Projet SR - Rapport Individuel : Backend du Jeu Distribué

Yazid BENJAMAA - ESIR 3 SI - Binôme D

Choix techniques

Côté backend, nous avons choisi de travailler avec TypeScript, car nous connaissons déjà assez bien ce langage qui dispose aussi de nombreuses bibliothèques disponibles à cet effet. L'utilisation de TypeScript facilite également le déploiement de l'application.

Le backend s'appuie sur la dépendance `Socket.IO`, cette librairie a été choisie pour les possibilités de communication bidirectionnelle en temps réel qu'elle offre. La dépendance `uuid` est également utilisée pour la génération d'identifiant unique qui serviront à identifier les joueurs.

Deux interfaces sont définies, à savoir `PlayerObj` et `Position` qui définissent respectivement un joueur et la nourriture.

Gestion des événements

Un challenge auquel nous avons été confrontés est celui de l'adéquation entre les données qui, et cerner quelles étaient les données qui étaient importantes et nécessaires au bon fonctionnement du jeu pour lesquels nous avions besoin d'une communication client/serveur. Cet aspect est important, car nous avons optimisé aux mieux le nombre de requêtes qui étaient envoyées afin de ne pas saturer la connexion, notamment lorsque plusieurs clients sont connectés.

Le backend reçoit et gère les événements suivants qui sont émis par le/les clients.

- `newPlayer` pour l'ajout d'un nouveau joueur et émettre sa position à tous les clients.
- `connect` qui gère la connexion de joueurs et la création d'un socket pour chaque joueur.
- `move` pour mettre à jour la position d'un joueur.
- `eatFood`, `eatMaliciousFood` et `eatPlayer` qui gèrent les interactions entre les joueurs et les nourritures.
- `Leave` qui gère le départ d'un joueur (quand il quitte la partie/ferme la session) + `disconnect` qui gère la déconnexion d'un joueur.
- `error` afin de gérer les erreurs de connexion.

Le backend va également faire en sorte de gérer les cas dans lesquels deux joueurs vont vouloir manger en même temps une nourriture en attribuant la nourriture au premier joueur qui l'aura mangé. Au moment où le deuxième player voudra manger la même nourriture, celui-ci se la verra refuser et non attribuée, car celle-ci n'existe plus et aura été retirée de la liste des nourritures, ce qui fera qu'elle aura un statut de variable `undefined` et donc non valable. Le backend effectue le check de la manière suivante dans le bloc de l'événement `eatFood` :

```
socket.on("eatFood", (uuid: string, foodId: string) => {
        const player = players.get(uuid);
        const food = foods.get(foodId);
        if (player !== undefined && food !== undefined && Math.sqrt((player.x -
food.x) ** 2 + (player.y - food.y) ** 2) < (scoreToSize(player.score)+4)){
            player.score += 1;
            foods.delete(foodId);
            console.log("eat food : " + uuid + " " + foodId);
        }
    });
```

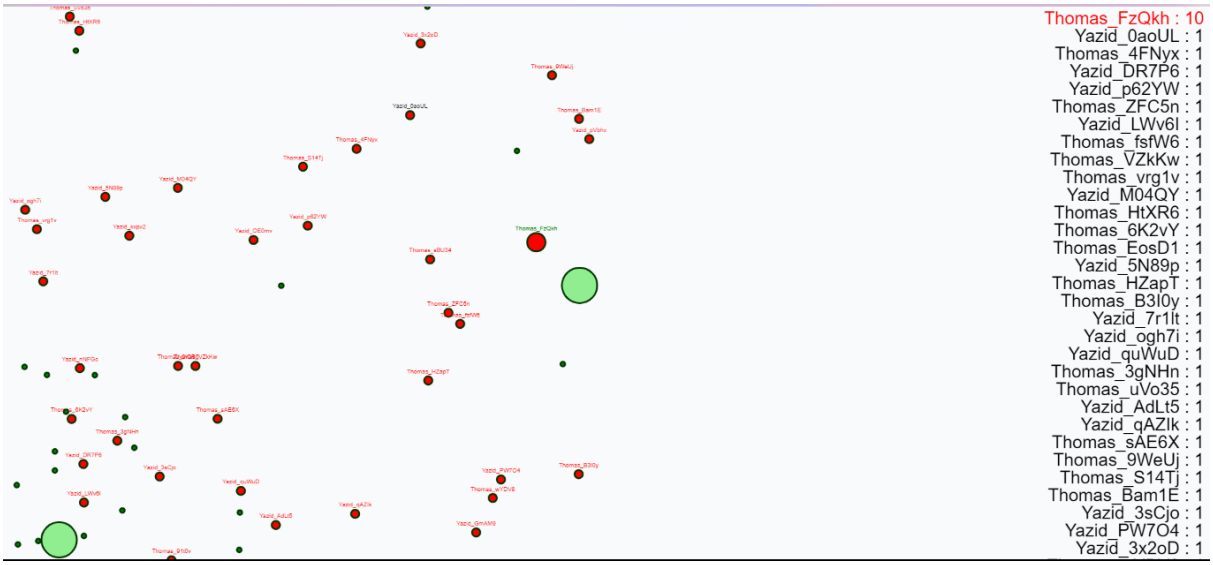
Peu de requêtes/acquittements sont émises de la part du serveur, elles ont pour but de le notifier le client d'effectuer certaines actions comme la mise à jour des coordonnées des joueurs / nourriture, toujours dans l'optique de réduire au mieux le nombre de requêtes qui transitent entre les deux entités.
Ce même principe est appliqué pour la nourriture "malicieuse" dans le handler d'événement `eatMaliciousFood`.

Tests de charge côté client

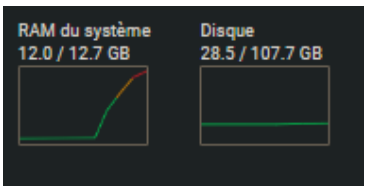
Nous avions également eu besoin de nous assurer que le jeu pouvait supporter la connexion simultanée de plusieurs clients et nous avons voulu voir comment le serveur de jeu réagissait à cela. Pour se faire, nous avons conçu un notebook avec Python et Selenium et profité de la puissance de calcul + RAM de Google Colab afin de générer plusieurs instances de Google Chrome où chaque instance représente un joueur établissant une connexion après avoir cliqué sur le bouton "Start Game" du menu principal.

Pour 100 connexions simultanées, aucune latence n'a été observée et le jeu était fluide, notamment lorsque l'on faisait bouger/simuler des déplacements aux joueurs. Une limitation à laquelle nous avons dû faire face est que chaque instance de navigateur prend pas mal de place en RAM et faire connecter 100 joueurs prend quelques 12Go de Ram, il était donc compliqué de faire des tests de charge avec un nombre de joueurs très élevé.

L'image suivante montre l'apparition de plus de 100 joueurs instantanément :



Consommation de RAM sur Colab pour 100 joueurs :



Mécanisme de bannissement des utilisateurs abusifs

Afin de prévenir les tricheries, brute-force et autres fourberies de la part des clients, nous avons mis en place un petit mécanisme de bannissement des joueurs qui fait déconnecter les joueurs qui dépassent un certain seuil de requêtes par seconde.