Programmation web avancée - TD 4 - 3h30

Université Lumière Lyon 2 - 2023/2024

M1 Informatique

Dans cet exercice, vous allez utiliser *Javascript* pour programmer un effet visuel interactif au sein d'une page *web*. L'effet en question consiste à simuler un choc élastique entre le pointeur de la souris et les caractères composant la page *web*. Ainsi, lorsque la souris sera déplacée par l'utilisateur, les caractères touchés par le pointeur le long de son déplacement seront mis en mouvement. Plus précisément, ces caractères subiront une translation dont la direction et la vitesse seront directement liées à la direction et à la vitesse du pointeur au moment de l'impact (*cf* figure 1).

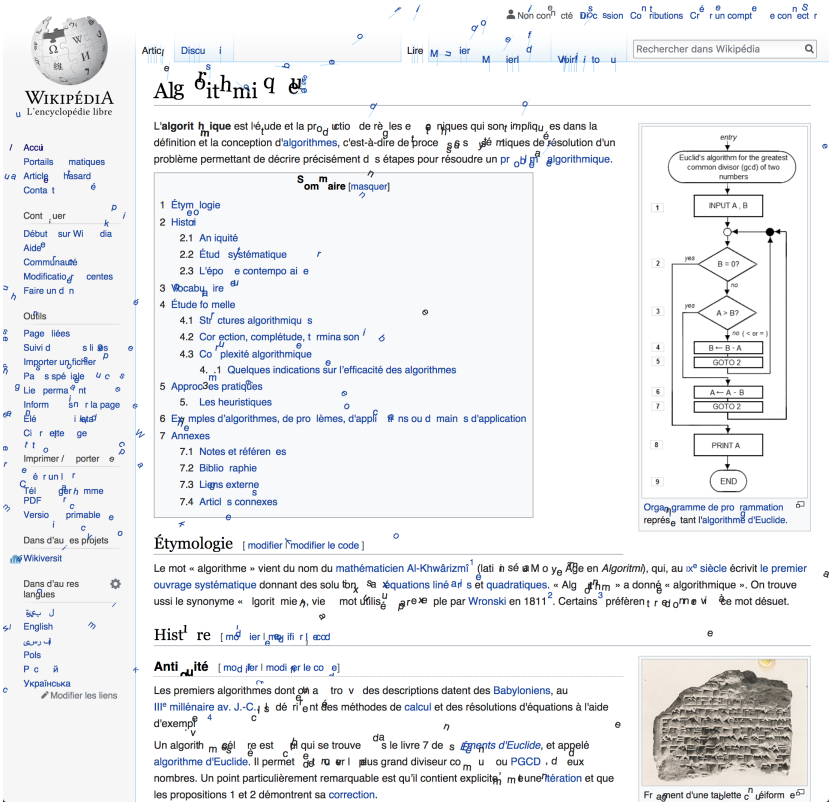

FIGURE 1 – Résultat de l'effet à implémenter sur une page *Wikipedia*

1) Fonction `_` : Écrivez une fonction `_` prenant en paramètre une chaîne de caractères contenant un sélecteur au format *CSS* et retournant un tableau contenant tous les éléments du *DOM* qui correspondent à ce sélecteur (1 ligne).

La première étape consiste à rendre les caractères indépendants les uns des autres, en encapsulant chacun d'entre eux dans une balise `<span>` individuelle. Chaque balise `<span>`

sera positionnée en `relative` de sorte que ses propriétés *CSS* `left` et `top` expriment un déplacement par rapport à son emplacement naturel dans le flux *HTML*. Et on ajoutera deux propriétés à chaque élément `<span>` : un tableau `aPos` pour mémoriser sa position horizontale et verticale par rapport à son emplacement d'origine, et un tableau `aVit` pour mémoriser sa vitesse de déplacement horizontale et verticale. C'est au moment du "choc" avec le pointeur de la souris que `aVit` recevra des valeurs non nulles (à savoir les composantes horizontale et verticale de la vitesse du pointeur de souris). À partir de cet instant, l'élément `<span>` commencera à se déplacer, c'est-à-dire que les deux valeurs du tableau `aPos` deviendront non nulles à leur tour et grandiront (en valeur absolue) au cours du temps. Plus précisément, c'est en répercutant ces deux valeurs dans les propriétés *CSS* `left` et `top` du `<span>` que son déplacement sera obtenu.

2) Fonction fGetNoeudsTexte : Écrivez la fonction `fGetNoeudsTexte` qui retournera un tableau contenant tous les noeuds de type texte (noeuds terminaux) du *DOM* (8 lignes).

3) Fonction fSepareCar : Écrivez la fonction `fSepareCar` qui remplacera chaque **noeud texte** du *DOM* par autant d'éléments `<span>` que le noeud original contient de caractères. Chaque élément `<span>` ainsi créé contiendra donc exactement un caractère du noeud texte original. Par ailleurs, vous ferez en sorte de fixer la propriété *CSS* `position` de chaque `<span>` à `relative`. Enfin, en vue de son déplacement, vous ajouterez à chaque `<span>` une propriété `aPos` pour stocker la position du caractère (initialisée à `[0,0]`) et une propriété `aVit` pour stocker sa vitesse de déplacement (initialisée aussi à `[0,0]`) (12 lignes).

Pour calculer la vitesse du pointeur de souris, on attachera un écouteur sur le document global pour surveiller le déplacement de la souris. À chaque exécution de cet écouteur, on récupèrera les coordonnées du pointeur et la date système (*timestamp* en millisecondes). En mémorisant ces informations dans un tableau global `aInfo`, on pourra accéder à la date et à la position du pointeur lors de la précédente exécution de l'écouteur, et calculer ainsi la vitesse horizontale et la vitesse verticale du pointeur entre les deux appels (on rappelle la formule *vitesse* = *distance parcourue* / *temps écoulé*). Dans ce même écouteur, on récupèrera dans un second temps la cible de l'événement et si celle-ci est l'un des éléments `<span>` créés à l'étape précédente, on affectera la vitesse du pointeur au tableau `aVit` du `<span>` impacté.

4) Fonction fEcouteSouris : Écrivez l'écouteur `fEcouteSouris` associé à l'événement de déplacement de la souris sur le document. En cas de survol d'une balise `<span>` créée à l'étape précédente, vous ferez en sorte de lui transférer la vitesse du pointeur de souris et de lui ajouter la classe `moving` pour marquer le fait qu'elle possède désormais une vitesse non nulle et que sa position devra donc être mise à jour régulièrement (12 lignes).

Il reste maintenant à écrire la fonction d'animation, qui sera invoquée à intervalles réguliers et réalisera le déplacement de tous les caractères dont la vitesse est non nulle (c'est-à-dire de tous les `<span>` possédant la classe `moving`). Pour que le déplacement soit réaliste, on diminuera progressivement la vitesse des `<span>` en appliquant un coefficient de frottement. Plus concrètement, avant tout déplacement de caractère, chaque composante `v` de la vitesse sera réduite de `v * tps * coefFrot` où `tps` est le temps écoulé depuis le déplacement précédent et `coefFrot` est la valeur du coefficient de frottement (0.005 donne de bons résultats). Suite à cette mise à jour de la vitesse, les éléments dont les deux composantes de vitesse ont atteint des valeurs inférieures à un certain seuil (0.001 par exemple) se verront retirer leur classe `moving` (leur déplacement prend fin). Tous les autres `<span>` seront déplacés (calcul de la nouvelle position et répercussion sur les propriétés *CSS* `left` et `top`).

5) Fonction fAnime : Écrivez la fonction `fAnime` qui réalise la mise à jour (et le déplacement le cas échéant) de tous les éléments `<span>` possédant la classe `moving` (18 lignes).

6) Fonction fInit : Vous pouvez maintenant écrire la fonction `fInit`, appelée au chargement de la fenêtre, qui pose les écouteurs, réalise la séparation des caractères en `<span>` individuelles et lance l'animation (6 lignes).

Vous allez implémenter à présent le retour des caractères à leur position initiale. L'animation de retour sera déclenchée par une frappe sur la barre d'espace. Pour que la fonction `fAnime` puisse détecter la phase d'animation (déplacement ou remise en ordre), on utilisera une variable globale `tBack` qui contiendra 0 en phase de déplacement, et le *timestamp* correspondant à la fin de la remise en ordre sinon. Par ailleurs, vous pourrez utiliser une classe `moved` pour marquer les caractères qui ont été déplacés par le pointeur de souris et doivent donc être remis à leur place originale.

7) Fonction fEcouteClavier : Écrivez la fonction `fEcouteClavier` qui réagira à une frappe sur la barre d'espace (de code clavier 32) en déclenchant la phase de remise en place des caractères (sauf si cette phase est déjà en cours). Cela consistera à affecter à la variable `tBack` la date système courante augmentée de 1,5 seconde (ou toute autre durée choisie pour la phase de remise en ordre des caractères) (4 lignes).

8) Modification de fAnime : Complétez la fonction `fAnime` pour animer les caractères pendant la phase de retour à leur position initiale. Vous ferez en sorte que tous les caractères parviennent à leur position d'origine au même instant (20 lignes).

9) Améliorations : Modifiez le code existant pour implémenter les améliorations suivantes :

- lors de la phase de séparation des caractères, ignorer les noeuds texte situés à l'intérieur des balises `<script>` et `<style>`, ainsi que les noeuds texte ne contenant que des espaces, tabulations et fins de ligne ;
- ajouter un effet de rotation des caractères impactés par le pointeur de souris ;
- ajouter une modification de la taille des caractères impactés pour donner une impression de mouvement en 3D, comme si les caractères étaient projetés en l'air par le pointeur de souris puis redescendaient sous l'effet de la gravité ;
- empêcher les caractères de sortir de l'écran, comme si les bords de la fenêtre du navigateur formaient des murs.

Remarque finale : un script *Javascript* peut être appliqué à n'importe quelle page chargée dans le navigateur (et pas seulement à la page *HTML* qui référence ce script) grâce au mécanisme des *bookmarklets*, ou *applications bookmark*. Pour créer une *bookmarklet*, on crée un *bookmark* (ou favori, ou encore signet) dans le navigateur mais au lieu de lui associer une *URL* classique, on lui associe un code *Javascript* précédé du tag `javascript:`. Un clic sur l'icône de ce *bookmark* exécutera son code *Javascript* sur **la page actuellement chargée** dans la fenêtre du navigateur. Il est donc possible d'appliquer l'effet décrit dans ce document à n'importe quelle page *web* (ce qui explique comment la figure 1 a été obtenue).

1 Annexe : *API Node*

La classe *Javascript* `Node` permet de manipuler *tous* les noeuds du *DOM*, non seulement les éléments *HTML*, mais aussi les noeuds texte, les attributs ou encore les noeuds *XML* en général (par exemple les primitives d'une figure *SVG*). En plus d'être universelles, ces manipulations sont plus précises et plus efficaces qu'en utilisant la propriété `innerHTML` de la classe `Element`. Elles sont aussi, hélas, sensiblement plus lourdes à écrire.

1.1 Informations sur le noeud

Les propriétés suivantes de la classe `Node` permettent d'obtenir de l'information sur un noeud :

- `nodeType` : contient un entier correspondant au type du noeud (1 pour un élément *HTML*, 2 pour un attribut, 3 pour un noeud texte, 8 pour un commentaire, *etc.*) ;
- `nodeName` : contient le nom de la balise pour un noeud de type élément, le nom de l'attribut pour un noeud attribut, `#text` pour un noeud texte, `#comment` pour un commentaire ;
- `nodeValue` : contient le texte d'un noeud texte, la valeur d'un noeud attribut, ou `null` pour un élément.

Notez que `nodeValue` (à la différence de `nodeType` et `nodeName`) est accessible en écriture. On pourra ainsi l'utiliser pour modifier le contenu d'un noeud texte ou la valeur d'un attribut.

1.2 Navigation dans le *DOM*

Les propriétés suivantes de la classe `Node`, accessibles en lecture seule, permettent de récupérer un noeud relativement à un noeud de départ `nd` donné :

- `nd.parentNode` contient le parent de `nd` ;
- `nd.childNodes` contient les enfants de `nd`, dans un tableau ;
- `nd.firstChild` contient le premier enfant de `nd` ;
- `nd.lastChild` contient le dernier enfant de `nd` ;
- `nd.previousSibling` contient le frère/soeur précédant `nd` ;
- `nd.nextSibling` contient le frère/soeur suivant `nd`.

Ainsi, en partant de la racine (pointée par `document.documentElement`) et à l'aide de ces propriétés, il est possible par exemple de réaliser un parcours complet du *DOM*.

Par ailleurs, il est important de comprendre que les caractères espace, tabulation, et retour à la ligne dans le code source *HTML* génèrent un noeud texte dans le *DOM*. Par exemple, avec le code *HTML* suivant :

```
<html>
    <body>
        <h1>Juste un titre</h1>
        <p>et un paragraphe</p>
    </body>
</html>
```

la propriété `childNodes` du noeud `body` ne contiendra pas 2 mais 5 enfants :

- un premier noeud texte contenant "\n\t\t" (un retour à la ligne suivi de deux tabulations) ;
- l'élément `h1` ;
- un autre noeud texte contenant "\n\t\t" ;
- l'élément `p` ;
- un dernier noeud texte contenant "\n\t" ;

1.3 Manipulation des attributs

À partir d'un noeud `nd`, les méthodes :
— `nd.getAttribute("nom")`,
— `nd.setAttribute("nom", "valeur")`,
— `nd.hasAttribute("nom")`,
— `nd.removeAttribute("nom")`,
permettent d'agir sur les attributs de `nd`.

Mais les attributs étant eux-mêmes des noeuds, il est également possible de les manipuler avec les méthodes suivantes :
— `a1 = n1.getAttributeNode("nom")` retourne, sous forme de noeud, l'attribut `nom` du noeud `n1` ;
— `n2.setAttributeNode(a1)` ajoute au noeud `n2` l'attribut correspondant au noeud `a1` ;
— `n2.removeAttributeNode(a1)` supprime du noeud `n2` l'attribut correspondant au noeud `a1`.

1.4 Création d'un nouveau noeud

Les méthodes de création appartiennent à l'objet `document` et diffèrent suivant le type de noeud à créer :
— `document.createElement("nomBalise")`
— `document.createTextNode("contenuTexte")`
— `document.createAttribute("nomAttribut")` : la valeur sera spécifiée ensuite, par exemple par `nd.nodeValue = "..."`

Il est également possible de créer un nouveau noeud par clonage d'un noeud existant, avec la méthode `cloneNode(bDeep)` de la classe `Node`. Le paramètre `bDeep` sera mis à `true` pour obtenir une copie en profondeur, *ie.* avec tout le sous-arbre du noeud original, ou bien à `false` pour une copie superficielle.

1.5 Insertion d'un noeud dans le *DOM*

Après avoir créé un noeud par l'une des méthodes du paragraphe précédent, on souhaitera généralement l'insérer dans le *DOM* :
— `p.appendChild(nd)` : le noeud `nd` est inséré comme dernier enfant du noeud `p` ;
— `p.insertBefore(nd, ndRef)` : le noeud `nd` est inséré parmi les enfants du noeud `p`, juste avant le noeud `ndRef`.

Notez bien qu'un noeud `nd` récupéré dans le *DOM* (avec `document.getElementById()` par exemple) puis inséré par l'une des méthodes précédentes sera *déplacé*. Pour le copier, il faudra insérer le noeud obtenu par clonage de `nd` (méthode `cloneNode()`).

Enfin, un noeud pourra être retiré du *DOM* avec la méthode `p.removeChild(nd)`, qu'on invoquera sur le *parent* du noeud à retirer.