|  |  |
| --- | --- |
|  | **Projet LO23 ChessP2P** |
|  | Auteur(s) :   * Communication et traitements * Gestion de données * IHM Connexion * IHM Grille |

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

Table des matières

[1 Introduction 3](#_Toc341181006)

[2 Communication et traitement 4](#_Toc341181007)

[2.1 Introduction 4](#_Toc341181008)

[2.2 Choix technologiques 4](#_Toc341181009)

[2.3 Difficultés rencontrées 4](#_Toc341181010)

[2.4 Points particuliers 4](#_Toc341181011)

[3 Gestion des données 5](#_Toc341181012)

[3.1 Introduction 5](#_Toc341181013)

[3.2 Choix technologiques 5](#_Toc341181014)

[3.3 Difficultés rencontrées 5](#_Toc341181015)

[3.4 Points particuliers 5](#_Toc341181016)

[4 IHM connexion 6](#_Toc341181017)

[4.1 Introduction 6](#_Toc341181018)

[4.2 Choix technologiques 6](#_Toc341181019)

[4.3 Difficultés rencontrées 6](#_Toc341181020)

[4.4 Points particuliers 6](#_Toc341181021)

[5 IHM grille 7](#_Toc341181022)

[5.1 Introduction 7](#_Toc341181023)

[5.2 Choix technologiques 7](#_Toc341181024)

[5.3 Difficultés rencontrées 7](#_Toc341181025)

[5.4 Points particuliers 7](#_Toc341181026)

[6 Conclusion générale 8](#_Toc341181027)

# Introduction

# Communication et traitement

## Introduction

Une fois l’établissement du dossier de conception terminée, l’équipe du module « communication et traitement » a entamé la phase de réalisation de son module et des différentes fonctionnalités qui lui sont associées. Nous avons veillé à répartir le travail en fonction des compétences et des disponibilités des membres du groupe. Au cours de cette phase de développement, nous avons dû faire face à des problématiques diverses et donc des choix techniques à faire.

Dans un premier temps, nous allons mettre en lumière les divers choix technologiques que nous avons faits en les expliquant. Ensuite nous nous attacherons aux principales difficultés rencontrées. Et pour finir, nous évoquerons divers points spécifiques.

## Choix technologiques

Nous présenterons dans cette partie les divers choix techniques que nous avons dû prendre tout en argumentant nos choix.

### La découverte des joueurs

Lorsque l’utilisateur lance une partie, il doit être capable de choisir son adversaire parmi la liste des utilisateurs connectés sur le même réseau que lui. De plus, il nous fallait un moyen de lister les utilisateurs sans serveur étant donné que l’application doit être décentralisée.

Après avoir fait quelques recherches et quelques tests, nous avons décidé d'utiliser le mode multicast pour l'envoie d'un message à tous les joueurs disponibles lors de la recherche d’un joueur. Il nous a semblé que le mode multicast était plus simple à mettre en œuvre que le mode broadcast car on n’a pas besoin de connaître précisément la topologie du réseau pour envoyer un message. De plus, on souhaite communiquer seulement avec ceux qui lance l’application, ce qui justifie aussi l’utilisation du Multicast.

Lors de la recherche d’un joueur, voici les points importants :

1) Envoi d’une demande d’informations de la part de tous les autres joueurs connectés au réseau. Lorsqu’un utilisateur reçoit cette demande, il envoie à l’émetteur un message contenant l’ensemble des informations constituant son profil.

2) Pour mettre à jour la liste un bouton permettant de renvoyer le message de demande d’information sera implémenté.

3) Les joueurs doivent être capables de gérer plusieurs demandes de parties simultanément.

Il faudra donc utiliser java.net.MulticastSocket pour lire les paquets reçus en mode multicast et java.net.DatagramSocket pour envoyer un paquet en mode non-connecté (UDP). Enfin, il faudra également utiliser une adresse IP multicast comprise entre 224.0.0.0 et 239.255.255.255 en vérifiant bien que cette adresse n'est pas déjà réservée.

### Les dialogues entre adversaires

Lorsque 2 joueurs se sont mis d’accord pour lancer une partie, les joueurs doivent être capable de dialoguer pour s’envoyer des messages via le chat ou les déplacements des pions.

Pour se faire, losrque deux joueurs lancent une partie, une connexion TCP entre les deux est initialisée. Nous utilisons java.net.Socket et java.net.ServerSocket qui permettent de créer une connexion TCP assez simplement. Par exemple, on peut envoyer et recevoir directement des objets Java sérialisables via java.io.ObjectOutputStream et java.io.ObjectInputStream et une exception se produit s'il se produit un problème avec la connexion.

La réception et l’envoi des messages se feront par le biais de handlers, lors de la reception d’un message, le message sera transmis vers le module correspondant (pour la plus part des cas, ce sera le module traitement de données).

## Difficultés rencontrées

Nous évoquerons dans cette partie les différentes difficultés que nous avons rencontrées et qui nous ont amenées à adapter notre code.

### Synchronisation des threads

Pour être capable de recevoir les messages venant de l’adversaire tout en gardant le contrôle sur l’application, l’utilisation des threads semble être le choix le plus judicieux.

Quand on veut récupérer des messages du réseau, les appels des méthodes concernées sont bloquants, c'est-à-dire que le thread est en attente jusqu’à la réception effective du message. Il faudra donc mettre en place des threads dédiés à la réception des messages mais aussi faire très attention aux problèmes de synchronisation entre les threads. Par exemple, si on envoie un message et qu’au même moment on a reçu un message du réseau alors il est possible d’avoir des problèmes d’accès concurrent à certaines ressources.

Pour répondre à ce problème, nous avons développé plusieurs handlers, un handler de reception de messages, un autre pour l’envoi de messages, et un dernier qui contient les deux précédents permettant la synchronisation des ressources.

### Thread MAJ interface graphique

Un autre problème vient de Swing qui ne peut mettre à jour son interface graphique que dans un thread particulier, le Event Dispacher Thread, appelé aussi EDT alors que nous avons différents threads dédiés au réseau. Nous avons décidé de traiter cette problématique dans notre module pour ne pas la répandre dans les autres modules notamment les modules IHM. Nous devrons donc utiliser des méthodes et des objets spécifiques comme SwingUtilities ou SwingPropertyChangeSupport pour exécuter les interfaces des autres modules dans le thread EDT.

### Apprentissage de JAVA

Il s’est avéré que certain membres du groupe n’avait jamais pratiqué le langage JAVA, une période d’apprentissage (auto-formation) a été prise en compte pour comprendre les principes et les outlis spécifiques à JAVA.

### Envoi des profil en UDP

Quand un joueur cherche un adversaire, l’ensemble des informations qui concernent les joueurs sont envoyées sur le réseau en multicast et donc en UDP. Lors de la phase de test des solutions techniques, nous avions testé d’envoyer des profils sans avatar. Cependant lorsque l’on rajoute un avatar au profil, les datagrammes à envoyer deviennent trop gros et ne peuvent plus être envoyé en une seule fois.

Nous avions donc plusieurs choix possible pour résoudre ce problème :

1) Etablir une connexion TCP pour tout échange de profil. L’inconvénient est que cette méthode risque d’être très couteuse à partir d’un certain nombre de joueurs connectés.

2) Mettre en place un système permettant de gérer l’envoi de datagrammes segmentés qui prend en compte le multicast. L’inconvénient de cette solution est qu’elle risquait de prendre beaucoup de temp de développement.

3) Limiter la taille des avatars de manière à contrôler la taille maximum des datagrammes. C’est la solution que nous avons choisi car c’est celle qui a le meilleur rapport Temps de développement / Sécurité.

## Points particuliers

Cette partie traitera de divers points ayant été soulevés avant, pendant et après la phase de réalisation du module communication et traitement.

### Polymorphisme

En effet, au lieu de tester la classe de chaque message reçu afin d’appeler le traitement correspondant, nous aurions pu utiliser le concept de polymorphisme. Nous aurions défini une classe abstraite « Message »possédant une méthode « traiter » et nous aurions redéfini cette méthode dans chacune des classes filles afin de spécifier le comportement à adopter vis-à-vis de chaque type de message. Ceci nous aurait évité de tester explicitement la classe du message reçu dans les Handlers liés à la réception de messages.

### Récupération adresse IP machine

L’application cliente étant identifiée sur le réseau via son profil (contenant une adresse IP) et le fait qu’un ordinateur muni de plusieurs cartes réseaux puisse disposer de plusieurs adresses IP a soulevé une question : quelle adresse IP devons-nous récupérer lorsque plusieurs sont disponibles (cas de la majorité des pc portables munis d’une interface réseau WIFI et Ethernet). Afin de répondre à cette question, il a été décidé que nous écrirons une fonction permettant de sélectionner la première adresse IP locale étant différente de celle de loopback.

# Gestion des données

## Introduction

## Choix technologiques

## Difficultés rencontrées

## Points particuliers

# IHM connexion

## Introduction

Après avoir établi le dossier conception, nous avons entamé la phase de réalisation du module « IHM Login » qui consiste à implémenter les interfaces de connexion, de gestion profiles, la liste des joueurs connectés, ainsi que la liste des parties terminées et celles qui sont en cours.

La répartition des tâches, au sein du groupe, a été effectuée par binôme dans le but d’assurer un équilibre dans les connaissances techniques.

Dans un premier temps, nous explicitons les différents choix technologiques pris avant le début de la phase de développement, ensuite nous allons aborder les difficultés rencontrées et enfin les points particuliers de cette phase de réalisation.

## Choix technologiques

1. **Netbeans**

Nous avons choisi d’utiliser l’IDE Netbeans pour coder notre application. Ce choix s’est fait en le comparant à Eclipse, l’éditeur de choix pour coder du Java il y a quelques années. De nos jours, il semble que Netbeans soit moins lourd dans son interface tout en gardant les mêmes fonctionnalités. De plus, la majorité des personnes de l’équipe connaissait ce logiciel ce qui a permis une prise en main simplifiée. Même s’il est moins lourd qu’Eclipse, il supporte quand même des fonctionnalités avancées comme le renommage de variable dans tout le code (*refactoring*), l’autocomplétion, l’intégration à des logiciels de gestion de version comme SVN ou Git ou l’ajout de getters/setters automatique.

1. **Swing**

Nous avons choisi d’utiliser la librairie Swing pour créer l’interface. Cette librairie est intégrée à J2SE et permet un développement d’interfaces rapide et multi-plateformes car elle est écrite complètement en Java. Cela lui permet d’avoir le même comportement sur plusieurs plateformes différentes au prix de performances un peu moindres qu’avec Abstract Window Toolkit dont les composants sont écrits directement pour une plateforme spécifique. Cette perte de performance est très faible et est apparemment imperceptible pour les utilisateurs finaux.  
Swing permet d’utiliser l’architecture MVC où les données et leur représentation sont découplées ce qui permet une plus grande modularité et une plus grande séparation des préoccupations.  
Elle utilise la programmation par événements pour signaler des changements du modèle. Par exemple si la liste des joueurs change, un signal sera envoyé à Swing qui “repeindra” le composant qui dépend de cette liste (modèle).

Il est possible d’utiliser plusieurs threads avec Swing ce qui est très important pour notre application qui devra avoir un thread pour écouter sur le réseau et un thread pour l’interface graphique.  
De plus, Swing est intégré à Netbeans ce qui permet de prototyper rapidement des interfaces grâce au designer d’interface. Il permet d’utiliser une interface WYSIWYG pour coder les différentes GUI.

## Difficultés rencontrées

## Points particuliers

# IHM grille

## Introduction

## Choix technologiques

## Difficultés rencontrées

## Points particuliers

# Conclusion générale