

Rapport

Arène Vectorielle Synchrones

Julien Bissey
Kelly

15 avril 2019

1 Introduction

Ce projet est divisé en deux parties, Serveur et Client.

Le serveur a été codé en OCaml, le client en Java.

Un manuel d'utilisation est disponible dans le dossier doc. Le dossier src contient les sources du client et du serveur.

Concernant les extensions :

- le modèle physique est basé sur des chocs élastiques.

2 Serveur

Le serveur est composé d'une thread principale lançant les autres threads, une thread recevant les nouvelles connections et associant les nouveaux clients à une nouvelle thread, une thread par client qui reçoit les requêtes de ce client et y répond si nécessaire, et enfin une thread qui contrôle le déroulement du jeu, met à jour les positions des différents objets, etc.

On conserve tous les objets de l'arène dans un unique tableau dans le module Arena. On a choisi un tableau car on a souvent besoin d'accéder à un élément particulier de la structure (pour exécuter les requêtes de type NEWCOMS notamment).

Concernant les ajouts au formulaire :

Lorsqu'un nouveau joueur se connecte à une partie déjà en cours, en plus du WELCOME, on lui envoie également un SESSION et un NEWOBJ pour qu'il ait immédiatement connaissance des autres joueurs et de l'objectif actuel.

Comme les chocs élastiques font se déplacer les obstacles, il faut pouvoir les identifier lorsqu'on donne leurs nouvelles coordonnées aux clients. On remplace donc le message coords des requêtes WELCOME et SESSION par coords (un

nom en plus des simples coordonnées) et on ajoute un autre message vcoords à TICK. Pour annuler ces changements et respecter la norme du formulaire, il faut lancer le serveur avec l'option -comp. Le modèle physique sera alors une inversion du vecteur vitesse d'un vaisseau lors d'une collision avec un obstacle.

3 Client

Le client est composé d'une thread principale lançant les autres threads, une thread déplaçant les objets de l'arène indépendamment du serveur, une thread recevant les messages du serveur, modifiant les données du jeu en conséquence, et affichant les informations envoyées par le serveur dans le terminal (arrivée et départ de nouveaux joueurs, et scores notamment), et enfin une thread consacrée à l'interface graphique du client.

Cette interface est composé d'une unique fenêtre affichant l'arène. A part pour l'objectif, nous avons utilisé des polygones pour représenter les différents objets de l'arène. Puisque tous les objets ont une hitbox en forme de cercle, nous avons utilisé des polygones ayant une forme proche d'un cercle.

Pour les vaisseaux : le polygone lie 3 points du cercle formant un triangle, ainsi que le centre du cercle, cela donne une forme de flèche permettant de facilement repérer l'orientation du vaisseaux. Pour les obstacles : un polygone de 12 points choisis aléatoirement, de tel sorte qu'ils soient suffisamment proches du cercle pour que leur forme soit en accord avec leur hitbox tout en étant irréguliers. De plus, on fait tourner chaque obstacle dans un sens et à une vitesse aléatoire. On a donc besoin d'ignorer la valeur d'angle envoyée par le serveur à chaque TICK (on pourrait ne pas l'envoyer, mais cela forcerait à traiter les vaisseaux et les obstacles encore plus différemment pour un gain de performance négligeable).

Pour quitter le jeu, il suffit de fermer la fenêtre. Cela n'envoie pas de message EXIT au serveur (de fait, le client tel qu'on l'a écrit n'envoie jamais de message EXIT), Mais le serveur a été fait de telle sorte qu'il considère un client qui ne répond plus comme un client sortant du jeu de manière contrôlée.

De même, si le serveur s'arrête de manière imprévue, le client va en informer le joueur dans le terminal puis quitter le jeu.