11/05/2020

Rendu de projet

La roue de la fortune

Compte rendu de la conception et du développement de notre projet

Professeur : Gilles Menez
Matière : Programmation
Auteurs : Armand Prévot**,** Steven Bouche, Pierre Griseri, Corentin Bailet.

Table des matières

I – Éléments d'une partie

1.1 – Déroulement et règles en œuvre de notre application

Tout d'abord, il est intéressant de préciser que nous nous sommes basés, comme demandé dans le sujet, sur l'émission « La Roue de la Fortune » de TF1 dont en voici l'énoncé :

A noter : chaque joueur possède une cagnotte de manche, une cagnotte de caverne et une cagnotte totale.

- 1. Avant de commencer, une énigme rapide se déclenche, celle-ci fait gagner 500 euros à celui ou celle qui la remporte. Cette énigme se déroule de la façon suivante : les lettres apparaissent toutes les deux secondes et le joueur ayant donné la bonne réponse le plus rapidement remporte cette énigme rapide (non implémenté, cf. la mention IMPORTANT ci-dessous).

- 2. Une manche commence avec une énigme rapide qui elle aussi fait gagner 500 euros à la cagnotte de manche du joueur mais qui en plus, détermine quel joueur prendra la main pour cette manche.

- 3. C'est alors que se lance une énigme principale : c'est au tour du joueur qui a prit la main durant l'énigme rapide de commencer. Durant un tour, le joueur ayant la main a alors trois possibilités, il peut :

    - a) Lancer la roue, s'il tombe sur une case d'argent, il doit forcément proposer une consonne (s'il essaye de faire autre chose, il perd la main). Si la consonne est bien contenue dans l'énigme, il gagne (dans sa cagnotte de manche) la somme de la case sur laquelle la roue est tombée multiplié par le nombre d'occurrences de la consonne dans l'énigme, sinon il perd la main. Le descriptif entier des cases de notre application est défini plus bas.

    - b) Acheter une voyelle, s'il n'a pas assez d'argent ou s'il essaye d'en acheter une deux fois de suite ou encore si la voyelle n'est pas contenue dans l'énigme, il perd la main. Si la voyelle est bien contenue dans l'énigme, il garde la main. Que la voyelle soit contenue ou non dans l'énigme, le montant de manche du joueur est débité de 150 euros.

    - c) Proposer une réponse pour l'énigme, si sa proposition est juste la manche se termine, sinon il perd la main. Lorsqu'elle se termine, seul le joueur ayant gagné la manche voit sa cagnotte de manche transférer à sa cagnotte totale, les autres ne la conservent pas.

- 3.bis A noter qu'après chacune de étapes ci-dessus, s'il le joueur a gardé la main et si l'énigme n'est pas résolue, il a toujours les mêmes possibilités.

- 4. Les étapes 2 et 3 sont répétées quatre fois. A noter que dans la seconde manche, l'énigme principale est une énigme à double-sens : une fois l'énigme découverte, le candidat ayant proposé la bonne réponse doit répondre à la question que pose l'énigme pour 500 euros en plus. S'il répond mal, il perd la main et la question est alors posé au candidat suivant (non implémenté, cf. la mention IMPORTANT ci-dessous).

- 5. Une fois les quatre manches passées, on détermine le joueur ayant la plus grosse cagnotte totale, celui-ci va alors en finale. La finale se déroule ainsi : le finaliste tourne une roue composée d'enveloppes d'argent, il ne prendra connaissance de la somme de l'enveloppe sur laquelle il est tombé qu'une fois la finale terminée. Ensuite, on affiche une énigme en montrant les lettres les plus utilisés de la langue française : RSTLNE. Une fois ceci affiché, le candidat à 10 secondes pour donner la bonne réponse ; s'il réussit, il gagne la somme dans l'enveloppe en plus de sa cagnotte, sinon il ne gagne que sa cagnotte. Dans les deux cas, s'il a gagné une caverne et/ou un voyage, il repart aussi avec.

IMPORTANT : Nous avons respectés ces règles à quelques détails près : dans notre application, il n'y a pas l'étape 1 décrite ci-dessus. De plus, nous n'avons pas non plus implémenté les énigmes à double-sens (celles sous forme de questions). Sinon, tout ce qui ait décrit ci-dessus a été implémenté.

1.2 – Les cases de notre roue et leurs rôles

Dans l'émission télévisé, en fonction de nombreux facteurs, les cases de la roue changent entre deux émissions. Nous avons donc fixé sur notre roue les cases les plus récurrentes, en voici la liste :

- Case CASH : elle contient une valeur en euros dont en voici la liste : 50, 100, 150, 200, 250, 300, 500.
- Case PASSE : fait perdre la main au joueur ayant tourné la roue.
- Case BANQUEROUTE : fait perdre la main au joueur et met sa cagnotte de manche à 0.
- Case SUPER CASH : contient une valeur en euros dont en voici la liste : 1.000, 2.500, 5.000, 10.000.
- Case MYSTERE : contient soit une case PASSE, soit une case CASH.
- Case CAVERNE : permet d'accéder à la caverne au cadeau, même si le joueur perd, le montant est conservé par celui-ci. Ici la caverne est modélisée sous forme d'un chiffre aléatoire entre 100 et 2000.
- Case HOLDUP : permet de voler la cagnotte de manche d'un autre candidat
- Case VOYAGE : permet de gagner un voyage qui lui aussi est indépendant de son résultat au jeu. Modélisé sous forme d'une chaine de caractère pioché aléatoirement dans une liste prédéfinie.
- Case FINAL : case uniquement contenue dans la roue finale, contient une valeur en euros dont en voici la liste : 20.000, 35.000, 50.000, 70.000, 90.000, 100.000.

1.3 – Les joueurs

Trois joueurs sont nécessaires pour lancer une partie. Comme précisé précédemment, chacun possède une cagnotte de manche, une cagnotte de caverne, une cagnotte totale, la cagnotte totale étant remplie uniquement lorsqu'un joueur remporte une manche.

Si un joueur tombe sur la case caverne et/ou la case voyage et perd la partie, il repart tout de même avec ses cadeaux et/ou son voyage. Chaque joueur est représenté par son pseudo, qu'il choisit avant de rentrer dans la partie.

II – Déroulement d'une partie

2.1 – L'accueil des joueurs : le lobby

Lorsqu'on lance l'application, l'utilisateur arrive sur un lobby. Ce lobby affiche la liste des serveurs de jeu disponibles et permet au joueur de se connecter à une partie, une fois qu'il a rentré son nom.

L'utilisateur, une fois qu'il a cliqué sur le bouton pour se connecter à un serveur, se retrouve dans une salle d'attente. Cette salle d'attente permet d'accueillir trois joueurs.

Chaque joueur se voit alors octroyer la possibilité d'indiquer au serveur qu'il est prêt via un bouton, il peut d'ailleurs l'annuler en recliquant sur le bouton, et ceci autant de fois qu'il le souhaite. Il peut aussi annuler sa venue dans la salle d'attente via un deuxième bouton, ce qui aura pour effet de le faire retourner sur le lobby.

2.2 – Le déroulement de la partie

Une fois la salle d'attente remplie de trois joueurs et les trois joueurs prêts, la partie se lance selon les règles expliquées plus haut (cf. 1.1).

2.3 – La fin de la partie

Lorsqu'une partie se termine, chaque joueur se voit déconnecter du serveur de jeu en cours ; ils sont alors individuellement redirigés vers le lobby et peuvent se reconnecter à un serveur de jeu.

Pour ce qui est du serveur de jeu qui vient de se terminer, il se réinitialise afin d'être à nouveau disponible dans la liste des parties du lobby. Il est intéressant de préciser que nous avons ajouté un bouton permettant de rafraichir la liste des serveurs de jeu.

2.4 – Quitter une partie en cours

Il n'y a que deux façons pour un joueur de quitter une partie en cours :

- Il subit une déconnexion à la suite d'une perte de réseau.
- Il ferme la fenêtre de l'interface de jeu.

Dans les deux cas, la partie s'arrête et les autres joueurs sont renvoyés dans le lobby. Une reconnexion à une partie en cours n'est donc pas possible.

III – Spécifications techniques du projet

3.1 – Technologies utilisées

Notre projet à été développé sous l'outil de gestion Maven.

Pour la partie serveur :

- Développé en C# sous Visual Studio.
- Framework .NET Core 3.1.
- Hébergé sur un serveur virtualisé OVH (adresse : 51.210.12.245, nom de domaine : vps-04a7f6d5.vps.ovh.net).

Pour la partie client :

- Développé en Java 11 sous IntelliJ.
- Interface développée en JavaFX 11 avec compléments de JFoenix.

3.2 – Protocoles réseau

Nous avons longtemps hésité entre le protocole UDP et TCP, et puis finalement, nous nous sommes demandé : « et pourquoi pas les deux ? ». C'est alors ce que nous avons fait ; le protocole UDP est utilisé pour le lobby et le protocole TCP pour la gestion des parties :

- UDP étant un protocole non-connecté, il nous paraît le plus judicieux pour gérer la connexion au serveur, car si les paquets sont perdus les clients peuvent facilement se reconnecter au lobby.

- TCP, lui étant en mode connecté, est selon nous le protocole le plus sûr pour nous assurer qu'aucune transmission client-serveur ne soit perdu pendant une partie. De plus, il nous permet de détecter facilement si un client a quitté la partie ou a perdu la connexion.

3.3 – Interconnexion UDP – TCP

Comme expliqué dans la partie II, les joueurs sont d'abord accueillis dans un lobby, puis dans une salle d'attente une fois qu'ils ont sélectionnés un serveur de jeu et qu'ils ont envoyés leur nom au serveur ; lorsqu'ils sont trois et qu'ils sont tous prêts, la partie se lance.

C'est ici que nous passons d'UDP à TCP, voici ce qui se passe :

- Lorsqu'un utilisateur lance l'application, il envoie un paquet UDP pour récupérer la liste des serveurs disponibles ; le serveur lui renvoie cette liste avec un autre paquet UDP.

- L'utilisateur peut alors sélectionner le serveur qu'il souhaite en cliquant dessus. Ensuite, il rentre son nom et appuie sur le bouton « Connect Server » : c'est à ce moment que le client ferme sa socket d'écoute UDP et se connecte en TCP au serveur de jeu.

- Il rentre alors en mode connecté dans la salle d'attente, jusqu'à ce qu'il y ait trois joueurs prêt et que la partie se lance.

3.4 – Gestion des processus serveur

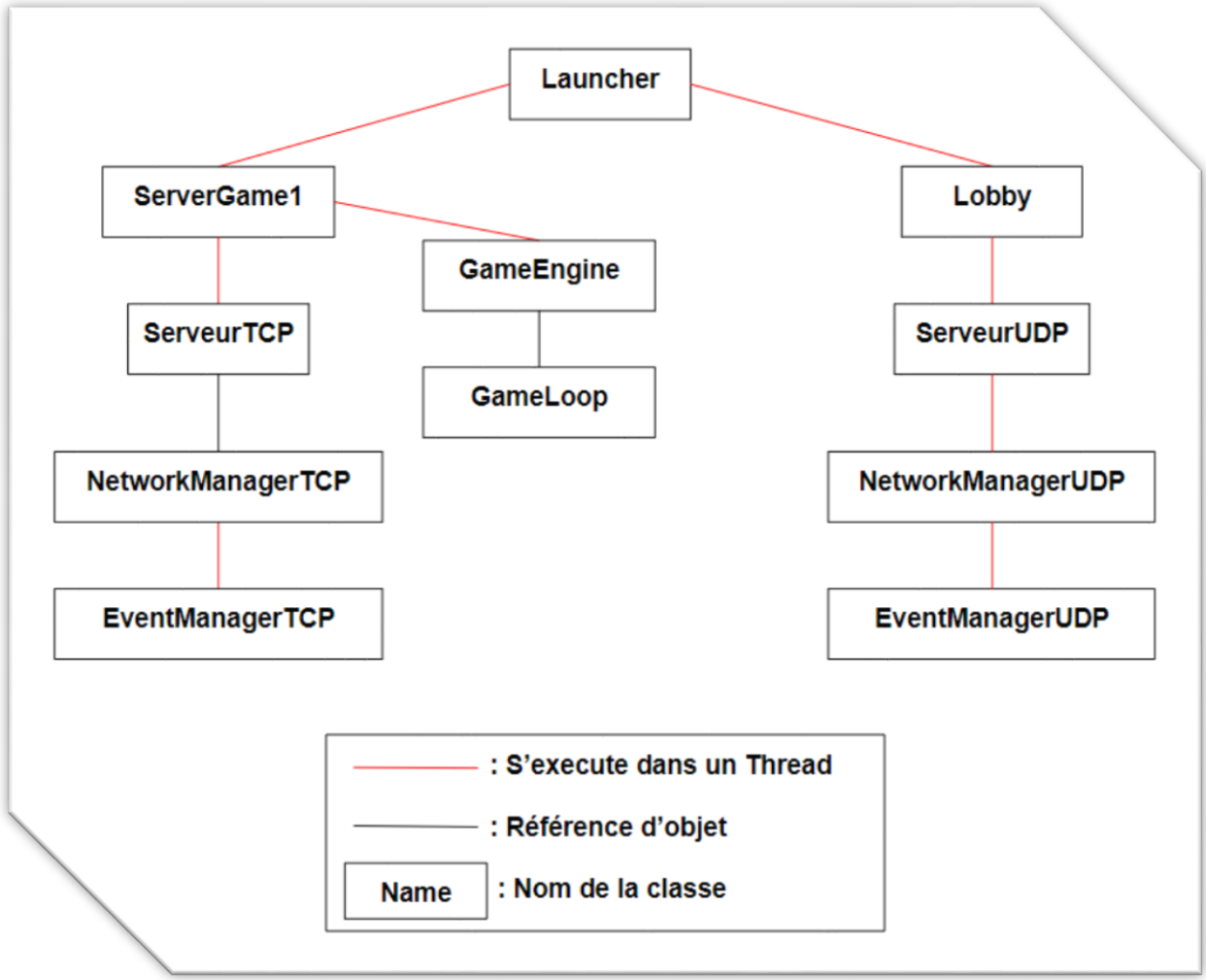

Figure 1 – Schéma de l'enchainement des processus (Serveur)

Le schéma ci-dessus représente l'exécution étape par étape du serveur, comme mentionné dans la légende, les liaisons en rouge indiquent une exécution dans un thread, alors que celles en noire représentent les références d'objets. Ainsi, on peut constater que la classe Launcher lance deux threads, un pour la gestion des parties et un pour la gestion du lobby.

De fait, la partie droite du schéma montre l'enchainement des processus du lobby et celle de gauche indique les processus d'une seule partie. Il est intéressant de noter que la classe Launcher possède une collection de la classe ServerGame, c'est pourquoi nous avons ajouté un « 1 » à côté du nom de cette classe ; en effet, la partie gauche du schéma est dupliquée pour chaque partie. En revanche, la partie droite elle, puisque nous ne voulons qu'un seul et unique lobby, n'est instanciée qu'une seule fois pour toutes les parties.

3.5 – Gestion des processus client

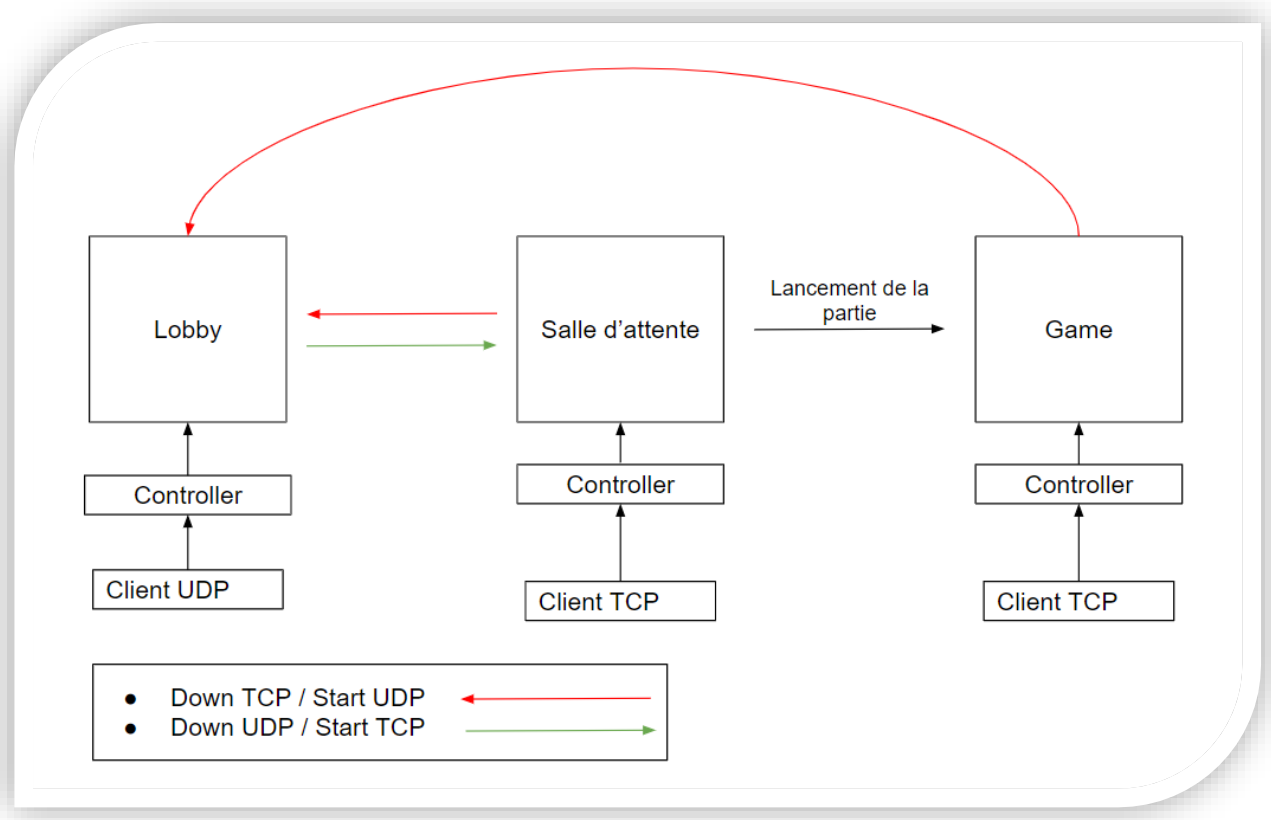

Figure 2 – Schéma des processus d'interfaces (Client)

Le schéma ci-dessus représente les processus de mise à jour entre les différentes interfaces d'un client. Lors de l'arrivé dans le lobby, nous initialisons un client UDP, muni d'un Controller pour faire le lien avec l'interface. Une fois son nom choisit et son serveur sélectionné, nous passons dans la Salle d'attente, pour ce faire nous utilisons un client TCP lui aussi muni d'un Controller. A la création de ce dernier, nous effectuons alors une destruction de la socket UDP. Une fois les trois joueurs prêts, le jeu commence.

Lors d'une de fin partie ou d'une déconnexion, les sockets TCP sont alors détruites, et nous recréons des clients UDP.

3.6 – Gestion des événements client - serveur

3.6.1 – Création du paquet à envoyer

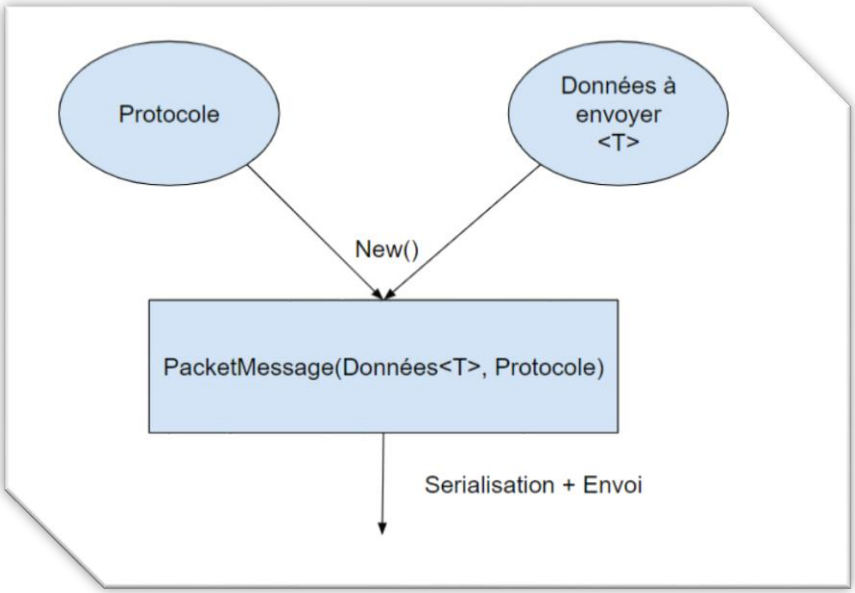

Figure 3 – Schéma du paquet à envoyer

La première chose à savoir sur nos événements client-serveur, c'est la généricité de ceux-ci. En effet, sur notre réseau les éléments envoyés sont toujours des PacketMessage ; avant chaque envoi, nous créons un PacketMessage contenant n'importe quel objet (d'où la généricité) associé à un protocole : en ce sens, l'objet PacketMessage encapsule les données. Nous voyons sur le schéma ci-dessus comment un PacketMessage est instancié.

3.6.2 – Envoi de données

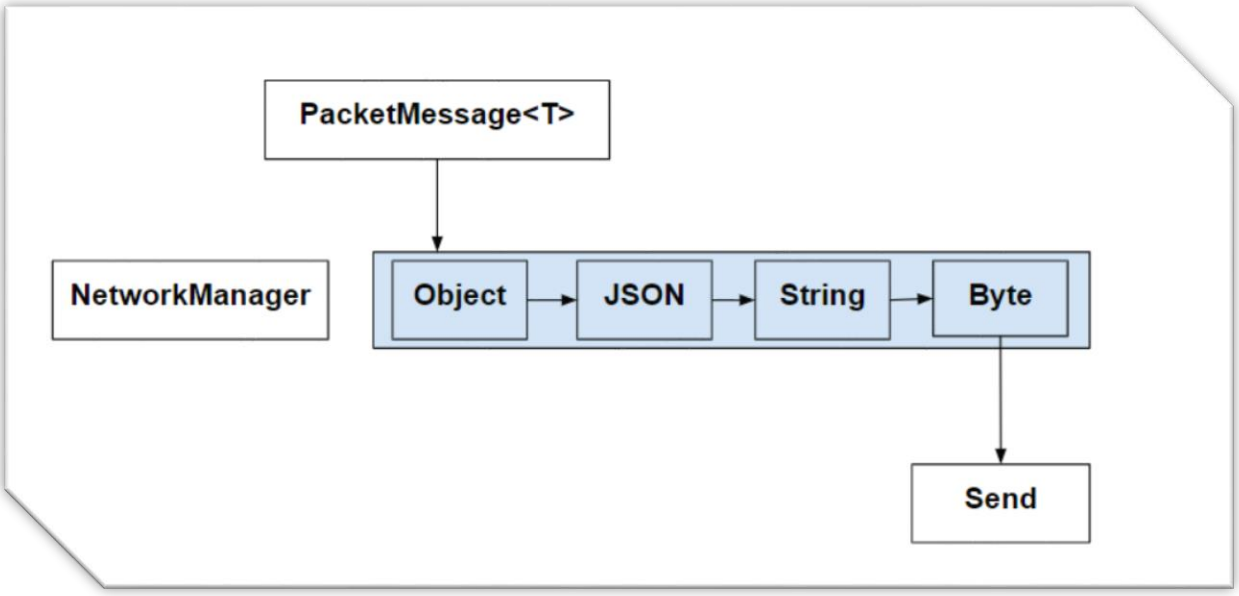

Figure 4 – Schéma de l'envoi d'un paquet

Pour ce qui est de l'envoi d'un PacketMessage, le schéma ci-dessus en résume le procédé : la classe NetworkManager reçoit un PacketMessage, le sérialise en Byte en passant par du JSON, avant de l'envoyer.

Il est intéressant de noter que ce système ne marche que si le client et le serveur communique exactement de la même façon : nous avons alors établit nos échanges réseaux de façon chirale, c'est-à-dire que chaque objet / protocole que nous créons côté serveur pour être envoyer doit être créer de la même façon côté client pour recevoir, et inversement.

En d'autres termes, pour que nos clients et notre serveur puissent échanger, il faut qu'il parle la même langue.

3.6.3 – Réception de données

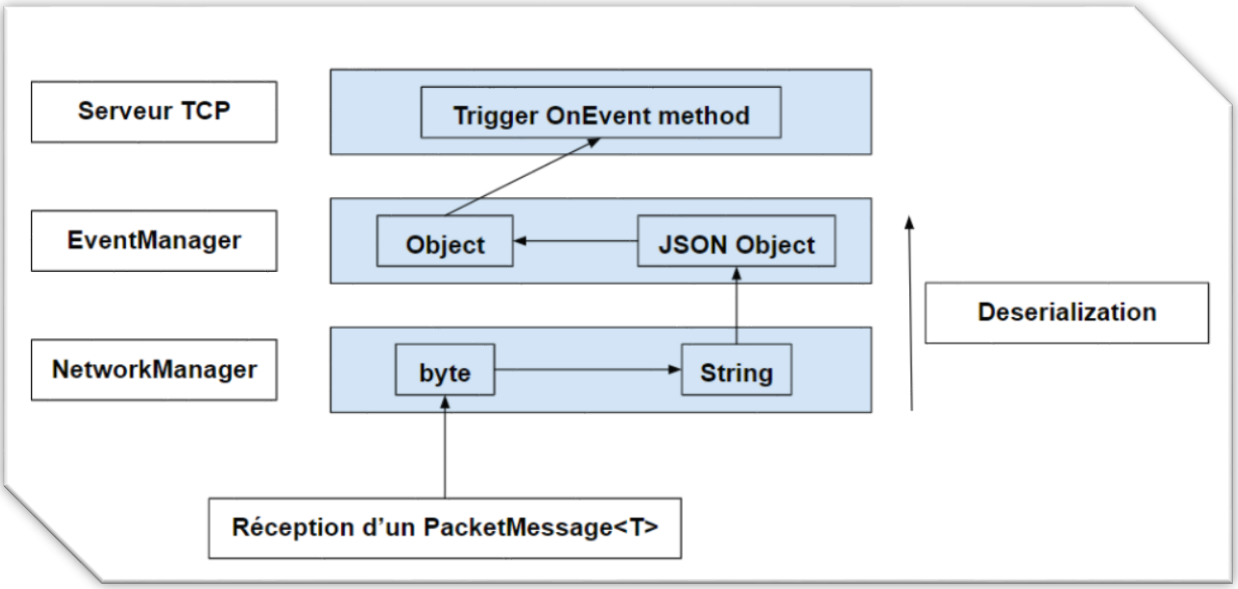

Figure 5 – Schéma de la réception d'un paquet

Ci-dessus est représenté la réception d'un PacketMessage et sa désérialisation : la classe NetworkManager reçoit le paquet puis le convertit en l'objet correspondant contenu dans les données du PacketMessage. Enfin, grâce à un système de pointeur de méthode (le « delegate » en C#), le serveur TCP activera la méthode correspondante au protocole envoyé pour traiter les données reçues.

3.7 – Gestions des actions du jeu

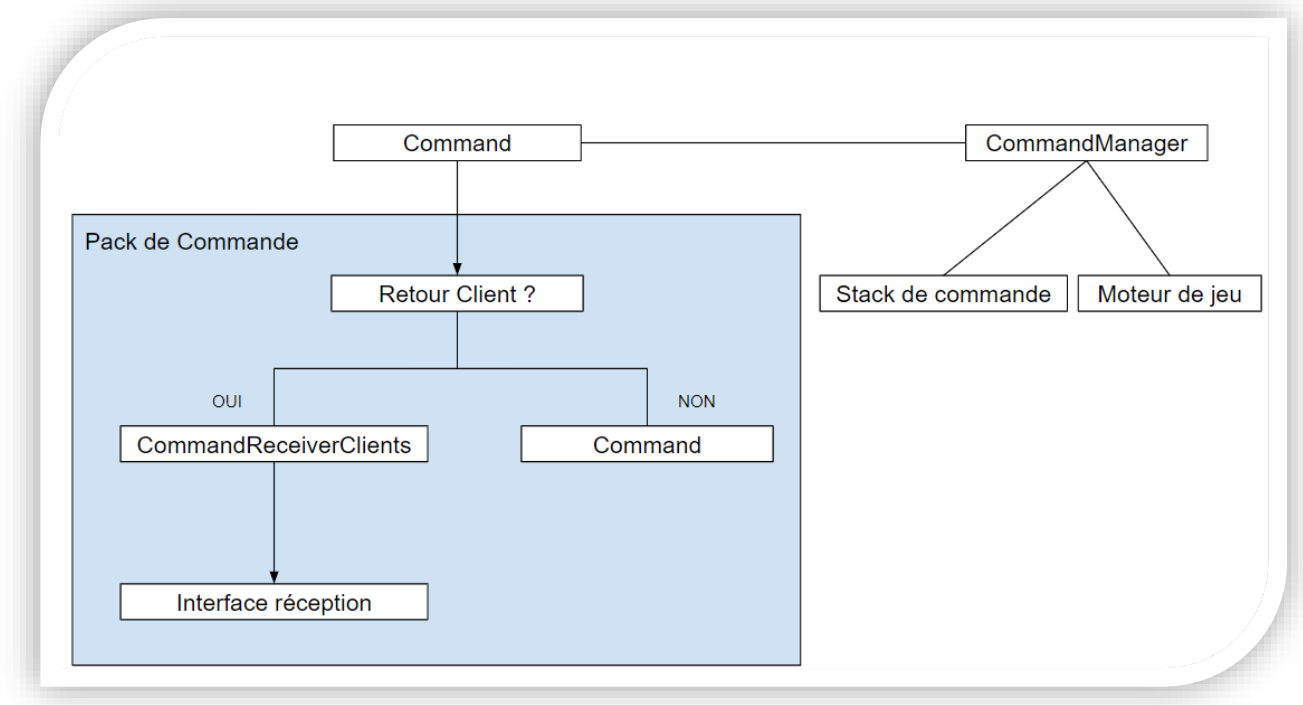

Figure 6 – Schéma du système de commandes (Serveur)

Ce schéma porte sur le Design Pattern Commande, des commandes liées au jeu. Il faut savoir que chacune des actions du jeu est encapsulé dans une commande.

Le CommandManager possède une pile de commande ainsi qu'une référence du jeu en cours. Il peut, via ordre du moteur, déclencher une action de jeu ; celle-ci peut se créer de 2 manières en fonction du besoin d'un retour client ou non.

Si la commande nécessite une action d'un client, on créera une commande implémentant l'interface nécessaire pour notifier le client en particulier. En revanche si le retour client n'est pas nécessaire une commande normale sera exécutée.

L'avantage principal de ce design pattern étant de pouvoir garder une exécution séparée des différentes actions du jeu.

IV – Conclusion

Ce projet a été développé en l'espace d'une semaine et grâce l'effort collectif et journalier de notre groupe, nous avons réussi à fournir une solution viable dans les temps. Nous sommes fiers de notre rendu et espérons qu'il sera considéré à la hauteur des efforts fournis.

Nous sommes bien conscients que notre solution mérite des améliorations, mais au vu temps imparti, nous avons fait au mieux. Il manque notamment la documentation et les classes des tests pour assurer une application stable.

Pour conclure, nous remercions notre professeur de nous donner l'opportunité de nous améliorer et de tester nos connaissances sur la conception et le développement d'applications entières.