avec beaucoup de conventions et normes différentes, nous avons décidé de commencer par une étape de recherche, c'est suite à celle-ci que nous avons défini les fonctions à développer impérativement, et celles à développer en bonus.

Nous avons ensuite commence par développer la partie saisie et interprétation de commandes, en intégrant Adagraph en deuxième partie.

Les Commandes :

Pour que l'interpréteur fonctionne correctement, il faut mettre un espace entre chaque commande, sa valeur associée, les crochets. Il ne faut pas oublier de mettre un point-virgule en fin de ligne.

Toutes les commandes suivantes ont été implémentées :

Avec Paramètre :

forward/fw : avancer : forward 10.

backward /bk : reculer : backward 15.

left/lt : rotation vers la gauche : left 50.

right/rt : rotation vers la droite : right 45.

repeat : boucle (équivalent à loop) : repeat 10 [forward 15 left 10].

Sans Paramètre :

home : la tortue se « téléporte » au centre de l'écran.

penup/pu : la tortue se déplace sans dessiner.

pendown/pd : la tortue se déplace en dessinant.

clear/cs : « nettoie » l'écran.

exit : quitte.

Exemple de commande : repeat 72 [left 10 repeat 15 [fw 20 lt 15] repeat 20 [fw 5 rt 10] home];

Détail des Spécifications du paquetage p_logo.ads

Nous allons ici lister les différents éléments du paquetage et les expliquer un à un.

Types de variables :

Rules : défini l'état de certaines variables d'environnement, ici il s'agit juste de l'état du stylo, (on aurait pu par exemple rajouter la couleur).

t_action : la commande à exécuter.

T_command : la commande, sa valeur, et si nécessaire, la liste de sous-commandes.

T_command_in_buffer : commande dans une liste chaine, avec pointeur vers la prochaine commande.

T_command_buffer : premier élément et longueur d'une liste chainée.

Turtle : position et orientation du « curseur » ou de la tortue.

turtleOutsideWindow : exception à lever si la tortue sort de l'écran (une méthode parmi d'autres…).

Fonctions :

Positionne la tortue sur sa localisation de départ (le centre de la fenêtre), définie les règles, crée la fenêtre Adagraph etc…

```
procedure build(t:in out turtle; interpreterRules:in out rules; op:in out pt_command);
```

Fonction principale, boucle tant que l'utilisateur ne rentre pas la commande « exit ».

```
Procedure interpreter;
```

Compare une t_action et une String.

```
function compareStrings(A,B:t_action) return boolean;
```

Va chercher la prochaine commande, son paramètre si nécessaire, et la liste de commande qui peuvent ou non y être associe (boucle).

```
function getCommand return pt_command;
```

Retourne vrai si la commande saisie n'a pas besoin de paramètre.

```
function inNoValueCommands(action: t_action) return boolean;
```

Retourne vrai si la commande saisie a besoin d'une liste de commande (boucle).

```
function inBufferCommands(action: t_action) return boolean;
```

Va chercher la liste de commandes.

```
function getBufferCommand return pt_buffer;
```

Ajoute une commande à la liste.

```
procedure addToBuffer(buffer: in out pt_buffer;command: in t_command_in_buffer);
```

Retourne la dernière commande d'une liste.

```
function lastCommandInBuffer(buffer:Pt_Buffer) return pt_command_in_buffer;
```

Affiche en ordre toutes les commandes d'une liste.

```
procedure listBufferCommands(Buffer:Pt_Buffer);
```

Retourne vrai si la commande est la dernière dans sa liste.

```
function endOfBuffer(buffer:Pt_Buffer;command:pt_command_in_buffer) return boolean;
```

Converti des degrés en radians.

```
function degreesToRad(d: Integer) return float;
```

Retourne une t_action grâce à une String.

```
function textToAction(S:String) return t_action;
```

Vide une liste (non fonctionnelle).

```
procedure emptyBuffer(buff:pt_buffer);
```

Afficher une t_action .

```
procedure put(act:in t_action);
```

Effectue une commande.

```
procedure doCommand(comm:pt_command;turt:in out turtle;interpreterRules:in out rules);
```

Effectue la commande Forward.

```
procedure doForward(comm:pt_command;turt:in out turtle;interpreterRules:in out rules);
```

Effectue la commande Backward.

```
procedure doBackward(comm:pt_command;turt:in out turtle;interpreterRules:in out rules);
```

Effectue la commande Left.

```
procedure doLeft(comm:pt_command;turt:in out turtle);
```

Effectue la commande Right.

```
procedure doRight(comm:pt_command;turt:in out turtle);
```

Effectue la commande Clear.

```
procedure doClear(comm:pt_command);
```

Effectue la commande Home.

```
procedure doHome(comm:pt_command;turt:in out turtle);
```

Effectue la commande PennDown.

```
procedure doPenDown(comm:pt_command;interpreterRules:in out rules);
```

Effectue la commande PenUp.

```
procedure doPenUp(comm:pt_command;interpreterRules:in out rules);
```

Effectue la commande Repeat.

```
procedure doRepeat(comm:in pt_command;turt: in out turtle;interpreterRules:in out rules);
```

Itère dans et effectue les commandes d'une liste de commande.

```
procedure doBuffer(buffer:in Pt_Buffer;turt:in out turtle;interpreterRules:in out rules);
```

Points intéressants :

La création de cet interpréteur a été intéressante dans la mesure où il nous a demandé un peu d'originalité pour pouvoir lire et effectuer une liste de commandes à l'écran, en respectant les contraintes imposées par le langage.

Il peut être intéressant de noter que grâce à la manière dont elles ont été codées, les boucles imbriquées fonctionnent aussi sans soucis nativement.

Nous avons laissé l'affichage de débuggage, pour permettre une meilleure compréhension du code.

Difficultés rencontrées :

La déallocation de Buffers ne fonctionne pas correctement. De plus, il aurait pu être intéressant de placer toutes les commandes dans un buffer, jusqu'à la fin de ligne ou elles seraient toutes interprétées d'un coup puis déalocalisées. On peut ensuite regretter le manque d' « icone » tortue.