GHOSTLAB
Projet de Programmation Réseaux

Licence Informatique - Université Paris Cité

2022

Attention : Prenez le temps de lire attentivement TOUT le document et ce dans ses moindres détails. Relisez plusieurs fois, même en groupe. Ne vous lancez pas immédiatement dans la réalisation à moins que vous ne soyez déjà à l'aise avec le développement réseau et la communication inter-applications. Faites donc des diagrammes, imaginez des scénarios de communication entre entités, essayez d'en analyser la portée, les problèmes, les informations dont les entités ont besoin au cours de leur vie, etc. Essayez d'abord de vous abstraire des problèmes techniques.

Note : Dans le document le signe ␣ représentera un simple caractère d'espacement (ASCII 32). De plus les messages circulant sont indiqués entre crochets, **les crochets ne faisant pas partie du message**. À la fin du document le format des parties constituant les messages est décrit de façon précise.

Introduction

Le but de ce projet est de programmer un serveur pour le jeu GHOSTLAB, ainsi que des clients. Le principe du jeu est le suivant : il y a un labyrinthe dans lequel les joueurs peuvent se déplacer et où se déplacent également des fantômes. Le but de chacun des joueurs est d'attraper le plus de fantômes possibles. Deux difficultés viennent s'ajouter au jeu : tout d'abord les fantômes sont invisibles et on ne les 'entend' que lorsqu'ils se déplacent et ensuite les joueurs ne savent pas comment est fait le labyrinthe, ils connaissent juste leur position et peuvent savoir si un déplacement est possible dans l'une des quatres directions (haut, bas, droite, gauche).

Afin de programmer ce jeu, le protocole 'basique' de communication entre les différentes entités vous est fourni.

Description détaillée du protocole

Le serveur de jeu

Les caractéristiques d'un serveur de jeu sont les suivantes :

— Un port d'écoute TCP pour communiquer avec les joueurs individuels

Les joueurs

Les caractéristiques d'un joueur sont les suivantes :

— Un identifiant faisant exactement 8 caractères alphanumériques
— Un port UDP pour recevoir les messages personnels des autres joueurs

Interaction joueur-serveur de jeu

Avant de commencer une partie.

Lorsqu'un joueur se connecte au serveur de jeu, celui-ci lui indique la liste des parties qui risquent de commencer (c'est à dire les parties pour lesquels des joueurs se sont inscrits mais qui ne sont pas encore commencées). Ainsi un joueur ne peut pas rejoindre une partie en cours et il ne peut être inscrit qu'à au plus un partie.

Dès qu'il se connecte, le joueur reçoit du serveur un message [GAMES␣n***] où n indique le nombre de parties avec des joueurs déjà inscrits et non commencées (ce nombre peut valoir 0). Le serveur envoie ensuite n messages de la forme [OGAME␣m␣s***] où m indique le numéro de la partie et s indique le nombre de joueurs inscrits pour cette partie. Le joueur peut alors envoyer un message de la forme [NEWPL␣id␣port***] pour dire qu'il souhaite créer une nouvelle partie, ou alors [REGIS␣id␣port␣m***] pour dire qu'il souhaite rejoindre la partie de numéro m. Dans ces deux requêtes id représente l'identifiant du joueur et port son port UDP. Le serveur lui répond alors soit [REGOK␣m***] pour lui dire qu'il est inscrit sur la partie m (dans le cas de création d'une partie ce numéro sera un *nouveau* numéro) soit [REGNO***] pour lui signaler que son inscription à une partie (nouvelle ou non) n'a pas été acceptée. Une fois inscrit à une partie le joueur peut envoyer quand il pense que la partie peut commencer un message [START***]. La partie ne commencera que losrque tous les joueurs inscrits à une partie auront envoyé leur message [START***].

Avant qu'une partie commence, si un joueur n'est pas inscrit à une partie ou s'il n'a pas encore envoyé un message [START***], il a la possibilité de faire les requêtes suivantes :

- Se désinscrire de la partie où il est inscrit en envoyant le message [UNREG***] et alors le serveur lui envoie le message [UNROK␣m***] où m est le numéro de la partie pour laquelle le joueur était inscrit (si le joueur n'est pas inscrit à une partie le serveur lui répond [DUNNO***]).
- Demander la taille du labyrinthe d'une partie (cf plus loin pour la description de la partie) en envoyant le message [SIZE?␣m***] où m est le numéro de la partie. Le serveur répond alors par [SIZE!␣m␣h␣w***] où m est le numéro de la partie, h la hauteur du labyrinthe et w sa largeur. S'il n'y a pas de partie m, le serveur répond [DUNNO***].
- Demander la liste des joueurs inscrits à une partie. Pour cela il envoie le message [LIST?␣m***] où m est le numéro de la partie. Ce à quoi le serveur répond [LIST!␣m␣s***] où s est le nombre de joueurs inscrits à la partie suivi de s messages de la forme [PLAYR␣id***] où id est l'identité d'un joueur inscrit. S'il n'y a pas de partie m, le serveur répond [DUNNO***].
- Demander la liste des parties pour lesquelles des joueurs se sont inscrits et qui ne sont pas encore commencées. Pour cela il envoie le message [GAME?***]. Ce à quoi le serveur répond[GAMES␣n***] où n indique le nombre de parties avec des joueurs déjà inscrits et non commencée (ce nombre peut valoir 0). Le serveur envoie ensuite n messages de la forme [OGAME␣m␣s***] où m est entier indiquant le numéro de la partie et s indique le nombre de joueurs inscrits pour cette partie.

Quand un joueur a envoyé le message [START***], il est bloqué en attente que la partie commence et ne peut plus interroger le serveur. À vous aussi de juger ce qu'il advient d'un joueur qui envoie [START***] sans s'être au préalable inscrit pour une partie.

Le début d'une partie

Tout d'abord à chaque partie est associé un labyrinthe. Il s'agit d'une plate-forme rectangulaire ayant des cases avec des murs et des cases sans mur et qui de plus doit être faite de telle sorte que depuis toute case sans mur il existe un chemin (où l'on ne traverse pas les murs) pour accéder à n'importe quelle autre case sans mur. Par exemple, sur la Figure 1, la plate-forme de gauche ne représente pas un labyrinthe car depuis la case $(3,1)$ sans mur, on ne peut pas atteindre la case $(3,3)$ (on notera toujours en premier le numéro de ligne puis le numéro de colonne), en revanche la plate-forme de droite représente un labyrinthe correct. On suppose que tout le tour du labyrinthe est entouré d'un mur, c'est-à-dire que l'on ne peut pas sortir d'un

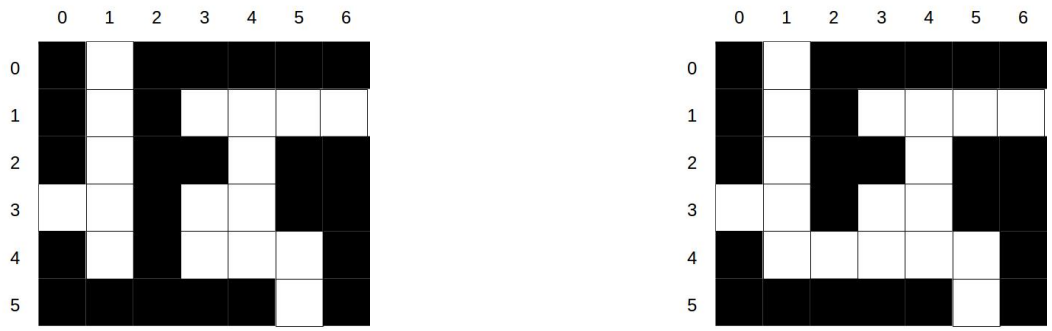

FIGURE 1 – Un labyrinthe mauvais et un labyrinthe correct

labyrinthe. La hauteur d'un labyrinthe est son nombre de lignes et sa largeur est son nombre de colonnes. Ainsi les deux labyrinthes de la Figure 1 ont pour hauteur 6 et pour largeur 7.

De plus, on associe à chaque partie une adresse IP et un port de multi-diffusion que le serveur utilisera pour communiquer des informations sur la partie, en particulier le mouvement des fantômes et les captures des fantômes.

Dans chaque partie on a aussi un nombre de fantômes qui sont placés au début au hasard sur les cases sans mur du labyrinthe.

Quand tous les joueurs inscrits à une partie ont envoyé leur message [`START***`], le serveur envoie à chacun de ses joueurs, un premier message [`WELCO␣m␣h␣w␣f␣ip␣port***`] où `m` est le numéro de la partie, `h` est la hauteur du labyrinthe (dans notre exemple `h` vaut 6), `w` est la largeur du labyrinthe (dans notre exemple `w` vaut 7), `f` est le nombre de fantômes présents dans cette partie, `ip` est l'adresse IP de multi-diffusion de la partie et `port` est son port de multi-diffusion. Puis elle envoie à chaque joueur un message [`POSIT␣id␣x␣y***`] où `id` est l'identifiant du joueur à qui le message est envoyé, (`x`,`y`) est sa position (`x` est le numéro de ligne strictement inférieur à la hauteur et `y` est le numéro de colonne strictement inférieur à la largeur). La position de chaque joueur est choisie par la serveur, les joueurs ne savent pas au début où se trouvent les autres joueurs, ni où sont les murs ni les fantômes.

Le déroulement d'une partie

La partie s'arrête quand il n'y a plus de fantômes ou quand tous les joueurs inscrits ont quitté la partie. Le but de chacun des joueurs est d'accumuler le plus de points possibles. Au début chaque joueur a 0 point. La façon dont les points sont augmentés est laissée libre sachant que l'on ne peut gagner des points qu'en capturant des fantômes.

Chaque joueur peut alors faire des requêtes de déplacement dans la labyrinthe en envoyant des messages de la forme [`UPMOV␣d***`],[`DOMOV␣d***`], [`LEMOV␣d***`] et [`RIMOV␣d***`] où le début du message indique la direction à prendre et `d` est la distance que le joueur souhaite parcourir. Le serveur fait alors bouger le joueur dans la direction correspondante du nombre maximum de cases inférieur à `d` sans rencontrer de murs. Il lui renvoie ensuite un message [`MOVE!␣x␣y***`] où (`x`,`y`) correspond à la nouvelle position du joueur si le joueur n'a pas croisé de fantômes sur son chemin et si le joueur a croisé un fantôme sur son chemin, le fantôme disparaît et le serveur envoie le message [`MOVEF␣x␣y␣p***`] où (`x`,`y`) correspond à la nouvelle position du joueur et `p` à son nouveau nombre de points. Par exemple, sur le labyrinthe de droite de la Figure 1, si le joueur est à la position $(3,0)$ et qu'il fait la requête pour se déplacer de 4 cases vers la droite, c'est-à-dire qu'il envoie le message [`RIMOV␣004***`] au serveur, alors si il ne croise pas de fantômes, le serveur lui répondra [`MOVE!␣003␣001***`]. Le joueur sait ainsi qu'il y a un mur à la position $(3,2)$.

Un joueur peut abandonner une partie en faisant [`IQUIT***`], dans ce cas là le serveur lui renvoie le message [`GOBYE***`] et il ferme sa connexion.

Au cours de la partie, un joueur peut aussi demander la liste des joueurs présents dans la partie en envoyant le message [GLIS?***]. Ce à quoi le serveur répond [GLIS!␣s***] où s est le nombre de joueurs présents dans la partie suivi de s messages de la forme [GPLYR␣id␣x␣y␣p***] où id est l'identité d'un joueur présent, (x,y) indique sa position et p son nombre de points.

Au cours de la partie un joueur peut envoyer un message à tous les joueurs en envoyant [MALL?␣mess***] où mess est le message à envoyer. Le message sera multi-diffusé sur le port et l'adresse IP de multi-diffusion de la partie. Le serveur répond alors au joueur [MALL!***].

Au cours de la partie un joueur peut aussi envoyer un message à un joueur en envoyant [SEND?␣id␣mess***] où id est l'identité du joueur à qui s'adresse le message et mess est le message à envoyer. Le message sera diffusé sur le port UDP et l'adresse ip du joueur correspondant. Le serveur répond alors [SEND!***] si l'envoi est possible ou [NSEND***] si l'envoi n'est pas possible (par exemple si le joueur id est inconnu ou n'appartient pas à la partie).

Si le joueur envoie un de ces messages alors que la partie est finie, la partie lui répond [GOBYE***] et ferme la connexion.

Les messages multi-diffusés

La partie multi-diffuse sur son adresse IP et port de multi-diffusion des informations sur le déplacement des fantômes, les messages envoyés par les autres utilisateurs et les points acquis par les utilisateurs.

Quand un fantôme non pris se déplace, la partie multi-diffuse le message [GHOST␣x␣y+++] où (x,y) indique la position d'arrivée du fantôme. Aucune règle n'est imposée sur la façon dont les fantômes se déplacent, il doivent juste toujours terminer leur déplacement sur une case sans mur.

Quand un joueur prend un fantôme, la partie multi-diffuse le message [SCORE␣id␣p␣x␣y+++] où *id* est l'identifiant du joueur qui a pris le fantôme, p son nombre de points et (x,y) indique la position du fantôme au moment où il est pris.

Quand un joueur envoie un message mess à tout le monde, le serveur multi-diffuse le message [MESSA␣id␣mess+++] où *id* est l'identifiant du joueur qui a émis le message.

Quand la partie est finie car il n'y a plus de fantômes, le serveur multi-diffuse le message [ENDGA␣id␣p+++] où id est l'identifiant du joueur gagnant et p son score. En cas de score égal, à vous de décider qui est le gagnant.

Les messages UDP

Lorsqu'un joueur demande d'envoyer un message à un autre joueur avec le message [SEND?␣id␣mess***], si cela est possible, la partie envoie sur le port UDP du joueur id le message UDP de la forme suivante [MESSP␣id2␣mess+++] où id2 est l'identifiant du joueur qui a fait la demande d'envoi.

Spécification des messages TCP

Comme vous avez pu le remarquer, les premiers octets d'un message TCP serviront à encoder dans une chaîne de caractères le type de message. Les types des messages étant : GAMES, OGAME, NEWPL, REGIS, REGOK, REGNO, START, UNREG, UNROK, DUNNO, SIZE?, SIZE!, LIST?, LIST!, PLAYR, GAME? WELCO, POSIT, UPMOV,DOMOV, LEMOV, RIMOV, MOVE!, MOVEF, IQUIT, GOBYE, GLIS?, GLIS!, GPLYR, MALL?, MALL!, SEND?, SEND! et NSEND.

Chaque message TCP finit de plus par trois octets contenant le codage en chaîne de caractères du signe arithmétique pour la multiplication (ASCII 42). **On rappelle les crochets [] ne font pas partie des message et les messages ne se terminent pas par \n.**

Voilà maintenant les caractéristiques pour chacun des champs contenus dans les messages :

— `n` est codé sur 1 octet et contient le nombre de parties.
— `m` est codé sur 1 octet et contient le numéro de la partie.
— `s` est codé sur 1 octets et contient le nombre d'inscrits à une partie .
— `id` représente l'identifiant d'un utilisateur. Il s'agit d'une chaîne de caractères formée d'exactement 8 caractères alpha numériques.
— `port` représente un numéro de port. Il s'agit d'une chaîne de caractères formée de 4 caractères numériques.
— `h` est codé sur 2 octets en little endian et contient la hauteur d'un labyrinthe (qui est forcément inférieure à 1000).
— `w` est codé sur 2 octets en little endian et contient la largeur d'un labyrinthe (qui est forcément inférieure à 1000)..
— `f` est codé sur 1 octet et contient le nombre de fantômes.
— `ip` est codé sur 15 octets et contient une chaîne de caractères correspondant à une adresse IPv4 complétée par des # (si sa taille est plus petite que 15). Par exemple, pour l'adresse 225.10.12.4, la chaîne de caractères correspondante sera `225.10.12.4####`.
— `x` est un numéro de ligne codé sur trois octets. Il correspond à une chaîne de caractères représentant le numéro et complété par des 0. Par exemple pour le numéro de ligne 11, on utilisera la chaîne de caractères `011` .
— `y` est un numéro de colonnes codé sur trois octets. Il correspond à une chaîne de caractères représentant le numéro et complété par des 0. Par exemple pour le numéro de colonnes 5, on utilisera la chaîne de caractères `005`.
— `d` est une distance codée sur trois octets. Il correspond à une chaîne de caractères représentant le numéro et complété par des 0. Par exemple pour la distance 55, on utilisera la chaîne de caractères `055`.
— `p` est un nombre de points codé sur quatre octets. Il correspond à une chaîne de caractères représentant le numéro et complété par des 0. Par exemple pour le nombre de points 789, on utilisera la chaîne de caractères `0789`.
— `mess` contient une chaîne de caractères d'au plus 200 caractères, et qui ne contient pas les chaines `***` ou `+++`.

Spécification des messages UDP

Les premiers octets d'un message UDP serviront à encoder dans une chaîne de caractères le type de message. Les types des messages étant : `FANT`, `SCORE`, `MESSA`, `ENDGA` et `MESSP`.

Les caractéristiques pour chacun des champs contenus dans les messages sont les mêmes que pour les messages TCP.

Chaque message UDP finit de plus par trois octets contenant le codage en chaîne de caractères du signe arithmétique pour l'addition (ASCII 43). **On rappelle les crochets [] ne font pas partie des message et les messages ne se terminent pas par `\n`.**

Réalisation

La réalisation se fera nécessairement en C ou en Java, et possiblement dans les deux (ce qui serait un très bon exercice et un point pris en compte lors de l'évaluation). Un groupe devra programmer au moins un client et un serveur.

Il est impératif de respecter scrupuleusement la spécification fournie dans le sujet ainsi que les formats de message. Toute violation sera jugée très défavorablement !

Il est vivement recommandé de proposer des extensions du projet, par exemple en proposant des nouvelles fonctionnalités, comme un mode triche, une façon de regrouper des joueurs par équipe, la capture de

joueurs, etc. Sur ces points **à vous de jouer** ! Ces fonctionnalités devront être décrites de façon précise au moment du rendu du projet et l'ajout de fonctionnalités ne doit pas avoir d'impact sur le protocole basique (c'est-à-dire que vous ne pouvez pas changer les messages de base et que votre serveur doit être capable d'interagir avec un client respectant ce protocole).

Nous mettrons en place un forum sur Discord pour que vous puissiez communiquer entre vous et avec nous, par exemple pour faire part d'imprécisions dans le sujet, ou pour signaler que vous avez un serveur qui tourne quelque part et ainsi donner l'opportunité à vos collègues de se connecter à elle. Ou encore pour préciser vos extensions. **Toute demande sur le projet devra passer par le forum.** Prenez garde à ne pas vous fier à la rumeur, la seule source d'information fiable sera le forum sur Discord ! Au moindre doute, n'hésitez pas à y poster votre question !

La communication verbale entre groupes est non seulement autorisée mais encouragée, cependant il est **strictement interdit** d'échanger du code ; ceci serait considéré comme plagiat et par conséquent jugé sévèrement. Les discussions doivent donc seulement porter sur le fonctionnement du protocole et son interprétation, ou les formats des messages ; il vaut donc mieux éviter de donner trop d'indications sur la façon de coder les fonctionnalités.

La **notation finale** prendra en compte le fait que des groupes auront réussi à faire communiquer leurs applications entre elles. Il faudra donc penser à en faire la démonstration. D'une certaine manière, ceux qui collaborent dans les limites indiquées ci-dessus seront récompensés ; pour ceux qui décident de faire quelque chose dans leur coin et qui ressemblerait vaguement à ce qui est décrit, ce sera l'inverse, la notation sera revue à la baisse. Un **programme qui s'exécute n'est pas suffisant** ! Vous **devez** écrire un programme qui se comporte comme indiqué ; et s'il ne communique pas avec le programme écrit par d'autres, c'est que vous n'avez pas compris ce qu'est la programmation réseaux.

Pour la soutenance, il sera **nécessaire de prévoir un mode de fonctionnement verbeux** dans lequel suffisamment d'informations seront disponibles à l'écran pour comprendre ce qui se passe (affichage des messages circulant, etc).

Vos programmes devront nécessairement pouvoir être exécutés sur les machines des salles de TP de l'UFR. Toute solution ne respectant pas ce critère sera jugée invalide.

Votre projet devra bien entendu être robuste (c'est à dire sans erreur) et devra être capable de gérer des messages erronés sans planter.

La réalisation du projet se fera par groupe **d'au moins deux** étudiants et **d'au plus trois** (cette règle ne souffrira aucune exception, un peu d'effort de la part de chacun pour ne pas prendre que des amis proches dans son groupe devrait aider). Et bien entendu, chacun dans un groupe devra travailler et il n'est pas exclu que des étudiants d'un même groupe n'aient pas la même note au final.

La composition des groupes devra être envoyée par mail à `sangnier@irif.fr` avant **le Jeudi 31 Mars 2022 23h59**. Toute personne n'ayant pas soumis de groupe avant cette date prend le risque de ne pas avoir de note au projet.

Des informations sur le rendu (qui aura lieu quelques jours avant la soutenance) et la soutenance (ordre de passage et instructions) seront aussi fournies plus tard. **N**otez seulement que les soutenances auront lieu après les examens).