Labo 6 – Manipulation du DOM

Introduction

Il existe de nombreuses ressources en ligne pour en savoir plus sur le DOM et la manière dont il peut être manipulé en Javascript. Voici un lien qui se révélera peut-être utile, et qui décrit la structure du DOM pour le navigateur Firefox.

- Mozilla Developer Network (*https://developer.mozilla.org/fr/docs/JavaScript*) : la référence en ligne pour le Javascript de Firefox, aussi disponible en anglais

Exercice 1 : un avant-goût du DOM

Pour vous rendre compte de la structure en arborescence qui se cache derrière toute partie d'un document HTML, construisez tout d'abord un fichier HTML contenant le code suivant.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1"/>
    <script>
      function crie () {alert("Bouh!");}
    </script>
  </head>
  <body>
    <p>Si tu <a href="javascript:crie();">cliques</a>,
      alors je <strong>crie</strong> !
    </p>
  </body>
</html>
```

Utilisez ensuite la console de Firefox pour définir la fonction dont le code est donné ci-dessous.

```
function affiche (elem) {
  var lgChildren = elem.children.length;
  var lgChildNodes = elem.childNodes.length;
  var msg = "ELEM : " + elem.nodeName + "\n";
  msg += lgChildren + " children\n";
  msg += lgChildNodes + " childNodes\n";
  msg += "Childnodes :";
  for (var i = 0 ; i < lgChildNodes ; i++)
    msg += " " + elem.childNodes[i].nodeName;
  console.log(msg);
  for (i = 0 ; i < lgChildren ; i++)
    affiche(elem.children[i]);
}
```

Étape 1

Observez le code de la fonction `affiche`.

Il s'agit d'une fonction récursive qui examine un élément (un nœud) de l'arborescence HTML et le décrit en affichant dans la console :

- le mot « ELEM » (pour « élément ») suivi du nom du nœud ; puis
- le nombre d'enfants-éléments (children) de ce nœud ; puis
- le nombre d'enfants-nœuds de (childnodes) ce nœud (blancs y compris) ; puis
- qui cite les noms de chacun des enfants-nœuds ; puis
- qui s'appelle récursivement pour afficher les détails de chacun des enfants-éléments.

Dans la console Javascript, utilisez la déclaration suivante pour cibler l'unique élément en gras (balise) du document HTML.

```
var elemGras = document.getElementsByTagName("strong")[0];
```

Appelez ensuite la fonction affiche sur cet élément.

```
affiche(elemGras)
```

Observez que l'élément en question n'a aucun enfants-éléments mais possède 1 enfant-nœud (il s'agit du texte contenu entre les balises et).

Étape 2

En utilisant à nouveau la méthode `document.getElementsByTagName`, ciblez dans la variable elemPara l'unique paragraphe du document HTML.

Puis exécutez la fonction affiche sur cet élément et observez le résultat.

Étape 3

Finalement, examinez les affichages produits par les appels suivants.

```
affiche(document.body);
affiche(document.head);
```

Observez également que chacune des variables globales que vous avez définies plus haut (comme affiche, elemGras ou elemPara) sont en fait des propriétés de l'objet window. Par exemple, vérifiez que les commandes suivantes produisent bien l'effet escompté.

```
var nbJours = 7;
alert(window.nbJours);

window.affiche(window.elemGras);
```

Exercice 2 : Bouton réactif

En manipulant le DOM, on peut modifier les actions associées à certains événements : plutôt que de fixer une fois pour toutes la tâche associée à un bouton, on peut modifier celle-ci à sa guise.

Étape 1

Créez un fichier HTML contenant un bouton dont le texte indique « Cliquez moi ! ».

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1"/>
  </head>
  <body>
    <button id="but">Cliquez moi !</button>
  </body>
</html>
```

Étape 2*

Modifiez le code de la page HTML pour que, lors d'un clic sur le bouton, son texte se transforme en « Merci, ça suffit ! ».

Si la variable « b » se réfère au bouton, vous pouvez accéder à son texte via `b.innerHTML`.

Étape 3*

Modifiez à nouveau le code pour que, après le premier clic, le bouton affiche « Encore une fois » puis, après le second clic, « Merci, ça suffit. ».

Concrètement, cela veut dire que le premier clic devra entraîner une modification non seulement du texte du bouton (`b.innerHTML`) mais également de l'événement associé au clic (`b.onclick`).

Étape 4*

Au début de votre script Javascript, définissez une variable globale `nbClicksRequis`.

```
var nbClicksRequis = 5;
```

Cette variable représente le nombre de fois qu'il faudra cliquer sur le bouton avant que celui-ci ne dise « Merci, ça suffit ! ».

Dans le cas où `nbClicksRequis = 5`, le bouton affichera donc « Cliquez moi ! » lors du chargement de la page, puis « Encore 4 fois ! » après un premier clic, « Encore 3 fois ! » après un second clic, … « Encore 1 fois ! » après un quatrième clic, et finalement « Merci, ça suffit ! » après le cinquième clic.

Étape 5

Dans l'étape précédente, vous avez sans doute utilisé une variable globale pour mémoriser le nombre de clicks déjà effectués sur le bouton en question.

Pour cette étape, il va sans doute falloir changer d'approche… ici, on vous demande d'ajouter 30 boutons sur la page HTML (vous pouvez le faire en usant du copier/coller ou en utilisant `document.write` et une boucle Javascript). Chacun de ces 30 boutons aura le même comportement que le bouton de l'étape précédente ; les nombres de clicks seront comptabilisés individuellement.

L'objectif est que chacun des 30 boutons utilisent la **même** fonction pour l'événement `onclick` (et pas 30 fonctions différentes utilisant chacune une variable globale différente). On pourrait s'en tirer avec une variable globale de type tableau… mais il y a plus simple : rappelez-vous que l'élément

```
var but = document.getElementById("monBouton");
```

est un objet Javascript (à qui on peut donc ajouter des propriétés à la volée) qui peut être référencé par le mot-clef `this` à l'intérieur de la fonction associée à l'événement `onclick`. Tirez avantage de cela en stockant le nombre de clicks déjà effectués sur un bouton en tant que « variable locale à un bouton », c'est-à-dire en tant qu'attribut/propriété de l'objet associé au bouton.

Exercice 3 : Voyage dans l'arborescence

Cet exercice se base sur le fichier HTML suivant et l'utilisation de la console pour voyager dans l'arborescence HTML.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1"/>
  </head>
  <body>
    <p>Un tiramisu pour 8 personnes :</p>
    <ul>
      <li>3 gros oeufs</li>
      <li>100g de sucre</li>
      <li>1 sachet de sucre <strong>vanillé</strong></li>
      <li id="masca">250g de mascarpone</li>
      <li>24 <i>biscuits</i> à la cuillère</li>
      <li>1/2 litre de café noir <strong>non sucré</strong></li>
      <li>30g de poudre de cacao <strong>amer</strong></li>
    </ul>
  </body>
</html>
```

Avant toute chose, définissez

```
var b = document.body;
```

directement dans la console. C'est à partir de cet élément b qu'on va tenter d'accéder à la plupart des autres éléments HTML.

Le but de cet exercice est de vous familiariser avec les expressions basées sur children, childNodes, nextSibling… pour voyager dans l'arborescence HTML. Il y a parfois des méthodes plus simples / rapides pour trouver un élément dans le document HTML mais, bien souvent, on est content de pouvoir retomber sur ces opérations « basiques » pour des déplacements plus locaux.

Question 1

On désire obtenir le texte du paragraphe, à savoir « Un tiramisu pour 8 personnes : » (par exemple pour le stocker dans une variable). Parmi les expressions suivantes, quelles sont celles qui donneront la réponse attendue ? Pourquoi les autres réponses ne sont-elles pas correctes ?

- ☐ `b.children[0].innerHTML`
- ☐ `b.children[1].innerHTML`
- ☐ `b.childNodes[0].innerHTML`
- ☐ `b.childNodes[1].innerHTML`
- ☐ `b.firstChild.innerHTML`
- ☐ `b.firstElementChild.innerHTML`
- ☐ `b.getElementsByTagName("p").innerHTML`

Question 2*

On désire obtenir le texte de la ligne concernant le sucre, à savoir « 100g de sucre ». Comment l'obtenir à partir de la variable `b` en utilisant (a) `children`, (b) `childNodes`, (c) `firstChild` et `nextSibling`, (d) `firstElementChild` et `nextElementSibling` ?

Question 3

Si on définit

```
var masca = document.getElementById("masca");
```

comment obtenir, à partir de la variable `masca`, le contenu en gras de la ligne correspondant au sachet de sucre (à savoir « vanillé ») ?

Que pensez-vous de l'expression suivante ? Comment la compléter pour obtenir le résultat désiré ?

```
masca.parentNode.getElementsByTagName
```

Question 4

Déterminez les valeurs des expressions suivantes (pour vérifier vos réponses, entrez-les simplement dans la console !)

```
b.children[1].childNodes[9].children[0].innerHTML

masca.parentNode.lastElementChild.lastElementChild.innerHTML

b.getElementsByTagName("strong")[1].parentNode.innerHTML

b.getElementsByTagName("strong")[1].parentNode.textContent
```

Question 5

Déterminez les actions opérées par les instructions suivantes (à exécuter les unes après les autres). Là encore, pour en voir l'effet, exécutez-les directement dans la console !

```
b.children[1].removeChild(masca)

b.children[1].appendChild(masca)
```

```
var alcool = document.createElement("li");
alcool.innerHTML = "un petit verre de marsala sec";
b.children[1].appendChild(alcool);

b.children[1].removeChild(masca)

b.children[1].replaceChild(masca, alcool);
```

Exercice 4 : Mini-calculatrice

Le but de cet exercice est de réaliser une mini-calculatrice en Javascript.

Étape 1

Considérez tout d'abord le fichier HTML suivant. Examinez son contenu.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
  </head>
  <body>
  <p>Valeur actuelle : <span id="spanVal"></span></p>
  <p>Opérations :
    <button id="bOppose">+/-</button>
    <button id="bCarre">carré</button>
    <button id="bFact">factorielle</button>
  <p>Ajuster :
    <button id="bDecremente10">-10</button>
    <button id="bDecremente1">-1</button>
    <button id="bRAZ">0</button>
    <button id="bIncremente1">+1</button>
    <button id="bIncremente10">+10</button>
  </p>
  <p>Multiplier :
    <button id="bMultiplie">×2</button>
    <button id="bDeux">2</button>
    <button id="bCinq">5</button>
    <button id="bDix">10</button>
  </p>
</html>
```

Le premier paragraphe permet d'afficher la valeur sur laquelle les opérations s'effectueront. Viennent ensuite trois autres paragraphes chaque fois avec plusieurs boutons correspondant à diverses opérations :

- Dans le premier paragraphe, les trois boutons correspondent aux opérations :
  - prendre l'opposé (changer de signe) : 3 devient -3, -7 devient 7 ;
  - remplacer la valeur actuelle par son carré ;
  - remplacer la valeur actuelle par sa factorielle (0! = 1! = 1, 2! = 2, 3! = 3×2×1, …) ou ne rien faire si la valeur actuelle est négative.

- Dans le second paragraphe, on trouve trois boutons permettant d'ajuster la valeur, soit en la décrémentant / en l'incrémentant (de 1 ou de 10), soit en la remettant directement à zéro.
- Finalement, dans le troisième paragraphe se trouve un bouton permettant de multiplier la valeur par un certain nombre. L'effet du premier bouton (la multiplication) pourra être modifié en cliquant sur un des trois boutons qui suivent, afin de multiplier par 2, par 5 ou par 10.

Dans cet exercice, on vous propose d'implémenter chacune des trois lignes de boutons de manière différente.

Étape 2

Tout d'abord, commencez par établir un espace de travail. Ajoutez dans le fichier HTML un lien vers un fichier Javascript (ou travaillez directement avec du code Javascript placé dans la balise <head> si vous préférez).

Déclarez-y une variable qui servira à stocker la valeur actuelle, ainsi qu'une fonction majValeur permettant de mettre à jour l'affichage de cette valeur (en modifiant le contenu de l'élément spanVal). Cette fonction sera appelée par la plupart des actions associées aux boutons.

Arrangez-vous pour que la valeur soit initialisée à 2. Ça peut sembler être un choix étonnant (pourquoi pas 0 ?), mais cela nous sera utile pour pouvoir tester les premiers boutons. Par la suite, vous pourrez modifier la valeur par défaut pour que la calculatrice affiche « 0 » au lancement.

Étape 3

Pour bien faire, il faudrait que la fonction de mise à jour de l'affichage soit appelée une première fois dès la fin du chargement de la page.

Ajouter un appel à cette fonction dans la partie <head> de la page web ne serait pas judicieux : en effet, cet appel serait effectué avant que la page web ne soit complètement affichée (avant donc que le « span » destiné à recevoir la valeur ne soit créé).

On pourrait ajouter un <script> en toute fin de document web mais il y a une méthode plus propre. Celle-ci consiste à indiquer au browser quel code exécuter dès que la page aura été entièrement chargée.

Pour ce faire, définissez tout d'abord une fonction init qui se chargera d'initialiser la valeur de la calculatrice (si vous ne l'avez pas déjà fait au sein de la déclaration) et d'appeler la fonction de mise à jour de l'affichage.

Puis, dans le script, ajoutez la ligne suivante.

```
windows.onload = init;
```

Cette ligne indique au navigateur que, dès que le document HTML aura été intégralement chargé en mémoire, il faudra exécuter la fonction init que vous venez de définir.

Étape 4

Occupons-nous du bouton permettant de calculer l'opposé de la valeur affichée. Définissez une fonction (dans ce cas-ci, il s'agira sans doute plutôt d'une procédure) qui modifiera la valeur (c'est-à-dire la variable globale) en son opposé puis qui fera appel à la fonction de mise à jour de l'affichage.

Pour associer cette fonction au bouton adéquat, modifiez le code HTML en ajoutant, dans la balise button adéquate, un attribut onclick.

Testez la calculatrice.

Étape 5

Faites de même pour le bouton permettant de mettre la valeur au carré.

Étape 6

Le bouton suivant doit calculer la factorielle de la valeur actuelle de la calculatrice. Afin de découper le problème en petites étapes plus abordables, définissez tout d'abord une fonction factorielle (x).

À l'aide de cette fonction, définissez une procédure calcFactorielle() mettant la valeur à jour ainsi que son affichage.

Pour associer cette fonction au bouton identifié par bFact, vous pourriez éditer le code HTML et ajouter un attribut onclick. Mais utilisez plutôt une autre option, qui consiste à établir ce lien une fois le document HTML chargé en mémoire. Pour ce faire, ajoutez la ligne suivante à la fonction init (qui est exécutée après le chargement en mémoire du document) :

```
document.getElementById("bFact").onclick = calcFactorielle;
```

Étape 7

Nous arrivons au second groupe de boutons. Occupez-vous tout d'abord du bouton remettant la valeur à zéro. Associez-lui son opération via la fonction init, comme ci-dessus.

Étape 8

Les quatre boutons suivants ont un effet similaire : +1, +10, -1, -10. Dans tous les cas, il s'agit d'ajouter un certain nombre (qui peut être négatif) à la valeur actuelle. Ce serait commode de n'écrire le code qu'une seule fois, de sorte qu'on puisse ajouter à init les lignes suivantes.

```
document.getElementById("bDecremente10").onclick = calcAjoute(-10);
document.getElementById("bDecremente1").onclick = calcAjoute(-1);
document.getElementById("bIncremente1").onclick = calcAjoute(1);
document.getElementById("bIncremente10").onclick = calcAjoute(10);
```

Pour que cela soit possible, il faudrait que chacune des expressions à droite du signe « = » soit une fonction. Autrement dit, il faut que

- calcAjoute(-10)
- calcAjoute(-1)
- calcAjoute(1)

- calcAjoute(10)

soient des fonctions. Qu'est-ce que cela implique au sujet de calcAjoute? Il faut que calcAjoute soit une fonction qui reçoit un nombre (-10, -1, 1 ou 10) et qui renvoie… une fonction ! Sur base du nombre qu'elle reçoit, calcAjoute construit une fonction et la renvoie.

Définissez la fonction calcAjoute. Il y a plusieurs manières de le faire (pensez notamment aux diverses manières d'écrire une fonction en Javascript : notation standard, notation littérale…).

Étape 9

Le dernier groupe de boutons… parmi ceux-ci, un seul réalise véritablement une opération : il s'agit du premier. Cette opération est ×2, ×5 ou ×10. Initialement, lors du lancement de la calculette, l'opération est ×2 mais, si l'utilisateur clique sur l'un des trois boutons qui suivent, l'opération change (pour devenir respectivement ×2, ×5 ou ×10).

Tout d'abord, comme dans l'étape précédente, définissez une fonction calcMultiplie(x) qui renvoie une fonction procédant au calcul et rafraîchissant l'affichage. Dans la suite, on utilisera calcMultiplie(2), calcMultiplie(5) et calcMultiplie(10).

Assurez-vous qu'initialement, on associe au bouton de multiplication l'opération calcMultiplie(2) : modifiez la fonction init en conséquence.

Que devra réaliser un clic sur le bouton « ×5 » ? Principalement deux choses :

- modifier le texte du bouton de multiplication, ce qui peut se faire grâce à :

```
document.getElementById("bMultiplie").innerHTML = "×5";
```

- modifier l'opération associée au clic sur le bouton en question :

```
document.getElementById("bMultiplie").onclick = calcMultiplie(5);
```

Un informaticien averti notera immédiatement deux choses.

Tout d'abord, dans ces lignes, on calcule deux fois « document.getElementById("bMultiplie") », ce qui revient à effectuer deux fois la recherche du bouton en question à travers le document HTML. Autant ne le faire qu'une fois puis de stocker le résultat dans une variable !

Ensuite, le code sera quasiment le même pour les trois boutons (×2, ×5 et ×10). Plutôt que de répéter du code similaire, il vaudrait mieux ne l'écrire qu'une seule fois et utiliser un paramètre (un argument) pour ce qui change (à savoir la valeur 2, 5 ou 10).

À vous d'appliquer ces principes et de terminer le code de la calculatrice !

Exercice 5 : De A à Z

Le but de cet exercice est de remplir une liste HTML de manière dynamique, en fonction des choix de l'utilisateur, puis de modifier l'ordre de présentation de son contenu à la demande.

Ètape 1

Créez tout d'abord un fichier HTML contenant le contenu suivant.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1"/>
    <style>
      li { color: black; }
      li:nth-child(odd) { background-color: #f3efe2; }
      li:nth-child(even) { background-color: white; }
    </style>
  </head>
  <body>
    <ul id="liste">
      <li>Début liste</li>
      <li>Milieu liste</li>
      <li>Fin liste</li>
    </ul>
    <button onclick="fillFruits();">Fruits</button>
    <button onclick="fillNavigator();">Navigator</button>
    <button onclick="tri();">Tri</button>
  </body>
</html>
```

À cette étape, le document HTML contient une liste minimaliste et trois boutons associés à des fonctions qui ne sont pas encore définies.

Étape 2*

Dans un premier temps, et pour manipuler un peu les tableaux, ajoutez au document un script Javascript qui initialise une variable globale `fruits` en tant que tableau contenant les chaînes de caractères suivantes : « Pomme », « Poire », « Abricot », « Fraise », « Noix de coco », « Banane », « Cerise », « Noisette ».

Codez également la fonction `fillFruits` qui devra remplacer le contenu de la liste `<li>` par les éléments du tableau `fruits` (arrangez-vous pour écrire un code générique qui fonctionne pour n'importe quel tableau de chaînes de caractères). Votre code pourra remplacer directement le « innerHTML » de la balise <ul>.

Étape 3*

Le DOM contient principalement des informations sur le document en cours, mais pas seulement. Ainsi, l'objet principal, `window`, contient une partie `window.document` relative au document mais également `window.location` (informations au sujet de la source du document actuel), `window.history` (historique du navigateur) et `window.navigator` (le navigateur).

On peut accéder à cette dernière partie soit via l'expression « `window.navigator` » soit via l'expression « `navigator` » (vu que le préfixe `window` peut toujours être omis).

Codez la fonction `fillNavigator` de manière à ce que celle-ci remplace le contenu de la liste par une liste constituée des différentes informations contenues dans `window.navigator`. Les éléments de la liste seront au format *nom* = *valeur*.

Souvenez-vous qu'il s'agit d'un objet Javascript, donc d'un tableau associatif, et que vous pouvez utiliser une boucle « `for… in…` » pour passer en revue toutes les propriétés (énumérables) de l'objet.

Étape 4*

Finalement, voici le cœur de l'exercice : implémentez la fonction `tri` qui devra trier le contenu de la liste. N'oubliez pas que le contenu en question peut varier en fonction des boutons que l'utilisateur aura cliqués (il pourra s'agir de la liste minimaliste affichée au chargement, de la liste de fruits ou de la liste des propriétés du navigateur).

Utilisez « `children` » pour passer en revue les éléments de la liste et « `innerHTML` » pour obtenir leur contenu. Si vous placez le tout dans un tableau, vous pouvez utiliser la méthode de tri prédéfinie sur le « type » `Array`. Il ne vous restera plus alors qu'à reconstruire le contenu de la liste dans le bon ordre.

Exercice 6* : Au saut du

Dans de nombreuses applications, lorsqu'on vous demande de choisir une ou plusieurs options parmi une liste donnée, on utilise une interface à 2 colonnes : la colonne de gauche reprend la liste de toutes les options et la colonne de droite cite les options que vous avez choisies. D'un simple clic, vous pouvez faire passer une option de la colonne de gauche vers la colonne de droite (c'est-à-dire l'ajouter à votre sélection) ou de la colonne de droite à la colonne de gauche (c'est-à-dire l'enlever de votre sélection).

Le but de cet exercice est d'implémenter une interface de ce genre en utilisant deux listes HTML.

Voici le fichier de départ.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1"/>
    <style>
      ul { border: 2px solid #4b3124; height: 200px;}
    </style>
  </head>
  <body>
    <table width="100%">
    <tr><td width="50%">
    <ul id="liste1">
      <li class="elem">Un</li>
      <li class="elem">Deux</li>
```

```
      <li class="elem">Trois</li>
      <li class="elem">Quatre</li>
      <li class="elem">Cinq</li>
    </ul>
    </td><td width="50%">
    <ul id="liste2">
    </ul>
  </body>
</html>
```

Celui-ci affiche deux cadres (représentant les listes). Au départ, 5 éléments se trouvent dans la colonne de gauche.

Pour vous faciliter la tâche, on a attribué les identifiants « `liste1` » et « `liste2` » aux deux listes et la classe « `elem` » à chacun des éléments.

Votre but est d'ajouter du code Javascript pour permettre à l'utilisateur de faire passer un des éléments d'une liste à l'autre en effectuant un simple clic. Ainsi, au départ, si l'utilisateur clique sur un des éléments de la liste, celui-ci devra disparaître de la `liste1` et apparaître dans la `liste2`. S'il clique à nouveau sur cet élément (alors qu'il se trouve dans la seconde liste), l'élément disparaîtra de la `liste2` et réapparaîtra dans la `liste1`. Vous ne devez pas tenir compte de l'ordre des éléments dans la liste.

Note. Faites votre code Javascript de manière indépendante du code HTML. Autrement dit, n'ajoutez pas manuellement des actions à l'événement « onclick » des éléments de la liste. Ces ajouts devront se faire après le chargement de la page, de manière dynamique (c'est-à-dire via du code Javascript). Pour ce faire, créez une fonction (par exemple `init`) qui se chargera de tout mettre en place et ajoutez la ligne « `window.onload = init;` » signifiant que, dès que tous les éléments de la fenêtre auront été chargés, la fonction `init` sera automatiquement exécutée.

Note (2). Pour faire changer un élément de liste, vous pouvez détruire le nœud HTML puis en recréer un nouveau dans la seconde liste… mais il y a plus simple : vous pouvez tout simplement réattacher le nœud existant à la seconde liste ; il sera alors automatiquement enlevé de la première (vu qu'un nœud ne peut se trouver qu'à un seul endroit dans l'arborescence).

Exercice 7 : Redimensionnements

Le document HTML de cet exercice affiche trois blocs colorés de tailles différentes (l'un est nettement plus grand que les deux autres). À eux trois, ces blocs colorés doivent se partager l'emplacement déterminé par un cadre. D'un simple clic, l'utilisateur devra pouvoir choisir lequel des trois blocs colorés va occuper la place la plus importante.

C'est un mécanisme d'affichage dynamique assez souvent utilisé dans les sites web (parfois appelé « accordéon » à cause des différentes parties qui se plient et se déplient). Certains outils, comme jQuery, permettent de réaliser des présentations de très grande qualité mais, ici, pour l'exercice, on va se contenter d'utiliser des fonctionnalités Javascript.

Étape 1

Voici le fichier HTML de départ.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1"/>
    <style>
      #cadre { border: 1px solid black; margin: auto;
               width: 600px; height: 220px; }
      #cadre div { height: 200px; margin: 10px; float: left; }
      #bloc0 { background-color: blue; width: 500px; }
      #bloc1 { background-color: red; width: 20px; }
      #bloc2 { background-color: green; width: 20px; }
    </style>
  </head>
  <body>
    <div id="cadre">
      <div id="bloc0" onclick="clic(0);"></div>
      <div id="bloc1" onclick="clic(1);"></div>
      <div id="bloc2" onclick="clic(2);"></div>
    </div>
  </body>
</html>
```

Le div « cadre » délimite l'espace disponible : 600px × 220px. À l'intérieur de celui-ci, on trouve trois div appelés « bloc0 », « bloc1 » et « bloc2 » faisant tous 200px de hauteur (+ 10px de marge en haut et en bas pour arriver au 220px du cadre) et ayant des couleurs différentes.

Horizontalement, les 600px sont divisés ainsi (de gauche à droite) :

- 10 px de marge ;
- 500 px pour le bloc0 (bleu) ;
- 20 px de marge entre deux blocs ;
- 20 px pour le bloc1 (rouge) ;
- 20 px de marge entre deux blocs ;
- 20 px pour le bloc2 (vert) ;
- 10 px de marge.

Les « petits » blocs auront donc une largeur de 20px alors que le seul et unique grand s'étendra sur une largeur de 500px.

Étape 2*

Pour mémoriser le numéro du grand bloc, ajoutez une variable globale `grandActuel` dans le script Javascript (elle sera initialisée à 0). Notez que, dans le fichier HTML, en cas de clic sur l'un des blocs, on appelle la fonction « `clic` » avec, comme argument, le numéro du bloc sur lequel on a cliqué.

Écrivez cette fonction clic sachant que

- si le numéro de bloc donné est déjà celui du `grandActuel`, elle ne doit rien faire ;

- dans le cas contraire, elle doit mettre la largeur de l'ancien grand à 20px, celle du nouveau grand à 500px et mettre à jour la variable `grandActuel`.

Note. Pour accéder à un cadre intérieur dont vous connaissez le numéro, vous pouvez passer par le grand cadre et utiliser « `children` ».

Note (2). Pour modifier la largeur d'un élément, comme il s'agit d'un composant de style, il faut utiliser `elem.style.width = "20px"` (n'oubliez pas les unités).

Étape 3*

Améliorons le côté esthétique : plutôt que de modifier les largeurs de 500px à 20px (et inversement) en une seule fois, pourquoi ne pas faire une boucle qui les modifie peu à peu. Visuellement, cela devrait donner une progression plus plaisante au lieu d'un changement brusque.

La progression pourrait s'effectuer de 10px en 10px.

Ainsi, initialement, l'ancien grand mesure 500px et le futur grand, 20px. Après une étape, le premier mesurerait 490px et le second, 30px. Après une seconde étape, on arriverait à 480px pour le premier et 40px pour le second. Et ainsi de suite…

Modifiez la fonction `clic` en remplaçant les modifications de largeur par une boucle progressive.

Étape 4*

Malheureusement, la progression reste trop rapide pour que l'œil humain puisse véritablement l'observer. Comment la ralentir ? En donnant des ordres à effectuer dans le futur, ce qui implique d'utiliser `setTimeout`.

La fonction setTimeout permet de demander l'exécution d'une fonction dans un certain nombre de millisecondes dans le futur. Ici, il s'agira de demander l'exécution dans le futur de deux modifications de largeur (celle portant sur l'ancien grand et celle portant sur le nouveau grand).

Dans un premier temps, construisez une fonction modWidth qui prend comme arguments le numéro d'un cadre (0, 1 ou 2) et la largeur qu'il faut lui donner et effectue la modification.

Ensuite, révisez le code de la fonction clic pour qu'elle planifie les appels à modWidth (vous pouvez faire une étape — c'est-à-dire ajouter/retirer 10px de largeur — toutes les 10 millisecondes par exemple).

Exercice 8 : le jeu de la vie

Le « jeu de la vie » est un petit jeu censé simuler la manière dont la vie se propage sur une grille quadrillée où chaque cellule peut être soit vivante soit morte. À chaque étape, l'état d'une cellule peut changer, c'est-à-dire passer de vivante à morte (la cellule meurt) ou passer de morte à vivante (la cellule naît).

Les changements d'état se font selon des règles très précises qui se basent sur le nombre de cellules vivantes parmi les 8 cellules voisines (en ligne droite et en diagonale) :

- Une cellule vivante qui a 2 ou 3 voisines vivantes reste vivante… sans ça, elle meurt.
- Une cellule morte qui a exactement 3 voisines vivantes naît. Sans ça, elle reste morte.

Selon la configuration initiale des cellules vivantes, l'évolution du jeu de la vie peut donner lieu à diverses figures aux propriétés étonnantes (plus d'informations sur Google).

Étape 1

Voici le fichier HTML de départ.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1"/>
    <style>
      td { width: 20px; height: 20px;
           border: 1px solid black; background-color: white; }
      td.vivant { background-color : blue; }
    </style>
  </head>
  <body>
    <p>
      <button id="bAvancer">Avancer</button>
      <button id="bContinu">Continu</button>
      <button id="bStop" disabled="true">Stop</button>
    </p>
    <table id="plateau"></table>
  </body>
</html>
```

La page affiche trois boutons intitulé Avancer, Continu et Stop et définit une table HTML (vide, pour l'instant) d'identificateur « plateau ». Les styles définis en début de fichier indiquent que les cases du tableau seront des carrés encadrés de fond blanc et que celles qui possèdent la classe « vivant », elles, auront un fond bleu.

Étape 2

Dans un premier temps, ajouter du code Javascript pour qu'au chargement de la page, on demande le nombre de lignes et de colonnes de la grille de jeu. Ne vous préoccupez pas trop des éventuels dépassements de taille (on supposera que l'utilisateur rentre des valeurs acceptables).

Prévoyez également une fonction pour « remplir » le tableau avec le nombre de lignes et de colonnes demandées. Cette fonction pourrait utiliser `innerHTML` mais, pour l'exercice, contraignez-vous à employer plutôt `document.createElement` pour chaque ligne (`tr`) et chaque case (`td`) ; cela se révélera plus pratique pour la suite.

Étape 3

On va souvent devoir intervenir sur les cases de la grille. Pour retrouver l'élément HTML qui correspond à une case de coordonnées `i`, `j` données, on pourrait passer à chaque fois par la recherche de la table HTML puis utiliser « `children[i]` » pour trouver la bonne ligne et encore « `children[j]` » pour trouver la bonne cellule.

On peut aussi décider de stocker dans une matrice / un tableau à 2 dimensions (des pointeurs vers) les cellules. Le meilleur moment pour le faire est lorsque les éléments HTML sont créés, c'est-à-dire dans la fonction écrite à l'étape précédente.

Ajoutez à votre code une variable globale `table` qui représentera le tableau de jeu. Garnissez-la au fur et à mesure de la construction de la table de manière à ce que `table[i][j]` soit la cellule située sur la ligne i (= 0, 1, 2…) et la colonne j (= 0, 1, 2…).

Note. Revoyez les slides concernant les tableaux : il faut initialiser ceux-ci sous la forme de tableaux vides avant de pouvoir y ajouter des éléments. Pour les matrices (= tableaux de tableaux), il faut non seulement initialiser la matrice mais également chacune de ses lignes !

Étape 4

L'utilisateur pourra indiquer la configuration de départ en cliquant tout simplement sur les cases à rendre vivantes. S'il clique par erreur sur une case, il pourra cliquer une seconde fois pour la remettre à son état initial. Il faut donc associer à chacune des cases une action pour l'événement « click » qui va se charger de modifier son état (vivante/morte).

Définissez une fonction « `clic` » qui sera l'action déclenchée lors d'un clic sur une des cases du tableau. Liez l'action à chacune des cases du tableau.

Note. Rendre une cellule vivante revient à lui ajouter la classe CSS « vivant » ; rendre une cellule morte revient à lui retirer la classe CSS « vivant ».

Étape 5

Cette étape vise à implémenter l'action associée au bouton « Avancer ». Celui doit faire avancer l'état global de la grille d'une étape. Cela revient à modifier l'état (a) des cellules vivantes qui ont moins de 2 ou plus de 3 voisins et (b) des cellules mortes qui ont exactement 3 voisins.

Cette partie relève plutôt de l'algorithmique générale… Vous avez donc le champ libre, mais pensez à écrire des fonctions annexes qui pourront faciliter l'écriture et la relecture de votre code. Si l'énoncé ne vous semble pas clair, n'hésitez pas à consulter la page « jeu de la vie » sur Wikipédia.

Note. Pour savoir si une cellule doit changer d'état, on se base sur l'état de ses voisines *au début* de cette étape. Ainsi, si lors d'une étape, une cellule a une voisine qui meurt, cette voisine comptera comme « vivante » lorsqu'il faudra déterminer le sort à réserver à la cellule.

Note (2). Les cases situées au bord de la grille ont évidemment moins de voisines que les autres. En fait, tout se passe pour elles comme si toutes les cases « voisines » en-dehors de la grille était mortes.

Étape 6

Le plus gros du boulot a été fait, mais il reste deux boutons : Continu et Stop. En cliquant sur le premier, l'utilisateur pourra déclencher l'avancement « continu » du jeu de la vie, avec une mise à jour de la grille toutes les demi-secondes. En cliquant sur le second, il pourra stopper l'avancement en continu.

C’est une bonne occasion pour utiliser `setInterval`, la méthode permettant de demander l’exécution d’une fonction à intervalles réguliers.

Pour améliorer l’interface, on va également griser les boutons qui ne sont pas utilisables. Ainsi, au départ, le bouton « Stop » est grisé (disabled). Lorsque l’utilisateur aura cliqué sur Continu, c’est ce dernier qui deviendra grisé et « Stop » sera accessible. Quand l’utilisateur appuiera alors sur « Stop », les deux boutons reviendront à leur état de départ.

Note. Si `b` désigne un élément HTML, on peut le griser / rendre inactif avec l’instruction `b.disabled = true`. Pour le rendre à nouveau actif, il suffit de mettre `b.disabled` à false.