

IDM - Soutenance 2

Groupe 1 - Les roboticiens

N. Salleron Y. Ghigoff K. Vu-Saintonge A. Archambault

12 Janvier 2018

Sommaire

- 1 Introduction
- 2 Editeur de langage
- 3 Fonctionnement du drone
- 4 Les commandes du DSL
 - Prologue
 - Instructions basiques
 - Instructions parallèles
 - Fonction
 - Le bloc "main" et les bibliothèques de fonctions
- 5 Environnement et tests de validations
 - Environnement

Introduction

Une utilisation de plus en plus populaire

- Utilisation dans le cadre militaire dans les années 1990.
- De plus en plus populaire au sein du grand public.
- Las Vegas, CES 2016, présentation du "drone dance" par la société Parrot
- La télévision avec "La France a un incroyable talent"



FIGURE – Exemple de drone

Introduction

Contexte

- Domaine de la danse.
- Préparation d'une chorégraphie et exécution sur le terrain.
- Création d'un langage de programmation spécifique.

Objectif

- Produire un langage de programmation **simple** et **compréhensible** pour l'utilisateur.
- Rendre l'utilisateur autonome.
- Adaptable à plusieurs drones.
- Faire danser le drone.



FIGURE — Autre exemple de drone

Editeur de langage

Description de l'éditeur

Notre éditeur possède les fonctionnalités suivantes :

- Auto-complétion du langage.
- Détection des fautes syntaxiques.
- Détection des erreurs de cohérence dans les scénarios.

Editeur de langage

Description de l'éditeur

Notre éditeur possède les fonctionnalités suivantes :

- Auto-complétion du langage.
- Détection des fautes syntaxiques.
- Détection des erreurs de cohérence dans les scénarios.

Écriture de la chorégraphie

- L'utilisateur écrit sa chorégraphie par une suite d'actions séparées par un retour à la ligne.
- Une chorégraphie commence par un décollage et un atterrissage.
- Usage possible de fonctions et d'instructions simples.

Editeur de langage

Description de l'éditeur

Notre éditeur possède les fonctionnalités suivantes :

- Auto-complétion du langage.
- Détection des fautes syntaxiques.
- Détection des erreurs de cohérence dans les scénarios.

Écriture de la chorégraphie

- L'utilisateur écrit sa chorégraphie par une suite d'actions séparées par un retour à la ligne.
- Une chorégraphie commence par un décollage et un atterrissage.
- Usage possible de fonctions et d'instructions simples.

Conditions de fonctionnement

- Le drone utilisé est un drone à hélices.
- Le drone est allumé, connecté via Wi-Fi à l'ordinateur.
- Le drone doit être utilisé dans un endroit sans un vent fort.
- Le programme utilisateur réalisé sous Eclipse Oxygen version 4.7.0.
- Java version 1.8.
- Système d'exploitation MacOS, Linux et Windows.

Fonctionnement du drone

Mouvements sur les axes

Le drone évoluant dans un environnement 3D, ce dernier peut effectuer différentes actions :

- Altitude : évolution sur l'axe Z.
Correspond aux instructions "**monter**" et "**descendre**"
- Roll : évolution sur l'axe X.
Correspond aux instructions "**gauche**" et "**droite**"
- Pitch : évolution sur l'axe Y.
Correspond aux instructions "**avancer**" et "**reculer**"

Fonctionnement du drone

Mouvements sur les axes

Le drone évoluant dans un environnement 3D, ce dernier peut effectuer différentes actions :

- Altitude : évolution sur l'axe Z.
Correspond aux instructions "**monter**" et "**descendre**"
- Roll : évolution sur l'axe X.
Correspond aux instructions "**gauche**" et "**droite**"
- Pitch : évolution sur l'axe Y.
Correspond aux instructions "**avancer**" et "**reculer**"

Éléments particuliers

- Décollage et Atterrissage : correspond aux instructions "**decoller**" et "**atterrir**"
- Pause : correspond à l'instruction "**pause**"
- Rotation : correspond aux instructions "**rotationGauche**" et "**rotationDroite**"

Fonctionnement du drone

Mouvements sur les axes

Le drone évoluant dans un environnement 3D, ce dernier peut effectuer différentes actions :

- Altitude : évolution sur l'axe Z.
Correspond aux instructions "**monter**" et "**descendre**"
- Roll : évolution sur l'axe X.
Correspond aux instructions "**gauche**" et "**droite**"
- Pitch : évolution sur l'axe Y.
Correspond aux instructions "**avancer**" et "**reculer**"

Éléments particuliers

- Décollage et Atterrissage : correspond aux instructions "**decoller**" et "**atterrir**"
- Pause : correspond à l'instruction "**pause**"
- Rotation : correspond aux instructions "**rotationGauche**" et "**rotationDroite**"

Cas de la caméra

- Caméra : Il n'est pas standard qu'un drone possède une caméra.



FIGURE — Exemple de drone sans caméra

Prologue

Définir 5 constantes de vol

- **define vitesse_hauteur_max** : vitesse maximale d'élévation du drone pour la chorégraphie.
- **define vitesse_deplacement_max** : vitesse maximale de déplacement sur le plan horizontal du drone pour la chorégraphie.
- **define vitesse_rotation_max** : vitesse maximale de rotation du drone pour la chorégraphie.
- **define hauteur_max** : Cette constante permet de limiter l'altitude maximale du drone en vol.
- **define eloignement_max** : Cette constante permet de contrôler la distance horizontale du drone en vol par rapport au point de décollage.

Exemple de define :

```
define vitesse_hauteur_max 50%
```

Instructions basiques

Le but est de rendre accessible le pilotage de drone aux chorégraphes.

Les 11 instructions basiques

- **decoller**
 - **atterrir**
 - **pause(durée : Seconde)**
 - **monter(durée : Seconde, vitesse_verticale : Pourcentage)**
 - **decendre(durée : Seconde, vitesse_verticale : Pourcentage)**
 - **avancer(durée : Seconde, vitesse_deplacement : Pourcentage)**
 - **reculer(durée : Seconde, vitesse_deplacement : Pourcentage)**
 - **gauche(durée : Seconde, vitesse_deplacement : Pourcentage)**
 - **droite(durée : Seconde, vitesse_deplacement : Pourcentage)**
 - **rotationGauche(durée : Seconde, vitesse_rotation : Pourcentage)**
 - **rotationDroite(durée : Seconde, vitesse_rotation : Pourcentage)**
-
- 1er paramètre est la durée du mouvement, en Seconde.
 - 2ème paramètre est la vitesse du mouvement, en pourcentage. Il représente la vitesse du drone par rapport à la vitesse définie dans la section "prologue".

Les Instructions parallèles

- Notre langage intègre un mécanisme d'exécution d'instructions parallèles.
- Il est possible d'ordonner au drone de faire plusieurs instructions en même temps.
- Mécanisme implémenté par le symbole '&'.

Les Instructions parallèles

- Notre langage intègre un mécanisme d'exécution d'instructions parallèles.
- Il est possible d'ordonner au drone de faire plusieurs instructions en même temps.
- Mécanisme implémenté par le symbole '&'.
- Disponible que pour certaines instructions comme :
 - 1 monter
 - 2 descendre
 - 3 avancer
 - 4 reculer
 - 5 gauche
 - 6 droite
 - 7 rotationGauche
 - 8 rotationDroite

Les Instructions parallèles

- Notre langage intègre un mécanisme d'exécution d'instructions parallèles.
- Il est possible d'ordonner au drone de faire plusieurs instructions en même temps.
- Mécanisme implémenté par le symbole '&'.
- Disponible que pour certaines instructions comme :
 - 1 monter
 - 2 descendre
 - 3 avancer
 - 4 reculer
 - 5 gauche
 - 6 droite
 - 7 rotationGauche
 - 8 rotationDroite
- Maximum de 4 instructions parallélisables.

Exemple :

`rotationDroite(2,25%) & avancer(5,20%)`

Les Instructions parallèles

- Notre langage intègre un mécanisme d'exécution d'instructions parallèles.
- Il est possible d'ordonner au drone de faire plusieurs instructions en même temps.
- Mécanisme implémenté par le symbole '&'.
- Disponible que pour certaines instructions comme :
 - 1 monter
 - 2 descendre
 - 3 avancer
 - 4 reculer
 - 5 gauche
 - 6 droite
 - 7 rotationGauche
 - 8 rotationDroite
- Maximum de 4 instructions parallélisables.

Exemple :

`rotationDroite(2,25%) & avancer(5,20%)`

-> Ok

`monter(1,10%) & descendre(4,20%)`

Les Instructions parallèles

- Notre langage intègre un mécanisme d'exécution d'instructions parallèles.
- Il est possible d'ordonner au drone de faire plusieurs instructions en même temps.
- Mécanisme implémenté par le symbole '&'.
- Disponible que pour certaines instructions comme :
 - 1 monter
 - 2 descendre
 - 3 avancer
 - 4 reculer
 - 5 gauche
 - 6 droite
 - 7 rotationGauche
 - 8 rotationDroite
- Maximum de 4 instructions parallélisables.

Exemple :

`rotationDroite(2,25%) & avancer(5,20%)`

-> Ok

`monter(1,10%) & descendre(4,20%)`

-> Impossible les deux commandes s'opposent.

`gauche(3,25%) & gauche(4,20%)`

Les Instructions parallèles

- Notre langage intègre un mécanisme d'exécution d'instructions parallèles.
- Il est possible d'ordonner au drone de faire plusieurs instructions en même temps.
- Mécanisme implémenté par le symbole '&'.
- Disponible que pour certaines instructions comme :
 - 1 monter
 - 2 descendre
 - 3 avancer
 - 4 reculer
 - 5 gauche
 - 6 droite
 - 7 rotationGauche
 - 8 rotationDroite
- Maximum de 4 instructions parallélisables.

Exemple :

`rotationDroite(2,25%) & avancer(5,20%)`

-> Ok

`monter(1,10%) & descendre(4,20%)`

-> Impossible les deux commandes s'opposent.

`gauche(3,25%) & gauche(4,20%)`

-> Impossible les deux commandes sont de même type.

Les fonctions

- Le langage permet de définir des fonctions.
- Les fonctions sont une suite d'instructions séquentielles.
- Il n'est pas possible de paralléliser deux fonctions.
- Une fonction ne peut pas s'appeler elle-même et les cycles ne sont pas autorisés.

Exemple :

```
func maFonction() {  
    rotationDroite(2,25%) & avancer(5,20%)  
    droite(3,80%)  
    avancer(4,10%)  
}
```

Exemple ne fonctionnant pas :

```
func A() {  
    B() //Erreur sur B()  
}  
func B() {  
    A() //Erreur sur A()  
}
```

Le bloc "main" et les bibliothèques de fonctions

Le point d'entrée "main { }" :

- Défini par le mot clé "main"
- Le contenu de ce bloc d'instructions sera exécuté.
- Il est le seul à pouvoir appeler des fonctions.

Le bloc "main" et les bibliothèques de fonctions

Le point d'entrée "main {}" :

- Défini par le mot clé "main"
- Le contenu de ce bloc d'instructions sera exécuté.
- Il est le seul à pouvoir appeler des fonctions.

Exemple :

```
main {  
    decoller()  
    gauche(1,10%)  
    avancer(4,34%)  
    maFonction()  
    atterrir()  
}  
func maFonction() {  
    rotationDroite(2,25%) & avancer(5,20%)  
    droite(3,80%)  
    avancer(4,10%)  
}
```

Le bloc "main" et les bibliothèques de fonctions

Le point d'entrée "main { }" :

- Défini par le mot clé "main"
- Le contenu de ce bloc d'instructions sera exécuté.
- Il est le seul à pouvoir appeler des fonctions.

Exemple :

```
main {  
    decoller()  
    gauche(1,10%)  
    avancer(4,34%)  
    maFonction()  
    atterrir()  
}  
func maFonction() {  
    rotationDroite(2,25%) & avancer(5,20%)  
    droite(3,80%)  
    avancer(4,10%)  
}
```

Les bibliothèques de fonctions :

- Possibilité d'utiliser des fonctions définies dans des fichiers .lib_drone.
- Doivent être dans le même répertoire que le fichier appelant les fonctions.

Le bloc "main" et les bibliothèques de fonctions

Le point d'entrée "main { }" :

- Défini par le mot clé "main"
- Le contenu de ce bloc d'instructions sera exécuté.
- Il est le seul à pouvoir appeler des fonctions.

Exemple :

```
main {  
    decoller()  
    gauche(1,10%)  
    avancer(4,34%)  
    maFonction()  
    atterrir()  
}  
func maFonction() {  
    rotationDroite(2,25%) & avancer(5,20%)  
    droite(3,80%)  
    avancer(4,10%)  
}
```

Les bibliothèques de fonctions :

- Possibilité d'utiliser des fonctions définies dans des fichiers .lib_drone.
- Doivent être dans le même répertoire que le fichier appelant les fonctions.

Exemple :

```
import <monFichier.lib_drone>
```

Le bloc "main" et les bibliothèques de fonctions

Le point d'entrée "main { }" :

- Défini par le mot clé "main"
- Le contenu de ce bloc d'instructions sera exécuté.
- Il est le seul à pouvoir appeler des fonctions.

Exemple :

```
main {  
    decoller()  
    gauche(1,10%)  
    avancer(4,34%)  
    maFonction()  
    atterrir()  
}  
func maFonction() {  
    rotationDroite(2,25%) & avancer(5,20%)  
    droite(3,80%)  
    avancer(4,10%)  
}
```

Les bibliothèques de fonctions :

- Possibilité d'utiliser des fonctions définies dans des fichiers .lib_drone.
- Doivent être dans le même répertoire que le fichier appelant les fonctions.

Exemple :

```
import <monFichier.lib_drone>
```

Utilisation du nom de fichier (ici : **monFichier**) pour référencer la fonction de la bibliothèque que nous souhaitons utiliser.

Exemple :

```
import <monFichier.lib_drone>  
main {  
    decoller()  
    monFichier.toto()  
    atterrir()  
}
```


Environnement

Application

- Disponible sur MacOS.
- Disponible sur Linux.
- Disponible sur Windows.

Environnement

Application

- Disponible sur MacOS.
- Disponible sur Linux.
- Disponible sur Windows.




Nom	^	Date de modifi
 eclipseLinux.zip		aujourd'hui à 2
 eclipseMacOS.zip		aujourd'hui à 2
 eclipseWindows.zip		aujourd'hui à 2

FIGURE – Distributions

Environnement

Lancement de l'environnement sous MacOS

- Lancement de l'application via le DroneDSLEditor.

Environnement

Lancement de l'environnement sous MacOS

- Lancement de l'application via le DroneDSLEditor.

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 – Lancement de l'application

Environnement

Création d'un projet

- Via l'interface graphique (file -> new -> project).

Environnement

Création d'un projet

- Via l'interface graphique (file -> new -> project).

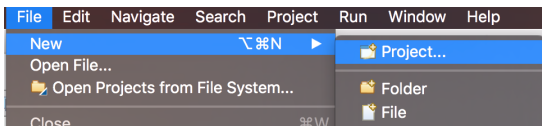


FIGURE – Création d'un nouveau projet

Environnement

Création d'un projet

- Créer un projet Xtext nommé DroneDSL Project.

Environnement

Création d'un projet

- Créer un projet Xtext nommé DroneDSL Project.

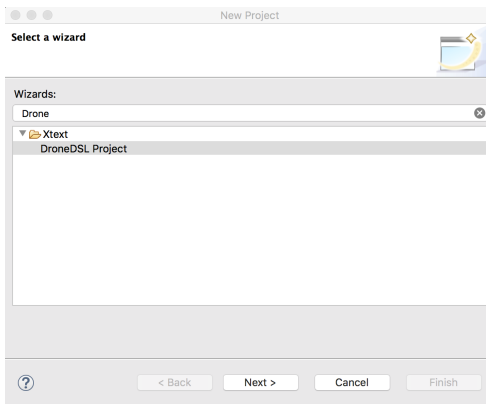


FIGURE – Création d'un nouveau projet Xtext

Environnement

Création d'un projet

- Création du fichier ".main_drone" automatique.

Environnement

Création d'un projet

- Création du fichier ".main_drone" automatique.

```
/*  
 * 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 – Fichiers auto-générés

Environnement

Lancer le projet

- Cliquer sur le dossier "src" puis sur le bouton Run (Bouton play avec un fond vert) une fois qu'il n'y a plus d'erreur syntaxique.

Environnement

Lancer le projet

- Cliquer sur le dossier "src" puis sur le bouton Run (Bouton play avec un fond vert) une fois qu'il n'y a plus d'erreur syntaxique.

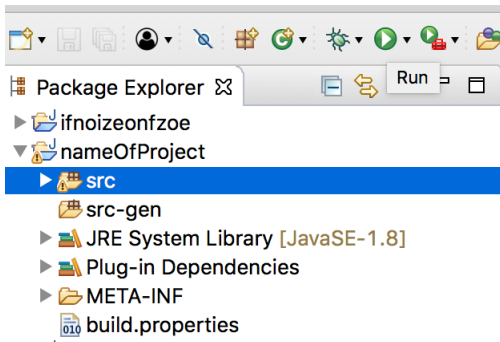


FIGURE – Lancement

Environnement

Lancer le projet

- Pour le moment juste des messages s'affichent sur le terminal.

Exemple :

Environnement

Lancer le projet

- Pour le moment juste des messages s'affichent sur le terminal.

Exemple :

```
import <carre.lib_drone>

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

main {
    decoller()
    carre.carre()
    atterrir()
}
```

FIGURE – Lancement du projet

```
<terminated> Main (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_151.jdk/Contents/Home/bin/
Execution de Prologue [vitesseVerticale=Pourcent [value=10], vitesseDeplacement=Pourcent [value=10]
Execution de Decoller □
Execution de Gauche [duree=Seconde [value=10], vitesse=Pourcent [value=5]]
Execution de Avancer [duree=Seconde [value=10], vitesse=Pourcent [value=5]]
Execution de Droite [duree=Seconde [value=10], vitesse=Pourcent [value=5]]
Execution de Reculer [duree=Seconde [value=10], vitesse=Pourcent [value=5]]
Execution de Atterrir □
```

Fonctionnement

Fonctionnement de la solution

- La chorégraphie écrite par l'utilisateur.
- Chorégraphie interprétée et compilée pour générer des classes JAVA.
- Chaque classe JAVA enverra des commandes au drone en fonction de la runtime

Fonctionnement

```
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 — Lancement du projet

Tests de validations

Validation :	TVEDT-02
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune
Scénario :	<pre>define vitesse_hauteur_max 100% define vitesse_deplacement_max 40% define vitesse_rotation_max 50% define hauteur_max 10 define eloignement_max 4 main { decoller() monter(1,20%) avancer(1,20%) reculer(1,20%) atterrir() }</pre>
Résultat attendu :	Aucune erreur n'est détectée par l'éditeur.

Résultats TVEDT-01 et TVEDT-02

```
main.main_drone  ⌘  
  
define vitesse_hauteur_max 100%  
define vitesse_deplacement_max 40%  
define vitesse_rotation_max 50%  
define hauteur_max 10  
define eloignement_max 4  
  
main {  
    decoller()  
    gauche(1,10%)  
    atterrir()  
}
```

FIGURE – Résultat de TVEDT-01

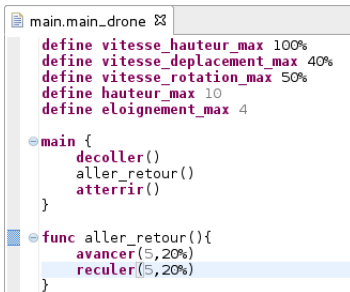
```
main.main_drone  ⌘  
  
define vitesse_hauteur_max 100%  
define vitesse_deplacement_max 40%  
define vitesse_rotation_max 50%  
define hauteur_max 10  
define eloignement_max 4  
  
main {  
    decoller()  
    monter(1,20%)  
    avancer(1,20%)  
    reculer(1,20%)  
    atterrir()  
}
```

FIGURE – Résultat de TVEDT-02

Tests de validations

Validation :	TVEDT-10
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune
Scénario :	<pre>define vitesse_hauteur_max 100% define vitesse_deplacement_max 40% define vitesse_rotation_max 50% define hauteur_max 10 define eloignement_max 4 main { decoller() foo() atterrir() } func bar() { avancer(5,20%) }</pre>
Résultat attendu :	Une erreur est détectée sur l'appel de la fonction <i>foo</i> .

Résultats TVEDT-09 et TVEDT-10

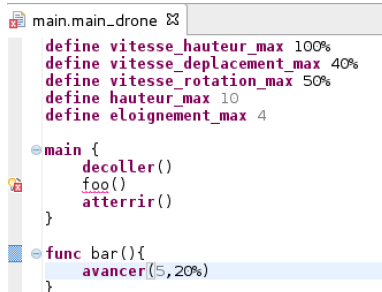


```
main.main_drone  ⌵
define vitesse_hauteur_max 100%
define vitesse_deplacement_max 40%
define vitesse_rotation_max 50%
define hauteur_max 10
define eloignement_max 4

main {
  decoller()
  aller_retour()
  atterrir()
}

func aller_retour(){
  avancer(5,20%)
  reculer(5,20%)
}
```

FIGURE – Résultat de TVEDT-09



```
main.main_drone  ⌵
define vitesse_hauteur_max 100%
define vitesse_deplacement_max 40%
define vitesse_rotation_max 50%
define hauteur_max 10
define eloignement_max 4

main {
  decoller()
  foo()
  atterrir()
}

func bar(){
  avancer(5,20%)
}
```

FIGURE – Résultat de TVEDT-10

Tests de validations

Validation :	TVEDT-12
Contexte :	L'utilisateur a démarré son éditeur.
Entrée :	aucune
Scénario :	<pre>define vitesse_hauteur_max 100% define vitesse_deplacement_max 40% define vitesse_rotation_max 50% define hauteur_max 10 define eloignement_max 4 main { decoller() monter(1,20%) & descendre(1,20%) atterrir() }</pre>
Résultat attendu :	Une erreur est détectée sur l'appel de la composition parallèle des fonctions <i>monter</i> et <i>descendre</i>

Résultats TVEDT-11 et TVEDT-12

```
main.main_drone ✕  
  
define vitesse_hauteur_max 100%  
define vitesse_deplacement_max 40%  
define vitesse_rotation_max 50%  
define hauteur_max 10  
define eloignement_max 4  
  
main {  
  decoller()  
  monter(1,20%) & avancer(1,20%)  
  atterrir()  
}
```

FIGURE – Résultat de TVEDT-11

```
main.main_drone ✕  
  
define vitesse_hauteur_max 100%  
define vitesse_deplacement_max 40%  
define vitesse_rotation_max 50%  
define hauteur_max 10  
define eloignement_max 4  
  
main {  
  decoller()  
  monter(1,20%) & descendre(1,20%)  
  atterrir()  
}
```

FIGURE – Résultat de TVEDT-12

Démonstration

Démonstration sous MacOS