


# Manipuler le DOM avec JavaScript



**ELAN**

14 rue du Rhône - 67100 STRASBOURG

☎ 03 88 30 78 30 ✉ [elan@elan-formation.fr](mailto:elan@elan-formation.fr)

[www.elan-formation.fr](http://www.elan-formation.fr)

SAS ELAN au capital de 37 000 € -

RCS Strasbourg B 390758241 – SIRET 39075824100041 – Code APE : 8559A

N° déclaration DRTEFP 42670182967 - Cet enregistrement ne vaut pas agrément de l'Etat

# SOMMAIRE

<b>I.</b>	<b><i>Le DOM ? Kézaco ?</i></b>	<b>3</b>
1.	Définition	3
2.	En pratique	3
a.	Atteindre des éléments du DOM	4
b.	Modifier des éléments du DOM	6
<b>II.</b>	<b><i>Jouons avec le DOM !</i></b>	<b>10</b>
1.	Les règles du jeu	10
2.	Prérequis	10
a.	index.html	10
b.	style.css	11
3.	Le code JavaScript	12
a.	Faire apparaître les boîtes à l'écran	12
b.	Mélanger les boîtes aléatoirement	15
c.	Ajouter un écouteur d'évènement	17
d.	Prendre en charge les phases de jeu	21
<b>III.</b>	<b><i>Aller un peu plus loin !</i></b>	<b>23</b>
1.	Animer les boîtes valides	23
2.	Supprimer les messages d'alerte	24
a.	La fonction showReaction()	24
b.	Placement des appels à la fonction	25
<b>IV.</b>	<b><i>Conclusion</i></b>	<b>26</b>

# I. Le DOM ? Kézaco ?

## 1. Définition

Selon Wikipédia :

*Le Document Object Model (DOM) est une interface de programmation normalisée par le W3C, qui permet à des scripts d'examiner et de modifier le contenu du navigateur web. Par le DOM, la composition d'un document HTML ou XML est représentée sous forme d'un jeu d'objets – lesquels peuvent représenter une fenêtre, une phrase ou un style, par exemple – reliés selon une structure en arbre. À l'aide du DOM, un script peut modifier le document présent dans le navigateur en ajoutant ou en supprimant des nœuds de l'arbre.*

Autrement dit, le DOM est l'interface par laquelle JavaScript passera pour atteindre, ajouter, modifier ou supprimer les balises d'une page HTML. Chaque balise est un objet `HTMLElement` contenant les propriétés de style, de position, de taille et de comportement de la balise.

## 2. En pratique

Ainsi, l'exemple suivant représente un titre et une liste à puces au sein d'une page HTML 5 valide, vue sous différents aspects :

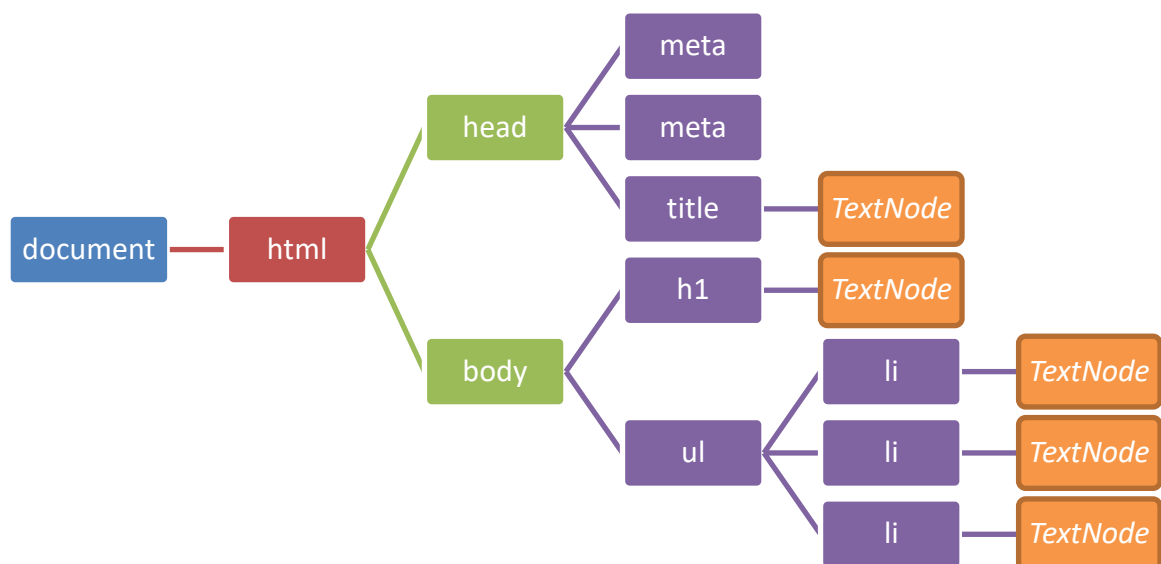
Le code du fichier :

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      <h1>Description du DOM</h1>
10     <ul>
11         <li>
12             le DOM est une interface de programmation
13         </li>
14         <li>
15             L'acronyme signifie : Document Object Model
16         </li>
17         <li>
18             Il représente les noeuds interconnectés (balises) d'une page HTML
19         </li>
20     </ul>
21 </body>
22 </html>
```

L'interprétation du navigateur (page affichée) :



Et sa représentation sous forme d'arbre (DOM Tree) :



Ainsi, pour atteindre un élément spécifique, JavaScript doit démarrer par le "document" (élément racine de la page) et parcourir l'arbre jusqu'à atteindre la cible. Une fois cette cible atteinte et stockée dans une variable, JavaScript peut depuis ce point se déplacer vers le parent ou les enfants de celui-ci.

#### a. Atteindre des éléments du DOM

Par exemple, si nous voulons atteindre la liste (ul) de la page, nous écrirons :

```

22     <script>
23
24         let list = document.querySelector("ul")
25
26     </script>

```

La méthode **querySelector()** de l'objet document permet de pointer le premier élément répondant à la requête spécifiée dans l'argument sous forme de chaîne de caractères. Cette requête s'écrit de la même manière qu'une règle CSS.

Néanmoins, si plusieurs ul étaient présentes sur la page, seule la première dans l'arbre sera pointée.

De ce fait, si nous souhaitons récupérer tous les éléments de la liste, voici le code :

```

22     <script>
23
24         let list = document.querySelector("ul")
25
26         let listElements = list.querySelectorAll("li")
27
28     </script>

```

Ici nous utilisons la méthode **querySelectorAll()** en partant de la variable list (et non plus du document) afin de **récupérer tous les éléments enfants de CETTE ul** répondant à la requête. Jetons un œil sur ce que la console nous présente si nous vérifions les valeurs de ces deux variables :

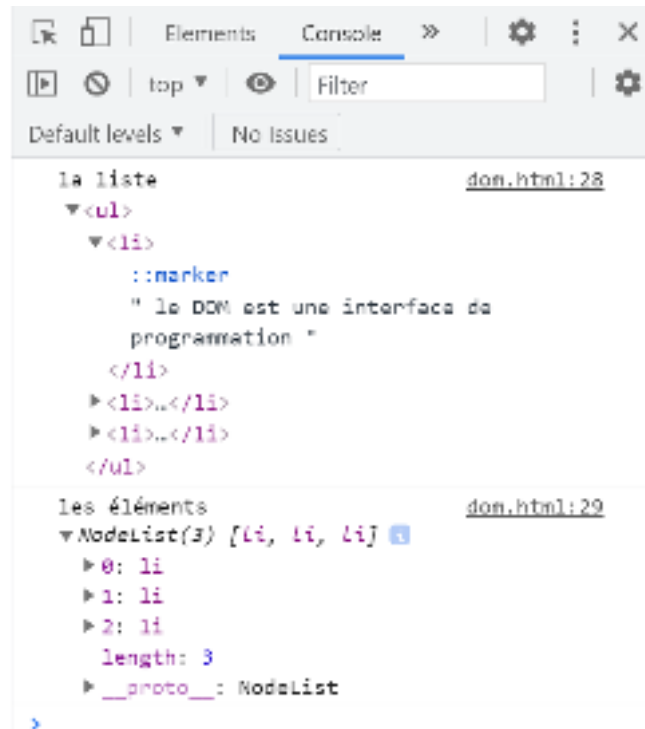
```

22     <script>
23
24         let list = document.querySelector("ul")
25
26         let listElements = list.querySelectorAll("li")
27
28         console.log("la liste", list)
29         console.log("les éléments", listElements)
30
31     </script>

```

## Description du DOM

- Le DOM est une interface de programmation
- L'acronyme signifie : Document Object Model
- Il représente les nœuds interconnectés (beliers) d'une page HTML.



La variable `list` présente une balise `ul` accompagnée de tous les nœuds enfants qu'elle contient. On constate également que chaque `li` à l'intérieur est composée d'un pseudo-élément `::marker` (la puce) et d'un **nœud de texte** (entre guillemets).

La variable `listElements` est sensiblement différente : elle comporte un objet `NodeList` contenant trois entrées (les trois `li`) dans un tableau indexé.

### b. Modifier des éléments du DOM

Nous pourrions effectuer une boucle sur la `NodeList` "`listElements`" pour, par exemple, les modifier une par une. Faisons cela pour changer la couleur du texte de chaque `li` en rouge :

```
22 <script>
23
24   let list = document.querySelector("ul")
25
26   let listElements = list.querySelectorAll("li")
27
28   listElements.forEach(function(element){
29     element.style.color = "red"
30   })
31
32 </script>
```

## Description du DOM

- Le DOM est une interface de programmation
- L'acronyme signifie : Document Object Model
- Il représente les nœuds interconnectés (beliers) d'une page HTML

JavaScript nous a ici permis de modifier la propriété `style` de chaque élément en effectuant une boucle `forEach()` sur la `NodeList`.

#### *Pour rappel :*

*`forEach` prend en paramètre une fonction dite "callback" (ou fonction de rappel) qui sera exécuté à chaque tour de boucle. Dans cette fonction*

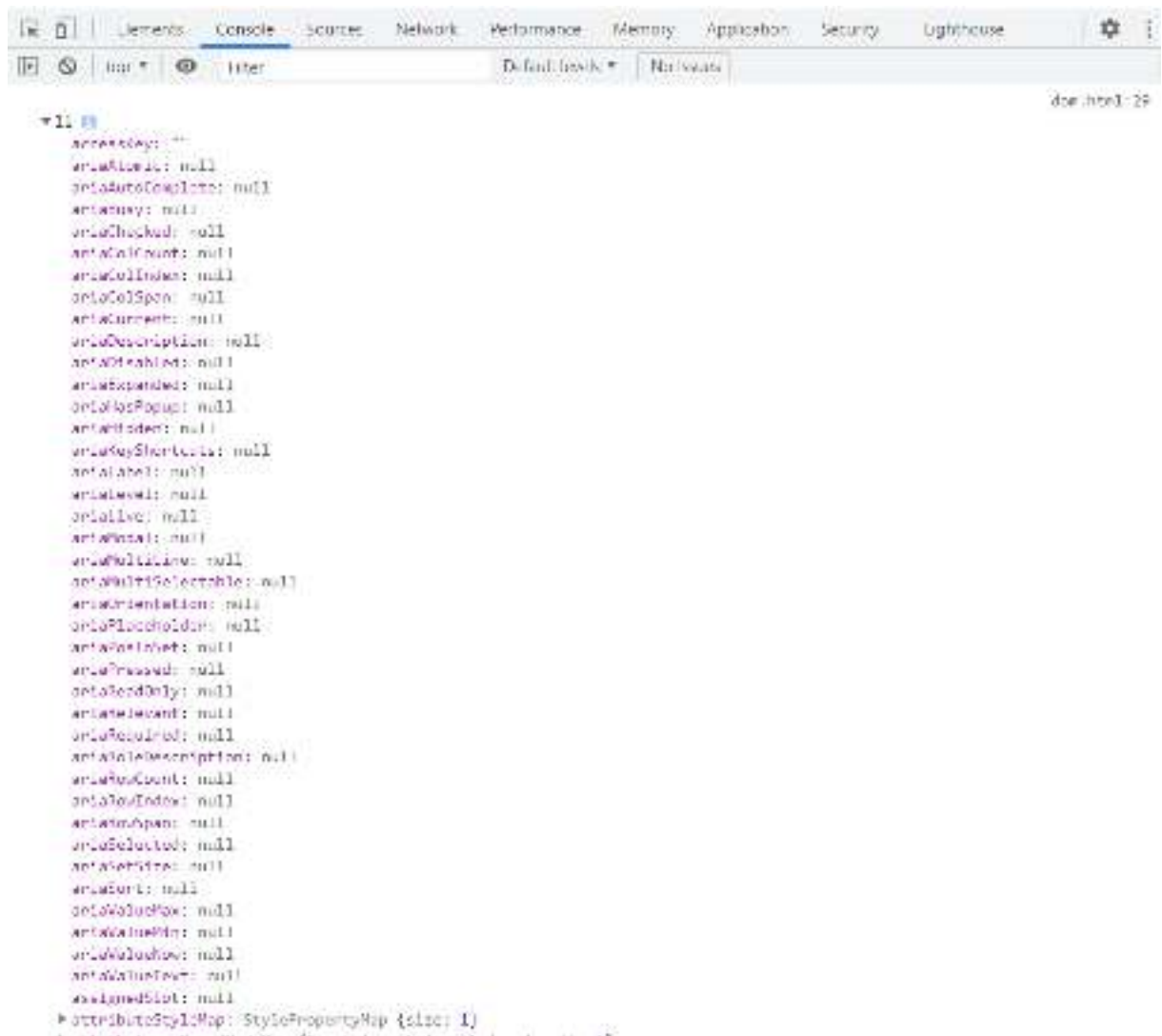
*anonyme, chaque li sera représenté par la variable passée en argument "element" (qui peut être nommée autrement si besoin).*

Comment connaître les propriétés d'un élément du DOM (comme ici, style) et ce qu'elles contiennent ? La commande **console.dir()**<sup>1</sup> vous permet de les consulter :

```
22     <script>
23
24         let list = document.querySelector("ul")
25
26         let listElements = list.querySelectorAll("li")
27
28         listElements.forEach(function(element){
29             console.dir(element)
30             element.style.color = "red"
31         })
32
33
34     </script>
```

---

<sup>1</sup> Là où console.log() présenterait l'élément de manière interactive, à la manière de l'inspecteur d'éléments du navigateur, console.dir() listera ses composantes d'objet JavaScript.



L'objet `HTMLElement` représentant cette balise `li` se compose de très nombreuses propriétés, comme évoqué plus haut, définissant toutes les valeurs nécessitées par le navigateur pour l'interpréter correctement (son style, sa taille, sa position sur la page, son contenu...). En descendant un peu dans cette longue liste, nous trouvons par exemple :

```

document: HTMLDocument {title: "Le DOM est une interface de programmation", ...}
parentElement: HTMLDocument
parentNode: HTMLDocument
previousElementSibling: null
previousSibling: null
scrollHeight: 1000
scrollTop: 0
scrollLeft: 0
scrollWidth: 1000
style: CSSStyleDeclaration {0: "text-align: center; font-size: 1.2em; font-weight: bold; color: red; background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"}

```



- `outerHTML` et `outerText` : ce sont les textes représentant le li soit en HTML, soit uniquement sa représentation textuelle, tel que le navigateur l'a reçu pour interprétation.
- `parentNode` : l'élément du DOM parent de ce li (nous voyons donc "ul")
- `style` : un objet `CSSStyleDeclaration` contenant l'intégralité des propriétés CSS du li et leurs valeurs. C'est cela qui a été atteint dans le code JavaScript précédent, pour modifier dans cet objet la propriété `color` et lui assigner la valeur "red".
- `tagName`: le nom de la balise utilisée pour cet élément (li)

Naturellement, il y a beaucoup trop de propriétés dans un `HTMLElement` pour les lister toutes ici, mais il faut savoir et acquérir le réflexe de se renseigner en utilisant la console, afin de déterminer si notre code JavaScript effectue les bonnes actions sur les bonnes propriétés.

## II. Jouons avec le DOM !

### 1. Les règles du jeu

Nous allons tout au long de la suite de ce support réaliser un petit jeu sur navigateur dont voici les règles :

- La page présente différentes boîtes contenant chacune un numéro. Ces boîtes forment une suite arithmétique logique (soit 5 boîtes : boîte n°1, boîte n°2, ... boîte n°5). Ces boîtes sont mélangées aléatoirement à chaque chargement de la page.
- Le joueur doit cliquer sur chaque boîte dans l'ordre (boîte 1 puis boîte 2 etc.). Une boîte cliquée est validée, son aspect devra changer pour informer visuellement le joueur de son action.
- Si le joueur se trompe de boîte (en cliquant sur la 4 juste après avoir cliqué sur la 2, par exemple), le jeu redémarre à zéro. Il lui faudra donc cliquer de nouveau sur la boîte 1 et ainsi de suite.
- Une fois que la dernière boîte est cliquée dans l'ordre imposé, un message de victoire doit apparaître.
- Enfin, si le joueur clique sur une boîte précédemment validée (le jeu attendait qu'il clique sur la boîte 4 mais le clic a lieu sur la boîte 2), un message le prévient que cette boîte a déjà été cliquée, mais le jeu ne s'arrête pas pour autant.

### 2. Prérequis

Nous nous concentrerons sur la partie JavaScript de la réalisation du jeu, donc voici les codes HTML et CSS nécessaires au préalable :

#### a. index.html

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <link rel="stylesheet" href="style.css">
8     <title>Document</title>
9   </head>
10  <body>
11
12    <div id="board"></div>
13
14    <script>
15      /* ici notre code JS */
16    </script>
17  </body>
18 </html>
```

## b. style.css

```
1  #board{
2      display: flex;
3      flex-wrap: wrap;
4  }
5
6  .box{
7      width: 75px;
8      height: 75px;
9      border: 1px solid black;
10     background-color: hsl(212, 25%, 76);
11     font-size: 3em;
12     text-align: center;
13     line-height: 75px;
14     margin: 5px;
15     animation: appear 1s;
16     border-radius: 6px;
17 }
18 .box-clicked{
19     background-color: #ccc;
20     color: #aaa;
21 }
22
23 @keyframes appear{
24     from{
25         transform: scale(0) rotate(180deg);
26     }
27     to{
28         transform: scale(1) rotate(0deg);
29     }
30 }
```

Sur la feuille de style ci-dessus, nous constatons deux classes **.box** et **.box-valid**. La première définit le style d'une boîte numérotée du jeu dans son état initial, la seconde modifiera la couleur de fond et celle du texte lorsque cette boîte sera cliquée ET valide.

L'animation nommée "appear" permettra au chargement de la page de faire apparaître les boîtes en tournoyant et zoomant légèrement, donnant un retour visuel au joueur lui signifiant que le jeu commence avec des boîtes mélangées.

### 3. Le code JavaScript

#### a. Faire apparaître les boîtes à l'écran

Vous remarquerez sans doute qu'aucune balise ne comprend la classe **.box** dans le code HTML précédent. Normal, nous allons faire générer à JavaScript ces éléments en les créant de toutes pièces.

```
10 | <body>
11 |
12 |   <div id="board"></div>
13 |
14 |   <script>
15 |     const box = document.createElement("div")
16 |     box.classList.add("box")
17 |
18 |   </script>
19 |
20 | </body>
```

En appelant la méthode `createElement()` de l'objet `document`, un nouvel objet `HTMLElement` est instancié, qui sera représenté par la balise spécifiée en argument de cette méthode (ici, `"div"`). Nous la stockons dans une constante (préférable aux variables lorsqu'on crée un élément du DOM, pour garantir l'intégrité de l'élément tout au long du code sans pouvoir réassigner sa valeur).

Ensuite, nous modifions sa propriété `classList` (contenant la liste des classes CSS attribuées à l'élément) pour lui ajouter la classe **.box** (appel de la méthode `add()` de `classList`)

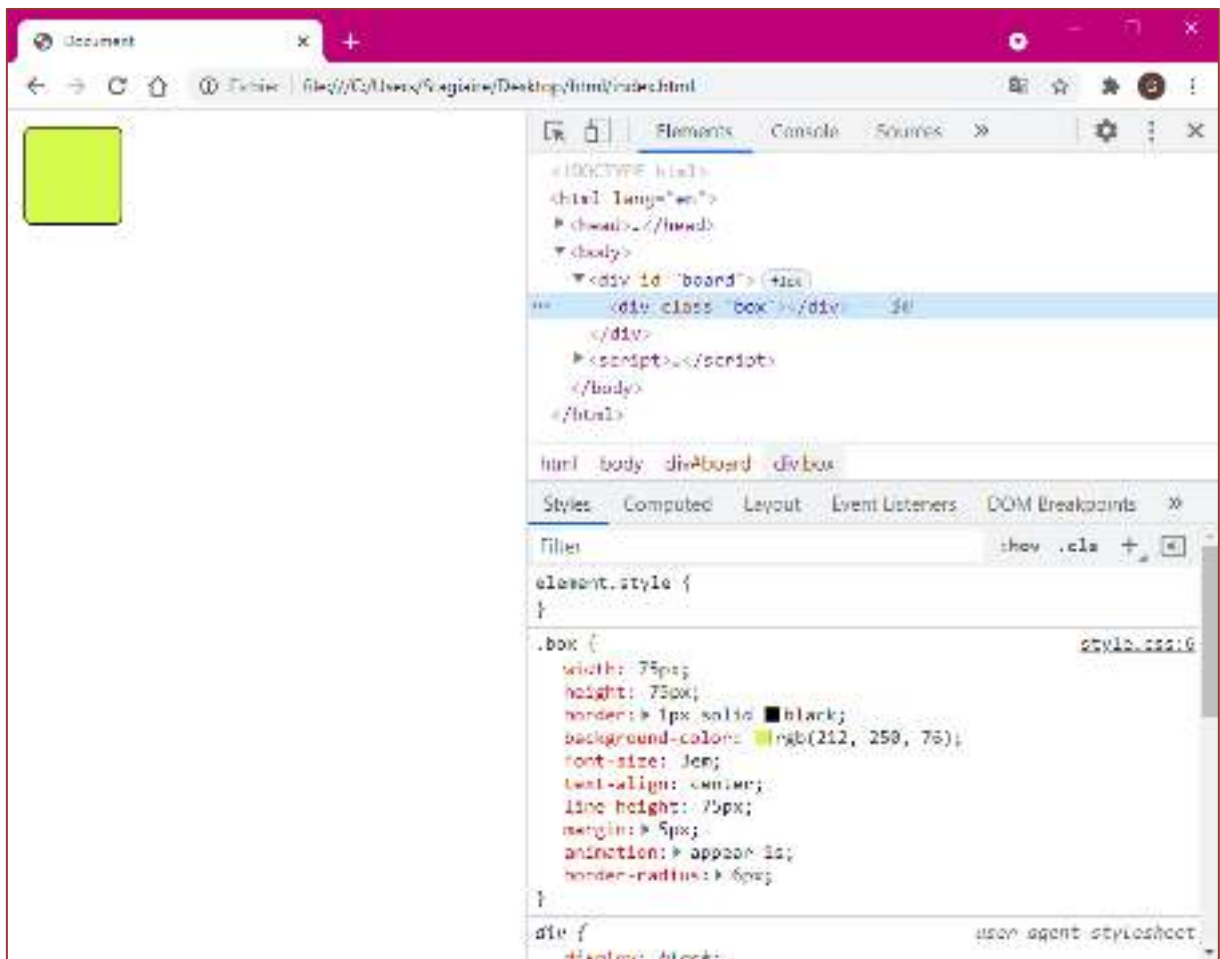
A ce moment, un élément du DOM nouveau (`<div class="box"></div>`) est instancié en mémoire mais non-présent sur la page. Il faut, pour cela, lui faire une place dans le DOM Tree. Nous allons le placer en enfant de la `div#board` en utilisant la méthode `appendChild()`<sup>2</sup> de l'objet `document` :

```
10 | <body>
11 |
12 |   <div id="board"></div>
13 |
14 |   <script>
15 |     const box = document.createElement("div")
16 |     box.classList.add("box")
17 |
18 |     const board = document.querySelector("#board")
19 |     board.appendChild(box)
20 |
21 |   </script>
22 |
23 | </body>
```

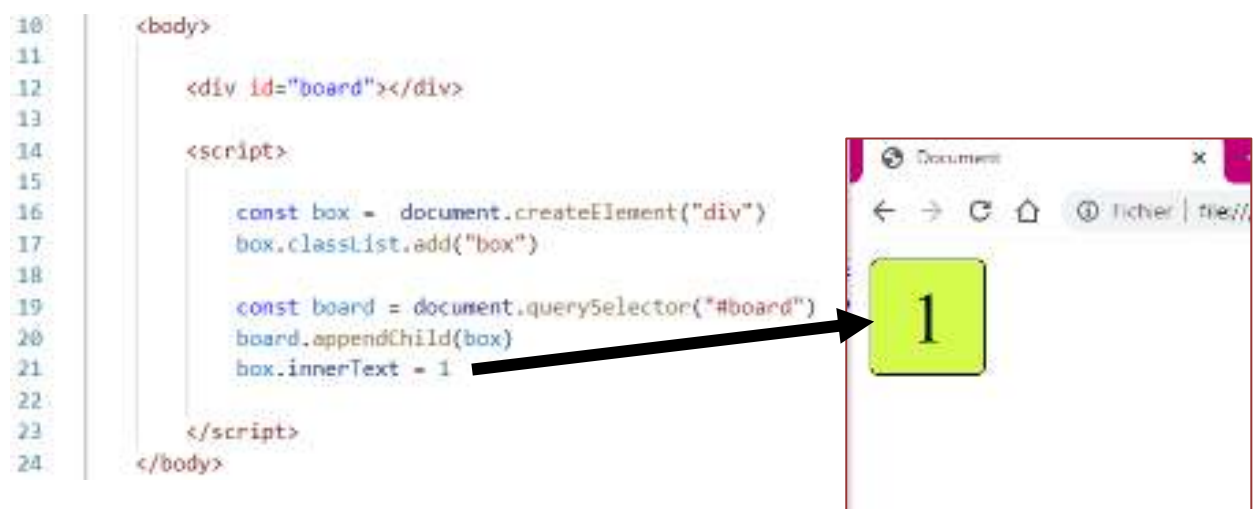
---

<sup>2</sup> `appendChild()` est une méthode qui place un élément du DOM à la fin du contenu de l'élément visé. Pour ajouter du contenu au début, on utilisera la méthode `prepend()`.

Vous verrez apparaitre, en chargeant la page dans votre navigateur, une boîte verte apparaitre en tournoyant ! Allez vérifier dans l'inspecteur d'éléments qu'elle se trouve effectivement à l'intérieur de `div#board`, comme ci-dessous :



La boîte ne comporte pas de numéro, ajoutons-le en écrivant à l'intérieur de l'élément un nœud de texte :



**Précision importante :** remarquez où se situe la ligne qui inscrit du texte dans la boîte par rapport à celle qui la précède. Il peut paraître étonnant de placer la boîte dans son parent

puis de la remplir avec du contenu, on peut penser qu'une fois placée dans le DOM, on ne peut plus revenir dessus. C'est sans compter que **nous sommes avec JavaScript dans un contexte objet : la constante box contient un pointeur vers l'élément du DOM** (placé ou non dans le DOM Tree) qui permet de l'atteindre à tout moment, tant que cet élément est présent en mémoire.

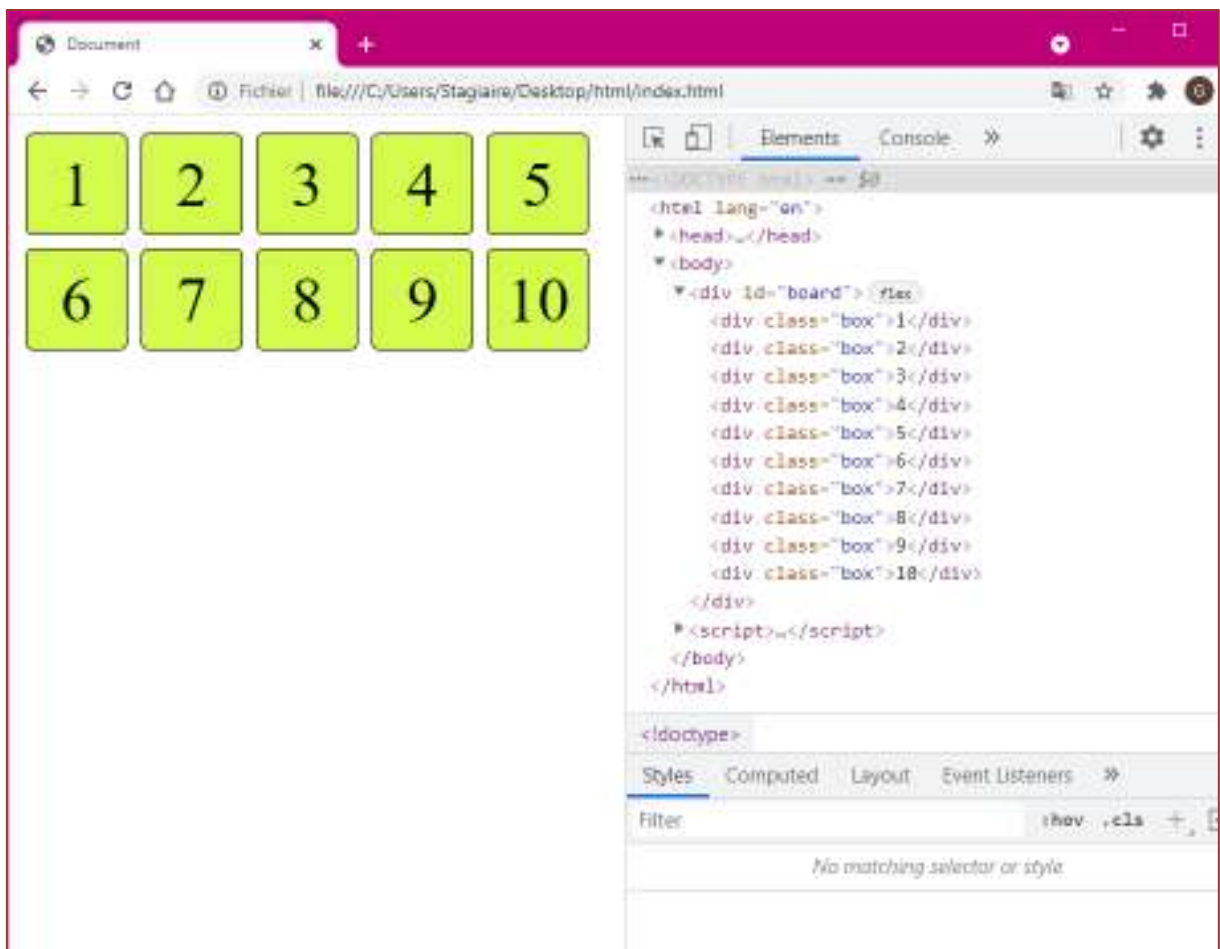
Pour finir cette première partie, nous allons faire apparaître ainsi plusieurs boîtes à l'écran. Partons alors du principe que nous voulons 10 boîtes dans le jeu. Nous n'allons bien sûr pas les créer une par une, prêtez particulièrement attention au code suivant :

```
10  <body>
11
12  <div id="board"></div>
13
14  <script>
15
16      const box = document.createElement("div")
17      box.classList.add("box")
18
19      const board = document.querySelector("#board")
20
21      for(let i = 1; i <= 10; i++){
22          let newbox = box.cloneNode()
23          newbox.innerText = i
24          board.appendChild(newbox)
25      }
26
27  </script>
28  </body>
```

Nous effectuons une boucle for, de 1 à 10 (vous pouvez changer ce nombre à votre convenance), pour générer autant de boîtes à l'écran. La particularité ici est qu'il faille créer une nouvelle variable (let newbox) qui aura pour valeur non pas l'élément box mais **une copie, un clone de celui-ci grâce à la méthode cloneNode()**.

N'oubliez pas le contexte objet expliqué plus haut : si nous n'avions pas procédé à une copie de l'élément box, nous aurions à chaque tour de boucle modifié et déplacé LE MEME ELEMENT !!! Essayez vous-même en commentant la ligne 22 et en renommant "newbox" en "box" : vous n'aurez qu'une boîte à l'écran, avec le numéro 10 !

Vous devriez désormais obtenir ceci à l'écran :



## b. Mélanger les boîtes aléatoirement

La prochaine étape du code consiste à mélanger aléatoirement les boîtes à l'écran, et ainsi obtenir un ordre différent des numéros à chaque rafraîchissement de la page.

Quelques recherches sur le net nous amènent rapidement à une solution très connue et très simple à implémenter : l'algorithme de **mélange de Fisher-Yates**<sup>3</sup>, aussi appelé mélange de Knuth ou algorithme P.

Ce support n'a pas pour objet de décrire l'algorithme en détail. Accompagnée de commentaires qui vous aideront à comprendre son fonctionnement, une de ses nombreuses implémentations en JavaScript se trouve dans la capture d'écran suivante :

<sup>3</sup> [https://fr.wikipedia.org/wiki/M%C3%A9lange\\_de\\_Fisher-Yates](https://fr.wikipedia.org/wiki/M%C3%A9lange_de_Fisher-Yates)

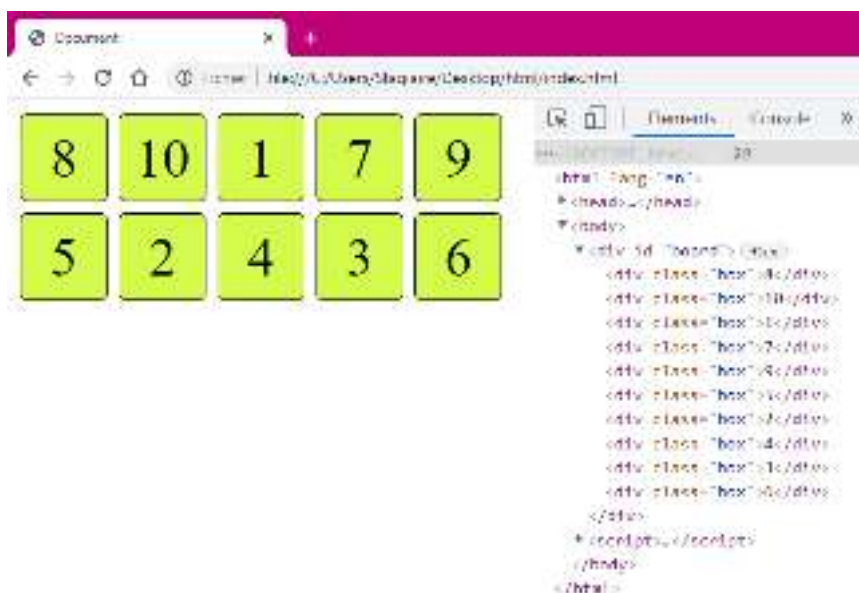


```

10 <body>
11
12 <div id="board"></div>
13
14 <script>
15
16     const box = document.createElement("div")
17     box.classList.add("box")
18
19     const board = document.querySelector("#board")
20
21     for(let i = 1; i <= 10; i++){
22         const newbox = box.cloneNode()
23         newbox.innerText = i
24         board.appendChild(newbox)
25     }
26
27     let i = board.children.length, k , temp
28     while(--i > 0){ // on boucle tant que l'oté de i est toujours positif
29         // k stocke un nombre aléatoire basé sur i
30         k = Math.floor(Math.random() * (i+1))
31         // temp pointe temporairement l'élément à la position k dans board
32         temp = board.children[k]
33         // remplace l'élément à la position k par l'élément à la position i
34         board.children[k] = board.children[i]
35         // place l'élément k pointé temporairement à la fin du contenu de board
36         board.appendChild(temp)
37     }
38
39 </script>
40 </body>

```

Rafraichissez votre page, un ordre différent s'effectuera :





**Optimisons notre code** : nous pouvons constater qu'à plusieurs reprises est fait référence à `board.children`. Ajoutons aussi le fait qu'il puisse être utile de réutiliser cette portion du code à d'autres moments (vous verrez...) voire dans d'autres projets.

Aussi, isolons l'algorithme dans une fonction dès maintenant :

```
10 <body>
11
12 <div id="board"></div>
13
14 <script>
15     function shuffleChildren(parent){
16         let children = parent.children
17         let i = children.length, k, temp
18         while( i > 0){
19             k = Math.floor(Math.random() * (i+1))
20             temp = children[k]
21             children[k] = children[i]
22             parent.appendChild(temp)
23         }
24     }
25
26     const box = document.createElement("div")
27     box.classList.add("box")
28
29     const board = document.querySelector("#board")
30
31     for(let i = 1; i <= 10; i++){
32         const newbox = box.cloneNode()
33         newbox.innerText = i
34         board.appendChild(newbox)
35     }
36
37     shuffleChildren(board)
38
39 </script>
40 </body>
```

**Note** : Les commentaires précédents ont été effacés pour des raisons de lisibilité, les variables ont été renommées pour découpler, "neutraliser" l'algorithme ce qui, à la lecture, lui confère une logique adaptable à toute situation et non plus dédiée à ce contexte de jeu.

La fonction elle-même a été nommée dans le même esprit, pour décrire au mieux sa procédure (elle mélange les enfants d'un élément parent donné, rien d'autre). La fonction est placée en début de code par convention.

### c. Ajouter un écouteur d'évènement

La situation de départ du jeu est mise en place, il ne reste plus qu'à le rendre jouable.

Le jeu nécessite que le joueur clique sur les boîtes, et le clic sera l'unique façon d'interagir avec le jeu (pas de réaction au survol d'une boîte, à la pression d'une touche du clavier, etc.). **Ces interactions avec la page sont appelées en JavaScript des événements.**

Il existe des dizaines d'évènements supportés par JavaScript et les éléments du DOM, certains spécifiques à des éléments précis : l'évènement "change", par exemple, n'est possible que sur des éléments dont la valeur peut être modifiée par l'utilisateur (liste déroulante, zone de texte...).

Ces évènements peuvent être incorporés à tout élément de la page, ce même si cet élément n'est pas prévu pour une telle action (comme survoler une image, cliquer sur un paragraphe, double-cliquer sur un lien...). Tout est possible... ou presque !

Pour associer un évènement, JavaScript fournit une méthode appelée "addEventListener" à tout élément du DOM. Le principe est d'ajouter un écouteur d'évènement<sup>4</sup> sur l'élément, celui-ci exécutera une fonction callback dès lors que cet évènement survient sur l'élément.

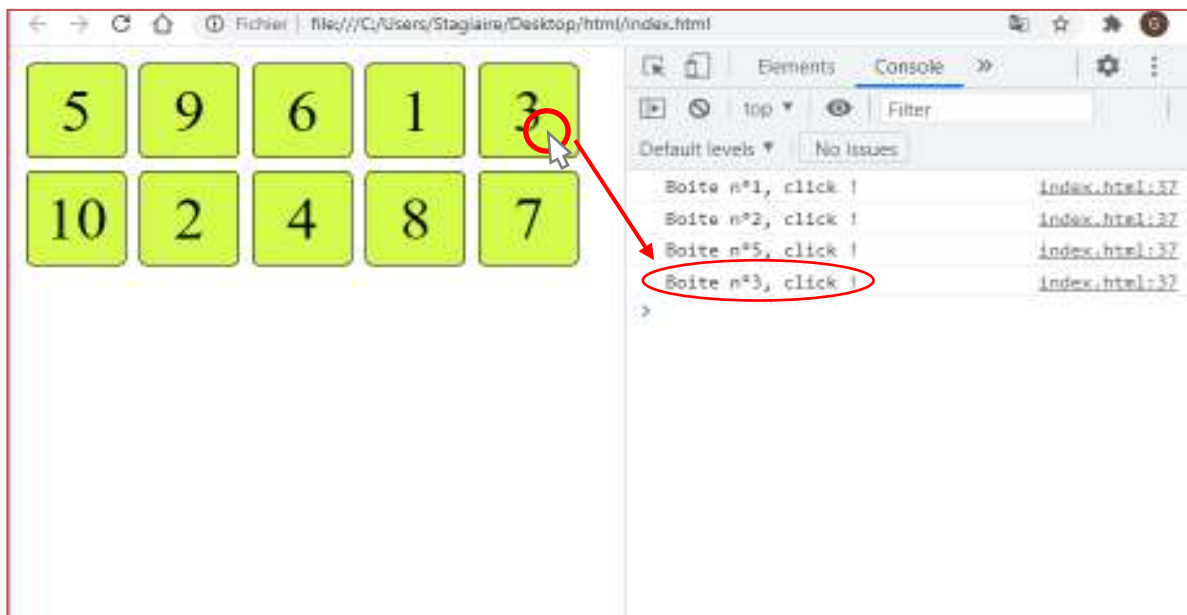
Associions donc l'évènement "click" aux boites du jeu (lignes 36 à 39) :

```
12      <div id="board"></div>
13
14      <script>
15          function shuffleChildren(parent){
16              let children = parent.children
17              let i = children.length, k , temp
18              while(--i > 0){
19                  k = Math.floor(Math.random() * (i+1))
20                  temp = children[k]
21                  children[k] = children[i]
22                  parent.appendChild(temp)
23              }
24          }
25
26          const box = document.createElement("div")
27          box.classList.add("box")
28
29          const board = document.querySelector("#board")
30
31          for(let i = 1; i <= 10; i++){
32              const newbox = box.cloneNode()
33              newbox.innerText = i
34              board.appendChild(newbox)
35
36              newbox.addEventListener("click", function(){
37                  console.log("Boite n°" + i + ", click !")
38              })
39          }
40
41          shuffleChildren(board)
42
43      </script>
```

---

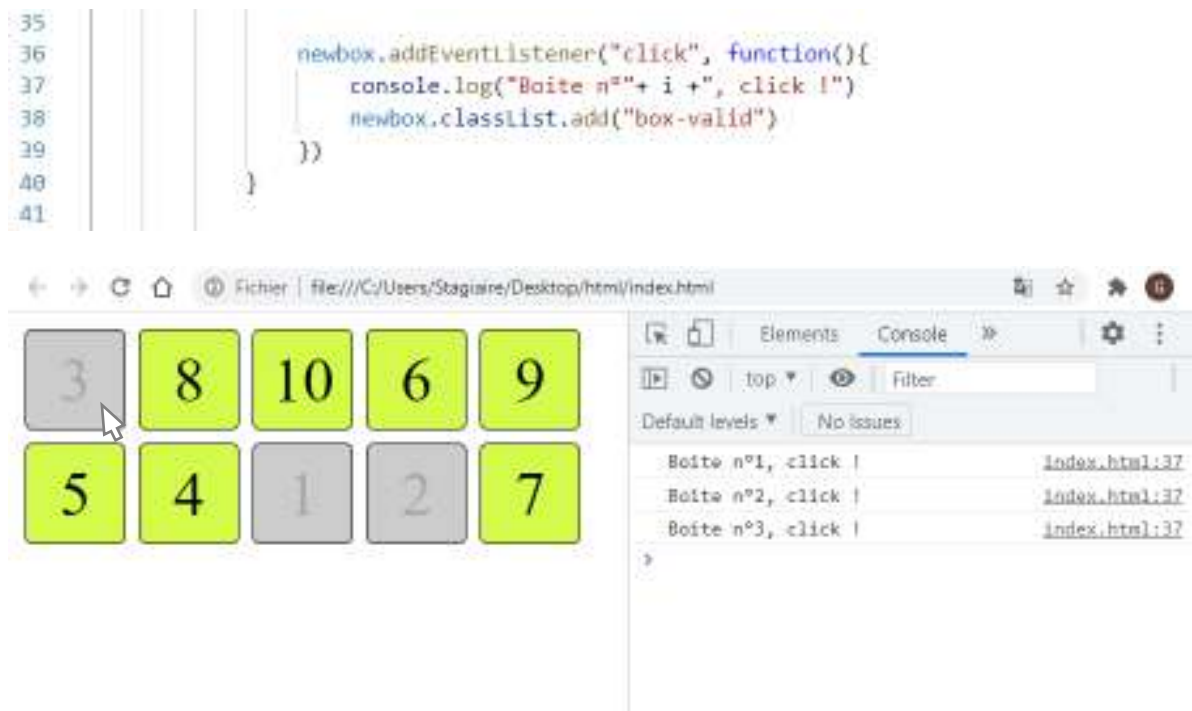
<sup>4</sup> [https://fr.wikipedia.org/wiki/Observateur\\_\(patron\\_de\\_conception\)](https://fr.wikipedia.org/wiki/Observateur_(patron_de_conception))

Et le résultat à l'écran :



Remarquez que nous plaçons l'appel à `addEventListener` à l'intérieur de la boucle, ce qui permet de ne pas avoir à effectuer une autre boucle sur toutes les boîtes pour leur adjoindre l'évènement. La variable `newbox` représente l'élément boîte au moment de sa génération, l'idéal est donc de le définir au maximum (style, comportement, données...) au même endroit dans le code !

Ça marche ! Etoffons un peu notre action au clic en modifiant visuellement la case cliquée, ainsi le joueur repère directement si la case est valide :



#### d. Prendre en charge les phases de jeu

Il ne nous reste plus qu'à déterminer si la case cliquée comporte le numéro attendu, selon que :

- Le tout premier clic doit se faire sur la boîte 1.
- Le prochain sur la boîte 2, si le joueur clique de nouveau sur la boîte 1, un message apparaît.
- A contrario, si le joueur clique sur les boîtes 3, 4 ou 5, le jeu redémarre et un message apparaît.

Procédons par étapes :

```
29     const board = document.querySelector("#board")
30
31     let nb = 1
32
33     for(let i = 1; i <= 10; i++){
34         const newbox = box.cloneNode()
35         newbox.innerText = i
36         board.appendChild(newbox)
37
38         newbox.addEventListener("click", function(){
39
40             if(i == nb){
41                 newbox.classList.add("box-valid")
42                 nb++
43             }
44         })
45     }
46
47     shuffleChildren(board)
48
49 </script>
```

**Ligne 31** : on déclare une variable nb qui représentera le numéro de la boîte attendue et qui s'incrémentera en cas de clic valide ! (ligne 42)

**Ligne 40** : on vérifie d'abord si la boîte sur laquelle le clic s'effectue possède le même numéro que ce que contient la variable nb. Si c'est le cas, on ajoute la classe CSS "box-valid" et on incrémente nb.

Rafraîchissez votre navigateur et testez le résultat en procédant à plusieurs cas d'utilisation :

- Cliquez sur 1, puis sur 2, etc. les boîtes doivent se griser une à une.
- Cliquez sur 1, puis sur 4 : rien ne se passe sur la boîte 4. Cliquez ensuite sur 2, l'ordre des boîtes à cliquer reprend (cela ne respecte pas encore les règles ci-dessus).

Le jeu est presque terminé ! Il nous faut pour cela émettre les messages prévus plus haut et redémarrez le jeu en cas d'erreur.

```

31     let nb = 1
32
33     for(let i = 1; i <= 10; i++){
34         const newBox = box.cloneNode()
35         newBox.innerText = i
36         board.appendChild(newBox)
37
38         newBox.addEventListener("click", function(){
39
40             if(i == nb){
41                 newBox.classList.add("box-out")
42                 // 1
43                 if(nb == board.children.length){
44                     alert("VICTOIRE !")
45                 }
46                 nb++
47             }
48             // 2
49             else if(i > nb){
50                 alert("Erreur, recommencez !")
51                 nb = 1
52             }
53             // 3
54             else{
55                 alert("Case déjà cliquée !")
56             }
57         })
58     }
59 }

```

// 1 : Si nb est égal au nombre de boîtes du jeu, c'est que le dernier clic était sur la dernière boîte → victoire du joueur ! (Il ne faut pas incrémenter nb avant !)

// 2 : Si le numéro de la boîte est supérieur à nb, c'est que le joueur a cliqué une boîte trop élevée → game over !

// 3 : Dernière possibilité : le joueur a cliqué sur une boîte déjà grisée. On l'informe simplement de cela, le jeu ne redémarre pas.

Et voilà, le jeu fonctionne et respecte les règles préétablies ! Néanmoins, si vous essayez de jouer, vous remarquerez qu'une action du joueur donne lieu à une situation déstabilisante : si le joueur se trompe et que le jeu redémarre, les boîtes grisées précédemment restent grisées.

En termes d'expérience utilisateur, ce genre de détail visuel involontaire peut perturber l'utilisation d'une application et induire en erreur sur l'état réel des informations et des composants présentés à l'utilisateur. Remédions-y à partir de la ligne 52 :

```

49     else if(i > nb){
50         alert("Erreur, recommencez !")
51         nb = 1
52         board.querySelectorAll(".box-valid").forEach(function(validBox){
53             validBox.classList.remove("box-valid")
54         })
55     }

```

L'idée est, lorsque le jeu doit redémarrer (nous nous plaçons donc dans la partie conditionnelle correspondante), de **sélectionner les boîtes grisées en passant par l'élément board** (puisque'il les contient). La méthode `querySelectorAll()` récupère un tableau d'éléments, sur lequel nous utilisons la méthode `forEach()` permettant de passer sur chaque élément, élément qui sera représenté ici par l'argument "validBox".

La fonction callback à l'intérieur de `forEach()` supprime tout simplement la classe CSS "box-valid" des attributs de la boîte courante "validBox", reprenant ainsi son aspect initial.

### III. Aller un peu plus loin !

Notre objectif est pleinement atteint ! Le jeu fonctionne et le joueur est sans cesse averti de la nature de ses actions. Seulement, nous pourrions encore améliorer certains points.

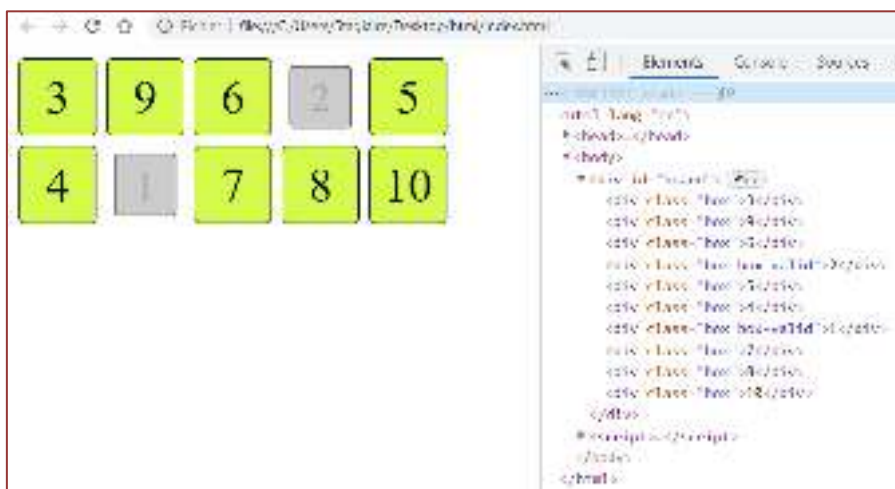
#### 1. Animer les boîtes valides

Le passage d'une boîte cliquable à une boîte validée est un peu abrupt pour le joueur. De même que le pointeur de la souris ne change pas au survol d'une boîte pour indiquer que celle-ci est cliquable. Ajoutons un peu de CSS pour rendre l'action plus explicite à l'écran :

```
6  .box{
7      width: 75px;
8      height: 75px;
9      border: 1px solid black;
10     background-color: #d3d3d3;
11     font-size: 1em;
12     text-align: center;
13     line-height: 1;
14     margin: 5px;
15     animation: appear 1s;
16     border-radius: 6px;
17     cursor: pointer;
18     transition: background-color 0.8s, color 0.8s, transform 0.8s;
19 }
20 .box-valid{
21     background-color: #90ee90;
22     color: #008000;
23     transform: scale(0.8);
24 }
```

Cela peut paraître anecdotique, mais faire apparaître le bon curseur (ligne 17) au survol est une bonne pratique depuis de nombreuses années sur le web.

Ligne 18, la propriété CSS **transition** permettra d'animer le passage d'état de la classe "box" à "box-valid". 0.8 secondes seront consacrées au changement de la couleur de fond, de la couleur du texte et de la propriété transform qui réduira la taille de la boîte de 20%.





## 2. Supprimer les messages d'alerte

Visuellement et ergonomiquement, les boîtes de dialogue affichées en utilisant la fonction `alert()` de JavaScript sont désuètes et "cassent" le rythme du jeu. Avec l'aide de JavaScript, nous pourrions faire passer le message au joueur tout aussi pertinemment avec un peu de CSS et... sans texte !

### a. La fonction `showReaction()`

Ajoutons une fonction, que nous nommerons `showReaction()`, dans notre script :

```
14      <script>
15      function shuffleChildren(parent){
16          let children = parent.children
17          let i = children.length, k, temp
18          while(--i > 0){
19              k = Math.floor(Math.random() * (i+1))
20              temp = children[k]
21              children[k] = children[i]
22              parent.appendChild(temp)
23          }
24      }
25
26      function showReaction(type, clickedBox){
27          clickedBox.classList.add(type)
28          if(type !== "success"){
29              setTimeout(function(){
30                  clickedBox.classList.remove(type)
31              }, 800)
32          }
33      }
34  
```

Le principe est le suivant :

Lorsque l'utilisateur provoquera un clic sur une boîte, la fonction `showReaction()` sera appelée pour provoquer une réaction visuelle sur cette même boîte. Les réactions sont les suivantes :

- error : la boîte cliquée est invalide (affichage en rouge)
- notice : la boîte cliquée l'avait déjà été auparavant (affichage en bleu)
- success : la boîte cliquée est valide et était la dernière, la partie est gagnée (affichage en vert)

La fonction attend deux arguments :

- type (une chaîne de caractères) correspondant au type de réaction souhaité
- clickedBox (HTML`Element`) étant la boîte sur laquelle l'effet sera appliqué

La fonction applique donc la classe CSS "type" (error, success ou notice) à la boîte passée en argument (ligne 27).



Puis elle vérifie que le type n'est pas égal à "success" pour exécuter un **setTimeout()**. La fonction `setTimeout()` de JavaScript permet de retarder l'exécution d'une ou plusieurs instructions (la fonction callback en premier paramètre) du délai voulu (le nombre en second paramètre, exprimé en millisecondes). Ici, nous retardons la suppression de la classe CSS ajoutée précédemment de 0.8 secondes.

## b. Placement des appels à la fonction

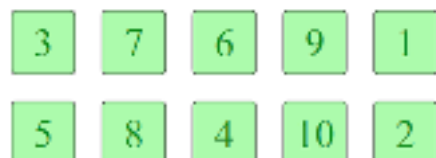
Nous allons désormais remplacer chaque `alert()` par l'appel à la fonction `showReaction()` correspondant :

```

42     for(let i = 1; i <= 10; i++){
43         const newbox = box.cloneNode()
44         newbox.innerText = i
45         board.appendChild(newbox)
46
47         newbox.addEventListener("click", function(){
48
49             if(i == nb){
50                 newbox.classList.add("box valid")
51                 if(nb == board.children.length){
52                     board.querySelectorAll(".box").forEach(function(box){
53                         showReaction("success", box)
54                     })
55                 }
56                 nb++
57             }
58             else if(i > nb){
59                 showReaction("error", newbox)
60                 nb = 1
61                 board.querySelectorAll(".box valid").forEach(function(validBox){
62                     validBox.classList.remove("box valid")
63                 })
64             }
65             else{
66                 showReaction("notice", newbox)
67             }
68         })
69     }

```

**Ligne 52** : de la même manière que lorsque nous avons bouclé sur toutes les boîtes valides en cas de redémarrage du jeu (ligne 61 ici), nous effectuons un `forEach()` sur toutes les boîtes du jeu pour appeler `showReaction()` en mode "success".



**Ligne 59** : `showReaction()` est appelée en mode "error" sur la boîte cliquée, le joueur verra donc cette boîte devenir rouge et, comme précédemment, toutes les autres boîtes se réinitialiser.



**Ligne 66** : Clic sur une boîte déjà validée, nous appelons donc `showReaction()` en mode "notice".



Il ne reste plus qu'à créer ces classes CSS dans la feuille de style :

```
19 }
20 .box-valid{
21     background-color: #00cc00;
22     color: #000;
23     transform: scale(0.8);
24 }
25 .box.error{ color: red; background-color: red; }
26 .box.success{ color: green; background-color: green; }
27 .box.notice{ color: blue; background-color: blue; cursor: not-allowed; }
28 }
```

Chaque classe modifie la couleur du texte et de l'arrière-plan de la boîte concernée. La classe "notice" modifie également le curseur de la souris pour indiquer au survol que le clic du joueur sur une case déjà validée ne fera pas avancer le jeu.

## IV. Conclusion

Ce petit jeu a eu pour vocation de mieux vous faire comprendre les principes de bases de la manipulation du DOM avec JavaScript. Il reste bien sûr tout à fait possible de l'améliorer encore, avec de l'imagination. Voici quelques pistes qui peuvent être implémentées :

- Demander le nombre de boîtes de départ au joueur (en utilisant `prompt()`)
- Remélanger les boîtes en cas d'erreur, ainsi le joueur ne pourra plus compter sur sa mémoire visuelle.
- On peut aussi penser à remélanger les boîtes à chaque clic valide, rendant le jeu plus difficile.
- Incorporer un timer qui prendrait fin à la validation de toutes les boîtes. En imaginant conserver d'une partie à l'autre les meilleurs temps, cela apporterait au jeu un petit côté "high score" intéressant.