GIGOUT Thomas - DAUNIQUE Wilfried

Cyril Rabat - Pierre Delisle

Compte rendu projet 3
NE TUEZ PAS LES POILLSONS SIOUPLAIT

SOMMAIRE

SOMMAIRE 2

Introduction 3

Connexion clients - serveur 3

Initialisation de l'affichage côté client 4

- Simulation 4
- Point 5
- Debug 5

Initialisation des poissons 6

Gestion des poissons 6

Lancer les items 7

Pêche 7

Scoring 8

Difficulté rencontrée 8

Implémentation manquante / solutions possibles 9

Conclusion 9

Introduction

Nous souhaitons développer une application de simulation de pêche à la ligne à deux joueurs. L'application se compose de deux parties distinctes.

D'un côté, le serveur : fait le lien entre nos deux joueurs et gère tous les traitements et toutes les requêtes que les clients lui enverront.

De l'autre côté, les clients : ils sont au nombre de deux (nombre de joueurs) par partie et interprètent et affichent aux joueurs toutes les informations et les réponses du serveur à leur requêtes.

Chaque joueur a à sa disposition des "pouvoirs" qu'il pourra débloquer en pêchant des poissons de différentes valeurs. Il peut utiliser un pneu, une dynamite, lâcher un requin...

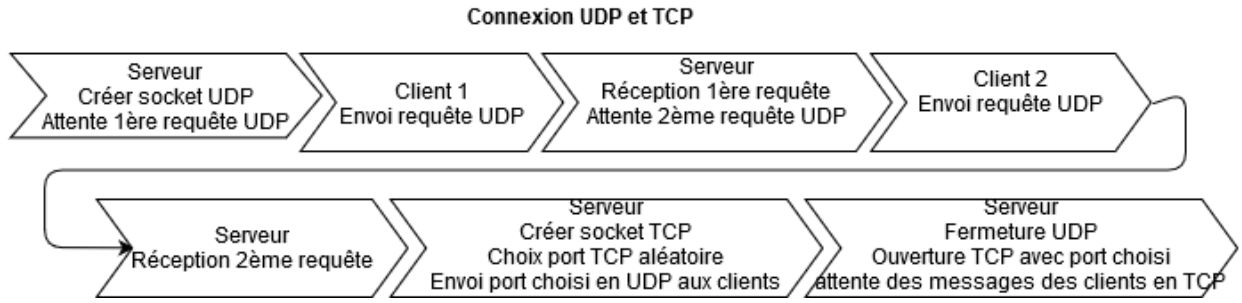
Pêcher un poisson rapporte un point et plusieurs *poireaus*, la monnaie du jeu. La partie s'arrête lorsque l'un des joueurs atteint le score limite (fixé à 15 par défaut).

Étant contre la cruauté animal nous ne tuons pas nos poissons mais les relâchons

SPOILER ALERT:(Tout compte fait peut être que ce détail sera expliquer dans les fonctions manquantes qui sait , SPOILER ALERT):SPOILER ALERT

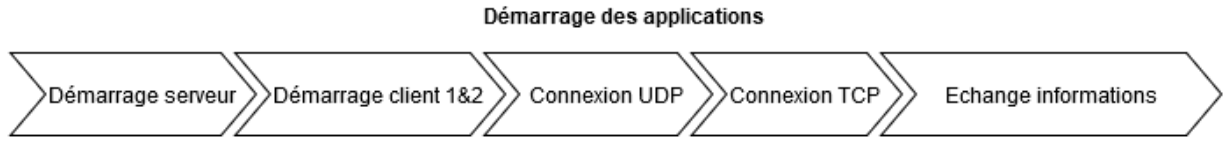
Connexion clients - serveur

Une fois le serveur démarré, il attend que deux clients lui envoient des requêtes de connexion en UDP. Une fois deux "clients" reçus, le serveur ouvre un socket TCP avec un port aléatoire disponible et l'envoie en réponse aux deux clients. Ainsi, les deux client peuvent fermer leur socket UDP et ouvrir un socket TCP avec le port que leur a envoyé précédemment le serveur.



Pour se faire nous utilisons donc certaines structures notamment requete_t et reponse_t qui contiennent à la fois le code de la requête ou de la réponse, le port TCP pour la réponse du serveur et les informations nécessaires au bon déroulement de l'application.

On peut donc résumer ce premier échange par ce diagramme :



Initialisation de l'affichage côté client

Une fois la connexion établie, le serveur envoie une première fois la zone de jeu aux clients par avec un write. Cette grille est affichée par les deux clients et le jeu peut commencer.

A chaque changement de position des poissons le serveur renvoie la grille de jeu.

Lorsque le client envoie un item la grille est ensuite envoyer au serveur qui compare la grille et la renvoie au client.

Il y a 3 fenêtre à l'intérieur de la console :

Simulation

Affiche les poissons et les attributs du client et seulement ceux du client (récupération du client via les sockets et mis dans une classe joueur_t avant le début de la simulation).

La simulation est update via la fonction update() et refresh avec la fonction refresh() ou wrefresh(window);

Elle est affichée via les informations de l'étang qui est une grille_t envoyée par le serveur via la fonction bothsend() qui appelle sendone();

Point

Affiche les informations de la classe joueur_t :

-id :permet de savoir quel est le client et savoir donc quoi afficher sur la simulation.

-point: compte les points du joueurs et cette valeur sert à arrêter la partie.

-poireaus : compte les poireaus équivalent à la valeur du poisson*100.

-outil: sert à sélectionner les différents “pouvoirs” liés à cette magnifique pêche.

les “pouvoirs” sont gérés via la fonction lancerTruc() : au clic elle envoie un “pouvoir”.

Debug

Tout bon développeur a besoin d’une petite fenêtre de debug pour les tests en tout genre.

Gestion du client

Tout d’abord le client envoie périodiquement à chaque action son étang au serveur.

Plusieurs choix sont possibles pour l’utilisateur ou plutôt deux dans cette version bêta de “Ne faites pas mâl au poillsson siouplait”. En effet, il peut au loisir lancer tout ce qu’il veut dans l’étang (“pouvoirs” au clic), ou changer de pouvoirs (avec la flèche du haut ou du bas).

Lorsque qu’un “poillsson” ou poisson passe sur une case autour de notre hameçon celui-ci est bloqué pendant trois secondes avant de refaire sa vie de “poillsson”.

Si le “poillsson” est sur une des quatres cases adjacentes à l’hameçon et que l’on remonte la canne, on gagne les points et les poireaus équivalents au “poillsson” attrapé.

Après vu que nous ne voulons pas faire de mal au “poillsson” ils sont relachés mais vous gardez vos points bien entendu (autrement dit ils ne disparaissent pas...).

Initialisation des poissons

Pour les petits “poillsson” ils sont créés dans une fonction generer_poisson() qui alloue dynamiquement un tableau de poissons et utilise un tableau de threads_poissons pour stocker les pthreads.

Leur valeur est générée par créer poisson et on met la valeur de notre poisson dans l’étang pour son premier “spawn”.

Gestion des poissons

Pour la gestion de ceux-ci il faut prendre en compte plusieurs facteurs :

-lorsqu’un poisson mord à l'hameçon il faut le “stun”. Pour cela j’utilise dans sa routine pthread_cond_timedwait() qui me permet de le stun une durée définie dans un timespec . Cette fonction est compliquée à utiliser et nous a prit pas mal de temps pour comprendre enfin qu’il faut une variable global en plus du while avec la condition de poisson_near() pour stun efficacement les poissons sans les bloquer indéfiniment ou ne pas les bloquer du tout.

-la routine permet les déplacements du poisson via la grille (case_t) et donne aussi l’information à l’étang(grille_t).

-la Fuite: pour la fuite nous étions parti pour la gérer via un attribut de classe dans la fonction poisson_t mais il est géré avec une variable global indiquant un timer motifié à 3 lorsqu'un des clients dépose un hameçon. Ceux-ci calculent alors leur position moins la position de l'hameçon et part dans le sens opposé. (Nous sommes moyennement convaincu par cette partie).

Lancer les items

Le lancé d'item est une fonction qui permet d'envoyer n'importe quels "pouvoirs" via l'event du clic de la souris. Il récupère les coordonnées du clic et va faire une action dépendant de l'item_actif sur le moment .

L'item_actif est initialisé à HAMMECONJ"client->id" (soit HAMMECONJ1 soit HAMMECONJ2) ensuite suivant la fonction switchUp() ou switchDown() on peut changer de pouvoir dépendant de quel joueur on est comme l'hamecon.

Cette fonction peut sembler être un fourre-tout mais c'est la façon qui nous a paru la plus utile.

A chaque fois que cette fonction est appelée on envoie l'étang au serveur pour faire des comparaison et tester si il y a déjà des objets présents ou autre.

lancerTruc() ne permet pas seulement de lancer des objets elle permet aussi de retirer l'hameçon et lorsqu'elle fait ce choix on retire l'ancien hammecon et on utilise la fonction *peche()* qui regarde autour du poisson et pêche en donnant les points au joueur. Cependant il faut bien relâcher le poisson après c'est pour cela que les pthread_cancel ne fonction pas dans leur boucle.

Pêche

La pêche comme expliqué ci dessus se fait via la fonction *peche()* appelée lors du retrait de l'hameçon par le client. Si le poisson se trouve en haut ou en bas ou à gauche ou à droite de l'hameçon on le pêche on obtient les points et les poireaus et on le relâche.

Le problème dans cette fonction c'est que le pthread_cancel est censé tuer le processus après l'avoir pêché mais avec la fonction timed_wait le processus semble introuvable et ne pas être sur la grille et donc crée un CORE DUMP ou ne passe pas dans la condition.

Scoring

Le scoring se fait grâce à la grille étang.

On obtient 1 point et 100 poireaus pour les POiSSON VAL1

On obtient 2 point et 200 poireaus pour les POiSSON VAL2

On obtient 3 point et 300 poireaus pour les POiSSON VAL3

Difficulté rencontrée

La plus grande difficulté de ce projet est d'avoir une modélisation juste au départ, avec une modélisation fausse on fait des erreurs et on recommence en essayant de trouver à tâtons la bonne manière de faire fonctionner les fonctions et nos attributs.

Le manque de temps à jouer car même si nous sommes en période de quarantaine , en n'ayant pas de travaux pratiques pour poser des questions et réfléchir, on avance à l'aveugle et on perd énormément de temps pour des erreurs bêtes telles que mettre les FD dans la boucle du serveur et pas au dessus car sinon elles sont remises à zéro et donc ne permettent aucun dialogue. Ou encore notre grille_t qui ne correspond défois pas à notre case_t.

De plus mettre en lien un client serveur et du ncurses ou du ncurses et des threads peut paraître être un challenge mais lier les trois l'est vraiment.

On apprend vraiment à lier toutes nos connaissances et à faire dialoguer ces parties différentes et c'est ce qui est la plus grande difficultés du projet

Implémentation manquante / solutions possibles

pthread_cancel on *peche()* ne fonctionne pas. La grille au moment ou l'on veut cancel ne contient pas l'élément POISSON

Changement de taille lors de l'ouverture, oubli de notre part de faire une taille variable pour les fenêtres à passer en args qui remplacerait les constantes définies dans le include.h.

La dynamite ne tuera pas non plus les poissons pour le même soucis que la pêche.

On ne dépose pas de requin via le pouvoir REQUIN.

Il faudrait calibrer la grille_t pour pouvoir cancel les thread.

Conclusion

NE TUEZ PAS LES POILLSONS SIOUPLAIT est un simulateur de pêche pacifiste ou les poissons ne sont pas nos ennemis.

Si nous avions pallié le problème de notre grille_t contenant nos thread et grâce à une meilleure modélisation du problème de base nous aurions pu finir ce projet.