



STARLYZE

INFO 601/602 – Jeu multijoueur en
réseau

TABLE DES MATIERES

1. INTRODUCTION.....	2
2. EDETEUR DE MONDE	2
A. LA REPRESENTATION D'UN MONDE DANS L'EDITEUR.....	2
B. LE FONCTIONNEMENT DE L'EDITEUR.....	4
3. LES APPLICATIONS CLIENT ET SERVEUR	4
A. LANCEMENT DES APPLICATIFS	4
B. COMMUNICATION ENTRE LES DEUX APPLICATIONS	5
C. PREMIERE CONNEXION D'UN CLIENT AU SEIN DU SERVEUR	6
4. L'ASPECT CLIENT.....	6
5. ORGANISATION DU TRAVAIL.....	6

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.

2. 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.

A. 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'écrit 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 sauvegardés.

2- 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.

3- 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 réservée 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 suivi 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 ».

B. Le fonctionnement de l'éditeur

Pour lancer l'éditeur il suffit d'utiliser la commande « ./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 suivi d'une table de vide qui est vide et enfin le premier niveau entièrement vide. A partir 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 sélectionné 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.

3. Les applications client et 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.

A. Lancement des applicatifs

Afin de lancer le serveur il suffit d'utiliser la commande « ./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 : « ./client [adresse IP du serveur] [numéro de port du serveur] ». Si les informations

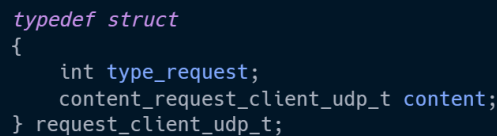
d'adresse IP et de port du serveur sont correctement renseignées et correspondent à ceux du serveur démarré, le client sera alors lancé.

B. 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 créées et qui attendent d'être lancées.

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 utilisée lors de l'envoi d'une requête depuis le client vers le serveur tandis que la deuxième est utilisée pour le chemin inverse, soit l'envoi de la réponse à la requête reçue.

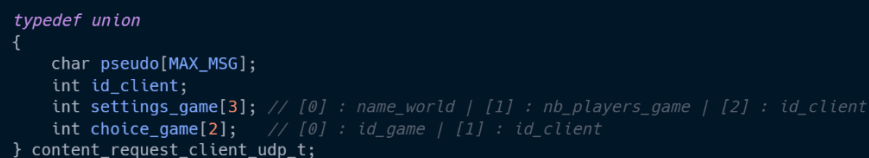
La structure *request_client_udp_t* est formatée de la manière suivante :



```
typedef struct
{
    int type_request;
    content_request_client_udp_t content;
} request_client_udp_t;
```

Figure 1 : 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'à requête toutes les champs devaient être remplis. C'est pourquoi notre variable *content_request_client_udp_t* est formatée comme suit :



```
typedef union
{
    char pseudo[MAX_MSG];
    int id_client;
    int settings_game[3]; // [0] : name_world | [1] : nb_players_game | [2] : id_client
    int choice_game[2]; // [0] : id_game | [1] : id_client
} content_request_client_udp_t;
```

Figure 2 : Union *content_request_client_udp_t*

Quant à la réponse du serveur aux différentes requêtes 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 :



```
typedef union
{
    int nb_clients;
    int id_clients;
    int port_tcp;
    list_world_response_t list_world;
    list_game_without_pointers_t list_game;
} content_response_server_t;
```

Figure 3 : 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.

C. Première connexion d'un client au sein du serveur

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

4. L'aspect client

5. Organisation du travail