

Projet Théorie Des Jeux

Jeu QUARTO



Membres du groupe :

- Houssem AYOUB
- Younes HADDAM

2021/2022

Code Source : https://github.com/Younext19/Quattro_Game

0. Installation

Étapes à suivre pour faire fonctionner le projet, exécutez sur un terminal ou cmd :

- git clone git@github.com:Younext19/Quattro_Game.git (Ceci est un lien de clone SSH)
- cd quartogame/
- npm install
- npm start

1. Introduction

Quarto est un jeu de société combinatoire abstrait au tour par tour, créé par Blaise Mullet primé en 1985 au Concours international de créateurs de jeux de société de Boulogne-Billancourt.

2. But du jeu

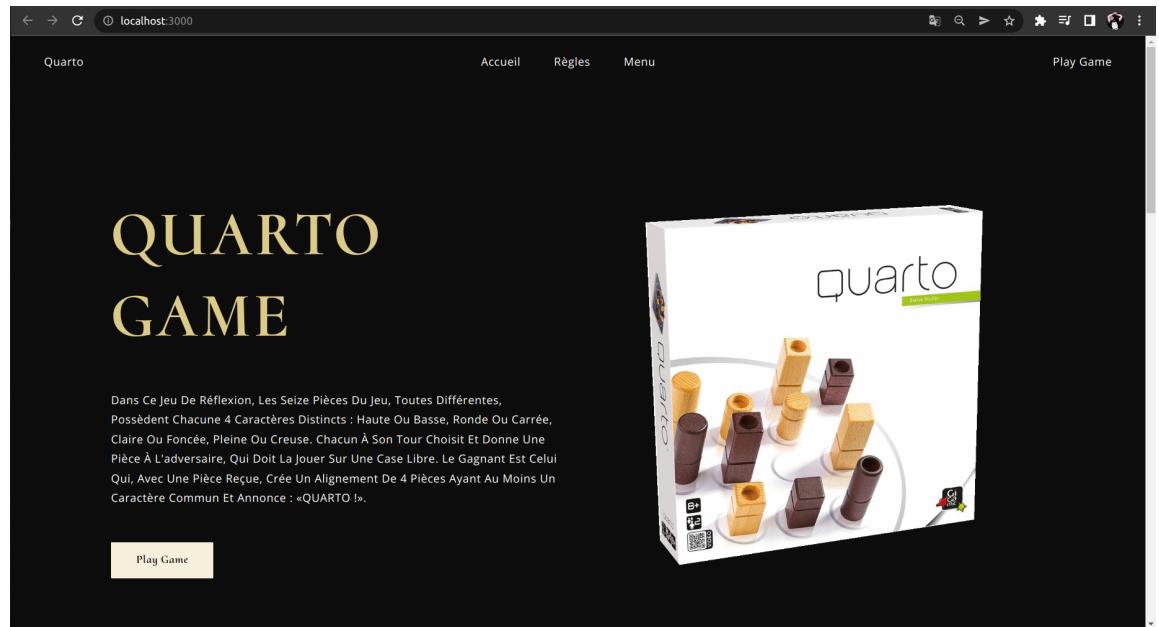
Créer sur le plateau un alignement ou un carré de 4 pièces ayant au moins un caractère commun. Chacune des 16 pièces possède quatre caractères distincts. Mais ce n'est pas vous qui choisissez la pièce que vous placez, c'est l'adversaire qui vous la donne.

3. L'interface

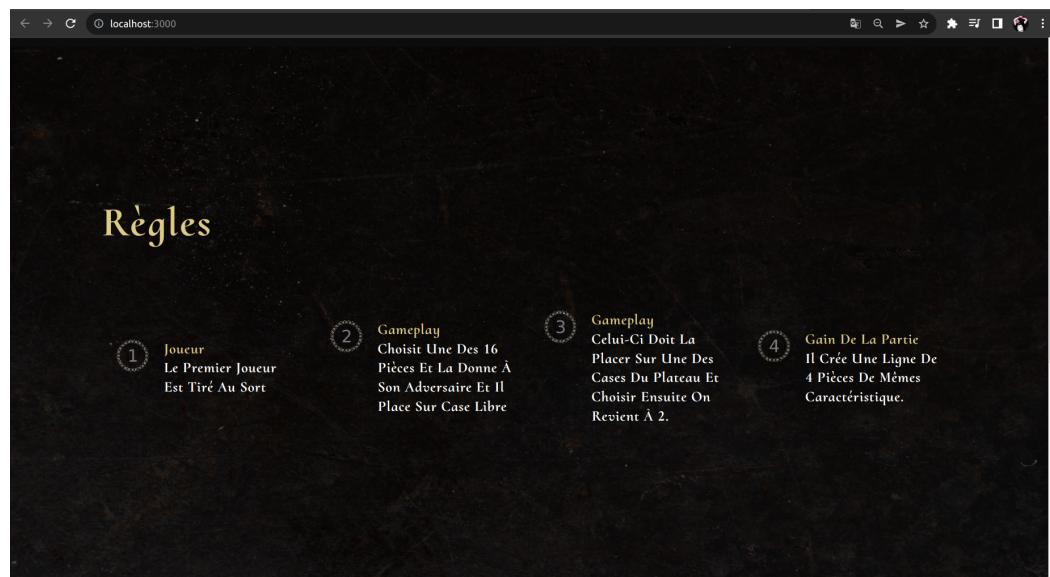
a. Page 1 (Accueil)

Sur cette page on peut voir une petite présentation sur le jeu ainsi que les règles du jeu et une petite démonstration vidéo.

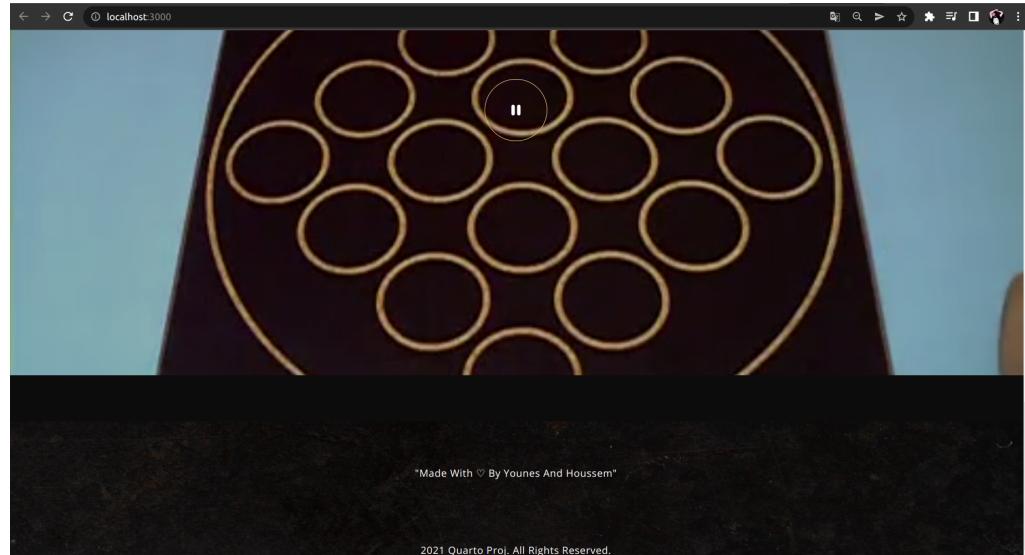
Partie 1 : Ici on peut voir une petite présentation sur le jeu et des boutons “Play Game” qui vont vous permettre de commencer à jouer



Partie 2 : Ici on peut voir les règles basiques du jeu.

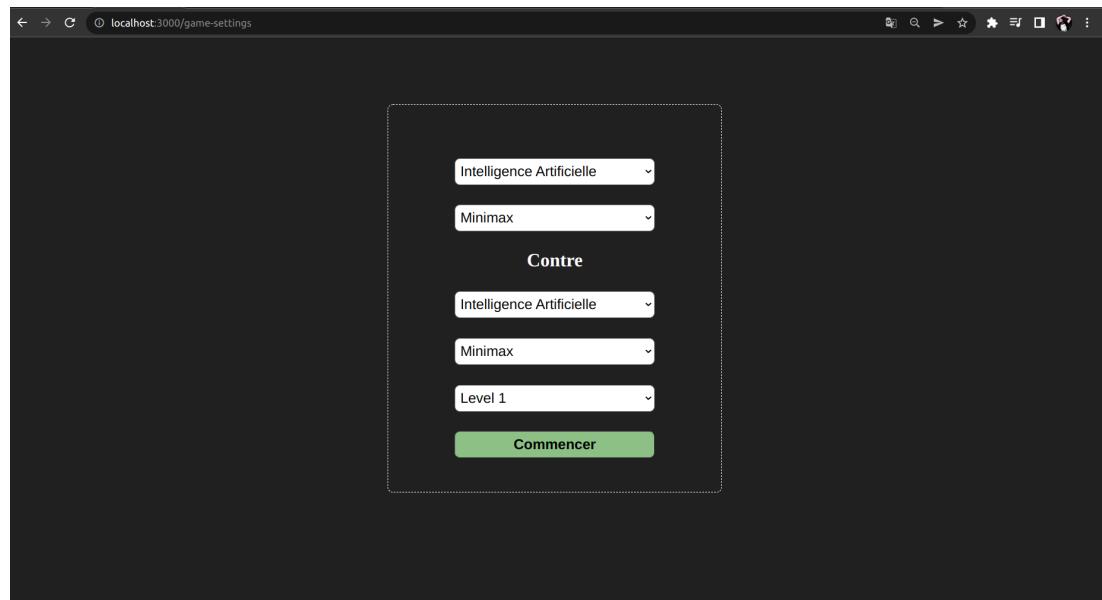


Partie 3 : Ici on peut voir une petite vidéo qui fait la démonstration du jeu.



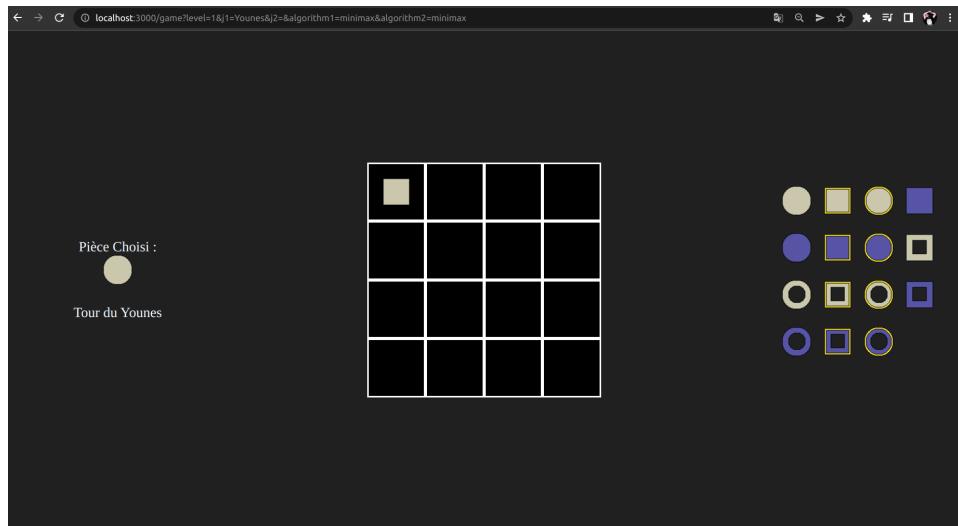
b. Page 2 (Configuration)

La deuxième page est une page pour faire la configuration du jeu et choisir l'algorithme souhaité pour l'intelligence artificielle si ce n'est pas un joueur qui va jouer. Aussi sélection du niveau du jeu.



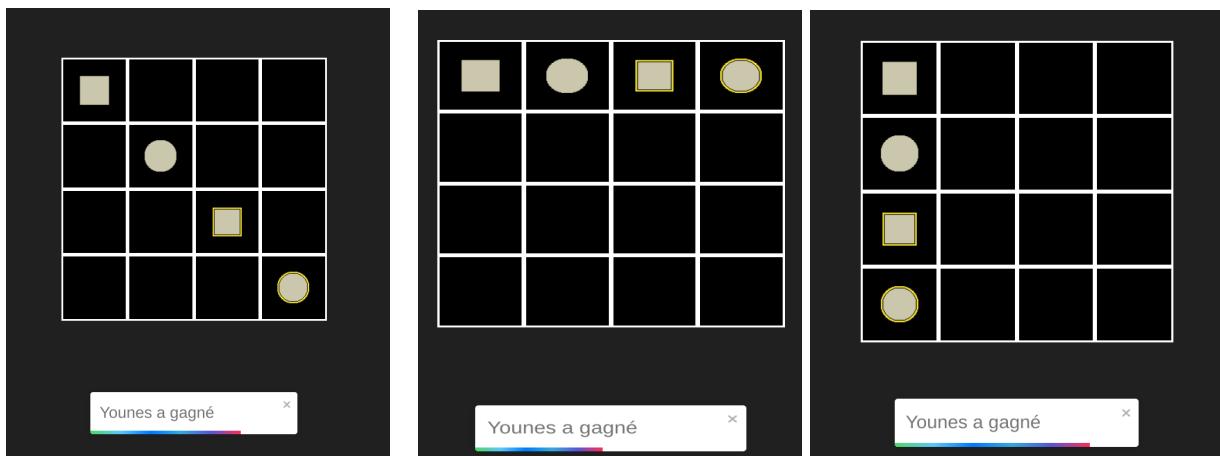
c. Page 3 (Jeu)

Arrivé à cette page ? Vous pouvez commencer à jouer, pour les pièces on n'a pas trouvé plus beau.
Pour les tailles les longues piece sont bordé en jaune



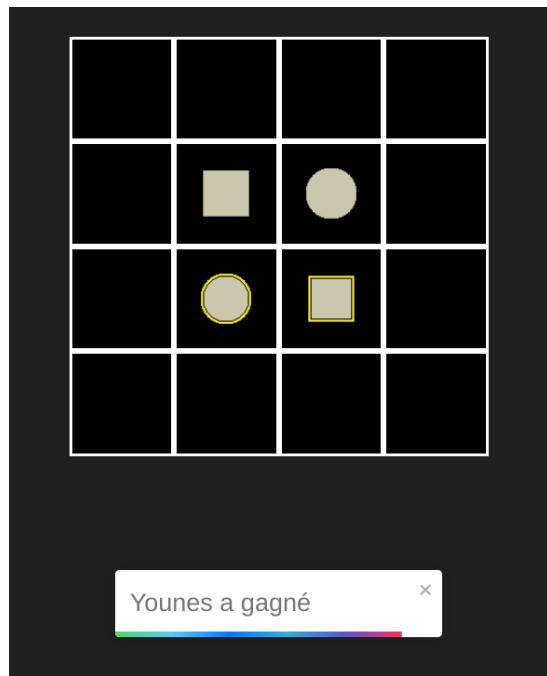
i. Niveau 1

Le Joueur gagne en diagonale, horizontale, verticale comme le montre les photos ci-dessous :



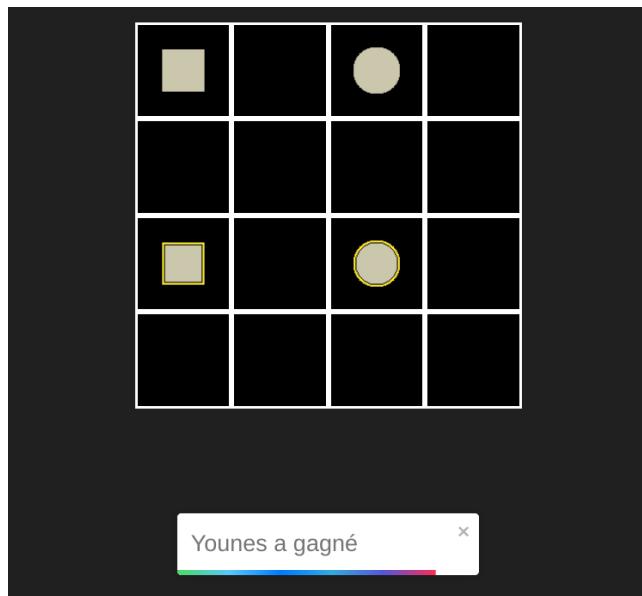
ii. Niveau 2

Le niveau 2 c'est un carré comme le montre la photo



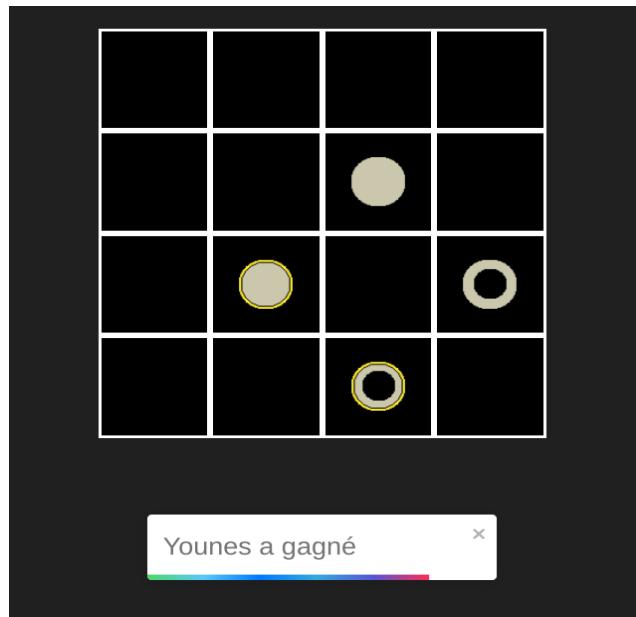
iii. Niveau 3

Pour le niveau 3 c'est un grand carré par exemple:



iv. Niveau 4

Un carré tournant



4. Données & Structures:

Pour le damier 4 x 4 on a utilisé une matrice 4x4 sous cette forme :

```
[  
    [null, null, null, null],  
    [null, null, null, null],  
    [null, null, null, null],  
    [null, null, null, null],  
];
```

Création des nodes et child et pour savoir next move (vous pouvez voir sur le fichier /strategy/Node.js

5. Algorithme & Logique

a. Configuration gagnante

Pour voir si c'est une configuration gagnante ou pas, on a utilisé des booléens pour chaque pièce spécifiée par leur couleur, taille, forme, trouée ou pas comme le montre l'exemple suivant :

```
{trou: false, black: true, circular:false,bordered:false}
```

Ainsi on a cette fonction qui fait la vérification si on a les 4 valeurs du tableau ont un même paramètre true pour toutes les pièces dans la ligne ou diagonale ou verticale ou pour les autres niveaux (2,3,4).

```
function check(p1, p2, p3, p4) {  
    return (  
        (p1.trou === p2.trou && p3.trou === p4.trou && p1.trou ===  
        p3.trou) ||  
        (p1.black === p2.black && p3.black === p4.black && p1.black  
        === p3.black) ||  
        (p1.circular === p2.circular &&  
        p3.circular === p4.circular &&  
        p1.circular === p3.circular) ||  
        (p1.bordered === p2.bordered &&  
        p3.bordered === p4.bordered &&  
        p1.bordered === p3.bordered)  
    );  
}
```

b. Minimax

Afin de pouvoir jouer contre l'Ordinateur, on utilisera l'algorithme Minimax.

l'algorithme Minimax effectue l'évaluation pour tous les nœuds de l'arbre de jeu d'un horizon donné.

Le but principal de cet algorithme est de renvoyer le meilleur coup à un instant précis. Pour cela, l'algorithme va

simuler toutes les issues possibles, à partir de la situation actuelle jusqu'à plus ou moins loin dans le futur. Une fois les prévisions faites, celui-ci doit faire remonter le coup menant à l'issue la plus intéressante pour l'ordinateur.

Au premier tour on aura 16 coups possibles, qui représente la pièce qu'on choisit de jouer pour commencer. Au deuxième tour, on doit alors poser cette pièce quelque part sur le plateau et choisir celle que posera l'adversaire. On a donc 16 positions et 15 pièces possibles, fois les 16 premiers coups, on a donc sur les premiers tours : $16 * 16 * 15 = 3840$ issues juste pour les deux premiers tours! Puis 806400 si on continue un tour de plus...

C'est pour cela qu'on fera le fonctionnement au début avec moins de possibilité.

Voici l'algorithme Minimax utilisé dans notre cas (
/quartogame/src/strategy)

```
export class MinMaxStrategy extends Strategy{
    resolve(node,depth) {
        if(node.isLeaf || depth ===0)
            return node
        else {
            if(node.isMax){
                return node.nextChildren().reduce((acc,e) => {
                    const val1 = this.resolve(acc, depth-1)
                    const val2 = this.resolve(e,depth-1)
                    return val1.h>=val2.h ? val1 : val2
                })
            }
            else {
                return node.nextChildren().reduce((acc,e) => {
                    const val1 = this.resolve(acc,depth-1)
                    const val2 = this.resolve(e,depth-1)
                    return val1.h>=val2.h ? val1 : val2
                })
            }
        }
    }
}
```

c. Alpha Beta

Basé sur Minimax, l'algorithme Alpha Beta, est une amélioration qui réalise un élagage de certaines branches qu'il est inutile de visiter.