



UNIVERSITE PIERRE ET MARIE-CURIE

IDM

LES ROBOTICIENS

---

## Rapport de la tranche 2

---

*Auteurs:*

Nicolas SALLERON

Yoann GHIGOFF

Kévin VU-SAINTONGE

Axel ARCHAMBAULT

*Numéros étudiant:*

3504018

3506454

3202944

3300807

11 Janvier 2018

# Sommaire

---

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Explication du fonctionnement de la solution</b>	<b>2</b>
<b>3</b>	<b>Notice d'utilisation</b>	<b>3</b>
3.1	Lancement de l'environnement de développement . . . . .	3
3.2	Langage de conception de chorégraphie . . . . .	8
3.2.1	Prologue . . . . .	8
3.2.2	Les types . . . . .	9
3.2.3	Décollage . . . . .	9
3.2.4	Atterrissage . . . . .	9
3.2.5	Les commandes de mouvements . . . . .	10
3.2.6	La commande pause . . . . .	12
3.3	Les fonctions . . . . .	13
3.3.1	Fonctions locales . . . . .	13
3.3.2	Bibliothèque de fonction . . . . .	14

## 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 quant-à elle consiste à transformer la chorégraphie écrite par l'utilisateur en un programme qui sera capable de faire exécuter des commandes au drone.

## 2 Explication du fonctionnement de la solution

Pour générer un code capable de faire exécuter des commandes au drone, il nous a fallu tout d'abord choisir le langage cible. Trois langages étaient disponibles : JAVA, C ou Objective-C.

Notre choix s'est porté sur le JAVA pour sa simplicité d'écriture et sa comptabilité native avec Eclipse.

Une fois le choix du langage cible décidé, nous avons dû concevoir un système qui interprète et compile la chorégraphie écrite par l'utilisateur pour ensuite générer des classes JAVA associées à cette dernière.

L'une des plus importante est la classe Main, elle contient la méthode main qui correspond à la chorégraphie écrite par l'utilisateur (voir la Figure 1 ci-dessous).

Le système générera d'abord les variables du prologue nécessaires à l'exécution de la chorégraphie (hauteur maximale, vitesse maximale...) qui ont été renseignées par l'utilisateur au début de sa chorégraphie.

Ensuite le système créera une méthode main quiinstanciera en nos classes JAVA en fonction des instructions renseignées par l'utilisateur dans le main qu'il aura écrit.

De plus, chaque commande sera exécutée par la runtime qui exécutera les commandes spécifiques à une API du drone.

```
public class Main {
    static DroneRuntime runtime = new DroneRuntimePrint();
    static Prologue prologue_1981352351 = new Prologue(new Pourcent(10), new Pourcent(10), new Pourcent(20), 20, 30);
    public static void main(String[] args) {

        runtime.execPrologue(prologue_1981352351);
        new Decoller().execute(runtime);
        Parallele p2_258366224 = new Parallele();
        p2_258366224.addCommande(new Gauche(new Seconde(10), new Pourcent(30)));
        p2_258366224.addCommande(new Avancer(new Seconde(10), new Pourcent(30)));
        p2_258366224.execute(runtime);
        Parallele p2_1987299330 = new Parallele();
        p2_1987299330.addCommande(new Droite(new Seconde(10), new Pourcent(30)));
        p2_1987299330.addCommande(new Avancer(new Seconde(10), new Pourcent(30)));
        p2_1987299330.execute(runtime);
        Parallele p3_1008265679 = new Parallele();
        p3_1008265679.addCommande(new Droite(new Seconde(10), new Pourcent(30)));
        p3_1008265679.addCommande(new Avancer(new Seconde(10), new Pourcent(30)));
        p3_1008265679.addCommande(new RotationGauche(new Seconde(3), new Pourcent(10)));
        p3_1008265679.execute(runtime);
        new Gauche(new Seconde(10), new Pourcent(10)).execute(runtime);
        A();
        new Droite(new Seconde(10), new Pourcent(20)).execute(runtime);
        new Atterrir().execute(runtime);
    }

    public static void A() {
        new Gauche(new Seconde(10), new Pourcent(10)).execute(runtime);
        B();
    }

    public static void B() {
        new Droite(new Seconde(10), new Pourcent(5)).execute(runtime);
    }
}
```

Figure 1: Classe Main

## 3 Notice d'utilisation

### 3.1 Lancement de l'environnement de développement

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 trois OS suivants : MacOS, Linux, Windows.




Nom	^	Date de modif
 eclipseLinux.zip		aujourd'hui à 14h
 eclipseMacOS.zip		aujourd'hui à 14h
 eclipseWindows.zip		aujourd'hui à 14h

Figure 2: Versions de la distribution

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

Une fois terminé, il aura les fichiers suivants :

Nom	Date de modification
configuration	aujourd'hui à 21:16
DroneDSLEditor.app	aujourd'hui à 21:15
Info.plist	aujourd'hui à 21:15
MacOS	aujourd'hui à 21:15
plugins	aujourd'hui à 21:15
Resources	aujourd'hui à 21:15

Figure 3: Fichiers extraits

Il lui suffira de lancer DroneDSLEditor pour pouvoir créer une chorégraphie. Au lancement de l'application il devra spécifier un répertoire pour stocker ses chorégraphies.

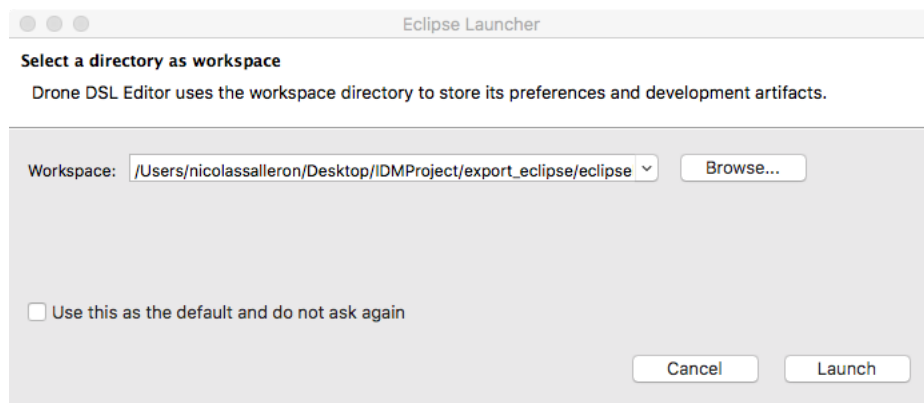


Figure 4: Répertoire de sauvegarde

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

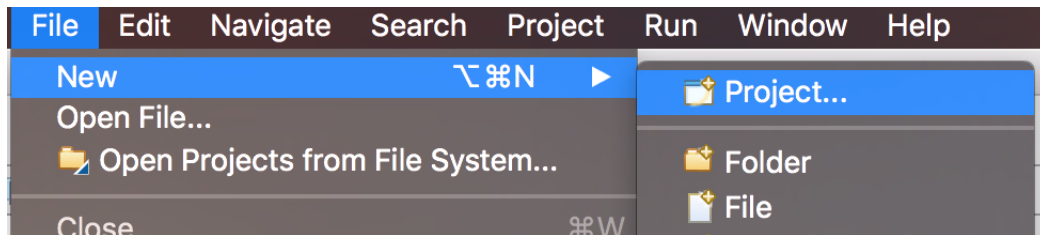


Figure 5: Nouveau projet

Il devra choisir un projet Xtext nommé DroneDSL Project.

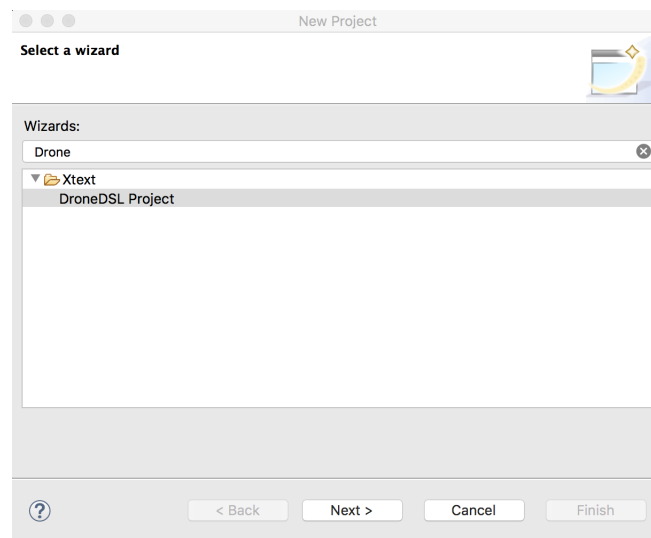


Figure 6: Nouveau projet

Ensuite l'utilisateur doit nommer son projet.

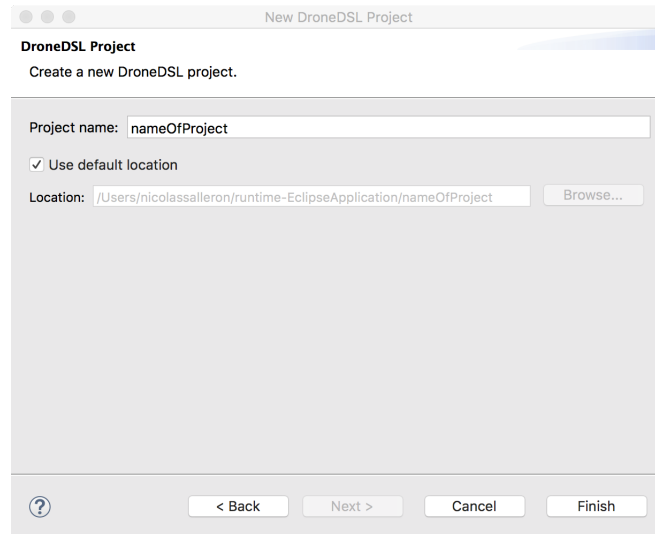


Figure 7: Nom du projet

Il crée sa chorégraphie en spécifiant tout d’abord 5 constantes de vol. Cela permet au programme d’adapter les instructions.

Il définit 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 doit 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 peut faire appel à des fonctions définies par l’utilisateur soit dans le fichier contenant le bloc main, soit dans un fichier `.lib_drone` situé dans le même dossier que le main.

```
/*  
 * 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 8: Exemple de projet

Une fois sa chorégraphie sauvegardée, le programme va générer des classes JAVA qui sont nécessaires pour exécuter la chorégraphie.

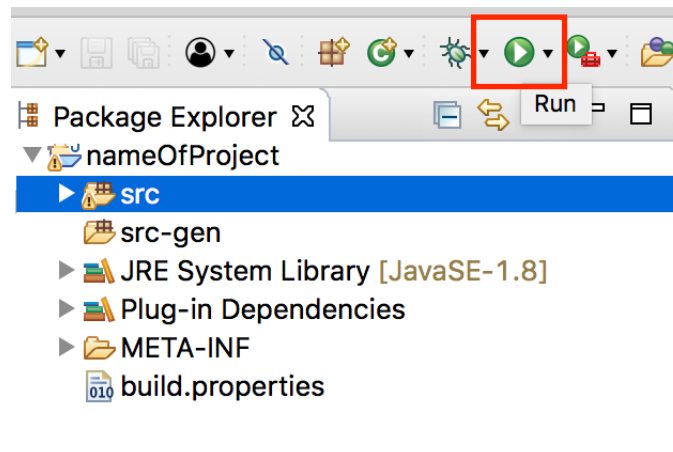


Figure 9: Structure du projet

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.



## 3.2 Langage de conception de chorégraphie

### 3.2.1 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.

### **3.2.2 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).

### **3.2.3 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.

### **3.2.4 Atterrissage**

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.

### 3.2.5 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\_verticale** 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\_verticale** 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.

### 3.2.6 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.

### 3.3 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.

#### 3.3.1 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

### 3.3.2 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%)  
}
```