|  |  |
| --- | --- |
|  | **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

La phase de réalisation correspond à la mise en œuvre concrète (programmation) du plan de conception. Elle comprend des choix technologiques concrets tels que les bibliothèques et algorithmes à utiliser. Lors de cette phase de réalisation, il est fréquent de rencontrer des difficultés techniques inattendues lors de la conception.

Dans ce rapport, nous présentons par module les choix technologiques faits, ainsi que les problèmes rencontrés. Enfin, nous présentons certains points intéressants qui méritent d’être mis en valeur.

# Communication et traitement

## Introduction

Suite à l’établissement du dossier de conception, l’équipe du module « communication et traitement » a entamé la phase de réalisation de son module. Nous avons veillé à répartir le travail en fonction des ressources humaines disponibles au sein de notre groupe. Au cours de cette phase de développement, nous avons du faire face à diverses problématiques.

Dans un premier temps, nous allons mettre en lumière les divers choix technologiques que nous avons faits. 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 auxquels nous avons du faire face tout en argumentant nos choix.

### La communication avec l'adversaire

Pour cette communication avec l'adversaire, nous avons décidé d'utiliser java.net.Socket et java.net.ServerSocket qui permet de communiquer une mode connecté (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 découverte des joueurs

Après avoir fait quelques recherches et quelques tests, nous avons décidé d'utiliser le mode multicast pour l'envoie d'un message à tout les joueurs disponibles. 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. 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 réservée.

## 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

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.

### 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’ayant pas pratiqué le langage JAVA, une période d’apprentissage (auto-formation) a été prise en compte.

## 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

Lors de la phase de conception, le module Gestion de données a procédé à des choix sur les technologies à employer sur certains aspects critiques du module. Nous discuterons ici de ces choix.

Par ailleurs, le travail a été effectué en binômes débutant/vétéran afin permettre la montée en compétence des membres du groupe qui n’avaient pas encore formé d’expérience sur les technologies du projet. Malgré les bonnes conditions de travail, nous avons rencontré des points de difficulté que nous exposerons.

Enfin, certains points particuliers nous ont semblé mériter quelques commentaires.

## Choix technologiques

### Sérialisation

La persistance ou sérialisation est un mécanisme technique qui permet de stocker les objets dans des fichiers (on aurait aussi pu imaginer une base de données mais cette solution n’est pas adaptée dans notre cas). Nous avons eu une réflexion sur le format de la persistance (XML ? JSON ? binaire ?) qui a donné lieu à une étude technique.

Au début, nous nous étions mis d’accord sur une persistance en JSON (privilégié par rapport à XML pour sa plus grande concision) puisque ce format aurait permis une évolution de l’application plus intéressante (réutilisation par des terminaux mobiles par exemple) par rapport à une persistance binaire.

Bien qu'il existe des librairies très puissantes pour la sérialisation en JSON  
(par exemple google-gson, developpée par Google : http://code.google.com/p/google-gson/), nous avons finalement privilégié la sérialisation Java native (binaire). En effet, celle-ci est la seule à supporter les références circulaires dans les objets sérialisés. Étant donné que supprimer les  
références circulaires dans notre graphe d'objet aurait représenté un travail important, nous avons décidé que le jeu n'en valait pas la chandelle.

La seule contrainte de ce mode de persistance est la nécessité de l’implémentation de l’interface Java Serializable. Cette dernière ne nécessitant aucune implémentation de méthode, la contrainte est extrêmement faible.

De plus, nous avons du réfléchir à la mise en place de la persistance (méthode de sauvegarde/chargement dans les objets vs objet externe chargé de la persistance).

Comme nous avons opté pour une architecture données/manager, il nous a semblé naturel de respecter le même schéma pour la persistance, pour des raisons de cohérence. Une classe (le Serializer, non présenté sur le diagramme de classe dans un souci de clarté, ce dernier n’apportant rien à la compréhension du problème) aura donc la responsabilité de faire sauvegarder et charger les objets dans des fichiers.

### Observation des événements

Au début du développement, nous avons mis en balance l’utilisation de l’interface Java Observable afin d’implémenter l’annonce d’événements avec l’utilisation de la classe PropertyChangeSupport. Finalement, c’est PropertyChangeSupport qui a été retenue car elle correspond à une solution clé-en-main pour notre problème particulier. En effet, la méthode publish() permet d’envoyer une information sur une chaîne dédiée. Tous les objets qui ont souscrit avec la méthode subscribe() sont notifiés du changement. Par ailleurs, on peut passer un string en complément d’information ce qui est précieux vis-à-vis de notre conception.

## Difficultés rencontrées

### Java, NetBeans, et Subversion

Dans le groupe, les compétences sur le langage Java et le JDK, l’IDE NetBeans, ou encore la manipulation de Subversion étaient hétérogènes au départ. Nous avons opté pour un travail par binôme avec un vétéran et un débutant, afin de faire monter en compétences les étudiants qui n’avaient jamais eu d’expérience sur ces technologies.

Le challenge était de taille car même si le langage Java est relativement accessible aux étudiants ayant fait de l’Objet auparavant (LO21), le concept de SVN ou même l’environnement NetBeans sont fortement étrangers au premier abord pour un débutant. Après la période de développement, nous avons constaté que la stratégie avait payé car les binômes se montraient très capables.

# 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

### 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.

### 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

Lors de la phase de réalisation, les modules ont rencontré des problèmes techniques concrets qui n’avaient pas été anticipés en conception. Cependant, des solutions ont été trouvées dans tous les cas, et l’étude et la documentation des problèmes rencontrés a permis aux membres du projet de gagner de l’expérience et de monter en compétence.