AWS : Jeu de Dames en ligne

Clément CAUMES & Mehdi MTALSI-MERIMI

UFR des Sciences Versailles - M1 Informatique

May 26, 2019

Fichiers de l'application

- server.js qui gère l'application côté serveur
- public/client.js qui gère presque toute la partie cliente de l'application
- public/connexion.js qui stockera les informations entrées par l'utilisateur lors de la connexion avec un mécanisme de session (sessionStorage)
- public/deconnexion.js qui videra la session de l'utilisateur à sa déconnexion
- views/*.html qui sont tous les fichiers HTML de l'application
- public/*.css qui représentent toutes les feuilles de style du site web

Outils de l'application

- express pour définir les différentes routes de l'application (/connexion, /play, /logout, /signin, /login)
- nunjucks pour générer dynamiquement les pages HTML
- express-session pour la persistance des données
- websockets pour obtenir une communication entre clients-serveur (envoyer la liste des joueurs ou assurer le jeu entre deux utilisateurs)

Les points obligatoires réalisés

- Une page permettant de s'enregistrer
- Une page présentant la liste des utilisateurs en ligne, leur nombre de parties gagnées/perdues
- Un mécanisme pour se connecter
- Un mécanisme permettant de défier un adversaire
- Une interface permettant de jouer le jeu

Les points optionnels réalisés

- montrer les pions déplaçables et leurs déplacements possibles
- intégrer une base de données pour le stokage des informations de l'état du jeu par le serveur (conséquence : plusieurs parties de joueurs différents peuvent se jouer en même temps)
- mettre plusieurs états des joueurs (CONNECTE, DECONNECTE, OCCUPE)
- permettre à un joueur de quitter par forfait (abandon)
- gérer la déconnexion inhabituelle d'un joueur en pleine partie
- stocker le hash des mots de passe afin de ne pas les stocker en clair (avec bcrypt)

Les points à ajouter

- implémenter plusieurs systèmes de règles différents
- utiliser une base de données No-SQL
- mettre un tchat sur la page de jeu à côté du plateau pour discuter avec son adversaire pendant la partie (non demandé)

Inscription et Connexion

- Lorsqu'un utilisateur crée un compte, le serveur vérifie que le login choisi n'est pas déjà utilisé par un autre joueur, puis le crée.
- Lorsqu'un utilisateur se connecte, il entre son login et son mot de passe dans le formulaire dont le serveur vérifie son identité.
- Après la vérification de son identité, le serveur le redirige vers la page /play. Il y a ensuite un établissement d'une websocket entre le client et le serveur. Ainsi, le serveur va associer le login du client avec le websocket du client. A chaque mise à jour de l'état des websockets entre le serveur et les clients, le serveur va envoyer la liste de l'état des joueurs (connecté, déconnecté, occupé). Le client va afficher cette liste avec les boutons de défi si le joueur est disponible (non occupé et connecté).

Initialisation du plateau

- Si un joueur défie un autre, celui-ci va envoyer un message d'invitation au serveur. Le serveur vérifie que les deux utilisateurs sont disponibles, crée le plateau associé à leur partie, les met en état d'occupé et envoie aux deux joueurs un message de **défi** avec de préciser les informations importantes (à qui est le tour, l'hôte, l'adversaire ...).
- Lorsqu'un joueur reçoit un message défi de la part du serveur, le client va créer un plateau complet et l'afficher. Ensuite, le client enverra un message **invitationRecue** au serveur.
- Le jeu commence et le plateau envoie aux deux joueurs un message **play** avec le nom de l'hôte, l'adversaire et le nom du joueur qui doit jouer.

Jeu entre deux utilisateurs

- Le client qui doit jouer met à jour son plateau et déplace son pion. Ensuite, le client envoie un message **playDone** avec le déplacement effectué.
- Le serveur reçoit le message et met à jour en conséquence son plateau. Puis, le serveur envoie le déplacement effectué sous la forme d'un message **play** au joueur qui doit maintenant joué. et qui attendait au tour précédent. Et ainsi de suite, en répétant l'étape précédente et celle-ci.
- A chaque fois que le serveur reçoit un nouveau déplacement, il vérifie sa validité et vérifie s'il y a un gagnant. Si le déplacement est invalide, s'il y a un abandon, si un joueur se déconnecte, le serveur enverra un message **error**. Si la partie est terminé, il enverra un message **défiFini**.

Défenses utilisées pour cette application

- protection contre les injections SQL avec des requêtes préparées
- protection contre les attaques XSS
- protection côté serveur du client qui essayerait de modifier manuellement ses sessionStorage
- mécanisme qui permet d'empêcher de se connecter deux fois au même compte