Wilhelm Daniel
Hamery Simon

L3 informatique

Université de Strasbourg
UFR de Mathématiques et Informatique

Rapport de projet

Systèmes Distribués

Programmation d'un jeu stratégique d'agents distribués en Java RMI

Strasbourg
2017

Table des matières dynamique

---

1 - Introduction

Ce programme réalisé pour l'UE systèmes distribués est une simulation d'un jeu d'agents en tour par tour ou bien en vitesse réelle.

Un certains nombres de paramètres peuvent être définis dans une interface graphique, et une fois la partie lancée, des terminaux indiquent l'état des joueurs en temps réel. A la fin de la partie, trois graphiques sont générés pour chaque ressource (or, bois et argent) et indiquent l'évolution du nombre de ressource de chaque joueur en fonction du temps ou du nombre de tour selon le mode de jeu.

Un contrôleur gère X joueurs sur X machines et Y producteurs sur Y machines avec leurs ports associés avec $X, Y \in [1, 20]$.

2 - Lancement du jeu

Pré-Requis : terminal xterm, java dernière version et gnuplot installés.

Le jeu doit préalablement être compilé avec la commande suivante :

```
> javac -cp ./JavaPlot-0.5.0/dist/JavaPlot.jar *.java
```

Enfin, pour lancer une partie il faut lancer la classe Test avec la commande :

```
> java -cp ./JavaPlot-0.5.0/dist/JavaPlot.jar:. Test
```

3 - Règles

Conditions de victoire

Les conditions de victoires peuvent être paramétrées selon deux possibilités.
La première est l'atteinte d'un nombre de ressources fini par le joueur pour qu'il gagne. Dans ce mode, il y également deux possibilité. La première est que lorsqu'un joueur gagne, tout le monde s'arrête. La deuxième est que les joueurs s'arrêtent progressivement. La deuxième fonctionnalité marche sur le terminal mais nous n'avons pas eu le temps de l'implémenter sur l'interface.
Le deuxième mode de jeu est l'atteinte d'un maximum de ressource dans un temps imparti. Pour cela un timer est lancé lorsque la partie est lancée. Lorsque ce timer expire, tous les joueurs et producteurs s'arrêtent. Celui qui possède le plus grand montant de ressource a gagné.

Comportements

Il y a, en plus de l'humain, 5 comportements disponibles. Nous avons supposé que le principal but/intérêt de ce jeu était soit de pouvoir tester des stratégies des joueurs, soit de pouvoir y jouer. Afin de pouvoir définir des stratégies un tant soit peu intelligente, nos joueurs, lorsqu'ils en ont besoin, demandent les ressources de chaque producteur et/ou joueur. Cela permet également de donner un avantage aux IAs face aux humains qui eux peuvent élaborer de meilleurs stratégies. Les comportements implémentés sont détaillés dans les parties qui suivent.

Individualiste

Ce comportement ne prend que des ressources aux producteurs. Il fait un choix avant de prendre des ressources.

D'abord, il demande à chaque producteur ce qu'il produit ainsi que le montant de ressources pour chaque type de ressource qu'il produit. Après avoir obtenu ces informations de chaque producteur, il les rassemble dans un tableau. Il y met, pour chaque type de ressource, la somme de la quantité de cette ressource de chaque producteur. Il trie ensuite ce tableau en faisant un calcul pour affecter une importance à chaque ressource.

L'importance d'une ressource est calculée ainsi : importance = QuantitéRessource + nbProducteurs * multiplicateur. QuantitéRessource représente la quantité totale de cette ressource (addition de ce que possèdent tous les producteurs pour cette ressource). nbProducteurs représente le nombre de producteurs qui produisent cette ressource. Multiplicateur est un facteur que nous avons choisi afin de donner plus d'importance au nombre de producteurs qui produisent une ressource, car nbProducteur est en soit une information plus importante que le montant de ressources vu qu'il donne une indication sur la production de cette dernière.Plus ce nombre est grand, plus il y a de producteurs qui produisent cette ressource et plus il y aura de l'approvisionnement.

De plus, puisqu'un joueur ne peut que demander des ressources à un producteur à la fois, il y a moins de risques d'avoir des producteurs vidés de leurs ressources lorsqu'il y en a plusieurs et un joueur ne peut pas prendre des ressources simultanément chez plusieurs producteurs. Nous avons mit le multiplicateur de l'importance à 10. Plus Cette importance calculée est petite, plus la ressource est importante car les joueurs recherchent en priorité les ressources rares et peu produites. Une fois que l'importance est calculée, le joueur s'en sert pour établir une liste de priorité de ressources, ordonnées par valeur d'importance croissante. Lorsque cette liste est calculée, le joueur choisi aléatoirement soit la première, soit la deuxième ressource (s'il lui manque au moins une deuxième ressource) du tableau. Nous avons fait le choix de ne pas toujours prendre la ressource la plus rare car cela conduit à des individualistes qui ont tous le même comportement.

Une fois la ressource choisie, le joueur doit encore décider chez quel producteur il veut la prendre. Pour cela, il ré-utilise les informations cherchées précédemment afin de voir quel est le producteur qui a le plus de ressources du type de la ressource demandée. Enfin, il demande la ressource ce producteur. Cela permet de diminuer le montant de ressources des producteurs qui en ont beaucoup car ces derniers produisent plus que ceux n'ayants rien (ils produisent selon une fonction linéaire). Il passe au prochain tour...

Coopératif

Le comportement coopératif est altruiste. Avant de prendre une ressource, il regarde donc si le montant total des ressources disponibles est suffisant pour tout le monde.

Pour cela il demande toutes les informations sur les ressources aux producteurs. Il demande également toutes les ressources dont disposent les joueurs. Grâce à ces informations, le coopératif fait un calcul pour savoir s'il y en a assez pour tous.

D'abord il fait la somme des ressources des producteurs par type de ressource. Ensuite il calcule les besoins par ressources qui sont :

$$multiplicateur \times Ressources\ Nécessaires\ \ pour\ la\ victoire \times\ \ nbJoueurs - Total\ des\ ressources\ des\ joueurs \times multiplicateur$$

Le coopératif utilise un multiplicateur pour s'assurer qu'il y ai plus de ressources que nécessaire. Cela permet d'augmenter les chances que tous les joueurs puissent chercher rapidement les ressources nécessaires. En effet, il se peut qu'il y ai plusieurs petits producteurs et quelques grands. Certains joueurs pourraient alors prendre les ressources dont ils ont besoin en 1 coup alors que d'autres devraient passer plusieurs producteurs.
Le coopératif cherche la première ressource qui remplit cette condition. Il la cherche chez le producteur qui a la plus grande quantité de cette ressource.

Voleur

Le voleur ne vole que les joueurs et ne s'occupe pas des producteurs. Ne faire jouer que des voleurs n'a donc aucun intérêt !
Il regarde d'abord ce que chaque joueur possède. Ensuite, il applique la même méthode que l'individualiste pour choisir quelle ressource voler. Il la vole chez le joueur qui en a le plus afin de " l'éloigner de la victoire".

Attentionnel

Ce comportement tire à chacune de sa boucle de jeu un nombre entre 0 et 1. Il a alors une chance sur deux de se mettre en état d'observation ou bien en état individualiste pour ce tour de boucle. Lorsqu'il est en état d'observation il remarque lorsqu'un joueur essaie de le voler et le pénalise (2 tours d'indisponibilité).

Brute

Là aussi, une chance sur deux d'être en état individualiste.
Sinon; il a 1 chance sur 10 de faire un echec critique (se blesser pour 3 tours), 1 chance sur 10 de faire un succès critique (blesser l'autre pour 3 tours), 1 chance sur 10 de ne rien faire, 6 chance sur 10 de blesser 2 tours l'autre et 1 chances 10 d'avoir 2 tours de pénalité.
Selon le mode de jeu (temps réel ou tour/tour), la pénalité est un "saut de boucle" ou un sleep de ms en temps réel.

Humain

Pour l'humain, nous avons implémenté une fenêtre permettant le contrôle de trois actions: Voler un montant de ressources à un joueur, prendre un montant de ressources à un producteur, ou bien ne rien faire et observer. Le joueur lance l'action à chaque tour avec le bouton jouer.

4 - Justifications des choix d'implémentation

Pourquoi avoir fait le choix de plusieurs ressources disponibles par producteur ?

C'est une simulation de fait réels, par exemple dans la vraie vie, un paysan peut fournir aussi bien du blé que de l'orge. Ici dans notre cas, nous avons choisis de l'or, de l'argent et du bois. A priori un mineur peut tout aussi bien aller miner de l'argent que de l'or et ensuite aller bûcher quelques arbres.

Ajouter d'autres ressources ?

Dans notre programme, il est possible d'ajouter d'autres types de ressources dans l'enum TYPE, car nous avons une liste de ressources qui gère la gestion des ressources. Par question de simplicité et surtout du fait qu'il y a déjà bien assez de facteurs modifiables (nombre de joueurs, comportement de chaque joueurs, nombre de producteur, mode de jeu, condition de victoire), nous avons décidé de ne pas nous impliquer davantage dans ces derniers qui suffisent à générer une infinité de simulations.

5 - Interface

Pour éviter de surcharger notre interface de lancement, les ports ont été fixés par défaut.

- Le port du Contrôleur dans l'interface a été fixé à 5000.
- les ports des joueurs ont été fixé à 5001+id_joueur.
- Les ports des producteurs ont été fixé à 5021+id_producteur (20 joueurs max, mais il est possible d'en rajouter avec le terminal.
- Il y a des valeurs minimales pour le nombre de ressources pour la victoire (5 ressources minimum) et le temps de jeu max en mode temps réel (20 secondes minimum).
- Le bouton Reset valeurs efface la liste des joueurs ajoutés

- Le bouton Lancer le jeu lance la partie avec tous les paramètres inscrits dans les champs à ce même moment.

- Le bouton Afficher les graphiques afficher les valeurs actuelles dans le fichier actionLog.dat, il afficher les 3 graphiques de l'évolution des nombre de ressources de chaque joueurs de la dernière partie effectuée.

6 - Exemple d'exécution

Voici un exemple d'exécution à 5 joueurs et 2 producteurs en tour par tour :

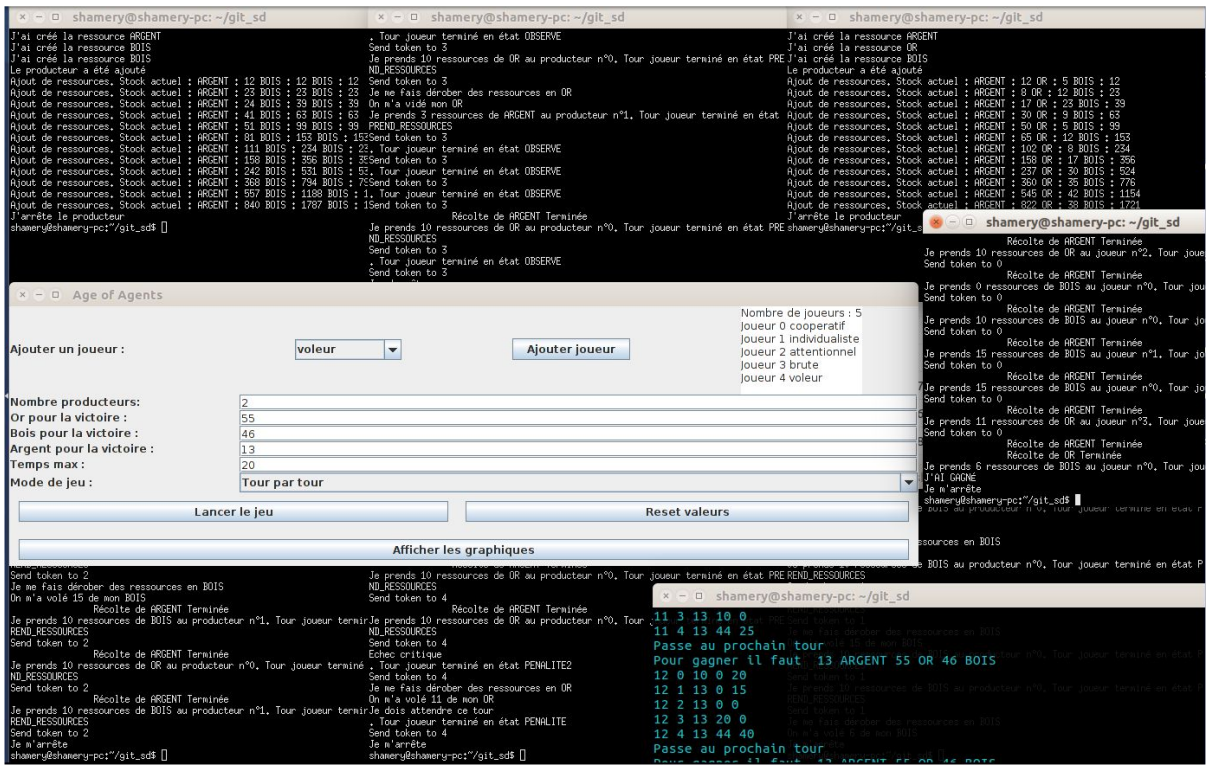


fig 1.1 exemple d'exécution : fin de partie

On a 5 terminaux xterm de joueurs, 2 terminaux xterm, un terminal au choix (ici celui avec l'écriture bleue) pour lancer le jeu et y voir apparaître les indication du controller (coordinateur) et une fenêtre graphique de lancement de partie.

Ici le voleur (terminal du milieu à droite) a gagné ! Le controlleur attends un nouvelle partie, on peut en relancer une après avoir appuyé sur reset et fermé tous les terminaux xterm (on peut utiliser la commande "pkill xterm" pour tout fermer d'un coup).

On peut ensuite afficher les graphes avec le bouton en bas et des graphiques avec Javaplot sont automatiquement générés :

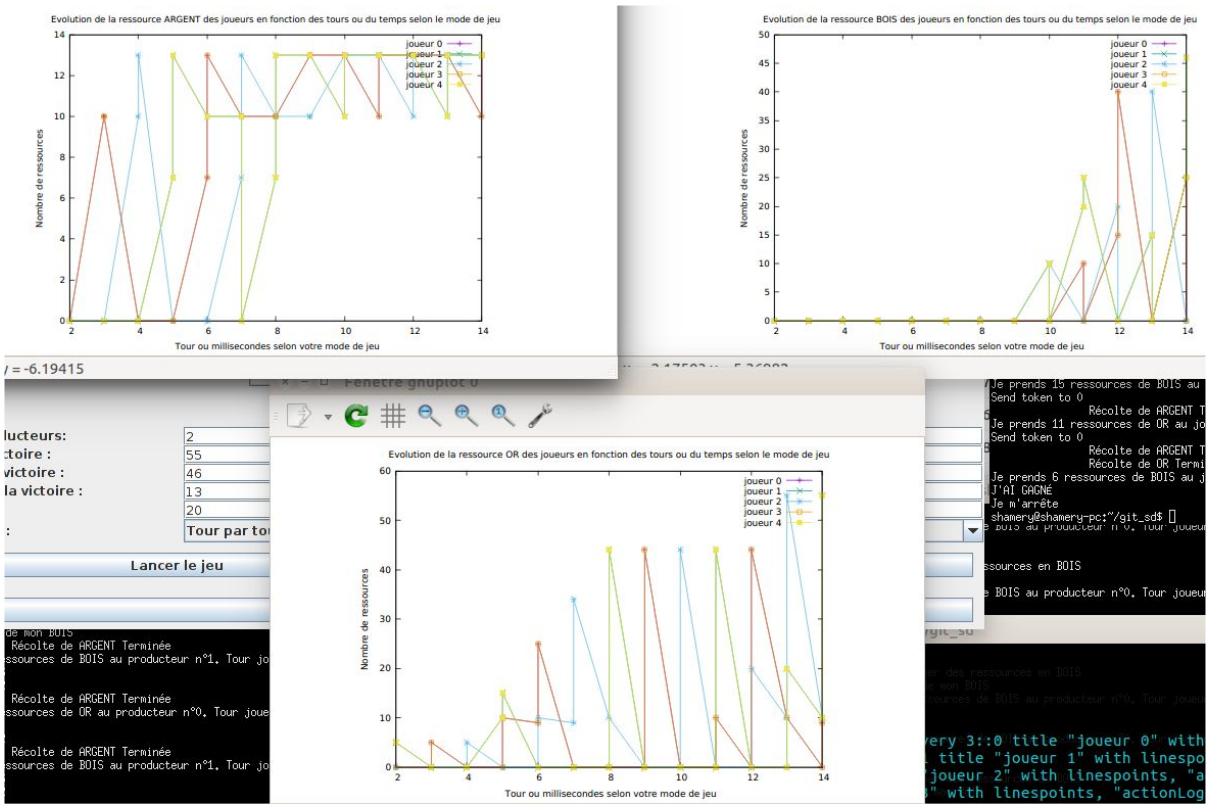


fig 1.2 exemple d'exécution : graphiques générés

En zoomant on voit bien l'évolution :

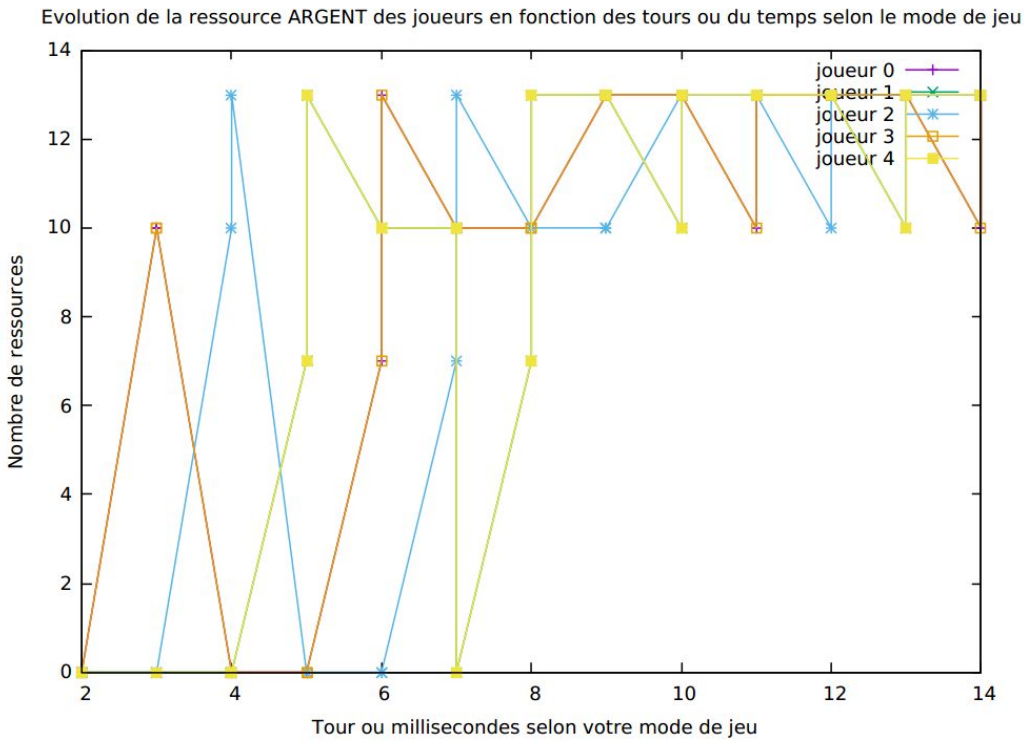
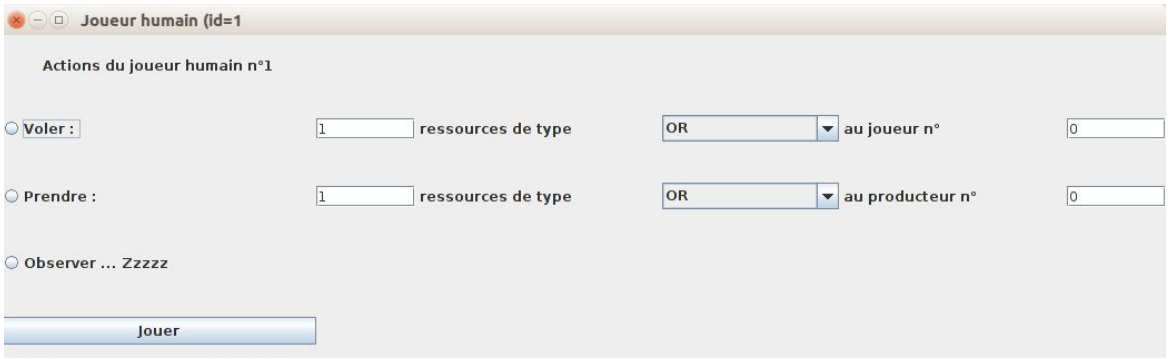


fig 1.3 exemple d'exécution : graphique de l'argent

Si un ou plusieurs humains sont choisis en joueur lors du lancement avec la première fenêtre, une nouvelle fenêtre pour chaque humain est générée comme çi-dessous



2.1 exemple d'exécution : fenêtre joueur humain

Le joueur humain peut donc à chaque tour choisir entre 3 actions : Voler un joueur, Prendre des ressources d'un producteur et Observer (ne rien faire en vérité, petit flemmard..). En tour par tour, le jeu se bloque grâce aux jetons pour attendre que l'humain joue.

7 - Divers

Les résultats des parties sont stockés dans 1 fichier "actionLog.data" qui est écrasé à chaque fin de partie.

Ce fichier est écrit de la manière suivante :

- Première colonne : numéro du tour ou temps, selon le mode de jeu.
- Deuxième colonne : numéro du joueur
- Troisième -> cinquième colonnes : ressources 1, 2 et 3 (or argent et bois)

Le code des graphes généré est en dur dans le fichier Graphs.java.

8 - Corrections appliquées suite à la présentation

Après la présentation nous avons encore apporté des modifications et ajouté des fonctionnalités qui n'étaient pas / mal implémentées.

Nous avons modifié la pénalisation des joueurs lorsqu'ils se font avoir en train de voler dans le mode en temps réel. Ils ont maintenant une pénalité correspondant à une attente de 50ms.
De plus, nos producteurs produisent maintenant uniquement à la fin de chaque tour dans le mode tour / tour et non plus toutes les x ms.

Enfin, les graphiques sont correctement générés lors de l'appui sur le bouton pour les afficher à partir de l'interface de lancement.

9 - Conclusion

Simon : Ce projet était intéressant dans le sens où j'ai appris concrètement ce qu'était que la programmation distribuée avec ses avantages et ses inconvénients.

Comme je suis généralement lent dans l'apprentissage de nouvelles notions, j'ai eu un peu de difficultés au départ, et Daniel a commencé la base du jeu, cela m'a bien aidé et après avoir passé beaucoup de temps à comprendre le fonctionnement de RMI que je n'avais pas pu bien comprendre en TP, je m'y suis adapté et j'ai pu réellement faire ce que je voulais. A ce moment là c'était très agréable, et ce projet est devenu tout de suite plus intéressant.

La partie majoritaire du travail a été de rechercher des solutions adéquates à tous les problèmes de fonctionnement/lancement. J'ai notamment essayé des frameworks comme jcommander pour le parsing du fichier de configuration et d'un script shell de lancement de terminaux xterm qui au final ne nous on plus été utiles par la suite car j'ai implémenté des interfaces graphiques. J'ai aussi participé à l'initiation de certains comportements et de fonctions rmi que Daniel a généralement optimisé. Enfin, j'ai réalisé la génération des graphiques de fin avec Javaplot et corrigé des bugs grâce à l'interface avec laquelle nous avons pu directement vérifier les fonctionnalités du jeu.

La coopération du projet avec Daniel était agréable. C'est un jeu qui pourrait être encore plus optimisé si nous avions plus de temps, cela nous a donné des idées pour la suite.

Daniel : J'ai bien aimé ce projet, organisé sous forme de jeu et dans lequel nous avions beaucoup de libertés. C'était parfois difficile de faire des choix de fonctionnement/ implémentation et certains se sont révélés inutilement complexes/limitants (comme l'utilisation des TYPE). Cela a été mon premier gros projet en Java. J'ai eu un peu de mal au tout début, mais une fois que RMI a fonctionné, la partie communication n'a plus posée de problèmes. Ce qui m'a aussi semblé difficile par moment, c'était que vu la taille du code, ça a été difficile de s'y retrouver des fois / savoir où mettre le code. Mais une fois passé les premières difficultés nous avons avancé plutôt rapidement.
Ma part du travail a été la gestion des communications avec RMI. J'ai d'abord travaillé sur le squelette (coordinateur, joueur et producteur). Simon et moi avons ensuite implémentés les différentes fonctionnalités. Je me suis plus concentré sur le code et sur les différentes étapes du jeu (début, tour/tour, temps réel, fin de jeu …) et Simon sur les difficultés

rencontrées comme le parsing des données, gnuplot, les interfaces. Nous nous sommes bien répartis les tâches et communiquions sur le projet très régulièrement.

Nous avons essayé de respecter la séparation en différentes classes ayants différentes fonctions. Cependant, notre nombre de classes a vite explosé.
La liberté de choix nous a plût. Elle nous a permis d'aller dans le sens qui nous plaisait le plus et nous étions donc plus motivés.
Nous avons essayé de gérer les transmissions RMI d'une façon "optimisée", en essayant d'uniquement transmettre ce qui était nécessaire pour le fonctionnement, et de le faire le moins possible.
Cependant il est vrai que nous nous sommes plus concentré sur le fonctionnement du jeu que sur la minimalisation des messages. Nous avons par exemple préféré transmettre les informations nécessaires à chaque tour, afin de pouvoir faire des comportements plus intelligents.