École Centrale Lyon

Projet javascript - Jeu d'échecs

Etudiant :
Koumou Jonathan ASSAMAGAN
Mouhamadou KANE

Enseignant:
Daniel MULLER

Table des matières

1 Guide de l'utilisateur

1.1 Mise en place

Pour la mise en place de l'application web, il faut au préalable installer les dépendances nécessaires. Il vous faudra donc installer nodejs et npm. Ensuite, après vous être placé dans le répertoire principale du projet avec la commande cd dans le terminal, il vous faudra exécuter les commandes suivantes :

$$1.\ npm\ init\ -y$$

$$2.\ npm\ install$$

$$2.\ node\ .$$

La dernière commande permet de lancer le serveur. Une fois ceci fait, rendez vous à l'adresse localhost :3000 sur votre navigateur. Vous voilà connecté à l'interface de l'application.

1.2 Utilisation

Une fois que vous vous êtes connecté, vous voudriez sûrement affronté votre premier adversaire. Pour cela vous avez le choix entre deux méthodes :

Première Méthode : vous et votre adversaire pouvez jour l'un contre l'autre sur le même appareil. Ouvrez deux onglets sur votre navigateur préféré et accédez, sur les deux onglets, à l'adresse localhost :3000 . Il vous suffira alors de copier l'identifiant affiché dans l'un des onglets dans le champ de l'autre onglet et de cliquer sur le bouton "Jouer". Votre partie est alors lancée. Le joueur ayant cliqué sur "Jouer" joue avec les pions blancs tandis que l'autre joueur joue avec les pions noirs. Vous pouvez alors jouer dans le respect total des règles du jeu d'échec

Deuxième méthode : Si votre adversaire et vous êtes connecté dans le même réseau, l'un d'entre vous (l'hôte) peut suivre la procédure de mise en place de l'application, tandis que l'autre peut accéder à la plateforme en saisissant l'adresse ip de son adversaire dans son navigateur : <adresseIp> :3000 . Assurez-vous tout de même qu'un firewall n'interdit pas la connexion à l'ordianateur hôte. Vous pouvez aussi héberger en ligne votre application mail et y accéder via son adresse url.
Pour les deux méthodes, la procédure de lancement du jeu est similaire.

FIGURE 1 – Interface de l'application

2 Implémentation

2.1 Algorithmes et logique

Trois éléments de logique sont importants à prendre à considération dans un jeu d'échecs : le déplacement des pièces, la capture de pièce et la mise en échec (et mat) du roi.

Configuration initiale Pour créer l'échiquier, nous avons utilisé une image sur laquelle nous avons superposé une grille de 8x8 représentant les cases du jeu. Les pièces sont ajoutées sur leur case de départ au début de chaque partie et chaque pièce possède une classe du type square-position colorPiece.

Déplacement des pièces Pour déplacer les pièces nous avons créer des classes pour chaque type de pièce (Pawn, Knight, Bishop, Rook, Queen, King). Et dans chaque classe on implémente la logique de détermination des coups possibles et légaux.

On utilise des objets javascript afin de sauvegarder la pièce qui est sélectionnée par un joueur ainsi que la case sur laquelle elle se trouve. On peut alors déplacer la pièce sélectionnée si cela nous est permis. Pour plus de facilité de jeu, les déplacements possibles sont en surbrillance, et les potentiels pièces à capturer également.

FIGURE 2 – Mouvement possible pour la dame blanche dans une position donnée

La fonction présentée à la figure 3 permet d'avoir tous les coups possible pour une pièce donnée sur une case donnée.

```
/**
 *
 * @param {T.Piece} selectedPiece
 * @param {Boolean} toDisplay if false, hide the moves
 *
 * @returns {T.Square[]} list of all available squares to move
 */
function getAvailableMoves(selectedPiece, toDisplay = true, noskip = true) {

    if (!selectedPiece) {
        return;
    }

    if ((selectedPiece.color === currentPlayer && currentIdColor === selectedPiece.color
) || (selectedPiece.color === currentPlayer && !noskip)) {
        const position = [selectedPiece.position.col, selectedPiece.position.row]
        /**
         * @type {T.Square[]}
         */
        let moves = [];
        if (selectedPiece.position.piece) {
            moves = selectedPiece.position.piece.getPossibleMoves(position, squares);
        } else {
            moves = selectedPiece.object.getPossibleMoves(position, squares);
        }
        displayAvailableMoves(moves, toDisplay);

        return moves;
    }

    return []
}
```

FIGURE 3 – Fonction `getAvailableMoves` pour obtenir les coups disponibles

Capture des pièces Pour capturer une pièce on vérifie si une pièce adverse est sur la trajectoire la pièce sélectionnée dans ce cas dès qu'on clique . (Cf image rouge dans la figure 3) ;

2.2 Communication et Réseau

Pour cette partie, l'utilisation du module socket.io a été centrale. Il nous a permis d'établir une connexion permanente entre le client et le serveur. Il ne fut donc pas nécessaire d'actualiser en permanence la page afin de vérifier si le joueur adversaire avait oui ou non déplacé son pion. Ce module a été utilisé aussi bien au niveau client qu'au niveau serveur.

Niveau Serveur : A ce niveau, le code se charge de l'ouverture d'une connexion entre les clients et le serveur (méthode socket.on()) ainsi que de la ré-émission vers tous le clients des informations émises par un client en particulier (méthode io.reemit()).

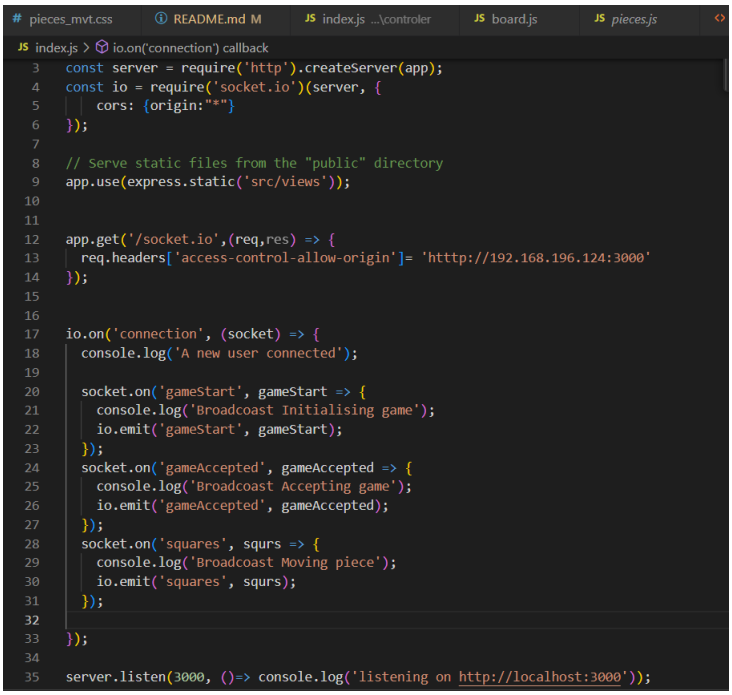

FIGURE 4 – Code côté serveur (index.js)

Niveau Client : A ce niveau, le code se charge de récupérer les données ré-émises par le serveur et provenant de l'adversaire et émet à son tour des données destinées à l'adversaire. Les données échangées sont principalement constituées des positions des pièces ainsi que des données de lancement de partie. Parmi les fonctions concernées, nous avons :
- socket.on('gameStart' ... : qui attend un événement start game envoyé par un adversaire. Il vérifie si l'id envoyé par l'event correspond à son id avant d'accepter la partie en émettant à son tour un event gameAccepted.

- socket.on('gameAccepted' ... : qui attend un événement game accepted envoyé par un adversaire en réponse à sa demande de partie. Il vérifie si l'id envoyé par l'event correspond à son id avant de démarrer la partie.

- socket.on('squares', .. : quant à lui attend un événement "squares" qui correspond au déplacement d'un pion par l'adversaire. Il met ainsi à jour la position des pièces du joueur qui devient la même que celle de l'adversaire en temps réel.

```
//Accepts a game started by an oponent
socket.on('gameStart', gameStart => {
    if (gameStart[1] == myId){
        console.log('Accepting game ...');
        socket.emit('gameAccepted', gameStart);
        oponentId= gameStart[0];
        currentIdColor= C.BLACK_PIECE;
    }
});

//Starts a game accepted by an oponent
socket.on('gameAccepted', gameAccepted => {

    if (gameAccepted[0] == myId){
        console.log('Starting game ...');
        oponentId= gameAccepted[1];
        document.getElementById("oponent-id-display").innerHTML= oponentId;
        currentIdColor= C.WHITE_PIECE;
    }
});

//Updates positions of pieces for the opponent
socket.on('squares', function(squrs) {
    if (squrs[3] != myId){
        console.log('Receiving move ...');
        squrs[0].element= document.getElementsByClassName(JSON.parse(squrs[4]).className)[0];

        let pc= JSON.parse(squrs[6]);
        squrs[0]= createPieceObjectByHtmlElement(squrs[0].element)
        // switch(squrs[0].type){
        //     case C.PAWN:
        //         squrs[0].object= new Pieces.Pawn(pc["color"], JSON.parse(pc["possibleMoves"]));
        //         break;
```

FIGURE 5 – Code principal côté client (board.js)