INSTITUT UNIVERSITAIRE
DE TECHNOLOGIE
UNIVERSITÉ D'ANGERS

SAE Jeu d'échecs

Raimbault Mathéo | Gaiffe Killian
GEII 2300

Encadrant : Philippe Lucidarme

Sommaire:

Remerciements:

Nous tenons à exprimer notre sincère gratitude à toutes les personnes qui ont contribué de près ou de loin à la réussite de ce projet. Leur dévouement, leur expertise et leur énergie ont été des éléments essentiels tout au long de cette expérience passionnante.

En premier lieu, nous tenons à exprimer notre reconnaissance envers monsieur Lucidarme pour son accompagnement et son suivi tout au long du projet, ainsi que pour tous ses conseils qui nous ont été précieux

Nous voulons également remercier, Arthur Dupont et Yanis Cousseau pour leurs astuces ayant contribuées à notre avancée dans ce projet, notamment en ce qui concerne l'utilisation de la librairie chess.js.

Pour conclure, nous voulons adresser nos remerciements à toutes celles et ceux qui ont contribué de près ou de loin à l'avancement de ce projet.

Introduction:

Les jeux d'échecs, riches en histoire et stratégie, ont toujours captivé l'imagination des joueurs du monde entier. Dans le contexte actuel du numérique, la fusion entre la tradition des échecs et les avancées technologiques a ouvert de nouvelles portes à l'expérience de jeu. Notre projet s'inscrit dans cette dynamique, visant à créer une plateforme en ligne permettant de jouer facilement et de manière immersive.

Pour comprendre l'ampleur de notre projet, il est essentiel de s'attarder sur l'évolution des échecs en ligne. Depuis les premières implémentations rudimentaires sur Internet jusqu'à aujourd'hui, les plateformes de jeu d'échecs ont continué à évoluer, offrant des expériences toujours plus sophistiquées aux amateurs du jeu royal.

L'un des défis majeurs dans le développement de jeux en ligne réside dans la gestion des interactions entre les joueurs et le plateau d'échecs. Nous nous sommes donc intéressés à ces interactions et les validations de mouvements concordants avec les règles du jeu.

Cahier des charges :

Le cœur de notre projet est donc de créer une interface de jeu permettant de jouer aux échecs en local avec un ami.

Pour cela il nous fallait premièrement créer une interface graphique intégrant un design intuitif afin de garantir une bonne expérience utilisateur et de l'intégrer sur une page web. L'enjeu ici était de nous initier aux langages HTML & CSS afin de réaliser une page structurée et au visuel « responsive », c'est-à-dire un site web réactif qui s'adapte à toutes tailles d'écrans sans déformation du contenu de la page web.

Exemple de plateau d'échecs en ligne [chess.com]

Dans un second temps notre mission était de mettre en place des déplacements des pions sur le plateau d'échecs et en parallèle de vérifier si le mouvement souhaité était valide, autrement dit qu'il respectait les règles du jeu d'échecs, ici l'enjeu était de s'initier au langage JavaScript.

Choix Technologiques :

Premièrement, pour réaliser l'interface graphique qui contient : le plateau de jeu, les pions, et les différents boutons pour relancer une partie ou encore retourner à l'accueil, nous devions choisir entre trois méthodes différentes :

- L'utilisation des CANVA
  Les CANVA sont très utilisés sur les sites web utilisant le javascript, car ils permettent de dessiner ainsi que de redessiner par-dessus un élément existant. Par exemple, le site « chess.com » utilise des canvas pour dessiner l'échiquier, ce qui permet d'intégrer des flèches par-dessus les carreaux existants. Cependant, cette méthode ne présente pas d'avantages réel pour la réalisation de notre jeu d'échecs.

- L'importation d'une simple image pour le plateau
  L'utilisation d'une image peut être une solution simple et efficace pour afficher un échiquier et les pions. Cependant, l'utilisation d'images ne permet d'identifier les différentes cases individuellement lors d'une sélection de pions par exemple.

- Créer chaque division/carreaux individuellement
  La troisième solution consiste à créer, à l'aide de boucles, les 64 cases ce qui permet de leurs attribuer à chacune un id, permettant d'identifier les pions sélectionnés.

Compte tenu du temps attribué au projet et de notre cahier des charges, nous avons décidé de créer 64 cases individuelles pour pouvoir facilement se repérer sur l'échiquier grâce aux id. De plus, cette solution est plus accessible, comparé aux canvas qui est une notion plus complexe sachant qu'elle n'apporte pas d'avantages à notre réalisation.

Réalisations :

Génération du plateau :

La première de nos réalisations a été de générer une grille de 64 cases qui sera ici notre plateau de jeu. De plus nous avons allié à cela le fait d'alterner la couleur de chaque case avec des classes (une noir, une blanche, ainsi de suite...) afin de pouvoir potentiellement modifier le design via le CSS, mais aussi l'attribution d'un ID pour chaque case. La fonction « initializeBoard() » ci-dessous utilise deux boucles « for » pour gérer les 8 colonnes et les 8 lignes de notre échiquier.

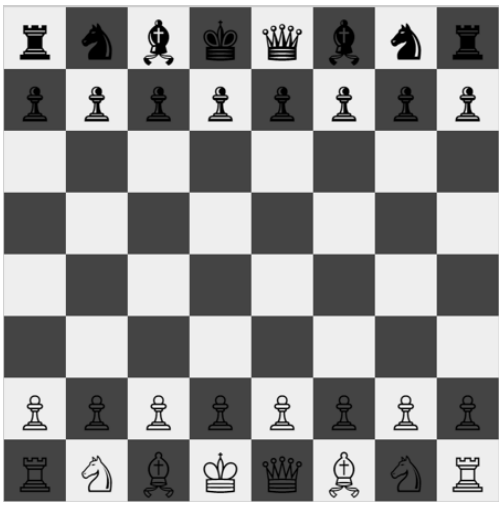
```
function initializeBoard() {
    board.innerText = '';
    for (let row = 8; row >= 1; row--) {  // Commencez avec 8 pour être compatible avec chess.js
        for (let col = 1; col <= 8; col++) {  // Commencez avec 1 pour être compatible avec chess.js
            const square = document.createElement('div');
            square.classList.add('square', (row + col) % 2 == 0 ? 'black' : 'white');
            square.innerText = '';
            square.setAttribute('id', String.fromCharCode(col + 96) + row); // Convertit le numéro de colonne en lettre (a-h)
            setInitialPieces(square, row, col);
            board.appendChild(square);
```

Initialisation du jeu :

Il nous a fallu par la suite créer l'initialisation du jeu sur le plateau, autrement dit placer les pièces dans la configuration initiale d'un début de partie, pour cela nous nous sommes servis des ID attribué à chaque case pour placer les pions au bon endroit, exemple ci-dessous avec les cavaliers

```
if ((id === 'b1' || id === 'g1')) piece = '♘';     // White Knight
if ((id === 'b8' || id === 'g8')) piece = '♞';     // Black Knight
```

Placement des cavaliers blancs & noirs à leurs cases respectives



Plateau d'échecs en configuration initiale

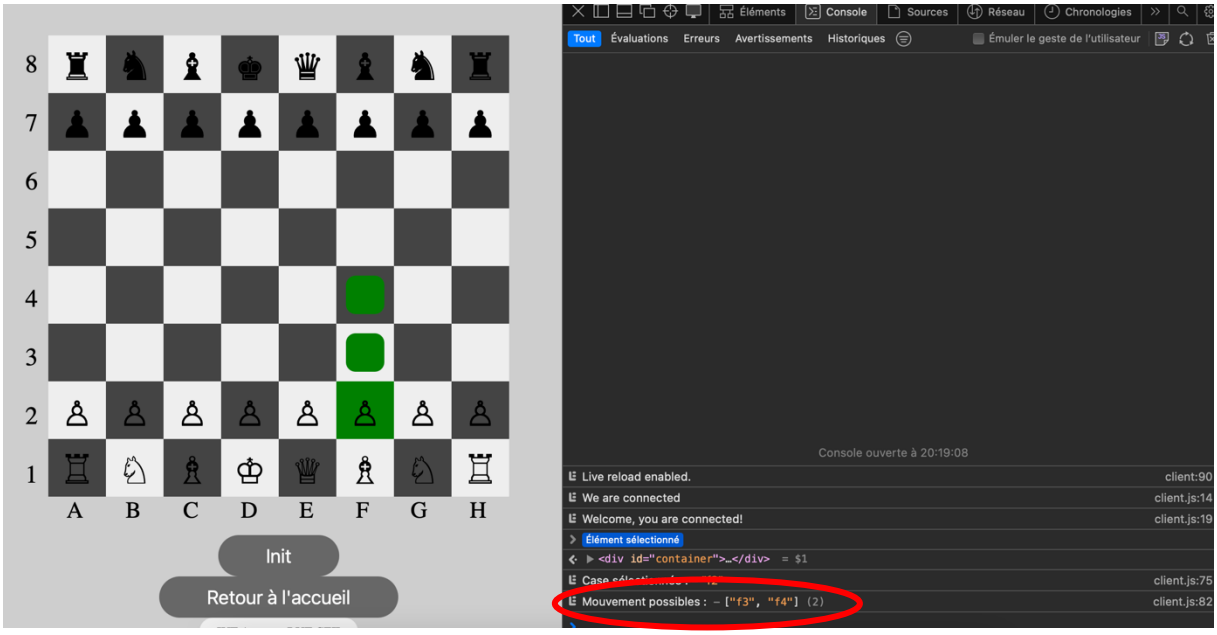
Déplacement des pièces :

Nous avons ensuite réalisé le déplacement des pièces sur le plateau, pour se faire nous avons utilisés la librairie chess.js, une librairie JavaScript contenant tout le nécessaire pour réaliser un jeu d'échecs dans ce langage. Cependant, il fallait bien vérifier que les ID attribuées, sont les même utilisées dans la librairie chess.js. Une fois, cela fait, nous pouvons donc utiliser chess.js pour vérifier que le mouvement réalisé par le joueur respecte les règles du jeu.

Première méthode :

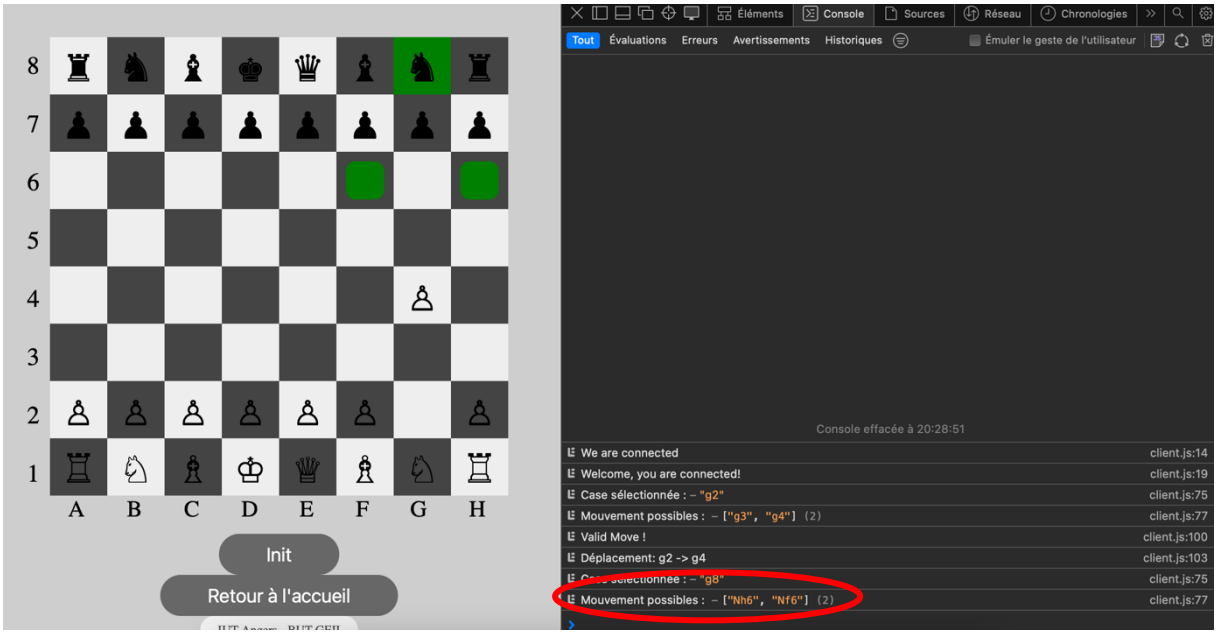
La première méthode que nous avons essayé, consiste à récupérer la liste des mouvements possibles lorsque l'on sélectionne un premier pion grâce à la fonction « moves()[1] » de chess.js. Puis vérifier si la case deuxième case sélectionnée appartient à ces mouvements.

Avec un console.log, nous pouvons afficher dans le terminal la liste des mouvements possibles après la sélection de la première case :



Nous avons ensuite ajouté des classes aux endroits où le pion peut se déplacer (s'il y en a) pour indiquer aux joueurs leur possibilités, ainsi qu'une classe (case verte) pour indiquer la première case sélectionnée.

Pour les pions simples c'est fonctionnel. Mais dans certains cas, les mouvements possibles retournés peuvent être différent de l'ID d'une case.



[1] : « moves() » retourne les mouvement possibles en fonction de l'ID passé en paramètre.
« move() » effectue le mouvement et met à jour l'échiquier (seulement dans les données et non pas graphiquement)
Ce sont deux fonctions différentes !

Le programme tel qu'il est conçu considèrera donc que le mouvement est invalide.
Nb : Cette notation avec la lettre correspondant que pion suivis de la case d'arrivé s'appelle le « SAN ».
Cette notation pose donc un problème lorsque l'on utilise cette méthode.

Deuxième méthode :

La deuxième méthode consiste à utiliser le « try…catch ».

Le try catch est très utilisé en javascript. Il permet d'essayer par exemple une fonction (qui sera insérée dans le « try »), puis avec le « catch » nous pouvons réaliser une suite de d'ordre qui se réalisera si et seulement si, l'élément passé en paramètre de catch est retourné par la ou les fonctions exécutées dans le try.

Dans notre cas, nous allons exécuter la fonction « move()[1] » avec le déplacement souhaité par le jouer passé en paramètre. Si la fonction retourne une erreur c'est que le déplacement est « illégal », dans ce cas nous ne déplaçons pas le pion. Sinon, dans le cas où aucune erreur n'est retournée, nous mettons à jour l'interface graphique (l'échiquier) et passons au tour suivant.

Voici ces quelques lignes de codes qui déterminent si le pion doit être déplacé ou non :

```
let valid = 0;
try {
    chess.move({ from: selected_square.id, to: square.id });
} catch (error) {
    console.log("Invalid Move");
    valid = 1;
}
```

Cette méthode est bien plus simple mais aussi plus efficace.

Le « Try..Catch » est vraiment un moyen très simple et clair pour réaliser quelque chose en fonction du résultat que retourne une fonction.

De plus, comparé à la première méthode, nous n'avons pas à appeler créer des variables pour stocker les résultats d'une fonction, ou utiliser plusieurs conditions pour vérifier le déplacement. Ici, nous « forçons » simplement chess.js à réaliser un mouvement, et si la fonction nous avertit qu'il est « illégal », nous ne le réalisons pas.

Partie CSS & Responsive Design :

Afin que le rendu final de notre projet soit le plus qualitatif possible, nous devions de nous concentrer sur des éléments de finissions dans la fin du temps qui nous était imparti, nous nous sommes donc chargés premièrement de rendre tout notre design harmonieux et cohérent, afin que tout aille bien ensemble et soit dans le même thème, pour cela nous avons par exemple fait des classes CSS pour que tous les boutons aient le même design.

Dans un second temps nous nous sommes chargés de rendre le design réactif (communément appelé responsive design). Concrètement cela consiste à adapter l'emplacement de tous les éléments

[1] : « moves() » retourne les mouvement possibles en fonction de l'ID passé en paramètre.
« move() » effectue le mouvement et met à jour l'échiquier (seulement dans les données et non pas graphiquement)
Ce sont deux fonctions différentes !

CSS qui constituent notre page web afin leur emplacement s'adapte en fonction de la taille de la page et/ou de l'écran sur laquelle elle est affichée. Cela évite que les éléments s'écrasent ou alors qu'ils se chevauchent lors d'un redimensionnement de page.

Pour faire simple, rendre son design réactif résulte majoritairement dans le fait d'appliquer des pourcentages sur les tailles et marges des éléments plutôt que des tailles et distances pré définis en pixel, cm, etc… Cela permet ainsi à la page de s'adapter proportionnellement à la taille de la fenêtre.

[1] : « moves() » retourne les mouvement possibles en fonction de l'ID passé en paramètre.
« move() » effectue le mouvement et met à jour l'échiquier (seulement dans les données et non pas graphiquement)
Ce sont deux fonctions différentes !

Perspectives :

Dans ce projet, nous avons su réaliser toute la partie graphique de notre jeu d'échecs ainsi que la partie « mécanique du programme », autrement dit le déplacement des pièces ainsi que la validation des mouvements. Nous avons à cela ajouté quelques détails tel que l'affichage des mouvements possibles ou encore un design réactif.

Néanmoins, si l'on se projette dans une optique d'amélioration pour ce projet, nous aurions par exemple pu héberger tout le projet sur un serveur afin de mettre en place une communication par websocket.

Cette amélioration aurait notamment pu nous permettre d'ajouter un mode de jeu contre un ordinateur, une IA. StockFish, un moteur de jeu d'échecs très connu pour sa capacité à prévoir un nombre de coup d'avance spectaculaire le rendant donc imbattable aurait très bien pu être intégré à notre jeu. Nous l'aurions fait tourner sur le serveur et l'aurait fait dialoguer par websocket avec notre client afin qu'il récupère les informations de chaque coup et nous retourne le meilleur coup à jouer puis, mettre à jour l'interface grapghique.

Pour finir, nous aurions également pu améliorer la structure de notre code en l'ordonnant de la façon suivante avec la méthode connue : (modèle, vue, contrôleur). Nous aurions eu le « modèle » qui serai la librairie chess.js, cette dernière contenant une bonne partie des fonctions de notre mécanique de jeu. La « vue » quant à elle contiendrait tous les éléments relatifs aux mises à jour de l'interface graphique, tous les mouvements de pièces à chaque coup joué, ou bien la surbrillance des mouvements possibles par exemple. Le « contrôleur » quant à lui aurait été là pour orchestrer tout le code, rediriger vers les fonctions de la librairie chess.js et de notre vue, etl aurait aussi effectué les communications entre le serveur et le client si nous l'avions mis en place.

Conclusion :

Ce projet ayant été mené à son terme nous pourrions dire qu'il s'est parfaitement déroulé. Néanmoins restons critique et soulignons que pour mieux faire nous aurions pu faire preuve de plus d'organisation sur le lancement du projet afin de ne pas partir sur trop de pistes diverses comme nous l'avons fait.

Si nous avions des conseils à donner à de futurs repreneurs du projet par exemple, nous conclurons que, pour mener au mieux un tel projet il est important de structurer ses recherches et ses phases de développement. Dans notre cas il nous a fallut apprendre les bases de l'HTML et CSS et l'appliquer sur notre projet, il est donc ici important de ne pas se précipiter à vouloir appliquer trop rapidement quelque chose que l'on ne maitrise pas. En revanche il est vrai que ces deux langages étant de type « descriptif » ils sont assez intuitifs à prendre en main. Il en est de même pour une potentielle future intégration de stockfish sur un serveur par exemple, il serait ici important de comprendre le fonctionnement de stockfish en ligne de commande dans un premier temps, afin de savoir de quelle manière il reçoit ou renvoie les FEN.

Resumé :

Ce projet de jeu d'échecs numérique est un projet de SAE réalisé par Raimbault Mathéo & Gaiffe Killian durant leur troisième semestre. Ils avaient pour objectif de créer un jeu d'échecs hébergé sur un site web afin de pouvoir jouer en local contre un ami, l'enjeu principal était ici de s'initier aux trois langages de programmations suivants : HTML, CSS & JavaScript.

Resume :

This digital chess game project is an SAE project having been carried out by Raimbault Mathéo & Gaiffe Killian during their third semester. His objective was to create a chess game hosted on a website to be able to play locally against a friend, the main challenge here was to learn the following three programming languages: HTML, CSS & JavaScript.