RAPPORT PROJET ROBOTWAR PROGRAMATION AVANCÉE M1 MIAGE SOPHIA ANTIPOLIS 2016 :2017

par

NICOLAS KIRCUN, TOM PHILY ET MOHAMMED TANOUTI

Dirigé par : Fabrice Huet

Table des matières

Ce document a pour but de décrire :

- La participation de chacun.
- La planification des grandes étapes pour chacun.
- Les procédures pour tester l'application.
- Les fonctionnalités de l'application.
- Mécanisme de gestion de plugins et de chargement dynamique.
- Le système de sauvegarde.
- Les problèmes de dépendances.
- La gestion du projet.

I. Participation

A. Nicolas Kircun

Mon travaille dans ce projet s'est concentré essentiellement sur les plugins, leurs architectures ainsi que la partie graphique. J'ai donc eu l'occasion de développer la plupart des plugins. Ceux 'par défaut' dans notre jeux (GraphismeBase, DeplacementSimple, AttaqueCourte), ainsi que le plugin de barre de vie.

J'ai également pu implémenté la plupart des interfaces du projet, cela m'a permis de consolider les connaissances que j'ai pu acquérir lors de mes études.

Ce projet m'a également permis de bien me familiariser avec le langage Java et sur les notions appris en TP. Mais également avec l'outil Maven, sur lequel j'avais déjà eu l'occasion de travailler durant mes années d'IUT.

B. Tom Phily

Mon travail au sein du groupe a été de concevoir l'architecture maven du projet, c'est-à-dire divisé le projet en trois modules. Géré les problèmes de dépendance au sein de ces modules. Permettre le lancement de l'application lors de l'exécution du fichier jar après la compilation de celui-ci.

J'ai travaillé en collaboration sur la gestion des plugins avec Mohammed Tanouti, sur la gestion de la vie et de l'énergie dans les plugins attaques, j'ai développé le plugin attaque à distance et implémenté le système de sauvegarde de l'application.

Le projet m'a permis d'utiliser les notions appris en cours concernant la réutilisation du ClassLoader des travaux pratiques sur le chargement de classes à partir d'un fichier jar. L'utilisation des classes internes anonymes pour lancer l'application avec un graphisme de base qui attend un décorateur

C. Mohammed Tanouti

Mon travail dans ce groupe est devisé en deux parties : la première sur le chargement des plugins à partir d'un fichier .jar, j'ai pu développer PluginRepository et SwingRepository en utilisant la notion de ClassLoader vu en cours et travaux pratiques, ainsi que la gestion de ces plugins en collaboration avec Tom Phily. La deuxième partie a été de développer deux plugins de graphisme (GraphismeFleur, GraphismePictureLoaded), et un plugin de déplacement (deplacementSansCollision). Le projet m'a permis de consolider les connaissances acquises durant ce semestre en programmation avancée, et de m'habituer a travaillé avec l'outil Maven et le langage Java.

II. Planification des étapes

Nicolas Kircun

Tâches réalisées	Issues correspondantes
Mise en place du plateau avec deux robots	N/A
Création du package interface	#3
Implémentation des premières interfaces	#3 / #7 / #8
Création du premier plugin GraphismeBase	#4
Mise en place des différents plugins par défaut	#7 / #8 / #9
Création d'une barre de vie et délégation de l'affichage aux plugins de graphisme	#12
Utilisation d'une liste de robots	N/A

Tableau 1 Planification de tâches par Nicolas

Tom Phily

Tâches réalisées	Issues correspondantes
[Architecture] Mise en place de l'architecture maven du projet	#1 / #2
[Moteur] Perte de vie et d'énergie	#11
[Moteur] Système de sauvegarde	#13
[Moteur] Gestionnaire de Plugins	#14 / #21
[Plugins] Patron Décorateur Graphisme	#20
[Plugins] Attaque à distance	#23
[Debeug] Correction de problème de compilation ou d'execution	#5 / #22
[Doc] Rédaction du rapport	#17

Tableau 2 Planification de tâches par Tom

Mohammed Tanouti

Tâches réalisées	Issues correspondantes
Gestions de plugins	#6 / #10 / #14
Graphisme Fleur	#18
Deplacement Collision	#19
ImagePictureLoader	#24
Rédaction du rapport	#17

Tableau 3 Planification de tâches par Mohammed

III. Procédure pour tester l'application

Pour installer et tout lancer :

A. Sous linux :

build.sh

B. Sous Mac OS :

sh build.sh

C. Sous Windows :

Si CigWin d'installé :

sh build.sh

Sinon :

mvn clean

mvn package

java -jar RobotWar-jar-with-dependencies.jar

IV. Les fonctionnalités de l'application

A. Lancement de l'application : Choix du fichier jar

Au lancement, l'application ouvre un *filechooser* dans le but de permettre à l'utilisateur à choisir le fichier jar qui contient la liste des plugins qu'il a développé pour lancer le jeu. L'application ne permet pas de sortir du *filechooser* tant que l'utilisateur n'a pas sélectionné un fichier ou kill le processus. Si le fichier sélectionner n'est pas valide et ne contient aucuns plugins, l'application va s'arrêter en envoyant une exception dans la console. Sinon, l'application se charge avec les plugins de bases et les robots s'affrontent jusqu'à qu'il n'en reste plus qu'un.

B. Choix des plugins

L'utilisateur peut sélectionner les plugins au cours de la partie dans la barre des menus.

Choisir un plugin relance la partie et sauvegarde l'état du jeu dans un fichier *Sauvegarde.xml.* Il ne peut y avoir qu'un plugin déplacement ou attaque au cours de la partie. La sélection d'un plugin attaque ou déplacement désélectionne le précédent pour lancer la partie avec le nouveau plugin.

Cependant l'application peut cumuler les plugins graphismes.

C. Sauvegarde et relance de la partie

L'utilisateur peut à tout moment appuyer sur le bouton *sauve* dans la barre de menu pour sauvegarder l'état des plugins au cours de la partie.

Si ensuite l'utilisateur appui sur le bouton *load*, la partie se relance avec les plugins sauvegardés.

D. Fonctionnement des plugins

Tous les plugins sont regroupés dans le même projet. Pour chaque utilisation différentes (Graphisme, Attaque, Déplacement), les fichiers sont séparés en package afin d'avoir un bon suivi et une bonne architecture dans le projet.

Afin de donner une visibilité globale des plugins dans le projet, ainsi que d'avoir le même fonctionnement dans chaque plugin, ils implémentent tous une interface qui leurs est dédié. Nous avons donc un autre projet interfaces qui regroupe le fonctionnement des objets qui sont partagés en interne du jeu : plugins, robots, projectiles.

V. Gestion des plugins

A. Utilisation des plugins

Les plugins sont affichés dynamiquement dans la barre des menus en fonction du fichier jar sélectionné. Les classes RepositoryLoader et PluginsLoader, développées en cours, vont permettre de parcourir le jar et de charger toutes les classes qu'il contient. RepositoryLoader va ensuite classer ces classes dans des listes attaques, déplacements et graphismes en fonction des interfaces qu'elles implémentent.

Si une de ces listes est vide, l'application s'arrête et renvoie l'exception avec le message d'erreur suivant : *« aucun plugin charge »*.

L'application Swing va ensuite parcourir les listes pour ajouter dans les menus qui leur correspondent de nouvelles actions avec pour nom le plugin et pour effet de sauvegarder et de relancer la partie avec le nouveau plugin sélectionné.

VI. Système de sauvegarde

Le système de sauvegarde utilise les outils XMLEncoder pour encoder des objets sous format XML dans un flux de sorties, XMLDecoder pour décoder des objets d'un flux d'entrée sous format XML, FileOutputStream pour écrire le flux de données dans le fichier « Sauvegarde.XML » et FileInputStream pour lire le fichier.

Nous enregistrons et chargeons la liste des robots qui possèdent des getters et setters sur les noms des plugins à enregistrer ou à charger. Nous pensions devoir enregistrer l'état de vie, d'énergie et les positions des robots pour permettre à l'utilisateur de relancer une partie non terminer. C'est pourquoi nous avons fait le choix d'enregistrer la liste des robots. La fonctionnalité n'a pas été implémentée mais la liste est restée.

Ce qui est intéressant à remarquer c'est la façon dont XMLEncoder écrit les attributs d'un objet, il n'écrit que les attributs possédant un getter et un setter sous la forme suivante :

- Dans une balise void dont l'attribut property de la balise est le nom de l'attribut de l'objet. A l'intérieur de la balise se trouve la valeur dans une balise représentant le type.
- Pour une lsite : dans une balise void dont l'attribut property de la balise est le nom de l'attribut de l'objet et l'attribut id précise le type de liste utilisée. A l'intérieur de la balise de trouve autant de balise void avec l'attribut method add qu'il y a d'éléments dans la liste.

VII. Dépendance entre les projets

L'ensemble des modules dépendent de junit version 3.8.1 pour l'exécution des test unitaires.

Les interfaces ne dépendent pas d'autre module mais utilisent maven-jar-plugin version 2.3.1 pour la création de leur fichier jar.

Les plugins et le moteur dépendent des interfaces et utilisent maven-assembly-plugin version 3.0.0 pour la création de leur fichier jar avec leurs dépendances.

VIII. Gestion de projet

Pour la gestion de projet, nous avons uniquement utilisé les outils proposés par Github sur leur site. Nous avons donc pu créer des tâches et les assignées à un des participants. Cela nous a permis de ne pas travailler sur la même chose et également de pouvoir se répartir les tâches en fonction du domaine sur lequel on développait.

Le plus gros avantage que propose cette méthode sur Github, c'est que les commit sont automatiquement reliés à la tâche correspondante. Il faut cependant indiquer le numéro de la tâche dans le commit. Cela permet d'avoir un bon suivi du projet, et de pouvoir facilement retrouver les modifications que chacun a apporté.

IX. Sources

XMLEncoder : https://docs.oracle.com/javase/7/docs/api/java/beans/XMLEncoder.html

XMLDecoder : https://docs.oracle.com/javase/7/docs/api/java/beans/XMLDecoder.html

FileOutputStream : https://docs.oracle.com/javase/7/docs/api/java/io/FileOutputStream.html

FileInputStream : https://docs.oracle.com/javase/7/docs/api/java/io/FileInputStream.html