LE LAN Quentin

12/02/2024
ESIR 3 INFO SI

SR – PM

I. Introduction

Pour ce projet de SR qui est de programmer un mini jeu de manière répartie. Nous sommes parties sur une architecture basique de client/serveur en Scala. Notre serveur gère la partie, il garde la liste de participants, des fruits, les positions, le score, etc… Le client ne sert juste qu'à informer le serveur des déplacements et de l'affichage.

II. Pourquoi Scala ?

Nous avons choisi de construire ce système en Scala avec la bibliothèque Akka. Elle permet de faciliter la construction de système distribué (la notion d'Actor qui est dédié aux partie asynchrone & concurrence), elle permet aussi de limiter les goulots d'étranglement, elle est robuste aux échecs. Akka permet de faciliter le traitement sur les flux de données (avec les Streams) avec des opérateurs dédiés. Scala avec son aspect fonctionnel se prête très bien à ce type de projet, de plus ce langage est très (très) performant.

III. Côté serveur

On trouve ici un Actor, le gameAeraActor qui permet de gérer notre jeu en lui-même, il va réagir au gameEvent (ajout d'un joueur, mouvement, etc). Par exemple, s'il reçoit "up" il va regarder si aucun autre joueur est sur la futur position (les Actors gèrent ça de manière asynchrone de base), si la case est disponible on change les coordonnées du joueur. On regarde aussi s'il y avait un fruit, on ajuste le score du joueur en fonction. On fait du broadcast, c'est à dire que les messages sont envoyés à chaque joueurs connectés au serveur. Il vérifie aussi si tous les fruits ont été mangé, si c'est le cas, il le notifie aux joueurs. A chaque début de partie le serveur va générer aléatoirement les positions des fruits. En plus de gérer les déplacements et les fruits, le serveur gère l'ajout ou la suppression d'un joueur, cela modifiera les messages qu'il enverra aux clients.
La classe gameService permet de transformer les gameEvent en message webSocket pour être envoyé sur le réseau (et inversement).

IV. Côté client

Nous avons choisi que le client ne puisse se déplacer uniquement à partir des touches directionnelles du clavier (up, down, right, left).
Le client va d'abord demander le nom du joueur en ligne de commande, puis on va créer une websocket avec le serveur et on lui dit qu'elle est le nom du joueur. `localhost:8080/?playerName=bob`
Il nous répond avec une liste de joueurs (position, nom et score) ainsi qu'une liste de nourriture (position).

{"players": [{"name":"Bob","position":{"x":0,"y":0},"score":0}], "foods": [{"position":{"x":3,"y":0}},{"position":{"x":5,"y":5}}]}

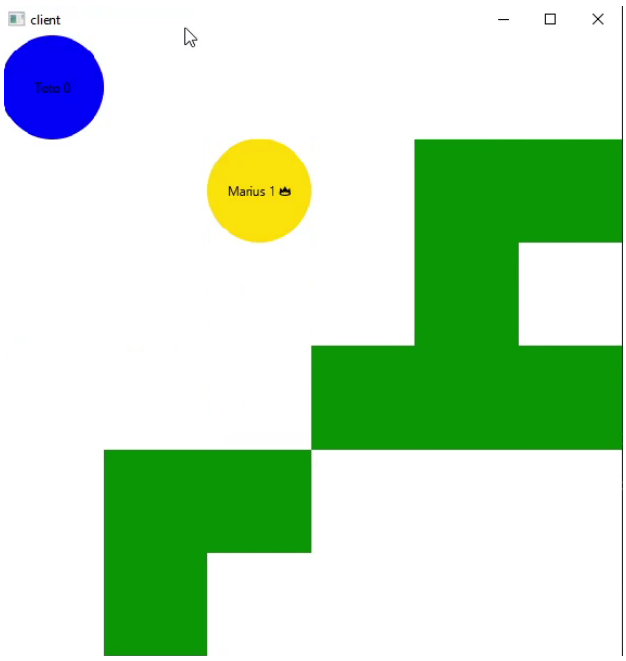
Il nous suffit alors de créer sur le GUI les formes correspondant au joueur, leur nom et ainsi que les mettre à leur place. Nous mettons le joueur qui gagne actuellement dans une couleur différente. Il faut aussi mettre en place la nourriture.

Chaque fois qu'un joueur appuie sur une touche directionnel un message est envoyé via la websocket au serveur, par exemple "up". Le serveur répondra avec la liste des joueurs et de la nourriture mise à jour. On met à jour le GUI en fonction de la réponse du serveur. Si le serveur à détecter que la partie est fini il envoie un message "`Game ended`". Cela signifie qu'il y a un gagnant, nous affichons alors une boite de dialogue où l'on demande le nouveau pseudo des joueurs et relançons la partie ainsi que la GUI (s'il refuse, le client se ferme).

V. Travail réalisé

Je me suis principalement occupé de l'interface graphique ainsi que des tests. Pour commencer, j'ai effectué de nombreuses recherches et il n'existe pas d'outils ou de framework pour effectuer des tests end to end ou d'interface en Scala. Il paraît donc compliqué de pouvoir tester entièrement notre application ou du moins son interface. J'ai cependant mis en place la suite de test (unitaire donc) côté serveur, côté client ce n'est que l'interface donc impossible. Pour tester le serveur, je simulais la connexion de nouveau client et l'envoie de message et je matchais la réponse du serveur en fonction de nos attendus.

Pour l'interface graphique, je ne m'y connaissais pas réellement en Scala et encore moins en GUI Scala. Je suis donc partie sur une interface basique blanche, les joueurs sont simulés par des ronds et les fruits par des carrés verts. Le joueur avec une couleur différente est le joueur qui a le plus de points au cours de la partie.



VI. Objectifs

Nous devions créer un mini jeu simple : plusieurs joueurs arrivent sur une map où se situe des fruits, celui qui mange le plus de fruits gagne. La partie est terminée lorsque plus aucun fruit ne se trouve sur la map.

Nous devions donc mettre en place différents objectifs :

- Collisions entre joueur, 2 joueurs ne peuvent pas être sur la même case
- Un fruit ne peut pas être mangé par plusieurs joueurs
- Affichage du gagnant actuel et du score
- La partie se termine lorsqu'il n'y a plus aucun fruit
- On peut relancer une partie

Tous ces objectifs ont été mis en place dans notre projet.

Nous avons donc bien un mini-jeu fonctionnel.

VII. Futur

Bien que tous les objectifs principaux soient mis en place, il reste cependant des améliorations possibles. On peut notamment améliorer l'interface graphique, elle reste pour l'instant très basique, pourquoi pas mettre des images à la place de carré vert pour simuler la nourriture. On peut aussi demander le nom de l'utilisateur pour la première fois avec une boîte de dialogue. On peut notamment penser à améliorer la gestion des bordures de la map.