Rapport d'implémentation du Jeu Distribué - Thomas Delapart

Introduction

Avant de nous plonger dans le code, nous avons cherché quel jeu nous pourrions réaliser. Nous avons choisi de garder les fonctionnalités de déplacement comme dans l'exemple de base, ce qui nous a fait assez converger vers notre jeu final : `Hagar.io`.

Le projet Hagar.io est un jeu multijoueur en temps réel, où le but est d'occuper la première place du classement en augmentant son score le plus possible sans mourir. Ce rapport détaille la conception et l'implémentation du jeu.

Le Frontend

Aperçu de la conception

Stack Technologique

Nous sommes partis sur une base https://create.t3.gg pour avoir une base pour le développement de notre projet avec les technologies suivantes :

`Next.js`, la backend de Vercel, très bien intégré à React. Cela nous permet de bénéficier des avantages du rendu côté serveur tout en conservant la réactivité du rendu côté client. Sa compatibilité avec Vercel nous permet également de déployer rapidement et efficacement le projet.

Le projet utilise donc aussi `React` pour son polyglottisme et sa facilité d'utilisation, qui permet certaines facilités pour le développement du jeu, notamment pour la gestion des événements et des états.

Nous avons cd plus utilisé `Tailwind CSS` pour la conception de l'interface utilisateur afin de garantir une expérience de jeu fluide et réactive.

Le projet est écrit en `Typescript` pour garantir la robustesse du code et faciliter la maintenance.

Pour communiquer avec le serveur, nous avons utilisé `Socket.io` qui nous évite de faire des requêtes HTTP pour récupérer les informations.

Dans le but de garantir la qualité du code, nous avons mis en place une série de tests d'intégration à chaque commit pour s'assurer du bon fonctionnement de la non-régression du projet grâce à l'outil `Playwright`.

Architecture

Notre projet est basé sur une architecture distribuée un peu spéciale, car la totalité des calculs du jeu sont effectués côté client pour une expérience de jeu fluide et réactive.

Le serveur ne sert que pour la communication entre les joueurs et la mise à jour des scores. Toutefois, celui-ci est capable de gérer les cas de triches et les mises à jour des scores en cas de déconnexion du joueur.

Communication avec le serveur

Dans notre communication avec le serveur, on commence par se connecter au serveur avec `SetSocket`, puis on émet un message `this.socket?.emit("newPlayer", ...);` afin de signaler au serveur qu'un nouveau joueur est arrivé.

Par la suite à chaque tick, on envoie au serveur la position du joueur avec `this.socket?.emit("move", ...);`. On récupère également les informations des autres joueurs avec `this.socket?.on("players", ...);` et les informations des boules de nourriture avec `this.socket?.on("food", ...);`.

Les tests

La première partie des tests est effectué avec l'outil `Playwright`. Ces tests permettent de vérifier le bon fonctionnement de l'application en simulant des actions utilisateur. Lancés par la CI, GitHub Actions, ils sont un gage de qualité pour le projet.

Test de Chargement de la page

C'est un ensemble de 3/4 tests qui regardent que les pages sont bien accessibles à leur URL ou en passant par les boutons de l'app.

Vérification de la fonctionnalité de changement de nom

C'est un test qui vérifie que la page profil permet bien de changer de nom en vérifiant après que le changement de nom est effectif sur la page principale.

Vérification des fonctionnalités du menu

C'est un test pour la barre de menu qui vérifie que les boutons sont bien cliquables et que la navigation entre les pages se fait bien.

Vérification de la fonctionnalité de déplacement

C'est un test qui vérifie que le joueur peut bien se déplacer en créant une instance de joueur et en vérifiant que le joueur bouge bien.

Test de charge pour 10 et 20 joueurs connectés

C'est un test qui vérifie que le serveur peut bien gérer 10 et 20 joueurs connectés en vérifiant que les joueurs sont bien créés et que les joueurs peuvent bien se déplacer.

Le Backend

Conception et Architecture

Choix Technologiques

Le backend de notre projet Hagar.io est beaucoup plus simple que le front, et donc utilise moins de technologies. Il est construit en `TypeScript` avec `Node.js` et utilise le framework `Socket.io` pour la gestion des connexions en temps réel.

Nous avons choisi cette combinaison en raison de la nature asynchrone de `Node.js`, qui se prête bien aux applications nécessitant une communication en temps réel avec de nombreux clients. Le backend s'inscrit dans la continuité du front, en utilisant les mêmes outils pour garantir la cohérence du projet.

Il n'y a pas de base de données dans notre projet, car nous n'avons pas besoin de stocker des informations sur les joueurs. Tout est géré en mémoire par le serveur. Les données sont stockées dans des objets JavaScript, ce qui permet de les manipuler facilement et de les envoyer aux clients et sont donc perdus à chaque redémarrage du serveur.

Gestion des Connexions

Lorsqu'un joueur se connecte, le serveur lui attribue un identifiant unique généré avec `uuid` et initialise ses informations de base, telles que la position aléatoire et le score initial. Le serveur maintient une liste des joueurs actifs et communique périodiquement les mises à jour de l'état du jeu aux clients connectés.

Le serveur stocke également une Map entre les joueurs et les sockets pour pouvoir communiquer avec les joueurs et savoir à la déconnexion d'une socket quel joueur déconnecter.

Détection de Triches

Une mesure anti-triche basique est intégrée, où les déplacements trop rapides sont ignorés. Les joueurs envoyant plus de requêtes que la fréquence de rafraîchissement du jeu sont aussi par ailleurs ignorés.

De même, lors de la connexion d'un joueur, on vérifie que le nom n'est pas déjà utilisé par un autre joueur. Lorsqu'un joueur envoie une requête pour manger une boule de nourriture, on vérifie que la boule est bien à portée du joueur. Enfin, lorsqu'un joueur envoie une requête pour manger un autre joueur, on vérifie aussi que le joueur ciblé est bien plus petit que le joueur attaquant et que la distance entre les deux joueurs est inférieure à la range de l'attaquant.

Ces mesures sont conçues pour empêcher les joueurs de tricher en se déplaçant trop rapidement, en envoyant des requêtes trop fréquentes ou en manipulant les scores.

Tests et Validation

Le backend est testé avec un script python `ipynb` qui permet de simuler des joueurs et de vérifier que les joueurs sont bien créés et que les joueurs peuvent bien se déplacer. Ces tests permettent de vérifier que le serveur peut bien gérer 50 à 100 joueurs connectés en même temps.

Nous avons également testé la détection de triches grâce à Cody Adam qui nous a simulé des déplacements trop rapides et des requêtes trop fréquentes. Que nous nous sommes efforcés de patcher.

Futures Améliorations

Des améliorations futures pourraient inclure des fonctionnalités anti-triche plus avancées, une meilleure gestion des erreurs, et une extensibilité pour prendre en charge un plus grand nombre de joueurs simultanés. Nous sommes confiants dans la solidité actuelle du backend tout en reconnaissant les opportunités d'optimisation et d'extension.

Conclusion

Notre projet Hagar.io est le résultat d'une réflexion approfondie sur la conception d'un jeu distribué, efficace et réactif. Malgré les défis rencontrés, nous sommes fiers du résultat obtenu et sommes convaincus des possibilités d'amélioration future.