UNIVERSITÉ CÔTE D'AZUR

IUT Nice Côte d'Azur

Programmation Web « Client Riche »
M413 - TD n°3 (séance 4)

1 Objectifs

Ce TD illustre la partie du cours sur la programmation web et en particulier les concepts liés à la conception d'application riches coté client, c'est-à-dire dans votre navigateur.

Les principaux concepts abordés seront :

- La prise en main des environnements de développement.
- Les principes de base du langage JavaScript.
- La découverte du DOM.

Vos « **livrables** » qui seront à déposer dans **la boîte de dépôt ad-hoc sur le LMS Moodle d'UCA avant la date spécifiée**, seront basés sur l'arborescence fournie et constitués **a minima** des **dossiers assets/, css/, js/** et des **fichiers** suivants :

- Les réponses à chacune des questions posées devrons être rédigées au format « Markdown » dans le fichier « **README.md** » fournie.
- Des fichiers index.html, *.html, js/td{x}.js js/*.js, css/td{x}.css, css/*.css et plus si nécessaire…

https://fr.wikipedia.org/wiki/Markdown

2 Documentation

Pour ce TD, mais également pour les suivants, sauf mention contraire dans l'énoncé ou si un lien vous est fourni, vous ne devez pas aller chercher des réponses sur internet !

Seuls les sites suivants sont autorisés :

- Le cours sur : https://lms.univ-cotedazur.fr/
- Le site de Mozilla : https://developer.mozilla.org/fr/
- Les sites du W3C : pour la validation HTML5 ou du CSS
  Mais aussi pour la documentation sur le DOM.

3 Prise en main de l'environnement

Pour réaliser ce TD, vous aurez également besoin d'un éditeur de texte pour écrire le code de vos pages. Vous pouvez par exemple utiliser un des logiciels **Brackets**, **Notepad**++, **Eclipse**, etc.

Pensez à sauvegarder régulièrement votre travail et à commenter votre code.

N'hésitez pas à sauvegarder les différentes versions, évolutions d'un même exercice de manière à pouvoir facilement réviser ou le réutiliser par la suite…

4 Les Outils pour parcourir le Document Object Model

De nos jours, des outils de débogage évolués sont intégrés aux différents navigateurs (cf. cours). Ces outils vous seront très utiles pour explorer l'arbre DOM de votre page, vérifier les propriétés CSS ou encore connaitre les erreurs de votre code JavaScript !

5 Rappel de cours

Lors de ce TD, l'ensemble des pages web écrites devra (sauf mention contraire) être constitué de pages XHTML5 valide ne contenant pas de mise en forme (squelette index.html et *.html fournis).

La mise en forme sera dans un ou plusieurs fichiers CSS valides
(squelette css/td{x}.css css/*.css fournis).

De même, le code JavaScript sera également dans des fichiers séparés
(squelette js/td{x}.js js/*.js fournis).

6 Exercice 1 : Le Jeux de Taquin (obligatoire)

Pour cet exercice vous devez partir de la page XHTML5 « **taquin.html** » et du fichier CSS « **css/taquin.css** » fournis. Dans un premier temps, aucun ces deux éléments ne pourra être modifié, seul le fichier JavaScript « **taquin.js** » le sera.

Vous trouverez le descriptif du Jeux de Taquin à l'URL suivante :

https://fr.wikipedia.org/wiki/Taquin

Bien lire l'ensemble des consignes avant de commencer à coder.

Pour cet exercice vous ne devrez pas utiliser les méthodes **getElementByXY()** et **getElementsByYZ()** !

A vous de trouver d'autres solutions…

Complétez la fonction **onLoad()** qui ajoute de manière propre un écouteur sur l'évènement « **click** » pour chaque élément HTML de type ** ** étant dans le classe CSS « **box** ».

Ecrivez une fonction **selection(event)** qui échangera la position de la case vide avec la case sur laquelle on vient de cliquer si et seulement si, ces deux cases ont un coté commun.

Pensez à utiliser l'objet JavaScript Math.

Vous n'avez pas le droit, pour le moment d'écrire d'autres fonctions.

- Que pouvez-vous dire de l'architecture de l'application ?

7 Exercice 2 : Le 2048 (obligatoire)

Vous trouverez les règles du 2048 à l'URL suivante :

https://fr.wikipedia.org/wiki/2048_(jeu_vidéo)

Alors que dans le premier exercice on s'est concentré sur la partie graphique de l'application (et on a utilisé les éléments HTML et le DOM pour modéliser la grille de jeu), dans ce deuxième exercice nous allons nous concentrer sur les fonctionnalités et la gestion des données en JavaScript.

En effet, le premier exercice mélangeait volontairement la partie « noyau fonctionnel » de l'application et la partie présentation.

Dans ce deuxième exercice, nous allons faire l'effort de séparer ces deux parties.

En partant du fichier XHTML5 « **2048.html** » (que vous n'avez pas le droit de modifier) écrivez le fichier CSS et JavaScript pour réaliser le jeu 2048 (vous devrez suivre les consignes données).

Bien lire l'ensemble des consignes avant de commencer à coder.

La grille sera modélisée sous la forme d'un tableau de tableau et initialisée dans une fonction init().

Chaque case de la grille sera modélisée par un objet :

```
// myBox object initializer
const myBox = {
      value: "",
      lastInsert: false
};
// The last Box
let lastBox = Object.create(myBox);
```

ayant une valeur (**value**) et un booléen (**lastInsert**, « false » à l'initialisation) qui servira à indiquer si la valeur contenue est une nouvelle valeur insérée dans la grille par la fonction **insertValue()**.

La fonction **init()** :

- Créera la grille, l'initialisera (chaque case aura pour valeur une chaîne vide).
- Abonnera la fenêtre à l'action « appui sur une touche du clavier » (la fonction appelée sera **keyboardAction()**).
- Ajoutera 2 valeurs aléatoirement dans la grille.
- Affichera la grille via la fonction **displayGrid()**.

La fonction **keyboardAction()** appelera les fonctions **moveUp()**, **moveDown()**, **moveLeft()** et **moveRight()** en fonction de la touche de direction pressée.

Les autres touches seront ignorées. Les touches qui nous intéressent ont un **keyCode** entre 37 et 40.

La fonction **displayGrid()** mettra à jour le DOM en fonction de la grille. La dernière case ajoutée sera en rouge. Pour le moment on ne s'occupe pas des autres effets graphiques (couleur des cases, animations, …).

Chacune des fonctions **move{Up,Down}()** fonctionnera de la fonction suivante :

- Pour toutes les lignes (ou les colonnes en fonction du sens),
- si la ligne courante (ou la colonne) n'est pas vide :
  - tasser les cases dans le sens demandé,
  - fusionner les cases de même valeur,
  - tasser à nouveau les cases pour supprimer les blancs générés par la fonction de fusion.
- S'il y a eu au moins un mouvement ou une fusion dans les actions précédentes
  - ajouter une nouvelle valeur dans la grille,
  - afficher la grille.

L'ajout d'une valeur dans la grille sera faites par la fonction insertValue(value, coordinate)

- la valeur sera déterminée par une fonction **getNewValue()** qui retournera un **2** avec une probabilité de **0.9** et un **4** dans le reste des cas. La fonction **Math.random()** vous retourne une valeur dans l'intervalle [0 ; 1[.
- la coordonnée sera un objet composé de deux attributs, ligne et colonne. Elle correspondra à une case vide de la grille, déterminée aléatoirement par la fonction **getEmptyBox()**.

Les fonctions **tampTowards{Left,Right}(row)** ou **tampTowards{Up,Down}(column)** déplaceront l'ensemble des éléments de la ligne (ou de la colonne) dans la direction spécifiées pour supprimer les cases vides. Il n'y a pas de fusion de nombre dans ce cas.

Les fonctions **mergeTo{Left,Right}(row)** ou **mergeTo{Up,Down}(column**) fusionneront les éléments de même valeur qui sont cote à cote. En respectant la direction spécifiée.

Attention, ici on ne déplace pas de cases, certaines doublent de valeur, d'autres deviennent vide.

Les fonctions **isEmptyRow(row)** et **isEmptyColumn(column)** retourne vrai si la ligne (ou la colonne) ne contient que des éléments de valeur vide.

8 Exercice 3 : Amélioration du Taquin (conseillé)

Modifiez votre jeu de taquin pour avoir un ensemble de fonction qui représente le modèle de la grille (initialisation aléatoire de la grille, taille de la grille dynamique, placement des cases en CSS dynamique, victoire au jeu, …).

Liez votre modèle aux DOM via un ensemble de fonctions qui ne font que modifier l'affichage en fonction de l'état du modèle.

9 Exercice 4 : Amélioration du 2048 (optionnel)

Même si nous avons mis en place, lors de l'exercice 2, la mécanique principale d'un jeu de type 2048, il reste encore pas mal de travail à faire pour s'approcher de l'original.

Pour cela :

- Ajoutez une fonction **isVictory()** qui retourne vrai si le joueur a obtenu une case d'une valeur égale à 2048 pour la première fois.
- Ajoutez une fonction **isDefeat()** qui retourne vrai s'il n'y a plus de case vide dans la grille et si aucun des déplacements ne provoque une fusion.
- Ajoutez une fonction, qui lors de la mise à jour de l'affichage, modifie la couleur (via le css) dans case en fonction de leur valeur.
- Ajoutez une fonction qui permet d'animer le déplacement des cases.