Université Nice Sophia Antipolis

Master 1 MIAGE

Documentation projet Montou

Balbis Robin, Dien Maxime, Fourmy Hugo, Mourton-Comte Noé

2017-2018

La participation de chacun des membres de l'équipe :

- BALBIS Robin : Réalisation des plugins “ConvergentMovement” qui permet à un robot de se rapprocher de son opposant, réalisation du plugin “RangeAttacksViewer” qui permet de visualiser la portée des attaques des deux robots.

- DIEN Maxime : Réalisation de plugin “HandToHandAttacks” qui permet l’attaque au corps à corps d’un ennemi.

- FOURMY Hugo : Réalisation du menu pour le lancement/rechargement d’une partie et choix des plugins à charger, réalisation du plugin “RangedAttacks” qui permet l’attaque à distance d’un ennemi.

- MOURTON-COMTE Noé : Réalisation du module de chargement des plugins, base du jeux, Plugin “RobotsGraphic” qui permet la visualisation des robots sur la grille de jeux, plugin “RandomMovements” qui fait bouger le robot de manière aléatoire.

Calendrier des étapes :

Balbis Robin :

- Semaine 1 : conception commune avec l'equipe
- Semaine 2 : résolution de différent problème et développement de plugins de mouvement
- Semaine 3 : développement du plugin d'affichage de la porté des attaques.

Dien Maxime :

- Semaine 1 : conception commune avec l'equipe
- Semaine 2 : réalisation d'une première version des modules du projet
- Semaine 3 : développement du plugin d'attaque

Fourmy Hugo :

- Semaine 1 : conception commune avec l'equipe
- Semaine 2 : développement du plugin d'attaque à distance.
- Semaine 3 : amélioration de la boucle de jeux et ajout d'un menu pour le choix des plugins a charger.

Mourton-Comte Noé :

- Semaine 1: conception commune avec l'equipe
- Semaine 2 : Réalisation de la base du projet, de la base de jeux, chargement des plugins et développement des plugins de déplacement aléatoire et affichage des robots sur la grille
- Semaine 3 : amélioration de la boucle de jeux, du menu de choix des plugins

Lancement du projet :

Exigences

1. Eclipse.
2. JDK ≥ 8.
3. Maven ≥ 3.3.9.

! Il est indispensable de modifier la valeur project.properties.jdk du Montou/pom.xml avant de continuer le lancement du projet.

Première solution - Utiliser la "LaunchConfiguration".

1. Sous Eclipse > Run > Run Configurations... > Maven Build > LaunchConfiguration.
2. Personnaliser les arguments de lancement à votre guise :
   exec.args > -pluginsJarPath -persistenceDirPath.
3. Run.

Deuxième solution - Lancement en ligne de commande avec Maven.

1. Taper dans la console Maven :
   mvn exec:java -Dexec.mainClass="com.montou.game.engine.Launcher" -Dexec.args="pluginsJarPath persistenceDirPath"
2. Pressez la touche "Entrée".

Troisième solution - Lancement avec JAR.

3. Se positionner (via la console) dans le dossier "Demonstration".
4. Exécuter la ligne de commande suivante : "java -jar Game-0.0.1.jar BasePlugins-0.0.1.jar /Persistence"

Fonctionnalité

Lorsqu'on lance l'application un menu s'affiche nous proposant soit de reprendre une partie déjà commencé, soit de lancer une nouvelle partie, ensuite nous avons la possibilité de choisir les plugins que l'on souhaite charger, notamment les plugins graphiques commun aux deux robots et les plugins d'attaques et de déplacements en fonction du robot.
Puis la partie se lance les deux robots sont initialement placé dans un coin opposé de la grille de jeu puis les robots attaque et se déplace chacun à leur tour jusqu'a ce qu'un des deux robots n'ai plus de points de vie ou que le joueur décide d'arrêter la partie.

Chargement Dynamique

Les plugins disponibles en début de partie sont recherchés dans le dossier que l'on aura spécifié en paramètre (`pluginsJarPath` dans la variable exec.args), ensuite la liste des plugins s'affiche, on peut ainsi choisir quels plugins devront être chargés pour chacun des robots. En fonction des classes chargées et des annotations présentes dans celles-ci, les classes (instanciées par types de plugin) sont ajoutées dans une liste de "Plugin".

Persistance

Persistance du core (Etat du plateau) :

A chaque fin de tour, le jeu sauvegarde la partie. Pour cela il serialize l'objet gameInformation qui contient la taille de la grille, les différents robots (position, hp, energie, …) l'etat de la partie (finie ou non) et les différents plugins qui étaient utilisés. Cet objet sérialisé est enregistré dans le dossier Persistence spécifié sous le nom de gameInformation.save.

Lors de la reprise d'une partie, on récupère les données de ce fichier et on les inclus par la désérialisation, ce qui permet de restaurer la partie précédente.

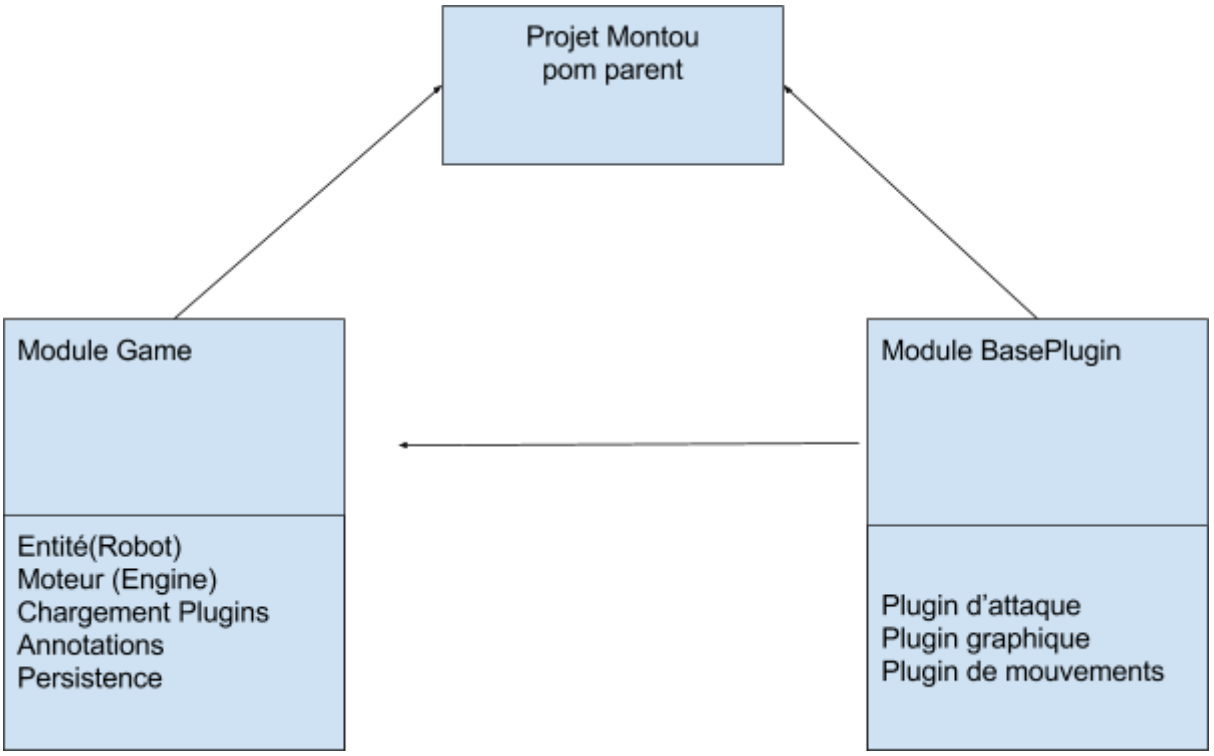
Persistance des plugins :

Pour les plugins souhaitant être persistant (RobotDesigner par exemple souhaite conserver les couleurs utilisées pour les joueurs) Il leur faut spécifier des méthodes.

- Save() qui servira à la sauvegarde de l'état du plugin à chaque tour de jeu (avec le même mécanisme de sérialisation que dans le coeur, un fichier NomDuPlugin.save est créé pour chaque plugin).
- Et onLoad() qui restaurera le plugin en cas de reprise de partie (avec le même mécanisme de dé-sérialisation que pour le coeur).

Modularité

Le projet possède l'architecture suivante :



Comme on peut le voir ci-dessus, le projet montou (le projet parent) dispose de deux sous projets (soit modules) qui se dénomment “Game” et “BasePlugin”. Le module “Game” dispose des fonctionnalités principales du jeu. Tandis que le module “BasePlugin” propose trois différents types de plugins pouvant être chargés dans notre application.

De plus, notre “BasePlugin” dispose d’un lien de dépendance vers le module “Game” car ce dernier nécessite de certaines classes appartenant au projet “Game” (notamment “Robot”, “Direction”,et ”GameInformations”) pour pouvoir implémenter des plugins compatibles avec le moteur de jeu.

Suivi

Lors de la réalisation de ce projet, nous avons utilisé Slack, une plate-forme de communication collaborative, pour une communication simple et efficace entre les différents membres du groupe.

De plus, pour un suivi constant du projet Mouton, le service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git, Git Hub a été exploité lors de ce projet.

Développer un plugin

Il existe trois types de plugins bien distincts à savoir :

1. Les plugins graphiques

Ils servent à afficher des objets ou des entités sur le plateau de jeu.

2. Les plugins de gestion de mouvements

Ils servent à définir un comportement concernant le prochain mouvement qu'un robot va effectuer.

3. Les plugins de gestion d'attaques

Ils servent à définir un comportement concernant la prochaine attaque qu'un robot va (ou non) effectuer.

Développement d'un plugin graphique

Annotation à préciser *(sur votre classe)* : @AGraphic.

Méthode à implémenter :

```
public void draw(GameInformations gameInformations, Graphics g)
```

Vous pouvez donc utiliser l'objet `Graphics g` pour dessiner sur le plateau de jeu ce que bon vous semble.

Vous disposez également d'informations concernant la partie grâce à l'objet `GameInformations gameInformations`.

Développement d'un plugin de gestion de mouvements

Annotation à préciser *(sur votre classe)* : @AMovement(energyCost = n).

Méthode à implémenter :

```
public Direction move(GameInformations gameInformations, int currentPlayer)
```

Vous devez donc retourner une `Direction` qui sera utilisée par le moteur de jeu afin de faire bouger le robot en question *(currentPlayer)* lors du prochain tour.

Développement d'un plugin de gestion d'attaques

Annotation à préciser *(sur votre classe)* : @AAttack(energyCost = n, range = n).

Méthode à implémenter :

```
public int attack(GameInformations gameInformations)
```

Vous devez retourner un entier qui correspond aux dommages appliqués au robot ennemi.

Rendre votre plugin persistant

Tous les types de plugins peuvent implémenter le système de persistence mis à disposition.

Attribut à préciser selon votre type de plugin : @AGraphic(useCustomData = true), @AMovement(useCustomData= true) ou @AAttack(useCustomData = true).

Méthodes à implémenter :

```
public void onLoad(Object[] objects)
```

Cette méthode est appelée lors de la reprise d'une partie pour que le plugin puisse retrouver l'état dans lequel il a été fermé.

```
public Object[] onSave()
```

Cette méthode est appelée lors de la fin d'un tour afin de sauvegarder l'état du plugin au cas où la partie s'arrêterait de manière innatendue.

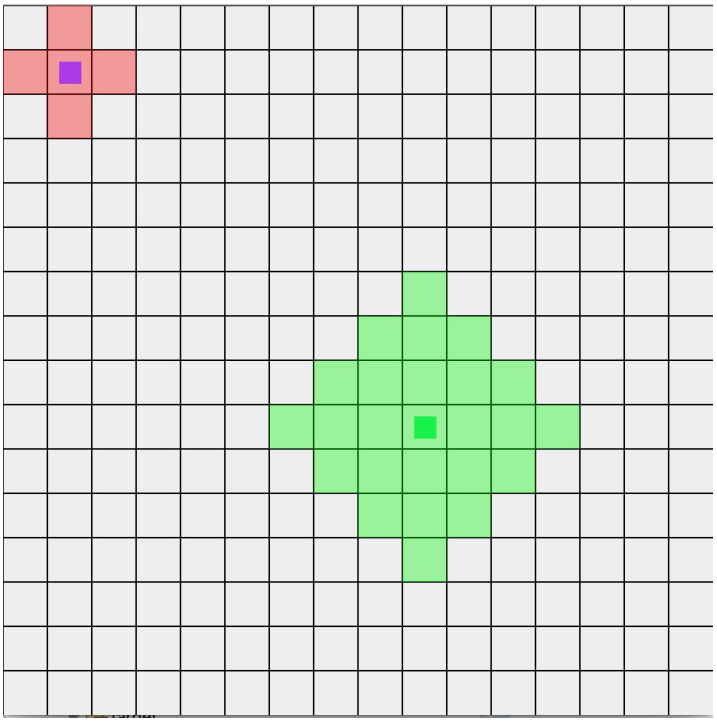
Pour une meilleure compréhension du système de plugins, nous vous conseillons d'étudier le module BasePlugins.

Présentations des plugins

- RangedAttacks
  Attaque à distance du robot ennemi inflige 10 points de dommages, portée de 3 cases autour du robot.

- HandToHandAttacks
  Attaque au corps à corps du robot ennemi inflige 15 points de dommages, portée de 1 cases autour du robot.

- RangeAttacksViewer
  Permet de visualiser la portée de l'attaque chargé par le robot sur la grille, un halo de couleurs s'affiche alors là où le robot peut attaquer.



- RobotDesigner

Robot graphique permet d'afficher des carrées sur la grille de jeux representant les deux robots.

- ConvergentMovement

Convergent movement permet de faire bouger le robot qui a chargé ce plugin dans la direction du robot ennemi.

- RandomMovement

Random movement permet de faire bouger le robot qui a chargé ce plugin dans une direction aléatoire.