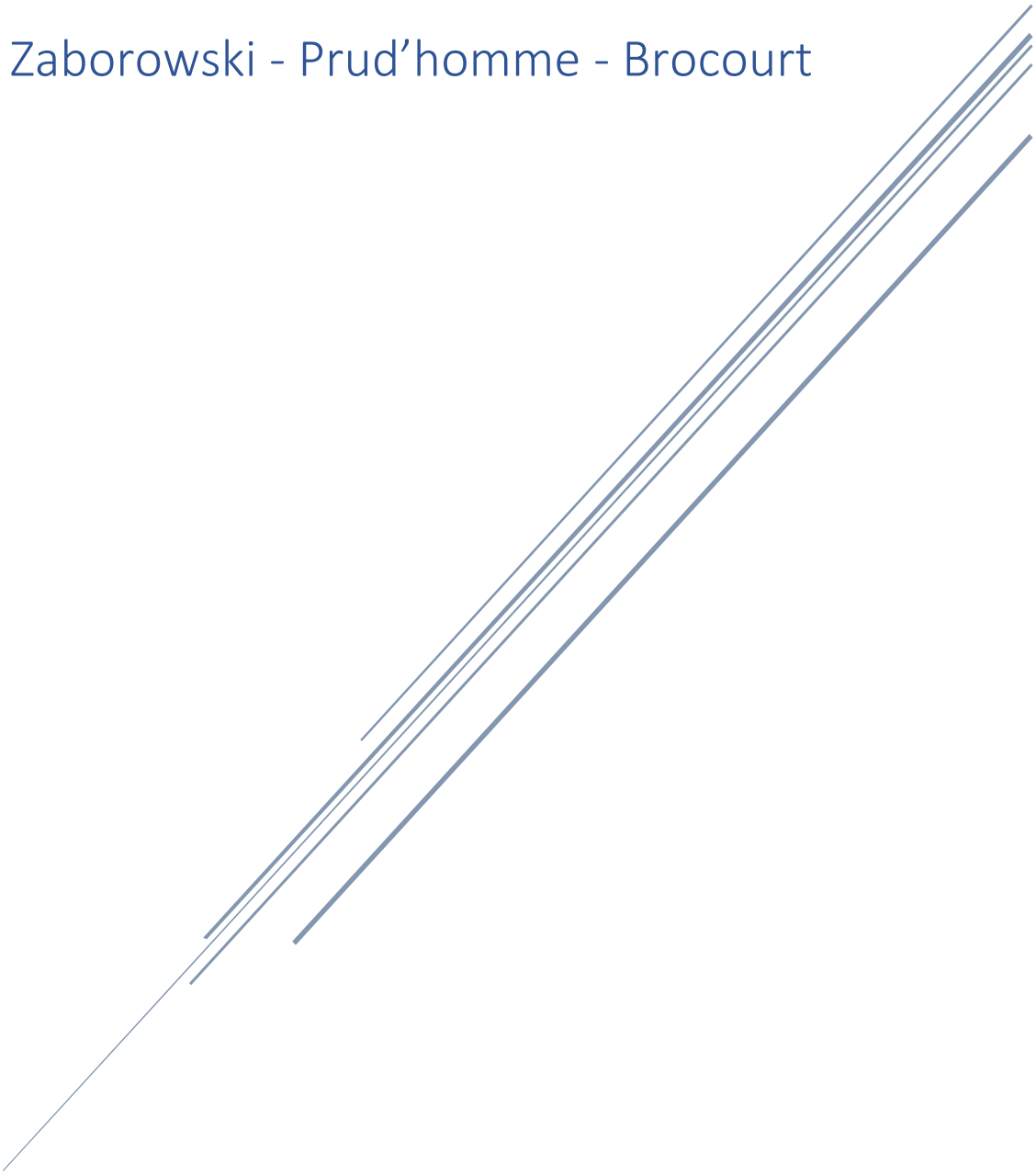
PROGRAMMATION AVANCEE

ROBOT WAR

Équipe FatalPicards

Zaborowski - Prud'homme - Brocourt



Master 1 MIAGE
Programmation Avancée Fabrice Huet

Introduction

Le projet réalisé devait mettre en évidence une architecture de plugin basé sur des annotations. Ce type d'architecture repose essentiellement sur le ClassLoader qui permet de charger dynamiquement les différents plugins. Une fois cette architecture bien mise en place, les plugins chargés servent à créer un jeu de robot 2D dans une arène ou ceux-ci s'entretuent jusqu'à ce qu'il ne reste qu'un seul robot en vie.

Le but des plugins est de pouvoir ajouter des comportements ou des visuels à un robot grâce à une interface graphique basique. Différents plugins de base sont obligatoires pour lancer le jeu, notamment un plugin graphique pour dessiner les robots, un d'attaque pour qu'il puisse s'entretuer et enfin un de mouvement afin qu'il puisse se rencontrer dans l'arène.

Participation des membres

Pour réaliser le projet en entier différents découpages ont été réalisés au sein du projet afin de distinguer 6 grandes phases de développement :

- **Phase 1 (Prud'homme et Brocourt et Zaborowski) :** Cette première phase a pour but la conception du projet ainsi que la mise en place des objectif et la stratégie de développement.

- **Phase 2 (Prud'homme) :** La seconde consiste à mettre en place un moteur de jeu basique contenant différents composants tels que : l'arène, les robots basiques, l'architecture du projet etc...

- **Phase 3 (Prud'homme et Brocourt) :** La troisième phase quant à elle permet de développer les plugins de base afin de pouvoir jouer au jeu basiquement en lançant le programme via notre IDE. Cette phase est composée de différents composants telles que : le design des robots (pour le moment des carré), un mouvement basique pour qu'il puisse se rencontrer et enfin une attaque courte portée pour qu'ils puissent se tuer. Tous ces composants reliés à une arène et un moteur de jeu faisant tourner le tout.

- **Phase 4 (Prud'homme et Brocourt) :** La quatrième phase permet de réaliser les plugins avancés permettant d'enrichir le jeu. Pour notre cas nous aurons donc une attaque longue portée, une barre de vie et des déplacements différents. Parmi les déplacements nous en aurons deux nouveaux un qui permet de trouver la cible la plus proche et un permettant de se déplacer intelligemment en fonction du type d'arme qu'il possède.

- **Phase 5 (Zaborowski) :** La cinquième phase consiste à pouvoir mettre en place le PluginLoader du jeu. Cette partie primordiale permettra de charger les plugins sous la forme de .jar. Une fois cela réalisé, différents mécanismes permettront de charger les fonctionnalités du jeu comme auparavant en ajoutant les plugins voulus via une interface graphique basique.

- **Phase 6 (Prud'homme) :** La dernière phase consiste à mettre les plugins sous forme de .jar mais également de les relier à des annotations comme précisé dans l'énoncé. Cette phase permet d'avoir un jeu fonctionnel et dynamique grâce au gestionnaire de chargement. Durant cette phase architecture du projet prendra la forme voulut avec un projet Maven suivi de 4 modules : Game, Shared, MandarotyPlugin, OptionalPlugin.

Toutes ces phases ont été décrites et réalisées afin de répartir les tâches au sein de l'équipe pour que chacun choisisse la partie du projet qui leur plaise. Bien sûr, certaines parties nécessitent de travailler à plusieurs et sont critiques au projet.

Avant la réalisation de ces 5 phases, plusieurs tâches ont été réalisés en commun :

Procédure de test du projet

Calendrier des tâches

Tâches communes

Conception du projet	Le 17 décembre
Objectifs et stratégie de développement	Le 18 décembre

Louis :

Créer une arène 2D	28 décembre
Créer des Robots	28 décembre
Créer le moteur	28 décembre
Création du graphique de base	28 décembre
Création du mouvement aléatoire	29 décembre
Création de mouvement avancé	4 janvier
Refractor de l'architecture	4 janvier / 8 janvier
Plugins avec annotations	10 janvier

Maxime :

Création du plugin Attaque courte portée	29 décembre
Mise à jour de l'attaque	4 janvier
Création de l'attaque longue portée	4 janvier
Modification du graphique de base	7 janvier
Création du graphique avancé barre de vie	8 janvier
Ajout de l'interface pour charger les plugin	9 janvier

William :

Créer le PluginLoader (première ébauche)	28 décembre
Création du PluginLoader final	7 janvier
Un ClassLoader par robot	8 janvier
Modification du PluginLoader	9 janvier
Modification de la structure	9 janvier
Chargement des plugins via l'interface	10 janvier

Fonctionnalité du projet

Implémentation

Afin de pouvoir réaliser le jeu de Robot, nous devions distinguer les différentes fonctionnalités à implémenter sur le projet.

La première fonctionnalité du jeu est de pouvoir créer une partie et mettre en place X robot(s) afin qu'ils se battent en duel jusqu'à la mort. Cet environnement est développé dans une d'architecture de plugin. Afin de pouvoir implémenter cette fonctionnalité différentes étapes seront nécessaires :

- Mettre en évidence les différentes classes qu'il faudra créer afin de parvenir à un affichage basique de 2 robots sur une JFRAME se battant en duel. En décomposant notre fonctionnalité on se rend compte que différents éléments seront nécessaires pour mettre en place ce visuel. Le premier est de créer les class permettant de créer un Robot basique, de créer une attaque basique et de créer un mouvement aléatoire pour se déplacer sur l'arène. Une fois ces classes créées nous devons associer toutes ces classes afin de pouvoir créer un Robot avec une graphique pouvant attaquer et se déplacer. Enfin nous allons créer notre moteur qui va générer plusieurs robots et qui vont s'attaquer tour par tour jusqu'à ce que mort s'en suive. Pour finir et générer un affichage graphique afin de visualiser les robots nous allons créer une main qui va lancer une JFRAME et ajouter dans celle-ci l'arène de combat avec les robots chargé dedans, le mécanisme de tour par tour est implémenté grâce à un Thread qui ne s'arrête qu'une fois un seul robot existant sur le champ de bataille.

- Mise en place de plugin, une fois la structure de base bien défini les différentes classe et implémentation des attaques, graphiques et mouvement devrons être transformé en plugin. De cette manière le futur PluginLoader pourra charger ces classes afin de lancer le jeu de manière dynamique. Cette modification nécessite une adaptation du code des plugin ainsi que de l'architecture Maven. Nous passons désormais avec une architecture composée de module. Chaque module à un but bien précis, le premier Game sert à lancer le jeu avec le moteur, le PluginLoader… Le second quant à lui sert à stocker les différents plugins de bases c'est ici que le PluginLoader ira piocher pour récupérer tous les plugins dont il a besoin. Enfin le dernier Shared sert à mettre en place des ressources qui seront partagé par les plugins et la Game.

- Mise en place de plugin avancé avec chargement dynamique, cette partie consiste à mettre en place différents types de plugins avancés. Dans cette partie nous aurons les différents plugins non nécessaires au fonctionnement du jeu mais qui permettre de l'enrichir. Nous pouvons par exemple ajouter un plugin d'attaque à distance, un pour ajouter une barre de vie ou encore certains pour changer le déplacement du Robot. HugeMove et SwarzyMove sont des déplacements spécifiques qui permettent aux robots de se déplacer sur la cible la plus proche ou encore pour SwarzyMove de se déplacer intelligemment en prenant en compte le type d'arme qu'il possède. De plus cette partie inclura également la notion du PluginLoader qui va charger tous les .jar. Une fois ceux-ci charger une interface graphique basique permettra de savoir toutes les 5 secondes les plugins disponibles et de les charger en cliquant dessus.

- Mise en place des plugins avec annotations, cette partie consiste à modifier le fonctionnement des plugins de base ou avancé afin de les charger grâce à différentes annotations. Ce changement inclut pour tous les plugins une profonde réforme ainsi que pour le PluginLoader qui ne se contentera plus de charger les méthodes des classes une fois le plugin récupérer mais, ici de charger les différentes annotations afin de pouvoir lancer telles ou telles plugins.

- Génération d'un jar exécutable pour lancer le jeu, cette étape correspond à la dernière du jeu. Une fois l'architecture de plugins réalisée avec le chargement dynamique des plugins s'exécutant grâce à des annotations, il ne reste plus qu'à générer un jar exécutable afin de pouvoir lancer le jeu

Aspects remarquables

Chaque phase compose différents aspects que nous pouvons noter dans cette partie :

- Les plugins sont tous configurés via des annotations, ces annotations sont chargées via le PluginLoader qui va aller chercher les différentes annotations qu'il trouve pour savoir qu'elle type de plugin nous avons à faire et qu'elle classe nous devons charger.

- L'architecture du projet, en effet dans celui-ci nous avons un projet Maven basique avec différents modules qui sont tous reliés grâce à différentes dépendances. Ces modules sont comme je l'ai dit précédemment au nombre de 4. Nous distinguons ainsi le moteur du jeu, le PluginLoader etc. Des différents plugins. Nous avons également deux sortes de plugins qui permettent de savoir qu'elles sont ceux obligatoires et qu'elles sont ceux optionnels, de cette manière nous avons une hiérarchie et un chargement basique avec ceux obligatoire.

Mécanisme de gestion de plugins et chargement dynamique

Implémentation

Dans cette partie nous allons montrer l'implémentation du PluginLoader ainsi que la gestion des différents plugins puis leurs constructions avec les annotations. Cette partie sera donc séparée en 3 grandes parties distinctes : le Plugin Loader, la gestion des plugins et, la construction d'un plugin.

La gestion du Plugin Loader

En ce qui concerne le Plugin Loader, nous nous sommes référés au TP3 effectuée en TD.

Dans un premier temps, on créé le PluginLoader en lui rentrant en paramètre le chemin où se trouve les jar files. Ainsi, lors de sa construction, cette classe va chercher tous les jar file dans le dossier et dans les sous dossiers et reporter le chemin de chaque jar file dans une liste.

Lorsque la recherche est terminée, et la liste prête, le PluginLoader va instancier en premier lieu URLClassLoader. Ce ClassLoader permet de charger les classes dans des jar file du moment où on lui informe le chemin de chaque jar file.

Ainsi, PluginLoader va parcourir chaque jar file et récupérer les fichiers se finissant par « .class » et en les charger grâce à URLClassLoader. Chaque classe chargée sera stocké dans une liste de classe.

PluginLoader dispose ainsi d'une liste contenant les classes chargées des plugins.

Lorsque qu'une classe veut charger un fichier d'un plugin, elle peut prendre la classe chargée voulue dans la liste et appeler la méthode désirée.

La gestion des plugins

Les plugins dans notre projet sont gérés dans un module. Comme je l'ai précisé plus haut l'architecture du projet est composée de 4 module. Le premier tous l'aspect Game (la gestion du jeu, le moteur, le PluginLoader), la seconde sur les parties partagées avec le module « Shared », et enfin la gestion des plugins.

Dans cette partie, les plugins sont séparés dans 2 modules distincts :

- Le premier module contient tous les **plugins de base**. Ce module est nécessaire car il permet au PluginLoader de charger les différents plugins nécessaires au fonctionnement du jeu. Dans ce module nous avons les 3 plugins de base :
  - Un plugin graphique qui permet d'avoir une visualisation des robots en petit carré,
  - Un plugin d'attaque qui permet à un robot d'avoir une attaque de courte portée, on peut caractériser ça sur une épée et enfin
  - Un plugin de déplacement qui permet à un robot de trouver la cible la plus proche afin de l'attaque.

  Une fois la partie lancée, les plugins se lancent pour finir la partie et seul un survivant pourra remporter la partie.

- Le deuxième module quant à lui contient **tous les plugins avancés**. Ces plugins sont comme pour ceux de base séparée en 3 différentes partie. Le plugin graphique avec l'ajout d'une barre de vie à un robot selon le patron décorateur permettant d'enrichir les petit carré en ajoutant une barre de vie en dessous des carrés. Nous avons également un plugin graphique

permettant de dessiner le rayon de portée autour du robot. Nous avons également les plugins d'attaque avancée avec celle longue portée que l'on peut caractériser avec un arc. Le second plugin avancé d'attaque est celle « Phoenix » qui permet à un robot de sacrifier de sa vie afin d'attaquer un autre robot. Pour finir la partie plugin avancé nous avons ceux de déplacement comme :

- Le plugin de déplacement aléatoire : les robots dans l'arène se déplacent aléatoirement jusqu'à ce qu'il rencontre un robot et donc va l'attaquer
- Le plugin de déplacement chaotique : ce type de déplacement va détecter la cible la plus proche pour aller l'attaquer. Une fois la cible taper il va constamment en chercher une autre pour l'attaquer. Cela conduit à la convergence des robots vers un point où ils s'entretuent
- Le plugin de déplacement « Swarzy » : de la même manière que M. Swarzy les robots vont détecter la cible la plus proche et l'attaquer jusqu'à ce qu'il meure ou que lui-même meurt. Cette opération s'exécute continuellement jusqu'à ce qu'il ne reste qu'un seul robot.

Construction d'un plugin

Pour finir cette partie nous allons expliquer comme les plugins de base/avancé se construisent. Chaque plugin possède différentes annotations, dès qu'une classe possède en haut de celle-ci l'annotation *@Plugin* cela signifie que nous avons un plugin de plus à l'intérieur de cette annotation nous aurons un type (Move, Attack, Graphic). Chacune des méthodes de cette classes seront également annotés afin de pouvoir les charger par la suite. Nous aurons donc des annotations de ce type sur les méthodes *@move(argument).* Dans les arguments, nous aurons différents types comme « MAIN », « POWER » … cela permet de pouvoir détecter le type de méthode dans le Plugin Loader.

Aspects remarquables

Bien que le jeu est fonctionnel est charge correctement les plugins. Les différents plugins et le PluginLoader ne sont pas testés, pour cette raison la stabilité du projet ne peut être garantie dans plusieurs environnements ni même l'apparition de bug ou d'exception non gérer grâce au test.

Ajout d'un Plugin

Les classes de plugin doivent être annotés avec @Plugin ; on y indique ensuite la nature du plugin (attaque, mouvement, graphisme).

Ensuite, en fonction du type de plugin, on doit aussi annoter certaines méthodes avec des annotations correspondant au type de plugin ;

- Pour une attaque, on doit utiliser l'annotation @Attack et annoter quatre méthodes en fournissant en paramètre de l'annotation son rôle ; MAIN pour la méthode permettant d'attaquer un robot, POWER pour obtenir la puisssance de l'attaque, RANGE, pour sa portée et CONSUMPTION pour l'énergie requise ;
- Pour un mouvement, l'annotation à utiliser est @Move et le seul rôle à définir est MAIN ;
- Pour un graphisme, l'annotation à utiliser est @Graphic et le seul rôle à définir est MAIN.

Modularité et dépendances

Implémentation

Game dépend de Shared.

Les Plugins dépendent aussi de Shared.

OptionalPlugins dépend de MandatoryPlugin (dont il étend certaines classes).

Aspects remarquables

Le PluginProcessor est laissé en llibre accès dans Shared, ce qui permet aux plugins d'accéder aux méthodes des autres, peu importe leur nature.

Documentation/ gestion de projet

Implémentation

Pour réaliser le projet dans une équipe composé de 3 personnes, différentes méthodes et outils sont nécessaires afin de pouvoir gérer au mieux les tâches, les échecs, les objectifs ou bien encore les grandes parties à réaliser. De cette manière nous avons pu utiliser différents outils :

- Le premier est Slack : cet outil bien que peu connu peut paraître semblable à Messenger mais celui-ci peut être relié à différentes applications afin de l'utiliser de la manière la plus optimale possible. Cet outil nous a servi de canal de communication unique, ces grâce à lui que nous avons pu gérer au mieux le projet, parler de nos difficultés dans nos tâches respectives, savoir ou en été le projet de manière générale ou encore demander de l'aide si nous en avons besoin à l'un des membres de notre groupe ou une personne tiers au sein de la formation MIAGE. Nous avons également relié cet outil à l'application gitHub afin d'avoir notre dépôt de projet en liaison avec l'application. Cela permet entre autres d'avoir une notification sur le canal de communication lorsque quelqu'un commit ou modifie une partie du code. Cette liaison est fort utile dans ce type de projet car elle permet de notifier tous les membres du groupe de l'avancement du projet et des modifications qui ont été faîte, de cette manière lorsqu'un commit est notifier les membres du groupe savent qu'ils doivent récupérer la dernière version du code avant de commit la leur.
- Le second outil comme je l'ai précisé n'est nul autre que GitHub. Cet outil comme pour beaucoup de projet nous permet de partager notre code à l'ensemble des membres du projet. Ces de cette manière que nous pouvons réaliser un avancement coordonné entre les membres ainsi que de l'avancement de telles ou telles tâches. Cet outil permet de commit via Git notre projet sur un dépôt distant mise à disposition. Pour notre équipe GitHub a été relié à SourceTree afin d'avoir une meilleure visualisation de l'ensemble du projet mais également de pouvoir gérer de façon simple et intuitive l'envoie de code ou encore la récupération de celui-ci. Cet outil nous permet d'avoir une interface pour chaque action nécessaire au projet comme :
  - L'envoie de code via un simple bouton
  - La mise à jour du code afin de récupérer ce qui a été réalisé
  - Le choix d'envoyer telles ou telles partie du code grâce à une interface indiquant ce que nous avons modifier sur notre code et les classes impactées.
  - La possibilité de supprimer les modifications que nous avons faîtes si celles-ci ne nous conviennent plus.
- Le troisième outil utilisé a été Trello, pour ceux connaissant le logiciel JIRA permettant de gérer de manière simple et efficace un projet. Trello fonctionne de manière plus intuitive et proposant globalement les mêmes fonctionnalités. Cet outil nous a donc servi à gérer l'ensemble de notre projet avec :
  - Les différentes parties du projet à réaliser grâce à des étiquettes
  - L'attribution à chacun de ce qu'il doit faire
  - Le déplacement des étiquettes sur un dashBoard afin de savoir ce qui est en cours de réalisation, ce qui a été fait mais doit être validé et enfin ce qui a été réaliser.
  - La notification par mail lors de la création d'une étiquette, la suppression, le déplacement à l'ensemble des membres du groupes notifier sur Trello.

Enfin nous allons parler de la méthode de gestion de projet utilisé. Ici nous pouvons comparer notre projet à une multitude de mini-Sprint réalisé tous le long du projet différentes parties ont ainsi pu être découper et réalisé en plusieurs petites tâches. Parmi ces parties je peux citer par exemple la première qui permettait de gérer la base du jeu. Dans cette partie différentes tâches devaient être réalisé pour pouvoir finir le sprint et avoir une première fonctionnalité pour lancé ce jeu. A la fin de celle-ci nous pouvions donc lancer le jeu et fournir un livrable afin de voir l'avancement de la partie. Pour cette première base nous avions tous simplement à l'issue de celle-ci deux robots représentés par des carrés se déplaçant et s'attaquant jusqu'à la mort. Nous pouvons donc dire que nous avons réalisé ce projet grâce à une méthode agile.

Aspects remarquables

Dans cette partie différents aspects sont à notifier. Notamment la partie de liaison avec le Slack qui a permet une grande agilité dans le traitement et les notifications des membres du groupe, l'utilisation de Slack comme moyen unique de communication a permis une meilleure coordination de chacun. La partie gestion de projet avec Trello a également était très utile car elle permettait de notifier les membres du groupe du temps qu'ils leurs restés pour réaliser la tâche attribuer ainsi que des différentes parties à faire dans le projet.