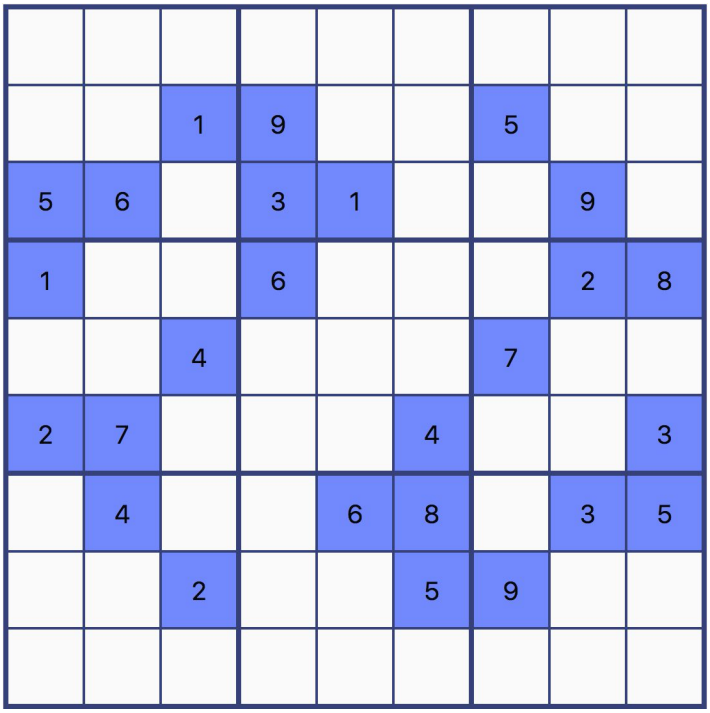
Dossier PWA



Manon FLORES - Flavie KOWAL - Inès PIEDERRIERE

Paris School of Business : M. STROPPA

15.06.2020

Voici notre rapport de code d'application PWA. Nous avions choisi de coder un sudoku diagonal.

Le Principe

Un **sudoku** classique contient neuf lignes et neuf colonnes, donc 81 cases au total. Le but du jeu est de remplir ces cases avec des chiffres allant de 1 à 9 en veillant toujours à ce qu'un même chiffre ne figure qu'une seule fois par colonne, une seule fois par ligne, et une seule fois par carré de neuf cases. Le sudoku diagonal embrasse la même logique dans une diagonale.

Au niveau de la structure technique, trois éléments composent le sudoku : lignes, colonnes, et box. Le but est alors de remplir une box avec un des chiffres manquants, en y énumérant les différentes possibilités (en ajoutant 1,2,3,4, …). Ensuite, nous vérifions quelle solution est bonne ou mauvaise, grâce à la mise en relation des row et des colonnes.

Pour imager : quelques exemples de fonctions expliquées…

Ici, nous expliquerons quelques unes des fonctions de notre jeu Sudoku.

1) Fonction rechercheSolution(tabs){

Elle permet de trouver une solution valide au problème.

Ici, dans le cas du sudoku, une solution est un chiffre entre 1 et 9. La solution valide au problème est trouvée grâce à :

if (tabs.length < 1){

return false (Si le tableau est vide)

} else {

Le code qui permet cette recherche de solution procède étape par étape c'est-à-dire en testant les éléments un par un. Si la valeur testée est correcte, nous passons au suivant, si elle est incorrecte, nous revenons au précédent afin d'y tester un autre chiffre :

var first = tabs.shift() → *Prend le premier élément*

const tryPath = solve(first) → *Essaie de résoudre avec le premier élément*

```
    if (tryPath != false){
        return tryPath  ⟶ On a trouvé
    }else{
        return rechercheSolution(tabs)  ⟶ On recommence
    }
}
```

2) Fonction boxesBon(tab){

Nous affectons des coordonnées à chaque chiffre dans la box :

```
const boxCoordinates = [[0, 0], [0, 1], [0, 2],
                        [1, 0], [1, 1], [1, 2],
                        [2, 0], [2, 1], [2, 2]]
```

Afin de s'assurer qu'il n'y ait pas de nombre qui se répète au sein d'une même boîte, nous codons :

```
for (var y = 0; y < 9; y += 3){
    for (var x = 0; x < 9; x += 3){
```

Nous examinons toutes les boîtes avec une double boucle pour les deux coordonnées :

```
        var cur = []
        for (var i = 0; i < 9; i++){
            var coordinates = [...boxCoordinates[i]]
            coordinates[0] += y
            coordinates[1] += x ⟶ On incrémente ici on rappelle que coordinates[1] = x + coordinates[1]
            if (cur.includes(tab[coordinates[0]][coordinates[1]])){
                return false ⟶ Au cas où les coordonnées retournent quelque chose de "null" on établit  alors dans une fonction if
            }
```

```
            else if (tab[coordinates[0]][coordinates[1]] != null){
                cur.push(tab[coordinates[0]][coordinates[1]])
            }
        }
    }
  }
  return true
}
```

Voici donc quelques exemples qui imagent le code que l'on a pensé être juste pour un sudoku. Notre fonction diagonale n'a cependant pas fonctionné : nous pensons en connaître la raison, sans pour autant avoir réussi à résoudre le problème. Ce point est détaillé dans les commentaires du code directement.

Enfin, le code que nous avons créé contient une erreur, qui s'oppose en ce sens à la bonne exécution du jeu. De ce fait, nous n'avons pas réussi à y ajouter les éléments pwa correspondants.