ÉCOLE
CENTRALELYON

S8 ELC-D3 - Applications Web

Application Web : jeu d'Othello

Auteurs :
Risa KAMBE
Emma DELAROZIERE

Enseignant :
M. Daniel MULLER

Version du
6 avril 2021

Table des matières

Introduction

On s'intéressera dans ce présent rapport à la programmation d'une application web d'Othello. Othello est un jeu de stratégie à deux joueurs, aux règles facilement assimilables mais tout de même assez complexes pour être un sujet de programmation intéressant.C'est ainsi l'opportunité de tester sur un projet plus conséquent Node js et des modules tels qu'express et socket.io.

Express est un module permettant de créer une application web. Socket.io est destiné à la création de websockets assurant l'interaction en continu entre le serveur et chaque client. Il nous permettra ici de mettre en place un système de salons pour que plusieurs parties puissent avoir lieu simultanément sans interférer les unes avec les autres.

1 Cahier des charges

— On souhaite organiser une partie de jeu othello entre deux clients. Il faut donc pour cela utiliser des websockets pour assurer la communication entre le serveur et les deux clients. Pour que le serveur puisse reconnaître ses joueurs, les sockets des clients ne doivent jamais être déconnectés du serveur. Une fois connecté, le client ne doit pouvoir interagir avec le serveur qu'à l'aide des commandes socket.io, au risque sinon de déconnecter le websocket.

— Pour que le joueur puisse facilement prendre en main son système, il faut une interface graphique claire.

— Si le joueur ne connaît pas les règles, il peut accéder immédiatement à une source expliquant ces règles

— Pour que plusieurs parties puissent être gérées simultanément par le serveur, on doit mettre un place un système de salons de jeu. Les joueurs peuvent choisir quel salon rejoindre à l'aide d'un identifiant de salon. La partie ne doit être lancée qu'à l'arrivée du second joueur dans le salon.

— On souhaite intégrer un chat une fois la partie lancée. Le client doit pouvoir correctement distinguer qui est l'expéditeur d'un message donné. Seules deux personnes dans un même salon devraient avoir le droit de communiquer.

À des fins de débugs, il est possible d'observer une partie des échanges entre serveurs et clients dans la console du serveur.

1.1 Description du code

1.2 Principe de fonctionnement général

— On met d'abord notre identifiant sur la page de login.

— Ensuite, on entre l'identifiant de salon. Si le salon n'existe pas, il est alors automatiquement créé avec l'identifiant donné.

— S'il n'y a pas d'autre joueur, on attend jusqu'à ce que l'autre personne entre dans la salle. Dès qu'un joueur rejoint un salon, un message de bienvenue est diffusé dans le chat.

— On commence le jeu.

1.3 Les problèmes rencontrés

Nous avons été confrontées à plusieurs difficultés lors de la programmation de l'application.

— La plupart des solutions que nous avons essayées pour vérifier si deux joueurs sont bien connectés dans un salon ont posé problème, n'étant valables que pour certaines versions de socket.io ou bien posant des problèmes de callback. L'event listener côté serveur pour rejoindre le salon étant le même pour les deux joueurs, pour savoir si le deuxième joueur est connecté, nous avons finalement opté pour un système "ping-pong". Lorsqu'un joueur clique sur le bouton pour rejoindre un salon, le client envoie par le biais du socket un event "join". L'event listener pour l'événement "join" envoie par le biais de socket.to(room).emit() un ping aux clients présents dans le salon, sauf au client qui vient d'envoyer au serveur l'événement "join". Ainsi, lorsque le premier joueur d'un salon se connecte, il ne se passe rien vu que personne ne reçoit le "ping". Cependant, lorsque le deuxième joueur se connecte au salon, le premier joueur à s'être connecté reçoit le "ping", et répond au serveur par un "pong", indiquant au serveur que la partie dans le salon concerné peut commencer. Le serveur envoie ensuite un signal à tous les clients du salon pour générer le plateau.

— Originellement, il était prévu d'initialiser à la même valeur pour tous les clients la valeur bwp en tant que variable globale, qui dicte la couleur du client, puis de modifier lors de la connexion les valeurs pour qu'elles soient différentes pour les deux clients d'un salon. L'assignation de la valeur de joueur est finalement devenue un attribut du plateau de jeu gp, et lors de l'échange de ping et de pong du salon, le client recevant le ping va générer son plateau avec un attribut gp.bwp égal à 1, et le serveur, quand il reçoit le "pong" de ce client, va envoyer un signal pour générer le plateau à l'autre client, avec gp.bwp=-1. La valeur assignée à bwp est alors un argument de la fonction gp_start responsable de la génération du plateau.

— Nous avons rencontré des problèmes de synchronisation des plateaux des deux clients lors d'une partie. Afin de régler ce problème, nous avons changé la façon dont un pion était joué : plutôt que d'envoyer les seules coordonnées du pion joué, le client envoie finalement au serveur l'intégralité du tableau représentant les pions sur le plateau, puis le serveur le transmet à l'autre client pour qu'il mette aussi à jour son plateau.

Conclusion

Nous avons pu ainsi obtenir le squelette d'un jeu othello en ligne. En dépit des problèmes techniques rencontrés lors du développement de l'application, ce projet a amélioré notre compréhension du fonctionnement de socket.io et de l'utilisation de l'objet canvas.