Université de Nice Sophia-Antipolis

Projet Java

2016/2017

SwingRobot

Auteurs :
Youness El Idrissi
Nathan Strobbe

Encadrants :
V. Granet

22 Mai 2017

Sommaire

1 Descriptif du projet

1.1 Principe

Le principe de ce projet est très simple : c'est un robot programmable qui effectue des actions dans un environnement 2D. Celui-ci peut se déplacer, et modifier son environnement : laisser une trace au sol, effacer une trace ou ne rien faire. L'utilisateur peut aisément programmer ce robot en lui donnant des instructions. Ces instructions peuvent être donnés directement par l'utilisateur ou alors par le biais d'un fichier `.txt` .

1.2 Règles

L'utilisateur doit respecter certaines règles pour que le robot puisse interpréter les instructions, en effet il ne comprend pas une instruction si la syntaxe de celle-ci est mauvaise. Dans ce cas, l'utilisateur sera prévenu de son erreur.

La syntaxe est très simple, une instruction doit s'écrire sur **une seule ligne**. Pour en ajouter une nouvelle, il suffit d'effectuer un retour à la ligne.

Liste des instructions disponibles :

- `forward x` : avance le robot de x pixels
- `backward x` : recule le robot de x pixels
- `left x` : tourne le robot de x degrés vers la gauche
- `right x` : tourne le robot de x degrés vers la droite
- `draw` : mode dessin
- `fly` : mode vol
- `erase` : mode effacer

Il est également possible de faire une boucle de type "`répéter n fois`" en écrivant :
⟶ `repeat n` : pour annoncer la boucle.
⟶ `end` : pour terminer la boucle.

Exemple :

```
repeat 5
forward 100
right 144
end
```

2 Programmation

2.1 Choix de programmation

Pour faire fonctionner ce robot, il faut plusieurs classes Java. *(Cf 5. Annexe)* Nous avons donc créer plusieurs classes qui ont chacune leurs utilités. Certaines d'entre elles utilisent les classes de l'interface Swing.

2.1.1 Console.java

Cette classe hérite de la classe `JFrame`, en effet c'est la fenêtre principale où l'on va retrouver toutes les informations. Elle est constituée d'un menu où l'utilisateur pourra obtenir des informations et gérer les fichiers, et constituée d'une grille (`Grid.java`). On va y gérer la vérification des instructions données par l'utilisateur. *(plus de détails dans 3. Algorithme)*

2.1.2 Grid.java

Cette classe hérite de la classe `JPanel`. Celle-ci va gérer les graphiques liés au robot. C'est pourquoi cette classe a un attribut de type `Robot.java`. En récupèrant une ArrayList contenant toutes les actions du robot, on peut changer les graphiques de la grille en fonction de ces actions.

2.1.3 Robot.java

C'est un nouvel Objet Java qui a comme attributs des coordonnées, un angle, un mode de fonctionnement, une liste d'actions effectuées, et une liste **constante** contenant toutes ces actions possibles. Cette classe contient des méthodes permettant de le faire évoluer dans son environnement.

2.1.4 Action.java

On définit un Objet Java qui associe une clé à une certaine valeur. Dans notre cas, nous utiliserons une chaîne de caractères `String` associée à un entier `Integer`.

2.2 Fonctionnement du code

Tout d'abord, il faut créer une console de type `Console.java`, cela est exécuté depuis le `Main.java`.
Ensuite, il y a la mise en place de la fenêtre : installation des paramètres de la fenêtre, ajout d'un menu, d'items, de boutons, du `textArea` et du `JPanel` grille. Également, il y a la création d'un robot dans la Grille.
L'utilisateur peut alors écrire un programme dans le `textArea`, le sauvegarder ou alors ouvrir un fichier texte. Tout cela grâce au menu de la fenêtre.
Lorsque celui-ci lance son programme, la console effectue une vérification de chaque instruction et stocke ces instructions dans l'`ArrayList` action du robot.
Après cela, on demande à la grille d'effectuer un `repaint()`, celle-ci va appeler la méthode `paintComponent(Graphics g)`.
On affiche alors sur le `JPanel` les actions du robot *(plus de détails dans 3. Algorithme)*

3 Algorithmes

3.1 Graphismes du JPanel

La gestion des graphismes se fait grâce à la méthode `paintComponent(Graphics g)`. Celle-ci est appelée lors d'un `repaint()`. On récupère l'`ArrayList` contenant les actions que doit effectuer le robot. Et pour chaque action, on exécute une méthode `paintAction(Action a, Graphics g)` qui va dessiner des graphismes en fonction de l'action donnée en paramètre.

3.2 Vérification des instructions

L'utilisateur doit respecter la syntaxe lors de l'écriture de ses instructions pour que le robot puisse correctement les interpréter. C'est pourquoi il faut faire une vérification de chaque instruction et les comparer à des instructions prédéfinies. On effectue donc une comparaison de chaîne de caractères.
Plus précisément, on récupère tous les instructions et on le stocke dans un `String`. On sépare celui-ci par le retour à la ligne dans un tableau `String[]`.
Finalement, si la syntaxe est correcte alors le robot effectue ses actions, sinon on "jette" une exception (`SyntaxErrorException`) et on "l'attrape" pour ensuite indiquer à l'utilisateur qu'il a fait une erreur de syntaxe.

3.3 Gestion de la boucle "répéter n fois"

Dans l'algorithme de vérification, on regarde en premier lieu si l'instruction est un procédé de boucle grâce à la méthode `checkLoop(String str)`. Si c'est la cas, on récupère le nombre de fois que l'on veut répéter l'action. On parcourt ensuite les autres instructions jusqu'à obtenir un `end` et on répète l'ajout de ces actions le nombre que l'on a stocké précédemment. Si le `end` n'est pas trouvé, alors il y a une erreur de syntaxe.

4 Perspectives

Ce projet est fonctionnel et peut être améliorer. Tant au niveau fonctionnel qu'au niveau algorithmique.
On pourrait penser à de nouvelles fonctionnalités, comme le fait de gérer d'autres procédés itératifs et même pourquoi pas des procédés conditionnels. Notamment le fait de faire plusieurs procédés "`répéter n fois`" imbriqués. Ce qui n'est pas le cas actuellement.
Concernant le robot, on pourrait gérer la couleur d'impression. Nous voulions le gérer mais nous avons eu un problème pour stocker la valeur de sa couleur dans un objet de type `Action<String, Integer>`, il aurait fallu trouver un autre moyen pour le stockage de cette donnée.
Pour ce projet, nous nous sommes inspirés de la *turtle* de Python. Nous avions aussi pensé à faire une animation pour voir le robot effectuer ces actions dans son environnement et même gérer sa vitesse de déplacement.
Ce projet ouvre beaucoup de possibilités.

5 Annexe

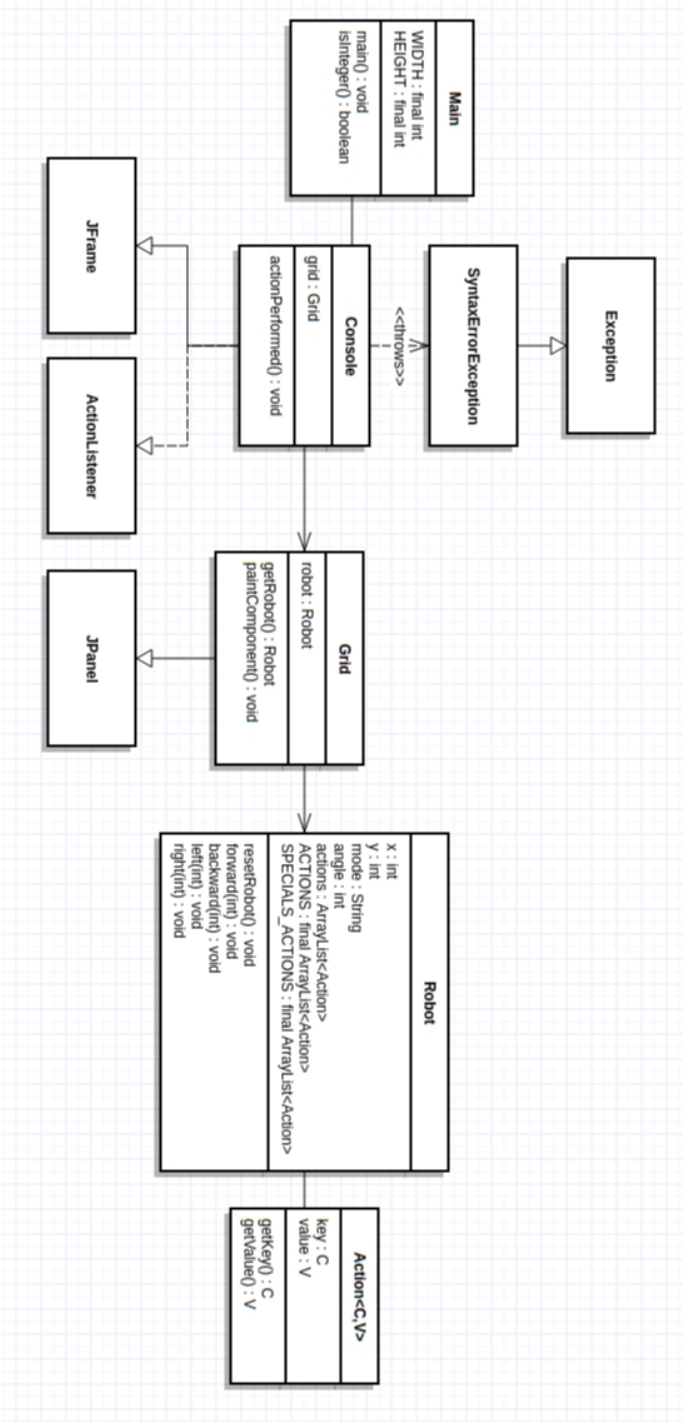

Figure 1: Diagramme UML