09/12/2016

RobotWar

FERRERO – BARNINI - DEÏ

Table des matières

I. Participation des membres de l'équipe

Les différences que l'on peut distinguer dans ces informations sont dues à une différence de niveau entre les membres. Nicholas BARNINI et Karl FERRERO se sont plus axés sur les points fondamentaux du projet pour permettre à Léonard DEÏ de développer sur bases solides.

II. Procédure pour tester le projet

- Dans une console, se placer dans le répertoire contenant le pom principal
- Exécuter la commande : **mvn package**
- Le jar se crée dans le répertoire RobotWar-Moteur
- Se placer dans ce répertoire et exécuter la commande : **java - jar RobotWar-Moteur.jar**
- Une fenêtre apparaît : cliquer sur "Choix du répertoire" et indiquer le chemin où se trouve le dossier *target* du projet *RobotWar-Plugins* (xxx\RobotWar\RobotWar-Plugins\target)
- Ensuite il suffit de cocher les plugins que l'on souhaite voir dans le jeu et de cliquer sur "Lancer la partie".

III. Calendrier des grandes étapes du projet (par membre)

Nous avons décidé, dès le début du projet, de découper le travail en utilisant un système de ticket. Nous allons donc résumer ici les grandes étapes du projet en reprenant les informations des tickets réalisés :

	Nicholas BARNINI	Karl FERRERO	Léonard DEÏ
Choix de conception			
Création de la Grille			
Création du Robot			
Création de Case			
Création du Moteur			
1er plugin de déplacement			
1er plugin d'attaque			
1er plugin graphique			
Gestion du jeu au tour par tour			
Développement de nouveaux plugins			
Chargement dynamique			
Persistance			
Système de log			
Fenêtre au lancement			
Découpage du projet en 3 projets			
Maven			
Perfectionnement du code			

IV. Critères d'évaluation

a. Fonctionnalité

Notre jeu permet à un grand nombre de robots de s'affronter dans une arène. Le jeu se déroule en tour par tour. Le comportement ainsi que le graphisme des robots est décidé par des plugins (choix au lancement du jeu). La partie se termine quand il ne reste plus qu'un robot en vie.

Le moteur de jeu assure le bon déroulement de la partie. Il s'occupe principalement de la gestion des tours, c'est lui qui demande aux robots de se déplacer, d'attaquer et également de se dessiner sur la grille de jeu.

Liste des plugins disponibles :

- DÉPLACEMENT :
  - *Déplacement de base* : Ce plugin permet au robot de se déplacer de manière aléatoire en fonction de ses points de mouvements. On remarquera que le robot peut choisir de rester sur place.
  - *Déplacement intelligent* : Ce plugin, quant à lui cherchera le robot le plus proche disponible et s'en approchera le plus possible. Il récupère dans un premier temps toutes les positions possibles sur lesquelles il pourrait aller. Ensuite il cherchera le robot le plus proche, et finalement il choisira la case la plus proche du robot visé.
- ATTAQUE :
  - *Attaque de base* : Une attaque basique, au corps à corps, faisant 50 de dégâts (un robot à 100 de vie max) et coûtant 50 d'énergies. Le robot peut attaquer à une case autour de lui (les cases diagonales comptent pour 2). Le plugin indique les robots à portée et sélectionne un robot au hasard dans cette liste.
  - *Attaque lourde* : Une attaque plus complexe, à distance, faisant 100 de dégâts et coûtant 100 d'énergies. Cette fois, le plugin désigne, dans la liste des robots à portée, celui qui a le moins de vie.

- GRAPHISME :
    - *Graphisme de base* : Ce plugin donne une couleur random à chaque robot dès le lancement.
    - *Tourelle* : Le premier plugin représentant le robot sous la forme d'une tourelle.
    - *Numéro* : Ce plugin permet d'afficher dans le coin inférieur gauche l'indice du robot.
    - *Barre de vie* : Ce plugin permet de dessiner une barre de vie, au-dessus du robot. La barre de vie se met à jour à chaque action.
    - *Barre d'énergie* : Ce plugin permet de dessiner une barre d'énergie, au-dessus du robot (même principe que la barre de vie). La barre d'énergie se met à jour à chaque action.
    - *Image* : Un plugin permettant de représenter un robot par une image (deux disponibles).

Les plugins de déplacement et d'attaque prennent en compte les diagonales. Ce qui signifie que par exemple pour un robot en (x, y), un déplacement ou une attaque en (x+1, y+1) lui coûtera 2 de portée.

On notera que chaque robot se dessine dans une Case (JLabel). Cela implique que chaque plugin graphique se dessine en fonction de la taille du JLabel, permettant ainsi d'avoir des dessins (Tourelle, barre d'énergie/vie et Image) proportionnels et toujours centrés.

b. Chargement dynamique

Au lancement du jeu l'utilisateur est amené à choisir un répertoire où se trouve les .class de tous les plugins. Le choix des plugins se fait donc uniquement avant le lancement de cette partie. Après cette sélection, la fenêtre se met à jour et affiche la liste des plugins, présents dans le dossier choisi : parcours du dossier en profondeur et récupération d'une liste de File que l'on place dans des JCheckBox personnalisées.

L'utilisateur peut indiquer, en cochant ces cases, les plugins qu'il souhaite voir apparaître dans la partie. Pour que la partie démarre, il doit appuyer sur le bouton "Lancer la partie". Le moteur se lance et le Gestionnaire de plugins prend le relai. Il récupère la liste des plugins choisie (liste de File) et a pour mission de les charger grâce à notre ClassLoader.

Le Gestionnaire possède 3 attributs pour les trois types de plugins possibles (ATTAQUE, DÉPLACEMENT, GRAPHISME). On retrouve le plugin ATTAQUE, le plugin DÉPLACEMENT et une liste de plugins GRAPHISME. Au moment du chargement, le gestionnaire doit donc identifier le

type de plugin de chaque plugin à charger pour les placer au bon endroit. Pour cela, nous avons décidé de nous baser sur les packages. En effet, tous les plugins GRAPHISME se trouve dans un dossier “graphisme”, DÉPLACEMENT dans un dossier “déplacement” et ATTAQUE dans un dossier “attaque”.

Tous les plugins héritent d’une interface en fonction de son type. Un plugin d’attaque va hériter de l’interface IPluginAttaque et devra obligatoirement posséder la méthode : “public IRobot choisirCible (IRobot robot, IGrille grille)”. C’est cette méthode qui sera appelé par le Gestionnaire à chaque fois qu’un robot devra attaquer.

Au cours du jeu, quand le moteur demande à un robot de faire une action (attaquer, se déplacer ou s’afficher), ce dernier va demander au Gestionnaire de plugins d’appeler la méthode du plugin concerné. Par exemple le moteur demande à un robot de se déplacer. Celui-ci va demander au Gestionnaire de plugins son déplacement. Quant à lui le gestionnaire va appeler la méthode seDeplacer(..) du plugin DÉPLACEMENT chargé.

c. Persistance

Notre persistance permet, une fois une partie lancée, de pouvoir retrouver, en relançant le jeu, les plugins sélectionnés précédemment, sans avoir à choisir un nouveau dossier de plugins.

Le fichier de sauvegarde est constitué de plusieurs lignes, chacune représentant un plugin. Chaque ligne est composée de trois éléments : un chemin (pour trouver le plugin), un enum TypePlugin (pour que le GestionnairePlugin le type du plugin) et enfin un booléen (pour connaître l’état du plugin lors du lancement).

La sauvegarde de l’activation des plugins se déroule en plusieurs étapes :

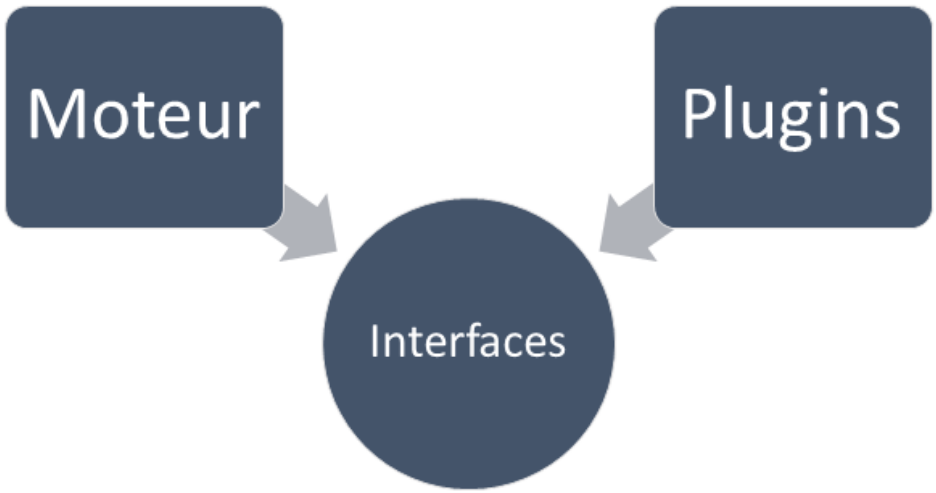
- Dans un premier temps, lors du lancement de l’application, on se retrouve confronté à deux cas. Dans le premier on a un fichier de sauvegarde vide. Si c’est le cas nous n’avons aucune checkbox pré-remplie, et nous pouvons choisir, par le biais d’un JFileChooser, un répertoire où se trouve les plugins. Dans le second cas, où le fichier est déjà rempli grâce à une partie réalisée précédemment, la classe “PanelChoixPlugins” va préremplir des checkbox avec les plugins.

- Dans un second temps, lors du lancement du Moteur, on lui donne un GestionnairePlugins contenant une liste avec les plugins qui ont été choisi par l’utilisateur. La classe GestionnairePlugins, va par le biais de la méthode “parserLigneFichier”; lire et parser la liste donnée précédemment. Pour chaque ligne il va charger le plugin grâce à la méthode “chargerPlugin. Cette méthode renvoi un booléen, s’il est à true, cela veut dire que le plugin a été chargé et on ajoute donc ce plugin et ses informations dans une liste qui sera renvoyée par “parserLigneFichier” et réécris dans le fichier plus tard grâce à la méthode “sauvegardeEtatPlugin”.

d. Modularité

Notre projet est découpé en trois modules distincts :

- RobotWar-Moteur : projet principal, contenant toutes les classes nécessaires au bon fonctionnement du jeu.
- RobotWar-Plugins : contient toutes les classes représentant les différents plugins disponibles.
- RobotWar-Interfaces : permet de faire la liaison entre le projet Moteur et le projet Plugins.



Le projet Moteur connaît Interfaces mais ne connaît pas Plugins. Nous avons dû créer des interfaces pour les plugins pour que le Moteur puisse quand même les utiliser.

Le projet Plugins connaît Interfaces mais ne connaît pas Moteur. Nous avons dû créer des interfaces pour les classes présentes dans Moteur et utilisées par les différents plugins (Grille, Robot, Case).

V. Procédure pour créer de nouveaux plugins

Pour créer un nouveau plugin il suffit se placer dans le projet RobotWar-Plugins puis dans le package faisant référence au type du plugin que l'on souhaite créer. On crée une nouvelle classe et on l'a fait hériter de l'interface correspondante. Ensuite, il suffit de détailler la méthode à implémenter.

Exemple : Création d'un nouveau plugin pour le déplacement

- On se place dans le package "déplacement" du projet RobotWar-Plugins.
- On crée une classe qui hérite de l'interface "IPluginDeplacement".
- On détaille la méthode principale, à implémenter obligatoirement : *public Point choisirDeplacement(IRobot robot, IGrille grille).*