Une image contenant texte, horloge, clipart

Description générée automatiquement

HADID Hocine et CHEMIN Pierre | 30/03/2023

Dirigé par **RABAT Cyril et DELISLE Pierre**

INFO 601/604 – Jeu multijoueur en réseau

Table des Matières

[1. Introduction 2](#_Toc131185829)

[2. Editeur de monde 3](#_Toc131185830)

[A. La représentation d’un monde dans l’éditeur 3](#_Toc131185831)

[B. Le fonctionnement de l’éditeur 4](#_Toc131185832)

[3. Utilisation du protocole UDP dans les applications client-serveur 6](#_Toc131185833)

[A. Lancement des applicatifs 6](#_Toc131185834)

[B. Communication entre les deux applications 6](#_Toc131185835)

[C. La gestion des parties et des joueurs 7](#_Toc131185836)

[D. Les différents échanges entre le serveur et le client 9](#_Toc131185837)

[E. Les différents échanges entre le serveur et le client via le protocole TCP 14](#_Toc131185838)

1. Introduction

Nous avons entrepris la conception d’un projet appelé Starlyze, qui consiste à développer un jeu de plateforme multijoueur en ligne. L’objectif est de créer un monde virtuel dans lequel chaque joueur peut contrôler un personnage et interagir avec d’autres joueurs de la même partie.

Le concept de base du jeu est simple : chaque joueur peut créer son propre monde et y jouer avec d’autres joueurs. Cela signifie qu’il n’y aura pas de monde prédéfini, et chaque joueur aura la liberté de créer le monde qu’il souhaite avec ses propres niveaux, décors, obstacles et ennemis. Ainsi, chaque partie de Starlyze sera unique et offrira une expérience de jeu différente à chaque fois.

Le projet est divisé en trois parties principales : l’éditeur, le serveur et le client. L’éditeur permet aux joueurs de créer leur propre monde et de le personnaliser selon leurs préférences. Les joueurs peuvent ajouter des objets, des niveaux et des décors à leur monde. Le serveur est la partie du jeu qui gère la mise en relation des joueurs et la gestion de la partie. Il est responsable de la création de parties, de la gestion des joueurs et de la communication avec les joueurs. Enfin, le client est l’application qui permet aux joueurs de visualiser et de jouer au jeu. Il est conçu pour être facile à utiliser et à comprendre, avec une interface utilisateur intuitive. Le client affiche la partie en temps réel et permet aux joueurs de contrôler leur personnage.

De plus, les applications « serveur » et « client » communiquent tout au long de la partie via différents protocoles afin de garantir la fluidité de l’expérience de jeu et une interaction en temps réel entre les joueurs.

En conclusion, le projet Starlyze est un jeu de plateforme multijoueur en ligne qui offre une expérience de jeu unique à chaque partie. Ce projet est une excellente opportunité pour les joueurs de se plonger dans un monde virtuel où ils peuvent jouer et interagir avec d'autres joueurs de manière créative et amusante.

1. Editeur de monde

Dans cette partie nous allons aborder l’application permettant d’éditer les mondes et niveaux du jeu. Il faut effectivement d’abord créer son monde et ses niveaux avant de pouvoir jouer. Nous verrons comment nous avons géré la sauvegarde des mondes, l’édition de ces derniers et comment leurs niveaux sont représentés dans l’éditeur ainsi que son fonctionnement.

1. La représentation d’un monde dans l’éditeur

Nous aborderons ici la représentation d’un monde et sa gestion uniquement du côté de l’éditeur.

Tout d’abord pour ce qui est de la représentation d’un monde. Il est composé d’une multitude de grille que l’on appellera niveau ou « level ». Un niveau est tout simplement une grille de 60x20, c’est-à-dire une matrice de 60 cases de largeur et de 20 cases de hauteur. Un monde proprement construit et complet est composé de plusieurs niveaux dont seulement un possède la « porte d’entrée » à savoir là où les joueurs apparaissent dans le monde et un autre niveau qui détient la « porte de sortie », c’est-à-dire la fin du monde, là où le joueur doit se rendre pour gagner.

1. Les éléments d’un niveau

Chaque niveau possède des éléments appelés « sprite », chacun d’eux possède un identifiant, la largeur ainsi que la hauteur, la position X, la position Y et la spécificité. Ce dernier attribut n’est valable que pour les portes, les portails et les clés. En effet, ces éléments ont besoin d’informations complémentaires qu’il est essentiel de conserver pour permettre le bon fonctionnement du jeu. Une porte a besoin d’un identifiant spécifique pour savoir vers quelle autre porte elle mène, le même principe s’applique aux portails et clés qui fonctionnent de pairs.

Lorsqu’un élément est enregistré dans un fichier, on n’écrira que son identifiant et sa spécificité puisqu’avec l’identifiant on retrouve le type d’élément de la case permettant alors de définir sa largeur et sa hauteur qui n’ont donc pas besoin d’être sauvegardé.

1. Le monde dans l’éditeur

Pour représenter notre monde dans l’éditeur nous avons créé une structure intitulée « game\_level » qui possède plusieurs paramètres :

* ***width* :** un entier représentant la largeur du niveau
* ***height* :** un entier représentant la hauteur du niveau
* ***elements\_map* :** une matrice en deux dimensions contenant des pointeurs vers les éléments du niveau

Le fait de représenter le niveau sous forme d’une matrice de pointeur nous permet de faciliter la gestion mémoire des éléments en évitant de multiples copies ou encore des oublis de libération de mémoire lors de la suppression ou modification d’un élément. Par exemple, pour ajouter un élément il suffit que chaque case occupée par cet élément pointe vers ce dernier. C’est pourquoi pour le supprimer ou le modifier on ne modifie que le pointeur de l’élément pour que cela soit effectif sur toutes les cases. Lors de la suppression toutes les cases occupées par l’élément sont mises à **NULL** et on libère la mémoire occupée par l’élément.

1. Le monde dans un fichier

Pour pouvoir sauvegarder au mieux un monde dans un fichier, il faut que ce dernier soit structuré. Par conséquent, le fichier possède des tables d’adresses qui sauvegarde la position de la représentation des niveaux dans le fichier. On retrouve également des tables de vides qui permettent de savoir où on peut insérer un niveau sans problème. Chacune de ces tables possèdent 10 adresses, la première étant réservé pour pointer vers la table précédente et la dernière pour la table suivante, ce qui fait qu’il reste 8 adresses utilisables pour renseigner nos niveaux.

On rappelle que pour un monde il faut un fichier qui sera composé au minimum d’une table d’adresses, d’une table de vide et d’un niveau. Il ne peut pas y avoir plus d’un seul monde dans le même fichier. Le nom du fichier est le nom du monde et son extension est « bin ».

Ensuite, pour minimiser les appels systèmes « read », « write » et « lseek », il a fallu réfléchir à une structure et une méthode spécifique pour enregistrer un niveau. En effet, nous évitons d’enregistrer toutes les informations de chaque élément car certaines peuvent être retrouvé notamment la largeur et la hauteur. C’est pour cela qu’un élément du niveau n’a pas la même représentation dans le fichier que dans l’éditeur. Dans un fichier nous enregistrons seulement son type suivit de sa spécificité. De plus, les éléments sont enregistrés en parcourant chaque case de la matrice représentant le level. C’est-à-dire que l’on enregistre en premier l’élément à la case {0,0}, ensuite la case {0,1} et pour finir à la dernière case {m,n} où m représente le nombre de ligne et n le nombre de colonne. S’il n’y a aucun élément dans la case alors on écrit des 0 pour le type et -1 pour la spécificité.

Avec cette représentation, il est maintenant facile d’aller lire ou écrire un élément spécifique pour le modifier ou le supprimer sans avoir à réécrire tout le niveau dans le fichier, ce qui permet de limiter le nombre d’appel à « read » et « write ».

1. Le fonctionnement de l’éditeur

Pour lancer l’éditeur il suffit d’utiliser la commande « ./bin/editor [nom du fichier monde] » dans un terminal. Si le fichier existe alors l’éditeur l’ouvre sinon il le crée avec un premier niveau vide. Notre fichier monde contient alors une table d’adresse avec une adresse vers le premier niveau suivit d’une table de vide qui est vide et enfin le premier niveau entièrement vide. A partie de là l’édition du monde peut commencer.

Tout d’abord, il va falloir se familiariser avec l’interface utilisateur. Dans la fenêtre principal (la plus grande) il y a la représentation du niveau actuel, dans la fenêtre secondaire qui se situe à la droite de la fenêtre principal ce sont les différents outils permettant d’éditer le monde, et enfin dans la dernière fenêtre située tout en bas nous avons les différentes informations données par l’éditeur.

Dans la fenêtre des outils il est possible de sélectionner des spécificités seulement pour les portes, gates et clés, il s’agira de la couleur pour les deux derniers éléments leur permettant de fonctionner ensemble. Une fois l’outil s’électionné il suffit de cliquer dans le niveau à l’endroit où l’on souhaite placer l’élément. Si des sprites sont déjà présents sur la zone occupée par le nouvel élément alors les supprime.

En ce qui concerne la sauvegarde, celle-ci s’effectue automatiquement dès qu’il y a un changement sur le niveau. De plus, la modification ne se fait que sur les éléments qui ont été modifié permettant de limiter les appels système.

1. Utilisation du protocole UDP dans les applications client-serveur

Au sein de ce projet, le serveur ainsi que le client occupent une place primordiale, en effet, ce sont ces derniers qui gèrent la gestion et le bon fonctionnement de la partie.

1. Lancement des applicatifs

Afin de lancer le serveur il suffit d’utiliser la commande « bin/server [n° de port] » dans un terminal. Dans un premier temps, le serveur effectue toutes ses communications avec le client via le protocole UDP. C’est pourquoi au lancement, ce dernier crée une socket UDP avec le numéro de port spécifié en paramètre lors du lancement de celui-ci. Une fois la socket UDP mise en place et fonctionnel, le serveur se met alors en écoute des diverses requêtes qu’il peut recevoir par le client.

En ce qui concerne le client, pour le démarrer, nous devons exécuter la commande suivante : "bin/client [adresse IP du serveur] [numéro de port du serveur]". Si les informations d'adresse IP et de port du serveur sont correctement renseignées et correspondent à ceux du serveur démarré, le client sera alors lancé.

1. Communication entre les deux applications

Afin de communiquer entre eux, le client et le serveur vont utiliser deux protocoles : UDP et TCP. L’usage du protocole UDP se fait dans le contexte du paramétrage des clients ainsi que des parties crées et qui attendent d’être lancés.

Afin de standardiser l’échange entre les deux entités, nous avons mis en place deux structures : *request\_client\_udp\_t* et *response\_server\_udp\_t*. La première est utilisé lors de l’envoi d’une requête depuis le client vers le serveur tandis que la deuxième est utilisé pour le chemin inverse, soit l’envoie de la réponse à la requête reçu.

La structure *request\_client\_udp\_t* est formatée de la manière suivante :

Une image contenant texte

Description générée automatiquement

Figure : Structure request\_client\_udp\_t

Cette structure comporte un élément de type union, ce qui évite la création de multiples structures pour chaque requête. En utilisant cette approche, nous réduisons considérablement la surcharge inutile de notre requête. Si nous avions opté pour une structure standard à la place de l'union, cela aurait engendré une charge supplémentaire à nos requêtes, étant donné qu’a requête toutes les champs devait être rempli.

C'est pourquoi notre variable *content\_request\_client\_udp\_t* est formatée comme suit :

Une image contenant texte

Description générée automatiquement

Figure : Union content\_request\_client\_udp\_t

Quant à la réponse du serveur aux différentes requête du client, nous retrouvons la même architecture que *request\_client\_udp\_t* dans *response\_server\_udp\_t*. Cependant, le contenu diffère, en effet, nous avons l’union suivante au sein de cette dernière :

Une image contenant texte

Description générée automatiquement

Figure : Union content\_response\_server\_t

Au sein de cette union, nous retrouvons trois variables de type entier, ainsi qu'une variable structurée contenant la liste des mondes existants, et enfin une dernière variable structurée contenant la liste des parties en cours.

Et en ce qui concerne le type de la requête, au sein de notre fichier contenant les structures précédemment évoqué, nous avons créé une liste de constantes qui nous permet d'identifier la requête reçue, ainsi nous pouvons déterminer le traitement à effectuer par la suite.

1. La gestion des parties et des joueurs

La gestion des parties et des clients se fait principalement au sein du serveur. En effet, c’est ce dernier qui met en lien toutes les parties de jeu et les clients qui ont lancé l’application.

1. La gestion des parties jeux

Afin de gérer les différentes parties de jeux qui peuvent être lancée sur le serveur, nous avons décidé d’opter pour une liste chainée contenant une structure représentant les différents paramètres d’une partie de jeu. Cette liste chainée est alloué dès le lancement du serveur et à chaque partie créée, celle-ci est directement enregistrement au sein de la structure de donnée.

Ainsi, une partie de jeu est représentée par la structure suivante :

Une image contenant texte, écran, capture d’écran, foule

Description générée automatiquement

Figure : Structure représentant une partie

Nous trouvons dans cette structure un ensemble d'informations essentielles : un identifiant unique pour la partie, le nombre de participants finaux attendus, le nombre actuel de participants, qui est incrémenté dès qu'un joueur rejoint la partie, le nom du monde sélectionné, ainsi qu'une liste chaînée de clients représentant les joueurs présents dans la partie en question.

1. La gestion des clients

Pour assurer la gestion des différents clients se connectant au serveur, comme pour les parties de jeux, nous avons également mis en place une liste chainée contenant une structure. Cette structure est composé de différents éléments utiles à l’administration des différents clients. Tout comme les partie de jeux, cette structure de donnée est alloué dès le lancement du serveur. Et dès lors qu’un client se connecte, il est automatiquement ajouté dans cette dernière.

Ainsi, la liste des clients connectés au serveur sont décrits par la structure suivante :

Une image contenant texte, tableau

Description générée automatiquement

Figure : Structure représentant un client

Dans cette structure, nous pouvons identifier plusieurs éléments : tout d'abord, un identifiant unique qui permet d'identifier chaque client. Nous trouvons également le pseudo associé à ce client ainsi que son adresse IP et le port correspondant, ce qui nous permettra ultérieurement de lui envoyer le numéro du port de la socket TCP lorsque la partie sera lancée. Enfin, les pointeurs *\*next* et *\*prev* permettent la mise en place d'une liste chaînée, facilitant ainsi la gestion de ces informations.

Cette structure est utilisé par les deux parties, en effet, le serveur et le client ont recours à cette dernière afin de gérer les informations du joueur.

1. Les différents échanges entre le serveur et le client

Durant cette partie, nous allons voir les différentes communications entre le serveur et le client effectuée par le biais du protocole UDP, protocole permettant des échanges fluides et rapides entre deux entités.

1. Première connexion d’un joueur au serveur

Maintenant que nos deux applications sont en cours d'exécution et que les formats des requêtes et les différentes structures de gestion ont été définis, la première étape pour le client sera de saisir le pseudonyme qu'il souhaite utiliser tout au long de la partie.

Après avoir saisi le pseudo, une première communication UDP est établie avec le serveur. En effet, le client envoie une requête soigneusement composée, incluant le type de requête ainsi que le pseudo choisi. A la réception de cette requête, le serveur répond à cette dernière en envoyant l’id généré au sein de son système de gestion des clients et enregistre le client au sein de la structure adéquate, structure que nous avons évoqué auparavant.

Nous avons donc le diagramme de séquence suivant, représentant la communication précédemment décrite :

Une image contenant texte

Description générée automatiquement

Figure : Diagramme de séquence correspondant à l'enregistrement du client au sein du server

1. Récupération du nombre de joueurs actuels

Une fois que le joueur a saisi son pseudonyme et que toutes les étapes de gestion ont été effectuées, un menu s'affiche côté client. Ce menu permet au joueur de créer une partie, de rejoindre une partie existante ou encore de connaître le nombre de joueurs actuellement connectés sur Starlyze. Pour obtenir cette information, une communication avec le serveur est alors nécessaire. À cet effet, en utilisant la même socket de communication que celle initiée lors de la première connexion, le client envoie une demande de récupération du nombre de joueurs actuels. Le serveur répond alors en envoyant le nombre correspondant au client.

Nous avons donc le diagramme de séquence suivant, qui représente cette communication :

Une image contenant texte, lettre

Description générée automatiquement

Figure : Diagramme de séquence correspondant à la récupération du nombre de joueurs connectés sur Starlyze

Suite à cette réponse, le client peut alors afficher le nombre de joueurs connectés sur Starlyze au sein du menu, comme nous pouvons le voir ci-dessous :

Une image contenant texte

Description générée automatiquement

Figure : Menu coté client

1. Créer une partie de Starlyze

Comme nous pouvons l'observer sur la figure n°8, le menu proposé côté client permet différentes options, dont celle de créer une partie de Starlyze. Si le joueur sélectionne cette option, cela déclenche une série de communications avec le serveur.

En effet, dans un premier temps, pour lancer une partie, il est nécessaire de récupérer les différents mondes disponibles dans le jeu. Pour ce faire, en utilisant la même socket UDP que celle utilisée lors de la première connexion, le client envoie une première requête au serveur afin signaler sa demande de récupération de la liste des mondes disponibles. Le serveur ayant chargée la liste des mondes au sein d’une liste chainée lors de son lancement renvoie celle-ci au client intéressé. Une fois cette liste récupérée, le client l'affiche alors à l'utilisateur.

Puis dans un second temps, une fois que le joueur a sélectionné le monde auquel il souhaite jouer et précisé le nombre de joueurs nécessaires pour que la partie soit lancée, une deuxième communication est établie avec le serveur. Cette communication permet d'envoyer au serveur les différentes informations concernant la partie qui vient d'être créée, afin qu'il puisse l'enregistrer au sein du système de gestion des parties. Cette étape est essentielle pour permettre à d'autres joueurs de rejoindre la partie nouvellement créée.

Ainsi, afin de créer une partie, nous avons donc les échanges suivants qui sont effectués :

Une image contenant texte

Description générée automatiquement

Figure : Diagramme de séquence correspondant à la création d’une partie de Starlyze

1. Rejoindre une partie créée

Hormis le fait qu’un joueur peut créer une partie, il peut également en rejoindre une déjà existante. Tout comme la création d’une partie, ce choix entraine une série de communication avec le serveur.

Dans un premier temps, afin de rejoindre une partie, il est nécessaire d’obtenir la liste des parties en attente et qui ne sont pas encore démarrées. C’est pour cela que le client envoie une première requête afin de pouvoir de récupérer la liste des parties en attente. Le serveur ayant une liste de toutes les parties en attente renvoie celle-ci, afin que le client puisse l’afficher.

Une fois que la liste des parties en attente s'affiche, le joueur peut alors effectuer sa sélection. À cet instant précis, une seconde requête est envoyée par le client au serveur, permettant ainsi de communiquer le choix du joueur quant à la partie qu'il souhaite rejoindre. Au sein cette requête, qui inclut non seulement l'identifiant de la partie sélectionnée, mais également l'identifiant du joueur lui-même. Cette information permet au serveur d’ajouter plus facilement le joueur sans qu’il ai a envoyé toutes ces informations telles que son pseudo, son adresse IP ou encore son port. En effet, grâce à l'identifiant fourni, le serveur peut directement récupérer le joueur concerné dans son système de gestion de client.

Suite à cela, le serveur effectue une vérification cruciale afin de s'assurer que le nombre de joueurs requis pour la partie est atteint. Si tel est le cas, le serveur crée une socket TCP et envoie le port TCP correspondant à tous les joueurs. Cette étape est essentielle pour assurer une connectivité stable et fluide pendant toute la durée de la partie ainsi que le début de la partie. Nous aborderons cette étape en détails dans les prochaines sections de ce rapport.

Ainsi, afin de rejoindre une partie, nous avons donc la série d’échange suivante qui sont effectués :

Une image contenant texte

Description générée automatiquement

Figure : Diagramme de séquence correspondant à la connexion d'un joueur à une partie

Il est important de noter que le diagramme illustre clairement que, lors de la réponse du serveur pour l'envoi de la liste des parties en attente, le type de requête peut prendre deux valeurs différentes. Si la valeur est SERVER\_SEND\_LIST\_GAMES, cela signifie qu'il y a effectivement des parties en attente et que le joueur peut en choisir une pour y participer. En revanche, si la valeur est SERVER\_SEND\_NO\_GAMES, cela indique qu'il n'y a aucune partie en attente, et que le joueur devra donc créer une partie pour pouvoir jouer à Starlyze.

1. Envoie du port de la socket TCP

Une fois qu’une partie a atteint le nombre de joueurs requis, nous créons donc une socket TCP au sein du serveur afin que celle-ci servent au déroulement de la partie. En effet, le déroulement de la partie se fait exclusivement en TCP. Nous ne rentrerons pas dans les détails de cela au sein de cette sous-partie, nous aborderons ce point ultérieurement au sein de ce rapport.

Afin que le client se connecte à la socket TCP, nous envoyons le port de notre socket via UDP vers le client. En ce qui concerne l’adresse IP, il utilisera la même que celle qu’il a utilisé pour UDP. Ce port a été généré automatiquement par le système d’exploitation. Une fois le numéro de port reçu, le client crée alors une socket TCP de son coté. Ainsi, à partir de cette étape, tous les échanges entre le serveur et le client se feront exclusivement en TCP.

Nous avons donc le diagramme de séquence suivant :

Une image contenant texte

Description générée automatiquement

Figure : Diagramme de séquence représentant l'envoie du port TCP

1. Déconnexion du client

Enfin, une fois qu'un joueur a créé ou rejoint une partie et qu'elle s'est terminée, ou bien si le joueur décide d'arrêter le jeu avant même d'avoir lancé une partie, il est essentiel de le déconnecter du serveur. Cette étape est importante pour garantir que le joueur ne soit plus comptabilisé dans la liste des joueurs connectés, évitant ainsi une surcharge du serveur. Cette fonctionnalité de déconnexion assure également une gestion efficace des ressources du serveur et une meilleure expérience de jeu pour tous les utilisateurs.

Dès qu'un joueur effectue l'une des actions mentionnées précédemment, la procédure de déconnexion est lancée. Pour cela, le client envoie une requête au serveur pour signaler qu'il souhaite se déconnecter, en incluant son identifiant dans la requête. Lorsque le serveur reçoit cette requête, il entame alors la procédure de déconnexion, qui consiste à supprimer toutes les informations relatives au joueur déconnecté qu'il avait sauvegardées jusqu'à présent.

Nous avons donc le diagramme de séquence suivant, représentant la communication précédemment décrite :

Une image contenant texte

Description générée automatiquement

Figure : Diagramme de séquence représentant la déconnexion d'un client

1. Les différents échanges entre le serveur et le client via le protocole TCP