BlueSnail - Documentation

Salma Belassal, Alexis Bonnin, Wallid Hassen, Nedhir Messaoud,
Gilles Sourisseau, Papa Traore

Avril 2017

Résumé

Cette documentation s'adresse d'une part aux utilisateurs souhaitant utiliser l'application *roger-runner* et d'autre part, aux développeurs qui veulent créer de nouveaux plugins pour cette application. Vous trouverez également dans cette documentation une description de la plate-forme *BlueSnail*.

Table des matières

1 Introduction

1.1 Présentation du projet

L'application *Roger-runner* est un jeu de d'arcade fonctionnant via la plate-forme *BlueSnail*. Ce jeu consiste à déplacer un personnage tout en évitant les différents obstacles. Il se joue au clavier et plus précisément grâce aux touches directionnelles ainsi que la touche "barre espace".

Cette documentation décrit comment installer la plate-forme et les différents plugins et comment développer un nouveau plugin. Elle présente également une description de la plate-forme *BlueSnail*.

Pour fonctionner, ce projet nécessite :

- Linux
- Java (version 1.7 ou plus)
- Maven (version 3)

1.2 Vocabulaire

Pour éviter toute confusion, voici quelques notions importantes à savoir :

- **plate-forme** : La plate-forme est le point d'entrée du programme. Celle-ci est chargée de gérer les différents plugins ainsi que de lancer ceux en mode "autorun". Tout besoin en terme d'extension doit être formulé à cette plate-forme.
- **application** : On distingue deux types de plugin : les *applications* et les *extensions*. Les applications sont des plugins en mode "autorun", c'est-à-dire qu'ils ne dépendent pas d'un autre plugin et sont exécutés dès le démarrage de la plate-forme.
- **extension** : les extensions sont des plugins dépendant d'une application. Ils ne seront exécutés par la plate-forme que lorsqu'un besoin sera formulé par une application.

2 Installer et utiliser l'application

2.1 Installation

Tout d'abord, le projet est disponible sur GitHub à l'adresse suivante : https://github.com/Asteip/Bluesnail.git. Pour installer le projet, veuillez exécuter les commandes suivantes dans un terminal :

- Cloner le dépôt :

```
$ git clone https://github.com/Asteip/Bluesnail.git
```

- Installer la plate-forme ainsi que tous les plugins (à la racine du projet) :

```
$ ./bluesnail.sh -i
```

- Afficher l'aide :

```
$ ./bluesnail.sh -h
```

2.2 Désinstallation

Pour désinstaller l'application, il suffit de supprimer le répertoire "Bluesnail" contenant le projet.

2.3 Lancer le programme

Pour exécuter l'application, veuillez lancer la commande qui suit dans un terminal :

➢ Exécuter les plugin autorun :

```
$ ./bluesnail.sh -r
```

Ceci aura pour effet de lancer toutes les applications disponibles, c'est-à-dire tous les plugins en mode "autorun". Si une application, une extension ou la plate-forme n'est pas installée alors celle-ci sera automatiquement installée avant l'exécution.

2.4 Comment jouer à Roger-runner ?

But du jeu L'objectif de Roger-runner est d'éviter les aliens apparaissant sur la carte. Le jeu se termine si Roger se fait tuer par un alien.

Commandes Tout d'abord, pour déplacer Roger, il faut utiliser les touches directionnelles : les flèches gauche et droite permettent de déplacer le personnage à gauche et à droite de l'écran. La flèche du haut permet de sauter. Enfin, la touche "barre espace" permet d'envoyer des projectiles sur les ennemis.

3 Développer un plugin

Tout d'abord, le projet est organisé en cinq répertoires :

- ➢ **platform** : Contient le code source ainsi que les fichiers compilés de la plate-forme.
- ➢ **applications** : Contient les différentes applications.
- ➢ **extensions** : Contient les différentes extensions.
- ➢ **documentation** : Contient la documentation technique ainsi que la documentation du code source (java-doc).
- ➢ **template** : Contient des templates utilisés pour la création de plugin.

3.1 Créer un nouveau plugin

Il est possible de créer deux types de plugin : application ou extension (cf. section 1.2). Pour créer un plugin, exécuter une des commandes suivantes :

➢ Créer une application :

```
$ ./bluesnail.sh -c <app-name>
```

Où <app-name> correspond au nom de l'application à créer.

➢ Créer une extension :

```
$ ./bluesnail.sh -c <extension-name> -p <app-name>
```

Où <extension-name> correspond au nom de la nouvelle extension et <app-name> correspond à l'application à étendre.

Une fois le plugin créé, un répertoire portant son nom et contenant le projet maven est créé, soit dans le répertoire "applications" si le plugin est une application, soit dans le répertoire "extensions" si le plugin est une extension. Pour importer un projet maven dans eclipse :

1. Ouvrir la version eclipse située dans */media/Enseignant/eclipse/*
2. Sélectionner "file" → "import" → "Maven" → "Existing Maven Projects" puis "next"
3. Choisir le répertoire contenant le pom.xml (racine du projet)
4. Cliquer sur "finish"

Note : il est possible d'avoir des erreurs de dépendances pendant le développement, pour les corriger un "update" sur le projet peut les corriger (Clic droit sur le projet → "Maven" → "Update project...").

3.2 Configurer un plugin

Lors de la création d'un plugin, celui-ci est configuré "par défaut" et contient les éléments suivants :

➢ Un fichier pom.xml configuré

➢ Un package "com.alma.application" ou "com.alma.extension" suivant le type de plugin.

➢ Une classe générée automatiquement.

Le projet contient également un répertoire pour les tests si l'utilisateur souhaite tester le plugin avant sa mise en service.

Important Le nom du package principal ne doit pas être modifié (il est possible de créer d'autres packages en cas de besoin). De plus, chaque classe créée (quelque soit le projet) doit avoir un nom unique.

Configurer la classe principale d'une application Si le plugin créé est une application, alors la classe principale doit impérativement implémenter l'interface "IMainPlugin" de la plate-forme (Note : la classe à importer est "com.alma.platform.data.IMainPlugin" ; les dépendances sont déjà configurées dans le fichier pom.xml). L'interface "IMainPlugin" impose de redéfinir la méthode run(), cette méthode est automatiquement appelée lors de l'exécution de l'application.

Configurer la classe principale d'une extension Si le plugin créé est l'extension d'une application existante, alors la classe principale de l'extension doit implémenter l'interface correspondant au besoin réalisé (Note : la classe à importer est de la forme "com.alma.application.<interface>" où "interface" correspond au besoin à réaliser ; les dépendances vers l'application son déjà configurées, d'où l'importance de ne pas modifier le nom du package principal).

Configurer la classe principale d'un Monitor Si le plugin créé est un monitor, alors non seulement il doit implémenter l'interface "IMainPlugin", mais il doit aussi implémenter "IMonitor". De plus, pour être pris en compte par la plate-forme, le monitor doit s'ajouter à la liste des monitor (addMonitor(IMonitor monitor), voir java-doc).

Fichier de configuration Le fichier "config.txt" contient les informations nécessaires au chargement des plugin dans la plate-forme. Il est situé à la racine du répertoire "platform". Lors de la création d'un plugin, une ligne est ajoutée à ce fichier contenant des règles préétablies. Celle-ci doit être modifiée afin de répondre au mieux au besoin de l'utilisateur. Les règles suivantes doivent être respectées :

- Un plugin est déclaré sur une seule ligne
- La syntaxe d'une ligne est la suivante (présentée ici sur plusieurs ligne pour une meilleure lisibilité) :

```
name=<nom du plugin>;
class=com.alma.extension.<nom classe principale>;
interface=<nom interface implementee>;
directory=extensions/<nom application>/target/classes/;
autorun=\{true|false\}
```

  Le nom de la classe principale correspond au nom de la classe qui va être chargée en première (ie : celle qui implémente l'interface correspondant au besoin) ; le nom de l'interface implémentée correspond au besoin réalisé.

3.3 Utiliser l'API de la plate-forme

La plate-forme (cf. section 4) offre plusieurs méthodes utilisables par les plugin. Ces méthodes sont détaillées dans la java-doc de la plate-forme (dossier documentation/doc-platform/). Les principales méthodes de l'API sont :

- getInstance() : Retourne l'instance unique de la plate-forme.
- getListPlugin() : Retourne la liste de tous les plugins disponibles.
- getListPlugin(java.lang.Class<?> need) : Retourne la liste des plugins disponibles correspondant au besoin (need). Le besoin est exprimé grâce à une interface que doit implémenter la classe principale du plugin.
- getPluginInstance(PluginDescriptor plugin) : Retourne une instance du plugin passé en paramètre.

Dans la plate-forme, un plugin est représenté par un objet PluginDescriptor. Cet objet contient une description du plugin (nom, classe principale, interface implémentée, etc.). Pour créer une instance d'un plugin, il faut utiliser la méthode "getPluginInstance" prenant en paramètre un objet de type PluginDescriptor.

Exemple d'utilisation de la plate-forme pour instancier un plugin "HighScore" (dans l'application roger-runner) :

```
/* Recuperer tous les plugins implementant l'interface
 * IHighScore. Ces plugins sont recuperes dans des
 * objets PluginDescriptor.
 */
List<PluginDescriptor> listHighScore = Platform.getInstance()
                                    .getListPlugin(IHighScore.class);

/* Pour chaque plugin de la liste, on lance une instance
 * de l'extension.
 */
for(PluginDescriptor plugin : listHighScore){
        IHighScore highScore =  Platform.getInstance()
                                        .getPluginInstance(plugin);
}
```

Listing 1 – Exemple d'extension utilisant la plate-forme

3.4 Compiler un plugin

Pour compiler un plugin, il existe trois possibilités :

- ➢ Sur eclipse : si le projet a été importé sous eclipse, il est possible de compiler directement l'intégralité des projets (le projet de la plate-forme ainsi que les autres plugins doivent être ouvert dans l'IDE).
- ➢ Avec maven : pour compiler le plugin en ligne de commande, il suffit de se déplacer dans le répertoire du plugin et d'exécuter la commande "mvn package".
- ➢ Avec le script bash : la commande "./bluesnail -i" aura pour effet de recompiler tous les plugins existants ainsi que la plate-forme.

4 Description de la plate-forme

4.1 Structure de la plate-forme

La plate-forme est découpée en deux parties : la partie "control" et la partie "data". La première contient les classes "contrôle" de la plate-forme, c'est-à-dire les classes d'interactions avec les classes "extérieures" (plugins). La seconde contient les classes gérant l'accès aux données, avec par exemple la classe PluginParser qui permet de parser le fichier de configuration. Un lanceur est situé à l'extérieur des premier package contient la méthode main permettant d'exécuter la plate-forme. Pour plus d'informations sur les fonctionnalités offertes par la plate-forme, consulter la java-doc de la plate-forme.

4.2 Illustration de l'utilisation de la plate-forme

L'application de base Roger-Runner possède une extension HighScore qui sauvegarde le score courant et le sauvegarde de manière permanente. Lors de l'utilisation de cette extension l'application fait appel à la plate-forme pour lui exprimer son besoin d'avoir une instance de l'extension pour fonctionner. Cet appel se fait de la façon suivante :

```
List<PluginDescriptor> listHighScore =
                        new ArrayList<PluginDescriptor>();
listHighScore = Platform.getInstance()
                        .getListPlugin(IHighScore.class);
```

Listing 2 – Récupération d'une liste de PluginDescriptor avec la contrainte IHighScore

Ici nous récupérons une liste des PluginDescriptors qui utilisent l'interface IHighScore.

```
highScore = (IHighScore) Platform.getInstance()
                        .getPluginInstance(listHighScore.get(0));
```

Listing 3 – Récupération d'une instance de HighScore

L'étape suivante consiste à demander à la plateforme une instance du plugins choisi, ici le premiers de la liste de PluginDescriptor.

Un autre exemple est celui du monitoring. L'extension Monitor est une extension configurée pour être autorun. Donc au lancement de la plateforme, lorsque celle ci parcourt la liste des plugins connus elle charge le monitor automatiquement. Le monitor ensuite dans son constructeur s'insére dans la liste des monitors de la plateforme. Le chargement des plugins en autorun se passe dans la classe launcher dans une boucle qui parcours les plugins autorun :

```
// Starting each plugin in autorun mode
for (PluginDescriptor plugin : platform.getAutorunPlugin()) {
        IMainPlugin mainPlugin = (IMainPlugin) platform
                                        .getPluginInstance(plugin);

        if (mainPlugin != null)
                mainPlugin.run();
}
```

Listing 4 – Chargement des plugins en autorun