|  |  |
| --- | --- |
|  | **Projet LO23 ChessP2P** |
|  | 19/10/2012    Auteur(s) :   * Auteur 1 * Auteur 2 * Auteur 3 * Auteur 4 |

|  |
| --- |
| **Dossier de conception**  **Réalisation d’un jeu d’échec en réseau décentralisé** |
|  |

Table des matières

[1 Le projet : Chess P2P 3](#_Toc338165057)

[1.1 Choix de conception 3](#_Toc338165058)

[1.2 Cas d’utilisation 4](#_Toc338165059)

[1.3 Diagramme de classes 4](#_Toc338165060)

[2 Les sous-projets 5](#_Toc338165061)

[2.1 Communication et traitement 5](#_Toc338165062)

[2.1.1 Analyse préliminaire 5](#_Toc338165063)

[2.1.2 Choix de conception 6](#_Toc338165064)

[2.1.3 Diagramme de cas d’utilisation 9](#_Toc338165065)

[2.1.4 Diagrammes de séquence 10](#_Toc338165066)

[2.1.5 Diagramme de classes 14](#_Toc338165067)

[2.2 Gestion des données 15](#_Toc338165068)

[2.2.1 Documents d'étude 15](#_Toc338165069)

[2.2.2 Choix de conception 15](#_Toc338165070)

[2.2.3 Cas d’utilisation 15](#_Toc338165071)

[2.2.4 Diagramme de séquence 15](#_Toc338165072)

[2.2.5 Diagramme de classes et interfaces 15](#_Toc338165073)

[2.3 IHM connexion 15](#_Toc338165074)

[2.3.1 Documents d'étude 15](#_Toc338165075)

[2.3.2 Choix de conception 15](#_Toc338165076)

[2.3.3 Cas d’utilisation 15](#_Toc338165077)

[2.3.4 Diagramme de séquence 15](#_Toc338165078)

[2.3.5 Diagramme de classes et interfaces 15](#_Toc338165079)

[2.3.6 Maquettage 15](#_Toc338165080)

[2.4 IHM grille 15](#_Toc338165081)

[2.4.1 Documents d'étude 15](#_Toc338165082)

[2.4.2 Choix de conception 15](#_Toc338165083)

[2.4.3 Cas d’utilisation 15](#_Toc338165084)

[2.4.4 Diagramme de séquence 15](#_Toc338165085)

[2.4.5 Diagramme de classes et interfaces 15](#_Toc338165086)

[2.4.6 Maquettage 15](#_Toc338165087)

# Le projet : Chess P2P

Dans le cadre de l’unité de valeur LO23 à l’Université de Technologie de Compiègne, concernant la conduite de projets informatiques, il nous a été demandé de réaliser un jeu d’échec en réseau. Le projet a pour objet toutes les étapes depuis la conception jusqu’à l’intégration, en passant par la réalisation.

La particularité de ce projet est son nombre important de contraintes. Des contraintes de temps, d’abord, puisqu’il doit être finalisé pour la fin du semestre d’automne 2012. La difficulté, ensuite, vient de l’organisation des acteurs : vingt-trois étudiants doivent travailler en accord pour mener à bien le projet atour d’un découpage structurel imposé en plusieurs sous-projets :

* Communication et traitement des données
* Gestion des données
* Interface humain-machine pour la connexion au jeu
* Interface humain-machine dans le jeu (grille de jeu)

Enfin, il s’agit de respecter les contraintes techniques imposées par le cahier des charges. Le logiciel utilise un réseau décentralisé, ce qui signifie que chaque joueur est propriétaire de ses données. Il a été demandé, de plus, d’intégrer les fonctions suivantes :

* Création du profil
* Connexion et découverte de joueurs connectés
* Lancement d’une partie
* Déroulement d’une partie
* Modification et portabilité du profil
* Sauvegarde d’une partie
* Visionnage d’une partie enregistrée

## Choix de conception

## Cas d’utilisation

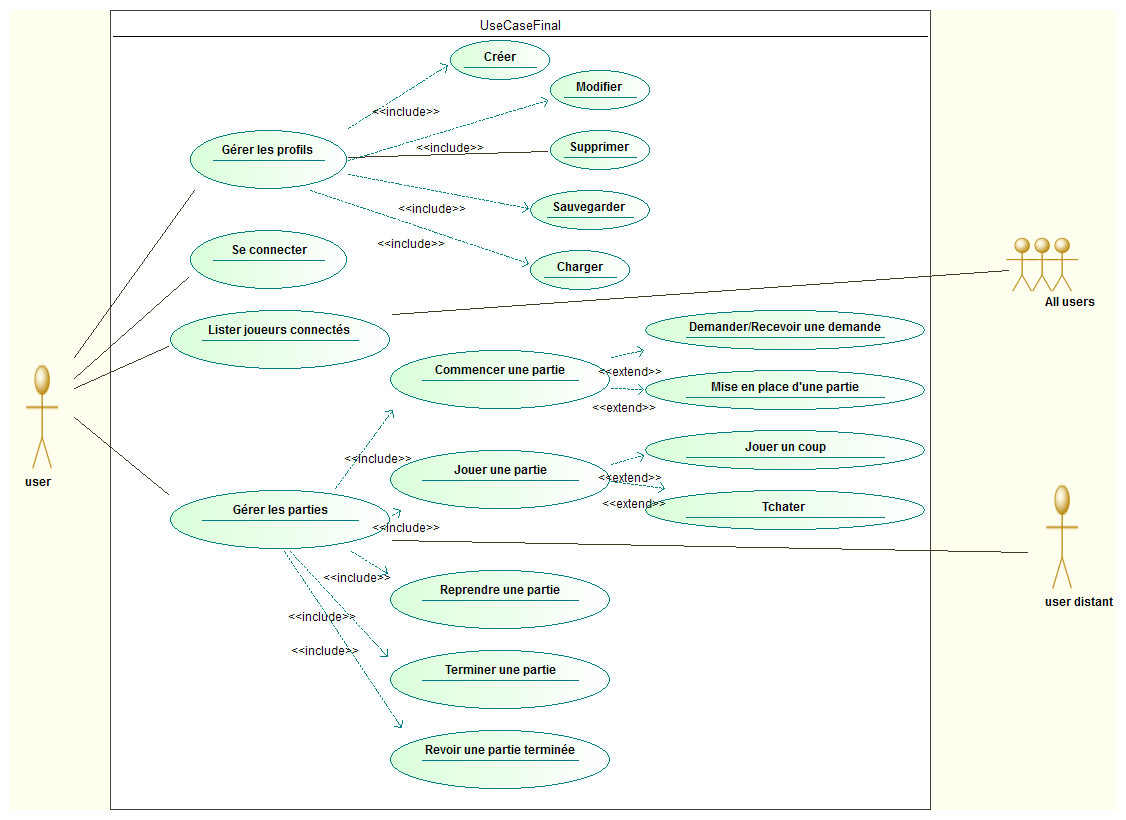


Figure 1 : diagramme de cas d'utilisation global du projet

L’ensemble des équipes a défini un diagramme de cas d’utilisation global. Ce diagramme a été défini sur un plan de haut niveau : pour plus de détails, on ira voir les diagrammes spécifiques à chaque équipe. L’intérêt de ce travail a été de définir plus précisément le rôle de chaque équipe tout en ayant une vision globale sur le projet général.

## Diagramme de classes

# Les sous-projets

## Communication et traitement

Ce module a la tâche de gérer l’acheminement des données dans le réseau décentralisé. Le but est donc de définir la logique d’envoi et de réception des messages dans les différentes situations imposées par les fonctionnalités offertes par le logiciel.

### Analyse préliminaire

Dans les prémisses du projet, notre équipe a réfléchi sur l’impact de la logique de communication au cours des différentes situations d’utilisation du jeu. Nous avons ainsi déterminé qu’il y avait deux cas à traiter : la communication joueur à joueur au cours d’une partie et la découverte des joueurs.

#### La communication avec l'adversaire

Il s'agit d'une communication bidirectionnelle entre deux joueurs. Dans cette situation, on doit pouvoir être informé que le message réseau a bien été reçu par le destinataire. Par exemple, si on envoie un message indiquant qu'un joueur a déplacé une pièce, il est primordial qu'il sache que l'autre joueur a reçu son message et que maintenant c'est à lui de déplacer une pièce. Il faut donc utiliser un protocole en mode connecté.

#### La découverte des joueurs

Pour la découverte des joueurs sur le réseau, il faut pouvoir envoyer un message à tous les joueurs disponibles. Deux solutions sont envisageables : soit en mode « broadcast », qui consiste à envoyer un message à toutes les machines d'un réseau, soit en mode « multicast », qui permet d’envoyer un message à un groupe de machines. Ces formes particulières de communication imposent d'utiliser un protocole en mode non -connecté car la communication est unidirectionnelle : le message part d'une machine pour arriver à un ensemble de machines. Dans le cadre de notre projet, une machine A enverrait d’abord un message à un ensemble de machines B pour demander de se présenter (profil + disponibilité) puis chaque machine B répondrait à la machine A en envoyant les informations nécessaires.

### Choix de conception

Durant la conception, l'équipe a été amenée à effectuer de nombreux choix importants parfois décisifs pour le bon déroulement du projet. Voici donc, en détails, les principales questions que s’est posée l'équipe, accompagnées des solutions possibles, d'arguments puis de la décision finale.

#### Objet brut ou objet intermédiaire ?

L'équipe s'est confrontée à un choix de conception dès la première séance. En effet, concernant l'envoi des données, la question était de savoir si les données envoyées à travers le réseau allaient être simplement celles offertes par le module *Gestion de données* à travers leur interface ou si notre équipe décidait de convertir cette structure en une autre plus adaptée à notre module et au transfert à travers le réseau.

Dans le premier cas, on récupère l'objet du module *Gestion de données* à travers l'interface liant nos deux modules et on décide de l'envoyer directement à travers le réseau, puis de le récupérer (au sein d'une autre application cliente) et de le rendre au module *Gestion de données.*

Figure 2 : cas d'un objet brut

*Gestion de données*

*Gestion de données*

*Interface*

*Interface*

*Réseau*

*Communication et traitement*

*Structure X*

*Structure X*

*Structure X*

*Structure X*

Schéma illustrant le premier cas : objet brut

Dans le deuxième cas, on récupère l'objet du module *Gestion de données* à travers l'interface et on décide de créer notre propre structure, de récupérer les paramètres indispensables et de les injecter dans notre objet. Ensuite, on peut alors envoyer l'objet nouvellement créé à travers le réseau, le récupérer, le reconvertir en un objet manipulé par le module *Gestion de données* avant de leur rendre à travers l'interface.

Figure 3 : cas d'un objet propre au module

*Gestion de données*

*Gestion de données*

*Interface*

*Interface*

*Réseau*

*Communication et traitement*

*Structure X*

*Structure X*

Schéma illustrant le deuxième cas : objet propre au module

*Structure Y*

*Structure Y*

*Conversion de X en Y*

*Conversion de Y en X*

*Structure X*

*Structure X*

L'avantage évident de la première solution est sa clarté et sa simplicité. Néanmoins, après réflexion, on se rend compte que cette solution lie de façon dérangeante notre équipe à l'avancement du module Gestion de données. En effet, dans ce premier cas, on utilise leur objet ; on se doit donc d'attendre que celui-ci soit finalisé pour pouvoir développer le module de communication à travers le réseau.

La deuxième solution permet, quant à elle, de gagner en indépendance par rapport à l'autre équipe (responsable de la gestion de données). On peut alors concevoir toute la partie envoi et réception de données à travers le réseau en se basant sur la donnée proprement créée pour transiter à travers le réseau. Une fois que le module Gestion de données nous fait parvenir la structure qu'elle veut faire transiter, il suffit alors à notre équipe de concevoir les 2 convertisseurs (un dans chaque sens) et la liaison sera opérationnelle. La seconde approche est, pour résumer, plus modulaire et apporte donc de la souplesse même si elle complexifie sensiblement le processus en passant par une structure intermédiaire pour le transfert à travers le réseau. Un autre avantage à la deuxième solution est le fait que cette structure intermédiaire ne contiendra que les informations indispensables contrairement au premier cas où l'on envoie l'objet entier d'origine.

L'équipe a donc choisi la deuxième solution (création d'un objet intermédiaire pour le transit) pour des raisons de souplesse principalement et donc pour conserver une certaine autonomie.

#### Structure globale

Concernant l’architecture globale de notre application, nous nous sommes entretenus à ce sujet, en premier lieu, avec le groupe relatif au module *Gestion de données*. Selon nous, il est plus propre et plus structuré de centraliser les communications inter modules autour de celui relatif à la gestion des données. En effet, dans le cas contraire, les modules IHM auraient de temps à autres court-circuité le module *Gestion de données* afin de s’adresser à nous directement, ce qui obligeait à prévoir une interface faisant à la fois référence à des fonctions appelées par le module *Gestion de données* ainsi que ceux liés à l’IHM. Cela ne nous semblait pas très propre d’un point de vue structurel. A cet effet, après diverses réunions avec l’intégralité des équipes, nous avons adopté le schéma structurel ci-après.

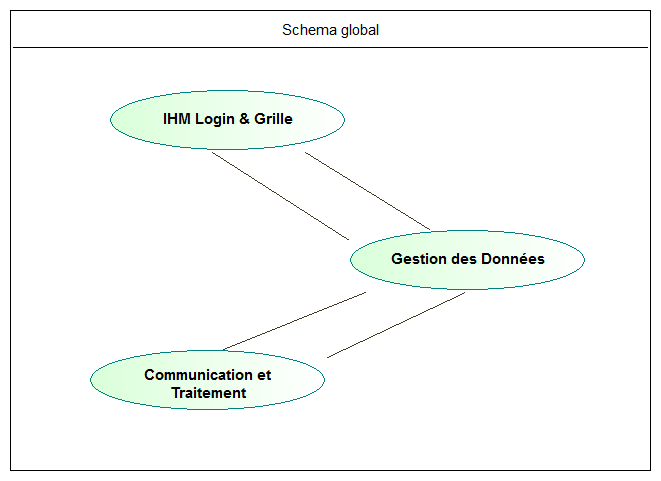


Figure 4 : schéma structurel des équipes

#### Interface

Concernant l’interface que nous allons fournir au module *Gestion de données*, nous avions le choix entre une seule méthode générale possédant de multiples paramètres en fonction du contenu des messages à transmettre ou l’utilisation de plusieurs méthodes différentes. Le module partenaire ne s’étant pas prononcé, nous avons décidé de fournir diverses méthodes afin de clarifier la structure des communications inter modules. De plus, ce choix nous permettra d’aisément répartir le travail de développement.

### Diagramme de cas d’utilisation

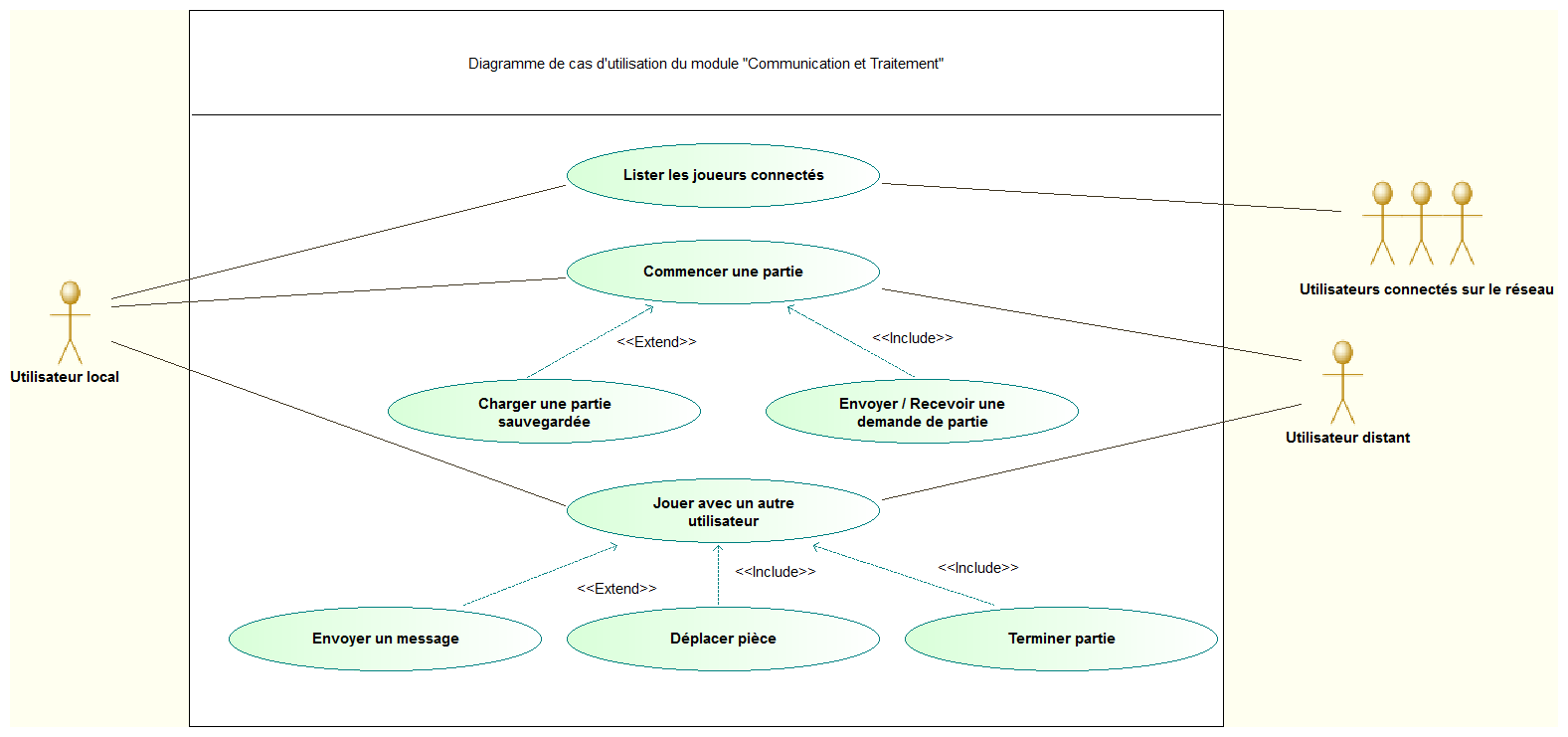


Figure 5 : diagramme de cas d'utilisation du module Communication et Traitement

Suite à la mise en place de ce diagramme, nous avons donc décidé d’étudier les diagrammes de séquence suivants :

* Lister les joueurs connectés
* Envoyer / Recevoir une demande de partie (chargement inclus)
* Envoyer un message
* Envoyer un statut
* Déplacer pièce
* Déconnexion

### Diagrammes de séquence

#### Lister les joueurs connectés

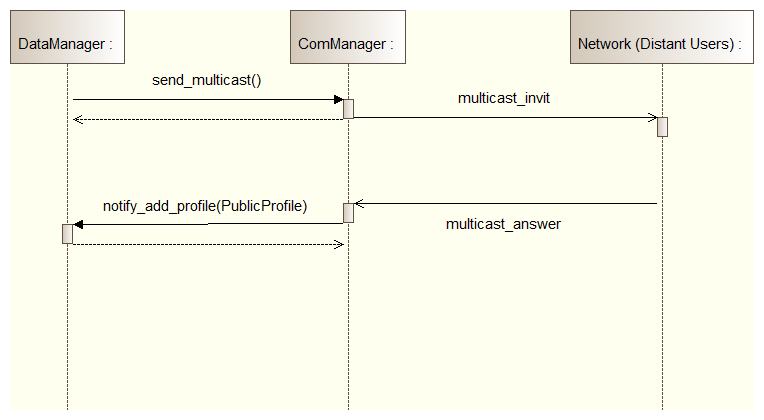


Figure 6 : diagramme de séquence de recherche des joueurs

On envoie un multicast sur le réseau local afin d’envoyer les utilisateurs détectés au « DataManager ».

#### Envoyer / Recevoir une demande de partie (chargement inclus)

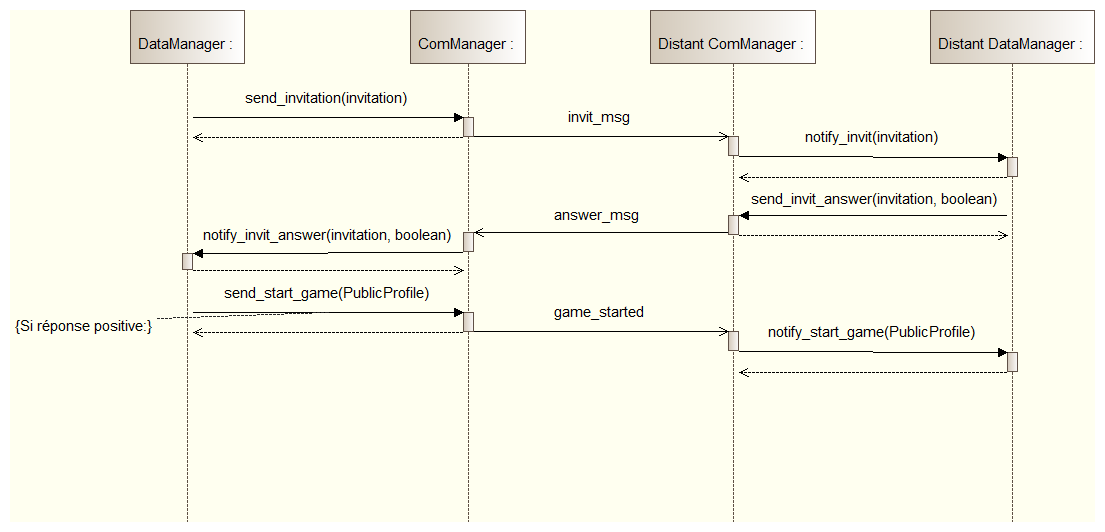


Figure 7 : diagramme de séquence de lancement d’une partie

Ici, on note que les requêtes sont envoyées de manière asynchrone, ce qui implique l’utilisation d’une pile de demandes afin de pouvoir permettre à l’utilisateur de répondre à celle de son choix. Ceci nous permettra également de gérer le cas de demandes de connexion simultanées. Néanmoins, lorsqu’une demande de connexion est acceptée, il faudra bien veiller à ce qu’un « flag » ou autre marqueur bloque l’acceptation de toute autre demande. On note également que ce scénario gère les demandes de nouvelles parties ainsi que les demandes de chargement de parties. A cet effet, la structure invit\_msg contiendra soit des éléments relatifs à une nouvelle partie, soit une partie sauvegardée localement qui sera envoyé à l’adversaire concerné.

#### Envoyer un message

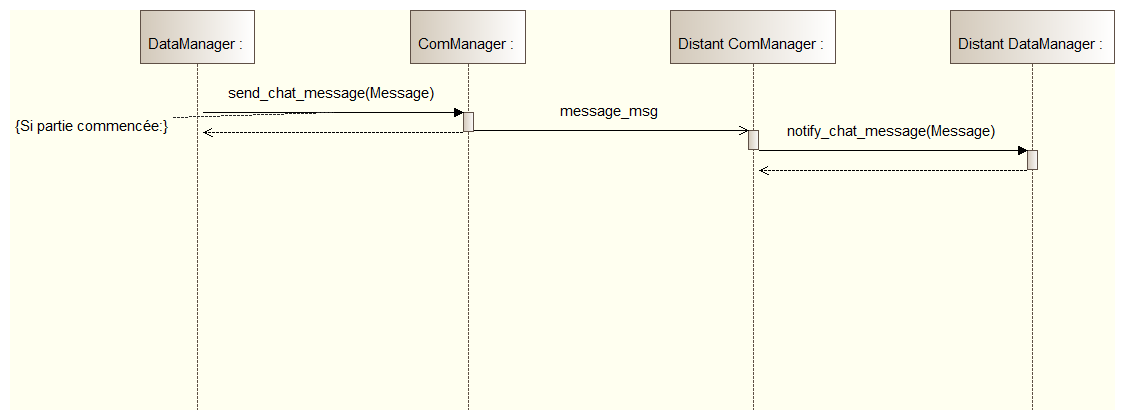


Figure 8 : diagramme de séquence d'envoi d’un message par chat

#### Envoyer un statut

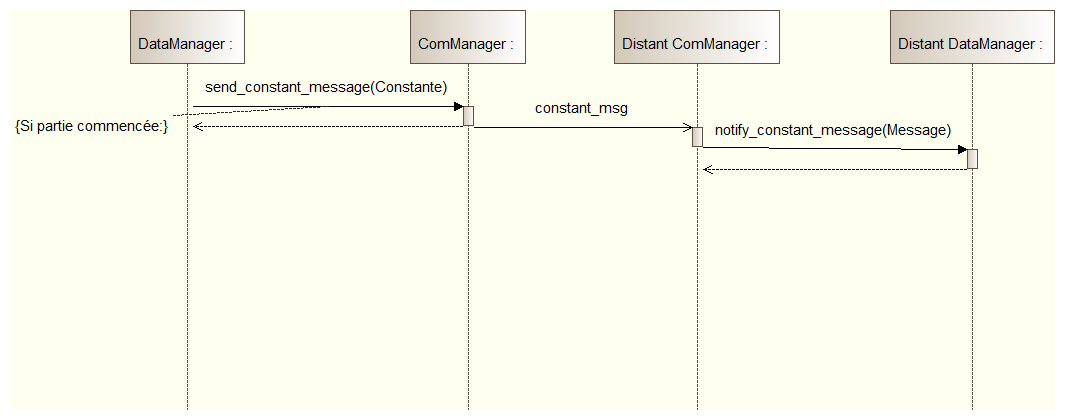


Figure 9 : diagramme de séquence d'envoi d’un message par chat

On envoie ici le statut de la partie. En effet, lorsqu’un joueur demande un match nul, qu’il abandonne ou qu’il quitte la partie, un message de type constant\_msg est envoyé pour prévenir le client adverse.

#### Déplacer pièce

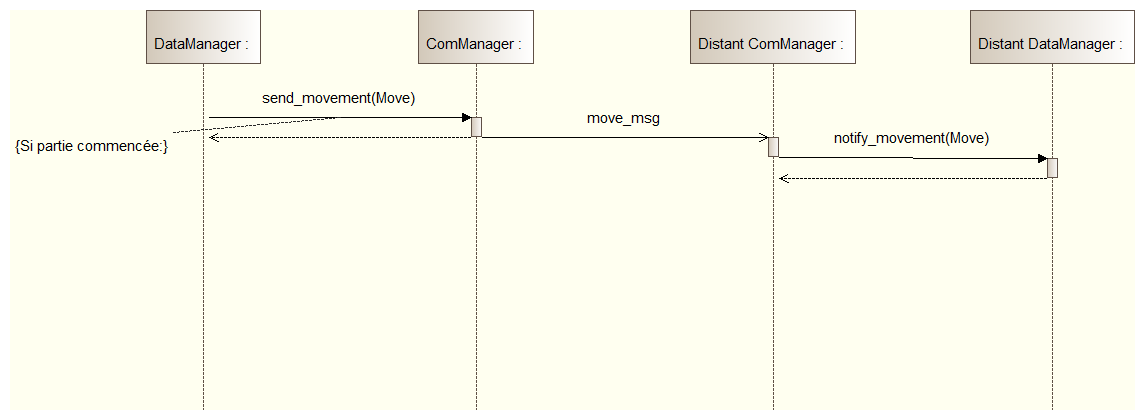


Figure 10 : diagramme de séquence de déplacement d'une pièce

On transmet le déplacement au joueur adverse par le biais de la structure move\_msg.

#### Déconnexion de partie

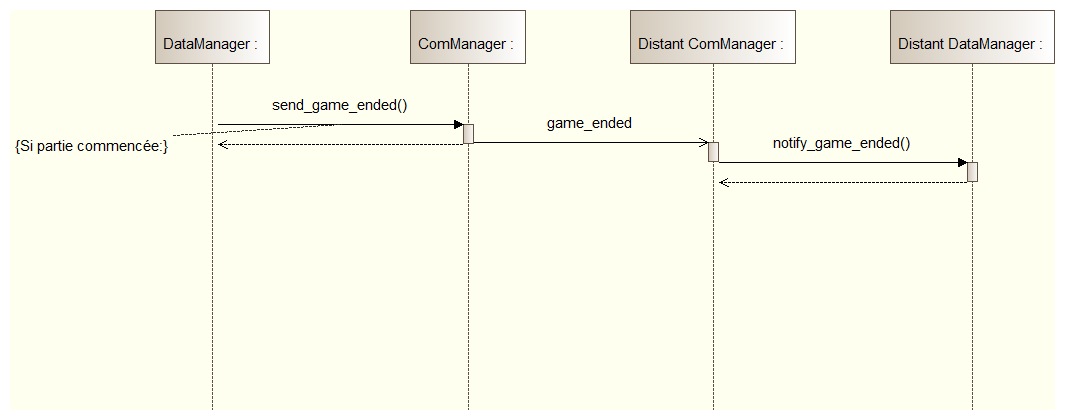


Figure 11 : diagramme de séquence de fin de partie

Une fois la parie finie ou si un des joueurs décide de quitter la partie, un message est envoyé à l’adversaire afin de lui notifier cette décision.

#### Interface

Etant donné que notre module communique uniquement avec le module *Gestion de données*, voici l’interface que nous avons définie avec eux :

|  |  |  |
| --- | --- | --- |
|  | **ISend** | **IReceive** |
| Lister les joueurs connectés | Send\_multicast() | Notify\_add\_profile(PublicProfile) |
| Invitation  Nouvelle partie / Chargement | Send\_invit(Invitation)  Send\_invit\_answer(Invitation, boolean) | Notify\_invit(Invitation)  Notify\_invit\_answer(Invitation, boolean) |
| Début de partie  Connexion | Send\_start\_game(PublicProfile) | Notify\_start\_game(PublicProfile) |
| Chat | Send\_chat\_message(Message) | Notify\_chat\_message(Message) |
| Fin de partie  (Match nul, se render…) | Send\_constant\_message(Constant) | Notify\_constant\_message(Constant) |
| Jouer | Send\_movement(move) | Notify\_movement(move) |
| Deconnexion | Send\_game\_ended() | Notify\_game\_ended() |

Les fonctions situées dans la colonne « ISend » sont celles qui seront implémentées par notre module ; celles situées dans la colonne « IReceive » seront implémentées par le module *Gestion de données*. A cet effet, nous allons développer une interface en JAVA afin que le module *Gestion de données* puisse commencer à développer et vice versa.

### Diagramme de classes

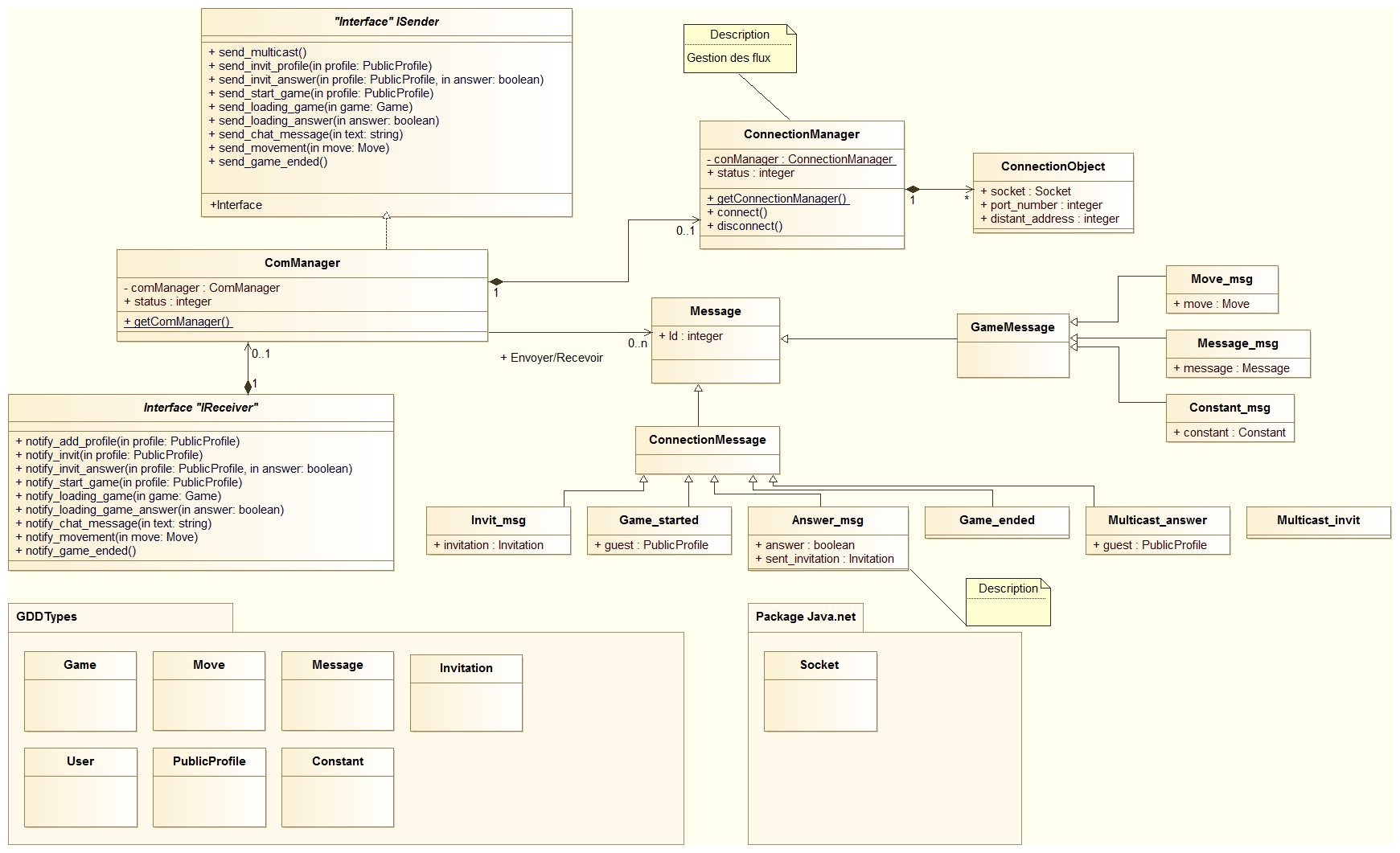


Figure 12: diagramme de classes du module Communication et Traitement

## Gestion des données

### Documents d'étude

### Choix de conception

### Cas d’utilisation

### Diagramme de séquence

### Diagramme de classes et interfaces

## IHM connexion

### Documents d'étude

### Choix de conception

### Cas d’utilisation

### Diagramme de séquence

### Diagramme de classes et interfaces

### Maquettage

## IHM grille

### Documents d'étude

### Choix de conception

### Cas d’utilisation

### Diagramme de séquence

### Diagramme de classes et interfaces

### Maquettage