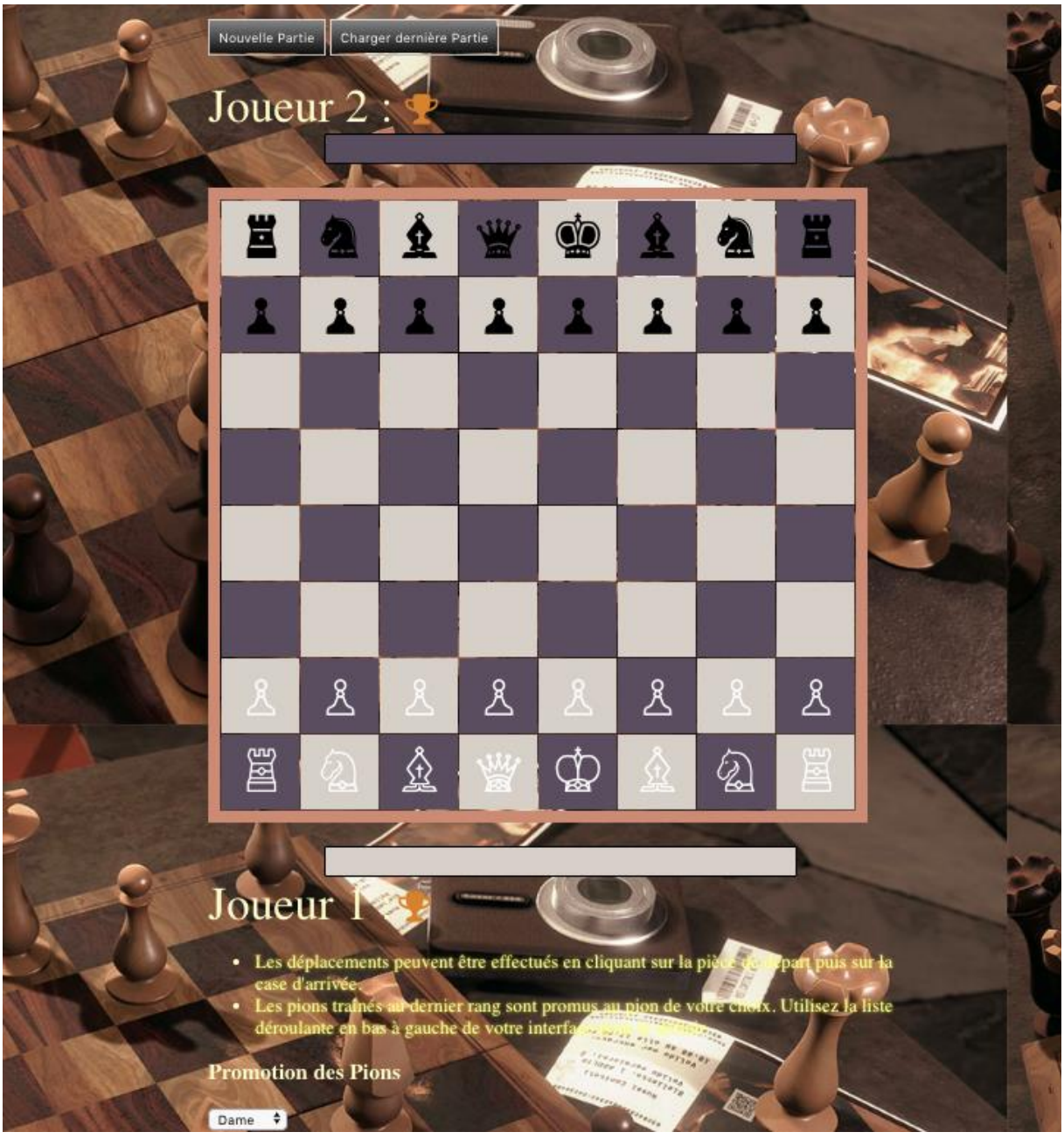
Rapport du projet réalisation d'un jeu en Java Script
Jeux d'échecs
Deux joueurs sur un même écran

Gestion de projets web 2018/2019 - Encadré par M. BUFFA



Projet réalisé par : Lys Ciella Niteka - TRAORE Mohamed - BOUREK Kamel

Introduction

L'objectif de ce projet est de réaliser un jeu d'échecs en JavaScript. On rappelle que ce jeu se déroule sur un échiquier de 8 cases par 8 cases, alternant cases grises et cases blanches. Deux joueurs humains (on ne s'intéressera pas ici à la réalisation d'une éventuelle intelligence artificielle), chacun d'entre eux doté de 8 pièces et 8 pions, s'affrontent sur cet échiquier. Chacun dispose d'un roi, et l'objectif pour son adversaire est de le mettre en échec et mat.
Les déplacements se feront avec des cliques case de départ et cliques case d'arrivée.

Fonctionnalités du jeu

- Jeux pour deux joueurs sur un même écran.
- Surbrillance de toutes les cases possibles pour chaque coup.
- Affichage des pièces capturées du coté du joueur adverse.
- Message Vocal et visuel alertant l'échec en spécifiant la couleur en question.
- Message Vocal et visuel alertant l'échec et mat en spécifiant la couleur en question.
- Message Vocal et visuel alertant le pat (une façon de faire un Match Nul).
- Possibilité de sélectionner la promotion des pions à partir d'une liste déroulante.
- Un bouton qui permet de lancer une nouvelle partie.
- Un bouton qui permet de récupérer une partie non terminée dans son état où elle a été laissée grâce au Local Storage.

Compatibilité Navigateurs

- Notre jeu fonctionne parfaitement sur google chrome.
- Fonctionne parfaitement sur Safari macOS.
- Fonctionne parfaitement sur Internet Explorer.

Choix de conception et solutions proposées

Les cases ont été construites comme des éléments td, dont les attributs contiennent les informations comme la ligne, la colonne et la couleur de la case. Si la case contient une pièce on met donc dans des attributs, la couleur et le type de la pièce.

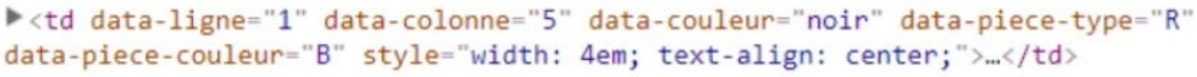


Figure 1 : Inspection d'une case de l'échiquier contenant le Roi blanc

Dans le but de faciliter l'évolution du travail on s'est penchés sur l'utilisation des classes en JavaScript. Nous détailleront ci-après la conception et la justification des classes implémentées (variables et méthodes):

Une classe **Echiquier** qui spécifie un échiquier contenant toutes les informations nécessaires pour représenter une configuration donnée de ce dernier.

Cette classe contient également des méthodes qui nous ont permis à leur tour d'ajouter des fonctionnalités pour le jeu. Ces méthodes sont:

· Une méthode initialiserPartie qui initialise un échiquier.
· Une méthode afficher qui permet l'affichage de l'échiquier sur l'interface graphique.
· Une méthode qui permet d'activer le déplacement de toutes les cases de l'échiquier.
· Une méthode surbrillerCases qui permet de donner tous les déplacements possibles d'une pièce après avoir cliqué sur cette dernière.
· Une méthode sauvegarderEtatPartie qui permet de sauvegarder l'état d'une partie dans le localStorage.
· Une méthode sauvegarderCimetiere qui permet de enregistrer dans le localStorage les pièces mangées.
· une méthode chargerEtatPartie qui permet de charger la dernière partie enregistrée dans le localStorage.

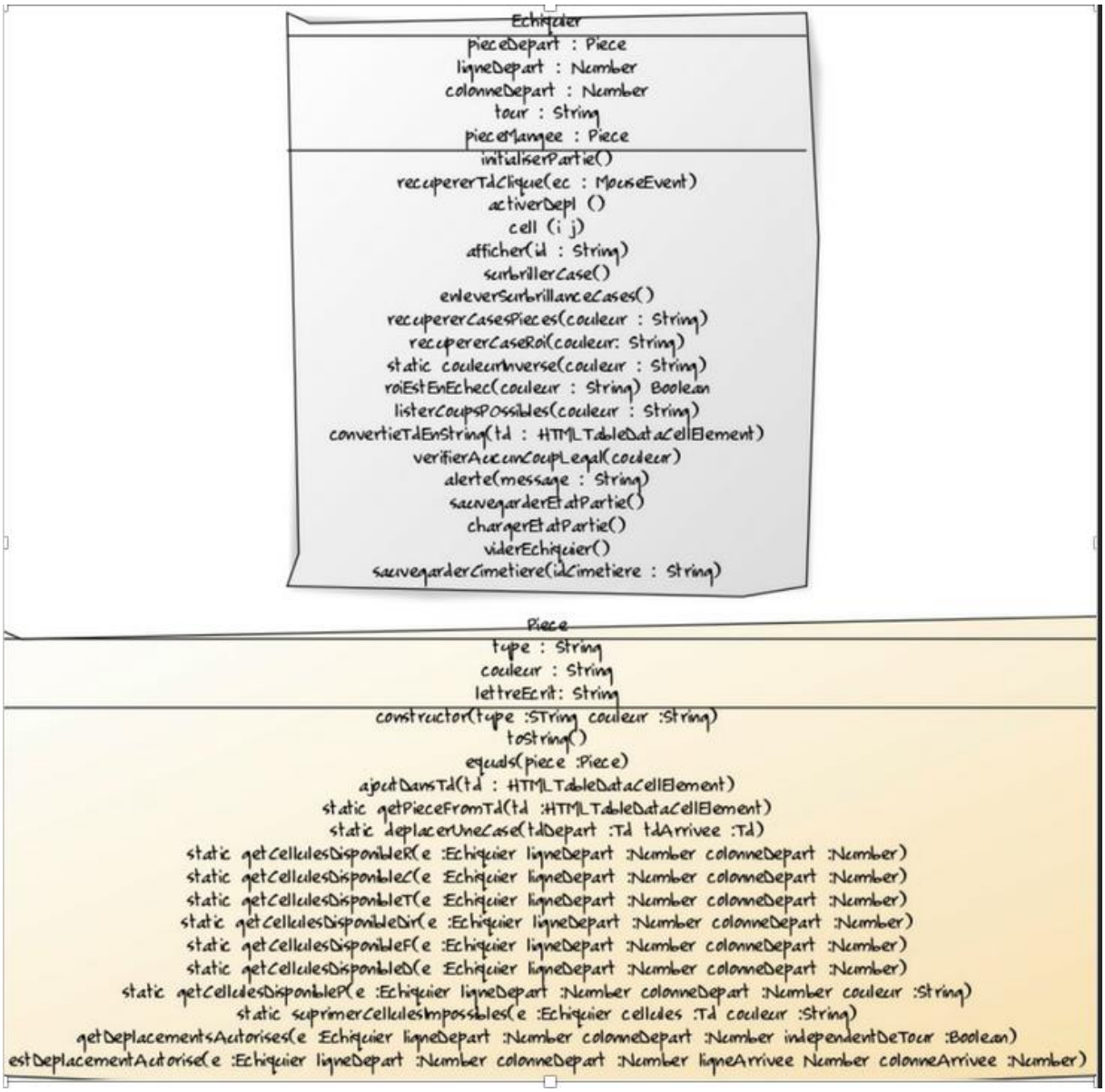
· Une méthode alerte qui prend en paramètre un message et l'affiche pendant 3 secondes et le prononce vocalement. Pour le message vocal on a utilisé speechSynthesis.

Pour être précis et clairs dans notre conception, nous avons décidé de créer une seconde classe nommée **Piece** qui va s'occuper de la représentation des pièces ou des pions.

Les méthodes de cette classe sont:

· une méthode toString qui permet de remplacer la représentation textuelle des pièces par des polices d'échecs. Nous avons utilisé la police Chess-7 pour dessiner les pièces comme du texte.
· une méthode equals qui vérifie si deux pièces sont équivalentes .
· une méthode getPieceFromTd qui permet de récupérer une pièce à partir d'une case.
· une méthode pour déplacer une case vers une autre case.
· une méthode ajoutDansTd qui permet de ajouter une pièce dans une case.
· une méthode getCelluleDisponibleX qui renvoie toutes les cases disponibles pour chaque type de pièces. (X représente le type de la pièce)
· une méthode getDéplacementsAutorisés qui permet de retrouver tous les déplacements disponibles pour une pièce peu importe le type et la couleur.
Ainsi que d'autres méthodes vérifiant la validité d'un coup pour une pièce donnée.

Représentation UML des classes Echiquier et Piece



Conclusion

La partie qui nous a demandé le plus d'efforts était de faire l'échec et mat. Mais on est parvenus à le coder finalement grâce à beaucoup d'efforts et de recherches pour contourner les difficultés rencontrées. On est globalement satisfaits du résultat final de notre projet. Notre jeu marche parfaitement sans aucun bug.
Une fois la partie codage terminée, on a travaillé un peu sur le css afin de rendre l'interface plus agréable.

On remercie M. BUFFA de nous avoir donné les pré-requis nécessaires en cours et de nous avoir aidé sur notre projet et l'avoir guidé dans la bonne voie.