M1 Informatique - Projet de PC2R

RAPPORT

Arènes Vectorielles Synchrones

Etudiants
Suxue Li
Julien Xavier

15 avril 2019

Table des matières

1 Introduction

Le devoir de programmation de cette année a pour objectif de réaliser une application de jeu clients-serveur où les joueurs jouent en concurrence dans un jeu de course dans l'espace, représenté par une arène torique.
Les parties s'organisent en sessions, les joueurs doivent atteindre des points particuliers nommés *objectifs* pour gagner une session. Des extensions ont été ajoutés par rapport à la version de base, vous trouverez une description plus complète dans le manuel du jeu à la fin de ce rapport. Pour ce projet, nous avons fait le choix d'implémenter la partie serveur en **Ocaml** et la partie client en **Java**.
Vous trouverez dans ce rapport, une présentation globale de l'architecture du projet, nos choix d'implémentation et les difficultés que nous avons rencontré au cours du développement. Enfin, un petit manuel de jeu vous guidera pour commencer à jouer.

2 Architecture générale du projet

2.1 Principe de communication

Pour réaliser la communication entre le serveur et les clients, nous avons utilisé un système de communication par socket, ce qui nous permet de réaliser des sessions TCP, afin d'envoyer des informations dans les deux sens de manière fiable.
La communication est basé sur le protocole texte donné dans le sujet.

2.2 Serveur

Nous avons préféré utiliser le côté fonctionnel du language Ocaml pour implémenter notre serveur. Le fichier *server.ml* contient tout le code du serveur. Toutes les valeurs constantes ont été définies au début du fichier, et une variable *current_session* contient toutes les données de la session en cours, notamment les joueurs qui sont présents, avec toutes les données qui les accompagnent. Ci-dessous les types *session* et *player* qui permettront de mieux comprendre la suite du rapport. Nous avons un point d'entrée principale dans le fichier, c'est la fonction *start_server* qui lance le serveur et écoute sur le port 2019. Mais avant d'écouter, il lance deux threads : *start_session* et *server_refresh_tick_thread* qui permettent respectivement, d'attendre la connexion d'au moins un joueur puis de patienter un délai avant de lancer le jeu avec les joueurs présents, et de faire les actualisations nécéssaire des positions de chaque joueur tous les 1
server_refresh_tickrate. Ensuite pour chaque client qui se connecte, un thread est lancé, et nous nous basons sur le partage des ressources code et données permis par le système de threading en Ocaml. En effet, tous les threads ont accès aux mêmes fonctions, et aux mêmes données, en l'occurence *current_session*. Sur le serveur, nous avons utilisé un seul Mutex et une seule Condition. Le mutex s'appelle *mutex_players_list*, il est là pour limiter l'accès à la liste des joueurs de la session en cours. Nous avions plusieurs mutex au début du projet, mais nous les avons retiré au fur et à mesure de l'avancement car cela devenait compliquer à gérer. Pour cause, l'utilisation de plusieurs mutex impliquerait des lock et unlock imbriqués, chaque fonction ayant son lot de modifications à faire sur les données, on arrivait facilement à des inter-blocages. Nous avons donc opté pour la sûreté, plus que l'efficacité. Et enfin la Condition est signalée à chaque connexion de joueur, permettant au thread qui lance le chronomètre de

C'est ici que se passe tous les systèmes de communication et d'interprétation des protocoles côté Serveur.
On divise ici le travail en plusieurs parties :
- Une partie d'initialisation/threading (start), qui permet de créer toutes structures de données utilisées pour l'application du serveur et de démarrer les différents services en paralèlles.
-Une partie réception (process), qui permet l'interprétation de chacun des services proposés par les protocoles de réception client->serveur.
-Une partie d'envoi (send), qui permet de préparer les commandes et d'envoyer aux différents clients les informations nécéssaire au bon déroulement du jeu.

2.3 Client

Pour la partie client, nous avons fait une structure qui s ?inspire du MVC ayant pour Modèle toutes les classes de représentation de données telles que Ship.java, Player.java ou encore Point.java qui représente comment on représente nos joueurs et leurs véhicules.

3 Manuel du jeu

3.1 Extensions