Sorbonne université

IDM

Les Roboticiens

Rapport de la tranche 3

Auteurs:	*Numéros étudiant:*
Nicolas SALLERON	3504018
Yoann GHIGOFF	3506454
Kévin VU-SAINTONGE	3202944
Axel ARCHAMBAULT	3300807

29 Janvier 2018

Nicolas SALLERON
Yoann GHIGOFF
Kévin VU-SAINTONGE
Axel ARCHAMBAULT

Sommaire

1 Introduction

La tranche 1 consistait à créer un langage spécifique à la création de chorégraphie pour les drones et à construire un éditeur de texte permettant l'édition et la complétion du langage.

La tranche 2 consistait à transformer la chorégraphie écrite par l'utilisateur en un programme qui sera capable de faire exécuter des commandes au drone en fonction d'une runtime.

La tranche 3 consiste à déployer la solution sur un serveur d'exécution (notre ordinateur ou celui du client) pour tester la distribution réalisée. De plus, une soutenance ou l'équipe présentera des vidéos démontrant la validité de la solution proposée.

2 Explication du fonctionnement de la solution

Comme expliquée dans le rapport de la Tranche 2, le code écrit par l'utilisateur générera du java qui respectera l'API du drone utilisé (via la Runtime).

Le code de Parrot étant en C, notre code java effectuera un "pont" entre lui-même et un fichier C (créé par nous même) qui communiquera avec l'API de Parrot.

Plus précisément notre code java a pour rôle de :

- Créer un flux de sortie en direction de notre programme C.
- Créer et gérer les Threads pour la durée des commandes.

3 Notice d'utilisation

Pour commencer, l'utilisateur devra choisir la distribution autonome du produit en fonction du système d'exploitation qu'il utilise pour le démarrer. En effet, la distribution est compatible pour les deux OS suivants : MacOS, Linux.

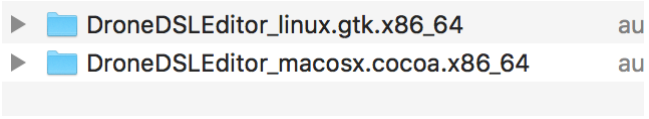

Figure 1: Versions de la distribution

L'utilisateur devra dézipper l'archive qu'il aura choisi.

Une fois terminé, il aura les fichiers suivants :

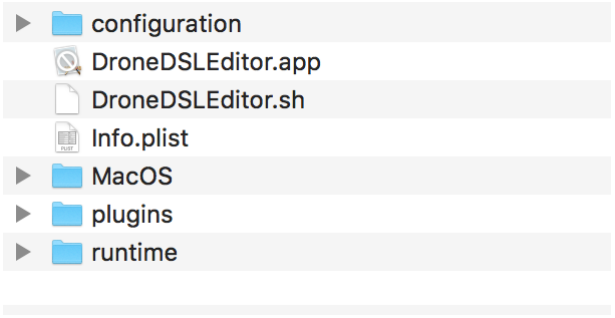

Figure 2: Fichiers extraits

Il lui suffira de lancer DroneDSLEditor.sh dans un terminal pour pouvoir lancer l'éditeur de création de chorégraphies.
Lors d'une première initialisation, il est nécessaire que l'application récupère et installe des dépendances, il faut donc lancer le script avec l'argument "install".

Exemple : "./DroneDSLEditor.sh install"

Pour les autres utilisations, cela n'est pas nécessaire et l'utilisateur ne devra taper que :

"./DroneDSLEditor.sh"

Au lancement de l'application il devra spécifier un répertoire pour stocker ses chorégraphies.

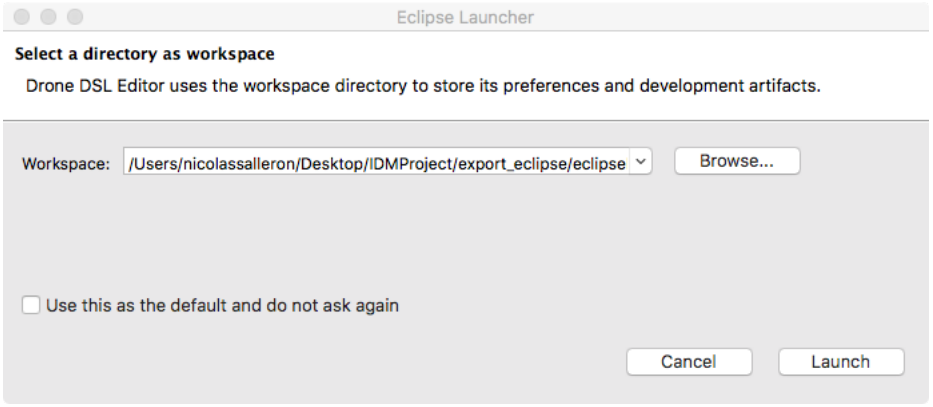

Figure 3: Répertoire de sauvegarde

Puis il créera un nouveau projet pour qu'il puisse éditer une chorégraphie.

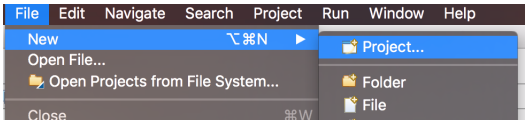

Figure 4: Nouveau projet

Il devra choisir un projet Xtext nommé DroneDSLProject.

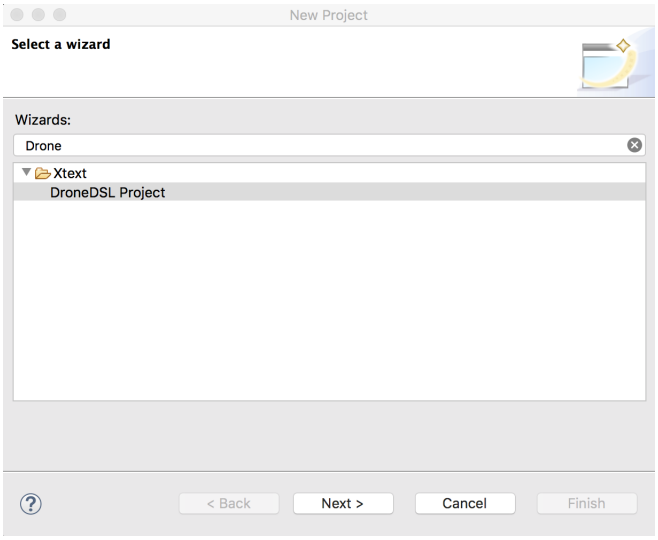

Figure 5: Nouveau projet

Ensuite l'utilisateur doit nommer son projet.

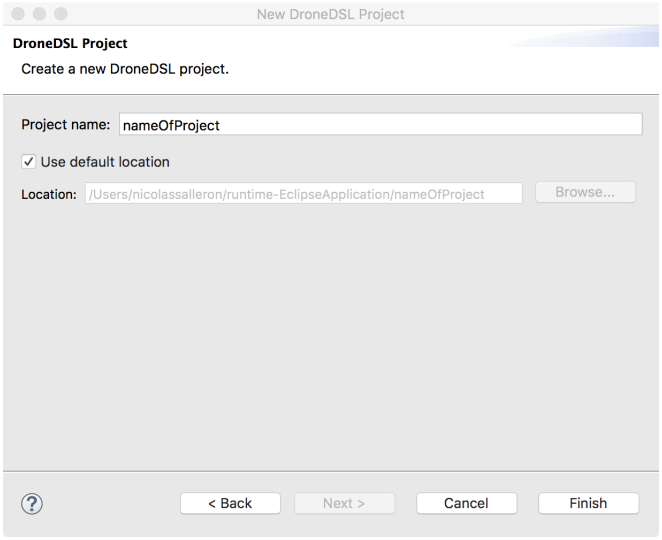

Figure 6: Nom du projet

Il pourra ainsi créer sa chorégraphie en spécifiant tout d'abord 5 constantes de vol qui sont générées automatiquement. Cela permet au programme d'adapter les instructions. Il définira ainsi la vitesse maximale d'élévation, la vitesse maximale de déplacement, la vitesse maximale de rotation, l'altitude maximale et la distance d'éloignement maximale du drone. Puis il pourra définir le corps main qui représente le point d'entrée du programme. Ce qui est contenu dans le bloc d'instruction suivant ce mot clef sera les instructions exécutées par le drone. Enfin il pourra faire appel à des fonctions définies par l'utilisateur.

```
/*
 * Paramètres d'exécution
 */
define eloignement_max 2
define hauteur_max 2
define vitesse_deplacement_max 100%
define vitesse_hauteur_max 100%
define vitesse_rotation_max 100%

main {
    decoller()
    // main auto-généré
    atterrir()
}
```

Figure 7: Exemple de projet

Une fois sa chorégraphie sauvegardée, l'utilisateur doit générer les classes JAVA dont il a besoin pour faire marcher son programme. Pour cela, il cliquera sur le bouton "généré". **Note : cela doit se faire pour chaque fichier .main_drone et .lib_drone (seulement s'ils ont été modifiés).**

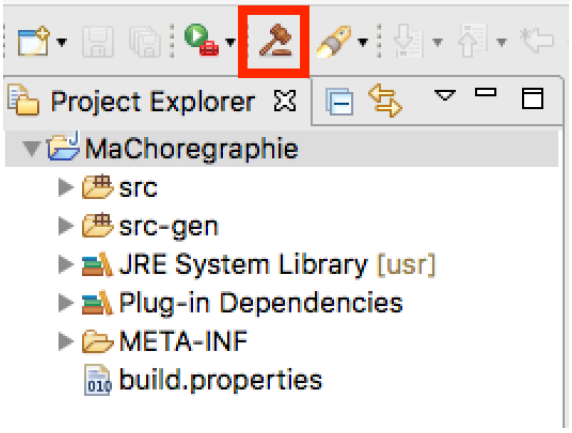

Figure 8: Structure du projet

Ainsi dans le dossier src du projet on retrouve le fichier main.main_drone qui représente la chorégraphie définie par l'utilisateur. De plus, les différents packages regroupent les différentes instructions.

L'utilisateur pourra lancer l'exécution de la chorégraphie en sélectionnant le dossier src puis réalisant un click droit, puis run as, puis Java Application. Il pourra également définir un bouton lui permettant d'éviter cette séquence.

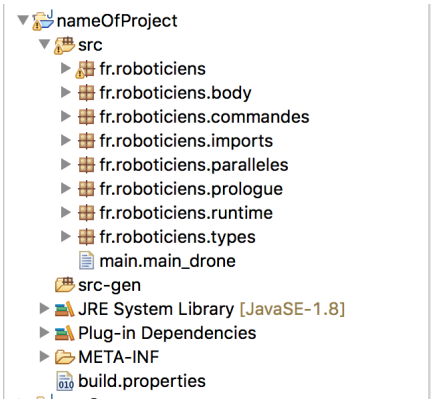

Figure 9: Structure du projet

3.1 Langage de conception de chorégraphie

3.1.1 Prologue

Pour fonctionner, l'utilisateur doit définir 5 constantes de vol. Cela permet au programme d'adapter la vitesse des mouvements en fonction de ces dernières. Il doit les définir comme ceci au début du fichier

```
define vitesse_hauteur_max 50%
define vitesse_deplacement_max 10%
define vitesse_rotation_max 20%
define hauteur_max 20
define eloignement_max 30
```

Figure 10: Exemple de define

define vitesse_hauteur_max 100%:
Cette constante permet de définir la vitesse maximale d'élévation du drone pour la chorégraphie par rapport à sa vitesse maximale possible.
La valeur de cette constante doit être comprise entre 1 et 100%.

define vitesse_deplacement_max 100%:
Cette constante permet de définir la vitesse maximale de déplacement sur le plan horizontal du drone pour la chorégraphie par rapport à sa vitesse maximale possible.
La valeur de cette constante doit être comprise entre 1 et 100%.

define vitesse_rotation_max 100% :
Cette constante permet de définir la vitesse maximale de rotation du drone pour la chorégraphie par rapport à sa vitesse maximale.
La valeur de cette constante doit être comprise entre 1 et 100%.

define hauteur_max 150 :
Cette constante permet de limiter l'altitude maximale du drone en vol. Lorsque le drone est sur le point de dépasser cette limite, il se stabilise automatiquement à la hauteur maximale.
La valeur de cette constante doit être un entier positif qui ne doit pas être supérieure à la hauteur maximale possible du drone.

define eloignement_max 3000:
Cette constante permet de contrôler la distance horizontale du drone en vol. Lorsque le drone est sur le point de dépasser cette limite, il se stabilise automatiquement à la distance maximale horizontale.
La valeur de cette constante doit être un entier positif qui ne doit pas être supérieure à la distance maximale horizontale possible du drone.

3.1.2 Les types

Type "Pourcent":
Déclaration : Pourcent a = 10%
C'est le type utilisé pour en paramètre des commandes de déplacement (monter, descendre, gauche, droite, avancer, reculer, rotation_gauche, rotation_droite).
La valeur doit être comprise entre 1 et 100

Type "Seconde":
Déclaration : Seconde b = 10
C'est le type utilisé pour en paramètre des commandes de déplacement (monter, descendre, gauche, droite, avancer, reculer, rotation_gauche, rotation_droite).

3.1.3 Décollage

Syntaxe: **decoller()**

Cette commande permet à un drone de décoller. Elle devra obligatoirement être la première commande exécutée dans le bloc main et ne peut pas être à nouveau exécutée si le drone n'a pas atterri.
Il est uniquement possible de décoller si le drone est au sol.

3.1.4 Attérrissage

Syntaxe: **atterrir()**

Cette commande permet à un drone d'atterrir. Elle devra obligatoirement être la dernière commande exécutée dans le bloc main et ne peut pas être à nouveau exécutée si le drone n'a pas décoller.
Il est uniquement possible d'atterrir si le drone est en vol.

3.1.5 Les commandes de mouvements

Syntaxe: **monter(durée: Seconde, vitesse_verticale: Pourcent)**

Cette commande permet à un drone de monter en altitude.
Elle ne peut être exécutée que si le drone est en vol.
Le paramètre **durée** est le temps en type Seconde durant lequel le drone s'élève.
Le paramètre **vitesse_vertical** est un type Pourcent. Il indique la vitesse d'élévation du drone par rapport à la vitesse verticale maximum définie par l'utilisateur dans le prologue.

Syntaxe: **descendre(durée: Seconde, vitesse_verticale: Pourcent)**

Cette commande permet à un drone de descendre en altitude.
Elle ne peut être exécutée que si le drone est en vol.
Le paramètre **durée** est le temps en type Seconde durant lequel le drone s'élève.
Le paramètre **vitesse_vertical** est un type Pourcent. Il indique la vitesse d'élévation du drone par rapport à la vitesse verticale maximum définie par l'utilisateur dans le prologue.

Syntaxe: **avancer(durée: Seconde, vitesse_deplacement: Pourcent)**

Cette commande permet à un drone de se déplacer sur un axe horizontal dans la direction vers laquelle sa face avant est tournée.
Elle ne peut être exécutée que si le drone est en vol.
Le paramètre **durée** est le temps en type Seconde durant lequel le drone avance.
Le paramètre **vitesse_deplacement** est un type Pourcent. Il indique la vitesse de déplacement du drone par rapport à la vitesse horizontale maximum définie par l'utilisateur dans le prologue.

Syntaxe: **reculer(durée: Seconde, vitesse_deplacement: Pourcent))**

Cette commande permet à un drone de se déplacer sur un axe horizontal dans la direction opposée à celle vers laquelle est tournée sa face avant.
Elle ne peut être exécutée que si le drone est en vol.
Le paramètre **durée** est le temps en type Seconde durant lequel le drone recule.
Le paramètre **vitesse_deplacement** est un type Pourcent. Il indique la vitesse de déplacement du drone par rapport à la vitesse horizontale maximum définie par

l'utilisateur dans le prologue.

Syntaxe: **gauche(durée: Seconde, vitesse_deplacement: Pourcent))**

Cette commande permet à un drone de se déplacer sur un axe horizontal dans la direction à gauche de celle vers laquelle est tournée sa face avant.
Elle ne peut être exécutée que si le drone est en vol.
Le paramètre **durée** est le temps en type Seconde durant lequel le drone se déplace.
Le paramètre **vitesse_deplacement** est un type Pourcent. Il indique la vitesse de déplacement du drone par rapport à la vitesse horizontale maximum définie par l'utilisateur dans le prologue.

Syntaxe: **droite(durée: Seconde, vitesse_deplacement: Pourcent))**

Cette commande permet à un drone de se déplacer sur un axe horizontal dans la direction à droite de celle vers laquelle est tournée sa face avant.
Elle ne peut être exécutée que si le drone est en vol.
Le paramètre **durée** est le temps en type Seconde durant lequel le drone se déplace.
Le paramètre **vitesse_deplacement** est un type Pourcent. Il indique la vitesse de déplacement du drone par rapport à la vitesse horizontale maximum définie par l'utilisateur dans le prologue.

Syntaxe: **rotation_gauche(durée: Seconde, vitesse_rotation: Pourcent)**

Cette commande permet à un drone de tourner sur lui même dans le sens inverse des aiguilles d'une montre.
Elle ne peut être exécutée que si le drone est en vol.
Le paramètre **durée** est le temps en type Seconde durant lequel le drone doit pivoter.
Le paramètre **vitesse_rotation** est un type Pourcent. Il indique la vitesse de déplacement du drone par rapport à la vitesse de rotation maximum définie par l'utilisateur dans le prologue.

Syntaxe: **rotation_droite(durée: Seconde, vitesse_rotation: Pourcent)**

Cette commande permet à un drone de tourner sur lui même dans le sens des aiguilles d'une montre.
Elle ne peut être exécutée que si le drone est en vol.
Le paramètre **durée** est le temps en type Seconde durant lequel le drone doit pivoter.

Le paramètre **vitesse_rotation** est un type Pourcent. Il indique la vitesse de déplacement du drone par rapport à la vitesse de rotation maximum définie par l'utilisateur dans le prologue.

3.1.6 La commande pause

Syntaxe: **pause(durée: Seconde)**

Cette commande permet à un drone de s'arrêter en position stationnaire pendant une certaine durée, soit en vol ou au sol.
Le paramètre **durée** donnera le temps en secondes durant lequel le drone doit se stabiliser.

3.2 Les fonctions

Le langage permet de définir des fonctions, ces dernières sont une suite d'instructions séquentielles qu'exécutera le drone.
Les fonctions ont un nom qui permet de les identifier. Il n'est pas possible de paralléliser deux fonctions.
La définition d'une fonction est de la forme suivante :

```
func nomDeLaFonction() {
    Mettre une instruction par ligne
}
```

Figure 11: Exemple de definition de fonction

Il n'est pas possible de réaliser des récursions de fonction.

3.2.1 Fonctions locales

Les fonctions locales sont des fonctions définies dans le même fichier que le bloc main (fichier d'extension .main_drone).
L'appel d'une fonction locale se fait sans précision du nom du fichier dans lequel elle est définie.

```
main {
    decoller()
    gauche(1,10%)
    maFonction()
    reculer(1,20%)
    atterrir()
}

func maFonction() {
    monter(2,10%) & droite(1,15%)
    avancer(2,20%)
    droite(2,15%)
}
```

Figure 12: Exemple d'appel d'une fonction locale

3.2.2 Bibliothèque de fonction

Il est possible d'utiliser des fonctions définies dans d'autres fichiers .lib_drone, ces fichiers doivent être dans le même répertoire que le fichier appelant ses fonctions.
L'inclusion d'une bibliothèque se fait en définissant via l'instruction **import** suivi du nom du fichier suivi de l'extension ".lib_drone".
L'appel d'une fonction définie dans une bibliothèque se fait en précisant le nom du fichier contenant sa définition.

```
/* import du fichier contenant la fonction maFonction() */
import <maLib.lib_drone>
/* Instructions de prologue */
main {
   decoller()
   maLib.maFonction()
   atterrir()
}
```

Fichier maLib.lib_drone :

```
func maFonction() {
   monter(2,10%) & droite(1,15%)
   avancer(2,20%)
   droite(2,15%)
}
```

4 Fonctionnement de la distribution

La distribution génère un programme JAVA. Ce dernier communique avec le programme C qui appelle les fonctionnalités de l'API du drone fournies par le SDK de Parrot.

Ainsi lorsque l'utilisateur entre une commande pour sa chorégraphie, elle appelle la commande execute de la classe JAVA associée à la commande. En fonction de la runtime (il y en a une par API de drone) passée en paramètre, la méthode fait appel au code C de Parrot.

De plus comme nous avons choisi d'associer une durée à une commande, nous avons décidé d'utiliser les Threads. Ces derniers nous permettent pour les instructions basiques d'attendre la fin d'une commande avant d'en exécuter une autre par le drone. Pour les commandes parallélisables, chaque commande est associée à un Thread et est lancée sans attendre la fin de la précédente.

En résumé :

- Chorégraphie écrite par l'utilisateur.
- Chorégraphie interprétée et compilée pour générer des classes JAVA (dont la principale nommée Main).
- La classe Main créer les classes représentant les mouvements que doit faire le drone.
- Le JAVA ouvrira un flux en direction de notre programme C.
- Chacune de ces classes utilisent la runtime pour envoyer des informations à ce programme C.
- Le C interprète ces informations et envoit des commandes au drone via l'API du constructeur.