Projet Java RMI

Multi-User Dungeon

But : le but de ce projet est de créer un jeu de rôle massivement multi-joueur de type MUD (Multi-User Dungeon) en Java avec les Sockets TCP, UDP et/ou RMI.

Étape 1 : MUD Simple

La première étape doit permettre à plusieurs utilisateurs de jouer simultanément sur un serveur qui va gérer un labyrinthe (ou souterrain ou donjon...). Lors de la première connexion, l'utilisateur devra fournir le nom de son personnage.

Le labyrinthe est composé de :

- Pièces (l'une d'elle étant l'entrée du labyrinthe) ;
- Portes (reliant les pièces entre-elles) ;

Pour simplifier, les pièces ne seront connectées qu'en damier et donc on aura, au plus, une porte au nord, au sud à l'est et à l'ouest. L'utilisateur pourra donc utiliser des commandes textuelles (N, S, E, O) pour se déplacer.

Lorsque plusieurs personnages se trouvent dans la même pièce le serveur doit les prévenir et ils pourront discuter, par exemple en tapant "**Bonjour** (le guillemet servant à différencier les discussions des autres commandes).

La discussion devra passer par un autre serveur qui ne servira qu'à cela pour éviter de saturer le serveur gérant le labyrinthe dans un cadre vraiment massivement multi-utilisateur.

Plan de travail :

- Lire le sujet jusqu'au bout ;
- Concevoir l'application avec UML ;
- Programmer la première version des clients et serveurs ;

Étape 2 : Et voilà les monstres et les combats...

Pour rendre le jeu plus intéressant on introduit des monstres et des combats lors de cette étape. A chaque fois qu'un personnage entre dans une pièce, un monstre apparait et attaque ce personnage. Le combat se déroule en plusieurs tours, chacun durant 1 seconde. A chaque tour, le personnage ou le monstre perd un point de vie (aléatoirement). Au départ, les personnages ont 10 points de vie et les monstres 5. Le personnage pourra à chaque tour de combat choisir de continuer ou de fuir.

Un combat se termine si un participant fuit ou se retrouve à 0 points de vie. Le participant qui atteint 0 points de vie meurt (le personnage ou le monstre disparait).

A l'issue d'un combat, le vainqueur gagne 1 point de vie supplémentaire (pour faire simple). D'autre part, dès qu'il n'y a plus de combat dans la pièce, on considère que tous les personnages ou monstres encore en vie regagnent tous leurs points de vie.

Ici aussi cette partie devra fonctionner sur un autre serveur. On pourrait même imaginer un ensemble de serveurs permettant d'équilibrer la charge en plaçant un nombre équivalent de combats sur chacun d'eux.

Étape 3 : Combats simultanés et entre joueurs...

Si, pendant un combat, d'autres personnages se trouvent dans la même pièce ils peuvent se joindre au combat en indiquant qui ils attaquent (par exemple en tapant **Attaque Monstre**). Dans ce cas on considère qu'il y a désormais 2 combats qui se déroulent en parallèle. Ainsi si le monstre M attaque le personnage A et le personnage B attaque le monstre M. On va déterminer, à chaque tour si M ou A perd un point de vie et si B ou M perd un point de vie (M pouvant donc perdre au maximum 2 points de vie par seconde).

Enfin, si plusieurs personnages se trouvent dans la même pièce, l'un deux peut décider d'attaquer un autre personnage.

Étape 4 : Labyrinthe Multi-Serveurs

Pour pouvoir gérer un plus grand nombre d'utilisateurs on va désormais répartir le labyrinthe sur plusieurs serveurs. Lorsqu'un personnage passe d'une pièce gérée par un serveur N vers une pièce gérée par un serveur M, le client va arrêter de dialoguer avec le serveur N et commence à dialoguer avec le serveur M.

Étape 5 : Persistance

On veut désormais gérer la persistance : si un joueur quitte le jeu et, plus tard, redémarre son client il doit apparaitre au même endroit avec le même nombre de points de vie et le même nom. Il faut donc un serveur de persistance des personnages qui stocke leurs caractéristiques et vers lequel on envoie, par exemple, à chaque changement de pièce et à chaque tour de combat des mises à jour si nécessaire (perte ou gain de points de vie).

Lors du redémarrage on récupère les informations utiles (en communicant avec le serveur de persistance) et on commence à dialoguer avec le serveur qui gère la pièce dans laquelle on se trouvait.

Note : toute fonctionnalité supplémentaire (IHM, 3D, déplacement des monstres, intelligence artificielle, gestion de trésors, épées, boucliers, armures, portes secrètes ou verrouillées, sortilèges, classes de personnages, règles complètes de Donjons&Dragons™ ...) ne sera prise en compte dans la notation que si toutes les étapes ont été correctement traitées.

Organisation du projet et travail attendu par trinôme :

Partie I : Conception de l'application

Concevez l'application permettant de répondre aux spécifications précédemment énoncées.

Constituez un dossier qui comportera les diagrammes UML adéquats exprimant le résultat de l'analyse de cette application (au minimum : cas d'utilisation, diagrammes de séquence détaillés et de classes/interfaces ou de composants). Note : il faudra détailler, autant que possible, les méthodes des futures interfaces Remote si vous utilisez RMI ou le protocole utilisé si vous utilisez les Sockets.

Partie II : Développement d'une maquette de l'application

Une maquette de la solution envisagée sera développée. Celle-ci devra donner une image fidèle de la solution réelle ensuite déployée. On s'attachera à y rendre opérationnelles les interactions entre entités.

Partie III : Évaluation

Des séances d'évaluation régulières seront organisées pour suivre la réalisation de votre projet. A chaque séance, chaque groupe devra présenter son projet et faire une démonstration à partir d'un jeu de tests. Ce même jour, vous devrez rendre le dossier de conception mis à jour sous forme de rapport **PDF** dans lequel vous rajouterez quelques éléments clés de programmation. De plus, vous devrez rendre les **sources courantes du projet évalué**. Ces différents fichiers devront être déposés sur le moodle de l'Université.