Extract of Les nouvelles technologies pour l'enseignement des mathématiques

http://revue.sesamath.net/spip.php?article525

Création d'exerciciels avec jQuery

- N°36 - septembre 2013 -

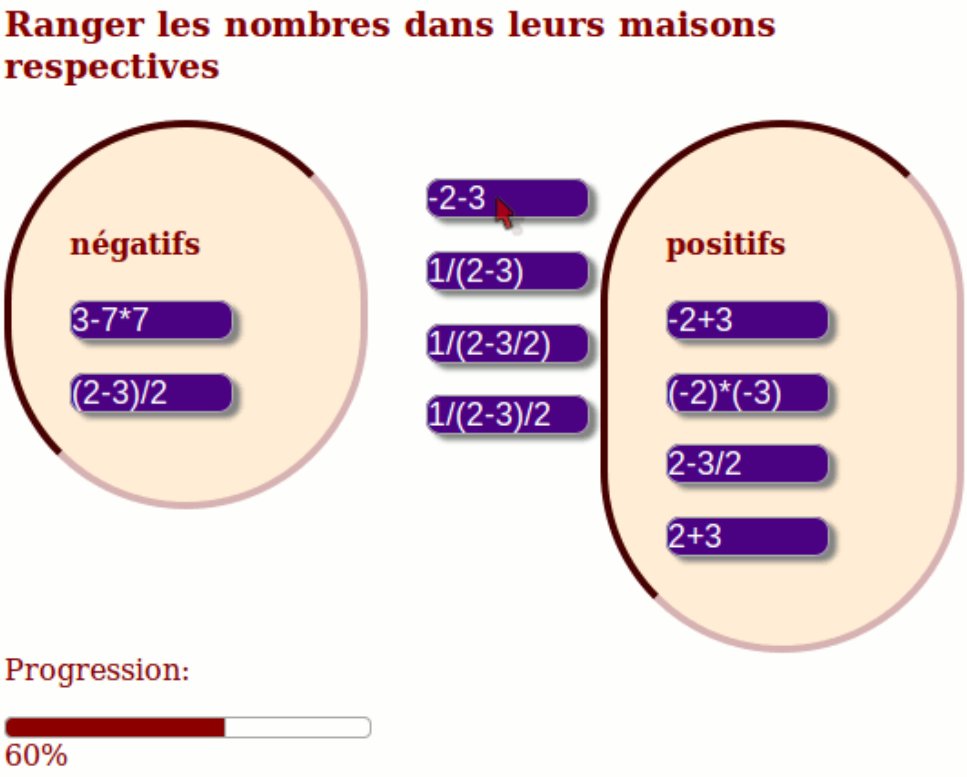
Publication date: vendredi 14 juin 2013

Description:

comment créer rapidement des exerciciels (comme webApp) utilisables sur tablette tactile (notamment), avec jQuery

La tablette tactile, n'étant pas munie de vrai clavier, ne se prête pas bien à des exerciciels classiques, où l'élève entre des données sous forme textuelle au clavier. Mais jQuery UI permet de créer des activités (exercices ou autres) où l'élève manipule au stylet (ou à la souris) des données déjà présentes : Tri, classement...

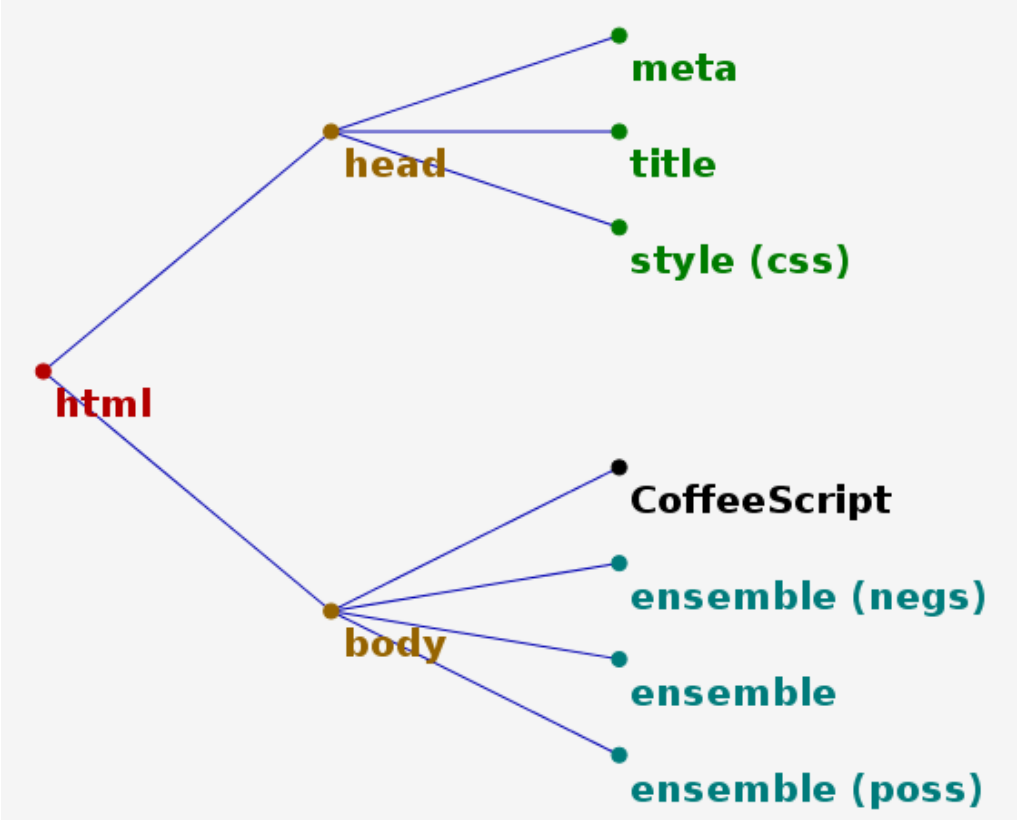
Lorsque les mathématiques se disaient "modernes", une activité fréquente dès le premier cycle consistait à classer des objets (éléments d'ensembles) selon certains critères. Par exemple, encercler les nombres positifs parmi des nombres disséminés au hasard sur une feuille de papier. La tablette permet de faire l'inverse, en demandant à l'élève de déplacer des nombres pour les caser dans deux ensembles fixes à l'écran, l'un pour les nombres positifs et l'autre pour les nombres négatifs. Cela produit l'exercice suivant [1] :



Dans cet article, nous allons montrer comment on peut construire le genre d'application web présentée dans la vidéo ci-dessus, en suffisamment peu de temps pour répéter ou décliner à l'envi ce genre d'exercice. Les outils qui serviront ici seront *jQuery* et ses ajouts *jQuery UI* et *jQuery Mobile*, ainsi que le langage de programmation CoffeeScript (une sorte de Python simplifié). Mais la base du travail de production, c'est le tandem *html/css*

structure html

Le langage Hypertext Markup Language est un langage à balises ; les balises sont comme des parenthèses, écrites entre des signes "inférieur" et "supérieur" pour la balise ouvrante, la même chose avec un "slash" pour la balise fermante. Or, qui dit parenthèses imbriquées, dit arbre. La structure générale de la webApp que nous allons construire, et la suivante : Dans l'entête ("head"), des utilitaires divers et des effets de style ("css") ; et dans le corps ("body"), le script qui implémente les algorithmes, et trois ensembles (des "div"), dont deux ont un nom :



Le code ressemble à ceci :

```
le titre de la webApp
....des liens vers les utilitaires

....la partie css avec les goûts et les couleurs dont on ne discute pas




ici on met les algorithmes


....ici se trouve l'ensemble des nombres négatifs, avec un titre (h4) et des éléments


....ici se trouve un ensemble sans délimitation, contenant le chaos initial
```

```
....ici se trouve l'ensemble des nombres positifs, avec un titre et éventuellement des nombres dedans
```

++++entête

L'entête ("head") contient des choses qu'on ne voit pas sur la webApp, mais qui sont utiles à son fonctionnement ; par exemple le titre "title" est une chaîne de caractères affichée dans l'onglet du navigateur ; il est obligatoire. La deuxième moitié du reste (la feuille de style en cascade) qui n'est ni simple ni indispensable, est réservée à l'onglet suivant. Ici on va donc détailler les liens et les balises "meta" :

1. La première balise "meta" est autofermante (avec un "slash" à la fin), et dit ; elle explique au navigateur que les caractères doivent être choisis dans la riche collection de l'unicode ("utf-8") qui comprend, outre des symboles mathématiques comme l'appartenance, l'infini, le signe de multiplication etc, les accents qui permettent par exemple d'écrire le mot "quadrilatères" sans qu'il ressemble trop à un hiéroglyphe...
2. La deuxième balise "meta" dit au navigateur que le machine sur laquelle tourne la webApp, est une tablette ou un smartphone ("viewport") et que la webApp occupe par défaut la totalité de l'écran : .

Cette balise est indispensable pour que la webApp fonctionne sur tablette tactile, et elle ne handicape nullement les utilisateurs d'ordinateur [2].

Plusieurs ressources nécessaires sont disponibles sur le net, et puisqu'on n'est pas censé utiliser le fichier autrement qu'en ligne, autant les charger à chaque utilisation, ce qui allège le fichier sans trop augmenter le temps de chargement. En voici la liste :

- Le look des objets jQuery-UI (curseurs par exemple) est stocké dans un fichier *css* (ou "stylesheet") appelé *jquery-ui.css*, téléchargé avant jQuery-UI lui-même. Il permet une certaine homogénéité de l'aspect de la webApp sur toutes plateformes, notamment les plateformes mobiles type tablette tactile.
- Pour se faciliter encore la tâche, on a choisi le langage de programmation CoffeeScript dont le compilateur est un petit fichier JavaScript appelé *coffee-script.min.js* ;
- Un autre facteur de simplification est *jQuery*, cité dans le titre de cet article. C'est également un fichier JavaScript appelé *jquery-1.7.2.min.js*.
- Celui-ci est accompagné de *Query-UI*, qui fournit des objets graphiques évolués et interactifs comme des curseurs [3]. Lui aussi est un petit fichier JavaScript, appelé *jquery-ui.min.js*.
- Enfin, le fichier *jquery.ui.touch-punch.min.js* ajoute à jQuery-UI la possibilité de simuler des évènements souris à l'aide du pointeur des tablettes. Il ne servirait à rien si la webApp n'était utilisée que sur ordinateur.

Le code de l'entête

```
classement de polygones
```

Le code complet de l'entête

```
 classement de polygones
```

```
body { color: maroon; min-width: 520px; }
.enspoly { width: 4em; float: left; padding: 2em; }
.p3 .p4 .p5 .p6 { list-style-type: none; background-color: rbga(255,255,255,0.99);}
#triangles, #quadris, #pentas, #hexas {width: 8em; border: 4px inset maroon; border-radius: 8em; background:
papayaWhip;}
.polygone { width: 80px; height: 80px; margin: 0 1em 1em 0; }
.ui-sortable-placeholder { background: white; border: 1px dotted black; visibility: visible !important;
height: 2em !important; }
#affres { width: 200px; height: 10px; }
#affres div {background: darkRed; }
```

Voici, avec la coloration syntaxique de html, l'entête du fichier (on voit que les noms des fichiers à télécharger sont longs, parce que leur adresse internet doit être complète) :

```
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>signes des nombres</title>
  <link rel="stylesheet" href="http://code.jquery.com/ui/1.10.2/themes/smoothness/jquery-ui.css" />
  <script src="http://cdnjs.cloudflare.com/ajax/libs/coffee-script/1.4.0/coffee-script.min.js"></script>
  <script src="http://code.jquery.com/jquery-1.7.2.min.js"></script>
  <script src="http://code.jquery.com/ui/1.8.21/jquery-ui.min.js"></script>
  <script src="https://raw.github.com/furf/jquery-ui-touch-punch/master/jquery.ui.touch-punch.min.js"></script>

<style>


</head>
```

L'entête comprend, comme on le devine ci-dessus, une balise de style, qui conditionne le look de la webApp. Elle est détaillée dans l'onglet suivant.

++++style

La partie la plus longue du fichier est aussi la plus inutile, puisqu'elle ne définit que l'aspect des objets présents. Mais

elle aide à comprendre l'exercice par des représentations évoquant la fonctionnalité :

- Les diagrammes de Venn étant généralement tracés au compas, les ensembles seront des rectangles à bord très arrondi (`border-radius: 8em`) et comme ils sont censés accueillir des éléments, on donne à leur bordure l'aspect d'un creux (`border: 4px inset`) ;
- Les éléments, par contre, volent au-dessus de la figure, et sont donc assortis d'une ombre (`box-shadow: 4px 3px 2px gray`).

Voici le détail des effets de style du document :

corps du document

Par souci d'unité graphique des couleurs, dans le corps du document, on écrit en marron, et la taille de celui-ci est bornée inférieurement ; cela s'écrit

```
body {
 color: maroon;
 min-width: 520px;
}
```

ensembles

Les ensembles sont des "div" de html (des sortes de sous-pages, ayant la forme d'un rectangle). Par défaut, leur largeur est de 4 caractères ("em"), ils se collent à gauche autant qu'ils peuvent, et leur bord intérieur ("padding") ne contient rien sur deux fois la largeur d'un caractère :

```
.ensemble {
 width: 4em;
 float: left;
 padding: 2em;
}
```

La notation ".ensemble" veut dire "tout ce qui a la classe d'un ensemble". Ces réglages s'appliquent donc aux trois ensembles de la figure. Mais deux d'entre eux doivent être mieux visibles puisqu'ils jouent le rôle de cibles où placer des éléments ; l'ensemble des nombres négatifs porte l'identifiant "negs" et se repère donc en CSS par *#negs*, et l'ensemble des nombres positifs porte l'identifiant "poss" et se repère donc en CSS par *#poss*. Le dièse, en CSS, signifiant "celui dont l'identifiant est"...

Ces ensembles ont donc

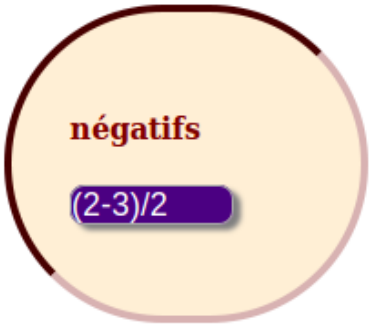
- une largeur de 8 caractères (`width: 8em`),
- une bordure de couleur marron, d'épaisseur 4 pixels, et creusée vers le bas (`border: 4px inset maroon`)
- des coins arrondis (`border-radius: 8em`)
- un fond de couleur "papaye" (`background: papayaWhip`)

Récapitulation :

```
#negs, #poss {
 width: 8em;
 border: 4px inset maroon;
```

```
 border-radius: 8em;
 background: papayaWhip;
}
```

Voici le rendu de l'ensemble des nombres négatifs sous Firefox (avec un élément dedans ; on voit le titre "négatifs" qui fait aussi partie de cet ensemble) :



éléments

Pour les mettre en exergue, les éléments (".nombre" parce que leur classe CSS s'appelle "nombre") sont foncés (fond indigo) avec une écriture blanche, et sont munis d'une ombre pour suggérer le fait qu'ils sont au-dessus de la figure. Un nombre

- a une largeur de 8 caractères (`width: 5em`)
- est séparé des autres nombres par un espace d'un caractère à droite et en bas (`margin: 0 1em 1em 0`)
- a des coins arrondis, d'un rayon de 8 pixels (`border-radius: 8px`)
- a un fond foncé (`background: indigo`)
- a un texte écrit en blanc (`color: white`
- a une ombre (`box-shadow: 4px 3px 2px gray`)

Au final :

```
.nombre {
 width: 5em;
 color: white;
 margin: 0 1em 1em 0;
 background: indigo;
 border-radius: 8px;
 box-shadow: 4px 3px 2px gray;
}
```

Le résultat sous Firefox :

Emplacements de tri

jQuery-UI fournit des emplacements vides destinés à accueillir les éléments en cours de tri. Pour faciliter le tri, on peut les rendre visibles par

```
.ui-sortable-placeholder {
 background: white;
 border: 1px dotted black;
 visibility: visible !important;
 height: 2em !important;
}
```

Voici les styles avec la coloration syntaxique de CSS (les trois dernières lignes seront expliquées dans le dernier onglet) :

```
<style>
body { color: maroon;
       min-width: 520px; }
.ensemble { width: 4em;
            float: left; padding: 2em; }
#negs, #poss {width: 8em;
              border: 4px inset maroon;
              border-radius: 8em;
              background: papayaWhip;}
.nombre { width: 5em;
          color: white;
          margin: 0 1em 1em 0;
          background: indigo;
          border-radius: 8px;
          box-shadow: 4px 3px 2px gray;}
.ui-sortable-placeholder { background: white;
                           border: 1px dotted black;
                           visibility: visible !important;
                           height: 2em !important; }
#triche { width: 200px;
          height: 10px; }
 #triche div {background: darkRed; }
 </style>
```

++++CoffeeScript

Description du langage

CoffeeScript est un langage fonctionnel [4], et les affectations portent sur des fonctions. Comme dans Xcas, la flèche qui sépare les antécédents de leurs images est simulée par un signe "moins" suivi d'un signe "supérieur". Comme en Python ou Ruby, les parenthèses sont optionnelles ; si la définition d'une fonction prend plusieurs lignes (regroupées en bloc par l'indentation, comme en Python), la dernière ligne est la valeur retournée. Exemple

```
leCarréDe = (x) ->
 x*x

alert leCarréDe 3
```

donne 9.

L'affectation se note par un signe "égal" ; de façon générale, CoffeeScript ressemble beaucoup à Python (indentation, affectations simultanées, listes en compréhension, programmation objet, etc). D'ailleurs la plupart des erreurs de programmation que nous avons faites en CoffeeScript portaient sur une indentation approximative, comme en Python...

Les tests s'écrivent avec "if...then" ou "if...then...else..." :

```
valeurAbsolueDe = (x) ->
 if x>0 then x
 else -x

alert valeurAbsolueDe -6
```

donne la même chose que

```
valeurAbsolueDe = (x) ->
 Math.abs x

alert valeurAbsolueDe -6
```

En effet, CoffeeScript est décrit par son auteur comme "du 100% JavaScript"...

En fait, il y a plus de jQuery que de CoffeeScript :

Pour les ensembles

Un ensemble est une *div* de html, dans laquelle l'élève va placer d'autres "div" : Les éléments. Pour éviter que les éléments se chevauchent, on va donner aux ensembles une propriété "sortable" [5]. Pour cela, on demande à jQuery de fournir la liste des ensembles (c'est-à-dire des objets ayant "ensemble" comme classe CSS), et on déclare ses éléments (il y en a trois) comme triables avec `$( ".ensemble" ).sortable` : C'est tout ce qu'il faut pour que les objets placés dans les ensembles puissent être triés par l'élève !

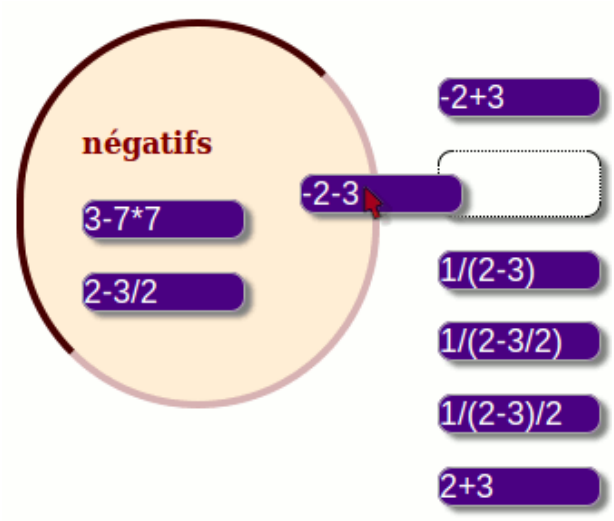
On en profite pour ajouter une propriété *connectWith* permettant de faire passer un élément d'un ensemble à l'autre (au début, parce que c'est le but de l'exercice ; ensuite, pour corriger les erreurs) :

```
$( ".ensemble" ).sortable
 connectWith: ".ensemble"
```

On remarque l'indentation qui précise que le `connectWith: ".ensemble"` est l'argument de la fonction `sortable` ; comme précédemment, `".ensemble"` désigne la liste de tous les objets de classe "ensemble", c'est-à-dire des 3 ensembles de la figure.

Pour les éléments

En fait, pour que les éléments puissent être placés dans les ensembles, il faut déjà qu'ils soient mobiles ; ce qu'on peut obtenir avec `$(".nombre").draggable`. Mais ici ce n'est pas suffisant : On souhaite que jQuery-UI gère aussi ce qui se passe lorsque l'élément est arrivé dans son ensemble ; notamment avec les emplacements possibles qui apparaissent en pointillés au survol :



Pour cela, on donne aux nombres tous les noms de classe CSS que jQuery-UI gère comme "widget" (aspect standard, gestion des évènements). Ce qui donne `$( ".nombre" ).addClass( "ui-widget ui-widget-content ui-helper-clearfix ui-corner-all" )`.

Réactivité

On souhaite qu'aucun ensemble ne soit mis en exergue, et que le ficher comprenne que le mouvement de la souris (ou du pointeur) s'applique aux éléments et pas aux ensembles. Pour cela, on confère à tous les ensembles une propriété "désactivé" avec `$( ".ensemble" ).disableSelection()`.

Enfin, pour éviter une avalanche de messages d'erreur, il faut s'assurer que les ensembles et nombres existent déjà dans la page html au moment où le script est exécuté. On stocke donc tout le script dans l'objet "ready" qui n'est actif qu'après le chargement de la page. Cet objet se note en abrégé par le symbole "dollar" et on lui ajoute la fonction (flèche avec `->` indentée) :

```
$ ->
 $( ".ensemble" ).sortable
 connectWith: ".ensemble"
 $( ".nombre" ).addClass( "ui-widget ui-widget-content ui-helper-clearfix ui-corner-all" )
 $( ".ensemble" ).disableSelection()
```

Ce programme JavaScript est tout ce qu'il faut pour que la webApp fonctionne ! En fait, jQuery est comparable au Djinn d'Aladin ou la Lampe merveilleuse, qui exauce tout ce qu'on lui demande, mais à condition qu'on lui demande. Quant à CoffeeScript, par sa concision, il rend les demandes plus simples à formuler.

++++corps

Le script CoffeeScript+jQuery agit sur le document html : Il permet de bouger des nombres dans des ensembles (ou hors d'eux). Encore faut-il que les nombres et ensembles existent ! Pour cela on les met dans le corps html du document, sans oublier de leur donner un identifiant ou une classe CSS qui permet de les repérer par la suite.

Comme on a vu dans le premier onglet, le "body" du html a une structure arborescente [6] : du "body" partent trois arêtes, chacune allant vers un des ensembles. De chacun des ensembles, partent un nombre variable d'arêtes, chacune vers un des éléments. Deux des ensembles ont, en plus, un titre (un "h4") rappelant qui ils sont.

Ensemble des nombres négatifs

Il contient au départ, deux nombres. Ce choix est censé guider l'élève dans la compréhension de l'exercice. D'ailleurs les deux nombres ne sont pas nécessairement négatifs...

négatifs

```
3-7*7
```

```
2-3/2
```

Chaos initial

Tout ce qui est extérieur aux deux ensembles *negs* et *poss* est un ensemble, contenant les nombres encore à classer :

```
-2+3
```

```
-2-3
```

```
1/(2-3)
```

```
1/(2-3/2)
```

```
1/(2-3)/2
```

```
2+3
```

Ensemble des nombres positifs

positifs

```
(2-3)/2
```

```
(-2)*(-3)
```

Nombres

Comme on le voit, les nombres à ranger ne sont pas des nombres ! Ce sont des chaînes de caractères...

Voici le corps du document avec la coloration syntaxique (les dernières lignes sont décrites dans l'onglet suivant) :

```
<body>
<script type="text/coffeescript">
<h3>Ranger les nombres dans leurs maisons respectives</h3>
<div class="ensemble" id="negs"><h4>négatifs</h4>
    <div class="nombre">3-7*7</div>
    <div class="nombre">2-3/2</div>
</div>
<div class="ensemble">
    <div class="nombre">-2+3</div>
    <div class="nombre">-2-3</div>
    <div class="nombre">1/(2-3)</div>
    <div class="nombre">1/(2-3/2)</div>
    <div class="nombre">1/(2-3)/2</div>
    <div class="nombre">2+3</div>
</div>
<div class="ensemble" id="poss"><h4>positifs</h4>
    <div class="nombre">(2-3)/2</div>
    <div class="nombre">(-2)*(-3)</div>
</div>
<p>
Progression:
<div id="triche"></div>
<div class="label"></div></p>
</body>
```

La partie CoffeeScript, décrite dans l'onglet précédent, a été "repliée" pour améliorer la lisibilité.

++++progression

Il manque encore un retour permettant à l'élève de savoir à quel point l'exercice est réussi. Comme jQuery-UI est pourvu d'une barre de progression, c'est ce biais qui a été choisi : Il donne une note sur 100 [7].

Barre de progression dans le html

À la fin du "body", on rajoute un paragraphe ("p") dans lequel trois objets sont ajoutés :

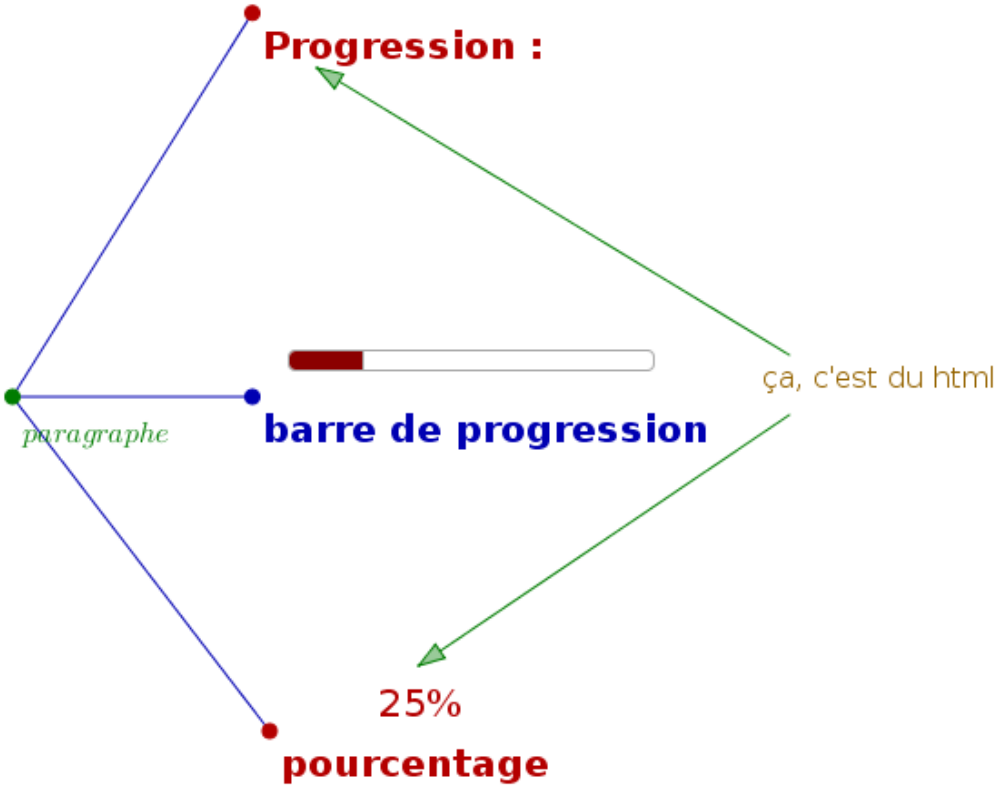
1. du texte html, disant "Progression :" ;
2. une div appelée 'triche", où sera la barre de progression elle-même :

3. une autre div, de classe "label", où sera affiché le pourcentage de nombres bien casés dans leur ensemble, sous forme d'un texte :
   .

En bref, on ajoute ceci à la fin du "body" de l'onglet précédent (visible en bas de l'onglet précédent, avec la coloration syntaxique) :


```
Progression:
```

En voici la structure arborescente :



tricher, oui, mais avec style

- La barre de progression doit être fine pour ne pas trop attirer l'attention ; on va donc lui donner les dimensions 200 pixels et 10 pixels, avec `#triche { width: 200px; height: 10px; }`
- le rectangle qui apparaît dans la barre de progression doit être en rouge foncé pour mieux s'harmoniser avec la charte graphique "papaya Whip" d'un goût douteux mais indiscutable ; comme ce rectangle est une "div" à l'intérieur de la barre de progression (qui s'appelle *triche*), il se trouve avec `#triche div` ; on écrit donc `#triche div {background: darkRed; }` pour le colorier en rouge foncé.

Action de coffeescript sur la barre de progression

La barre de progression et l'affichage du pourcentage doivent être mis à jour chaque fois que l'élève cesse une des opérations de tri (relâcher le bouton gauche de la souris, ou soulever le "touch"). Or jQuery-UI possède une fonction "stop" qui s'active dans ce cas ; elle accepte un évènement *event* (non utilisé ici) et un "ui" de jQuery, à savoir l'objet dont le mouvement vient de cesser.

Une variable *n* contient le nombre de nombres bien placés [8].

- Pour commencer, *n* est initialisé à 0 ; ce qui permet de compter à partir de là.
- Puisque jQuery retrouve l'ensemble des nombres négatifs avec `$("#negs")`, la liste des "div" contenus dans cet ensemble s'obtient par jQuery avec `$("#negs div")`. Cette liste possède une méthode "each" avec laquelle on peut boucler. Il suffit alors de fournir la fonction qui va tourner dans la boucle : `$("#negs div").each (x) ->`. Cette fonction effectue les opérations suivantes :
  - La div sur laquelle on est en train de boucler s'obtient par `$(this)` ; c'est un objet de type "div" ;
  - La chaîne de caractère qu'elle contient (comme "2+3" par exemple) s'obtient par `$(this).text()`
  - À ce texte, est appliquée la fonction "magique" de JavaScript : *eval*, qui exécute la chaîne de caractères, considérée comme du code JavaScript [9] : `eval($(this).text())` transforme par exemple, "2+3" en le nombre 5 ;
  - on peut maintenant voir le signe de ce nombre par comparaison avec 0 : Si le nombre est négatif, cela fait un nombre négatif de plus qui est bien classé. Dans ce cas, la variable *n* est incrémentée avec *n*++ et le test sur les nombres négatifs s'écrit `if eval($(this).text())<0 then n++`. Finalement le comptage des nombres négatifs bien placés se fait par ces deux lignes de CoffeeScript :

```
$("#negs div").each (x) ->
 if eval($(this).text())<0 then n++
```

- De même, les nombres positifs bien placés sont comptés, et *n* incrémenté ; *n* contient donc, à l'issue des deux boucles, le nombre de nombres bien placés (parmi les 10 au total). Donc son décuple est le pourcentage de nombres bien placés : La fameuse note de l'exercice.
- On s'adresse alors à la barre de progression (appelée `$("#triche")` pour jQuery) : On lui demande de se (re)mettre en mode "progressbar" et d'afficher la valeur `10*n`
- Enfin, on demande au label d'afficher la note suivie du caractère "pourcents", avec `$(".label").text(10*n+'%')`

En résumé, la fonction "stop" donne ceci :

```
stop: (event, ui) ->
 n=0
 $("#negs div").each (x) ->
 if eval($(this).text())<0 then n++
 $("#poss div").each (x) ->
 if eval($(this).text())>0 then n++
 $("#triche").progressbar
 value: 10*n
 $(".label").text(10*n+'%')
```

Cet algorithme, agissant par comptage automatique, fonctionne donc également avec des expressions arithmétiques engendrées automatiquement ou au hasard : Tout ce qu'il faut, c'est qu'il y en ait 10 au total ; et même ça, c'est assez facile à modifier, c'est juste un exercice sur les pourcentages.

Résumé

Voici le script complet, avec gestion de la barre de progression :

```
$ ->
 $( ".ensemble" ).sortable
 connectWith: ".ensemble"
```

```
stop: (event, ui) ->
n=0
$("#negs div").each (x) ->
if eval($(this).text())<0 then n++
$("#poss div").each (x) ->
if eval($(this).text())>0 then n++
$("#triche").progressbar
value: 10*n
$(".label").text(10*n+'%')
$( ".nombre" ).addClass( "ui-widget ui-widget-content ui-helper-clearfix ui-corner-all" )
$( ".ensemble" ).disableSelection()
```

Et voici le même script, avec la coloration syntaxique de Geany :

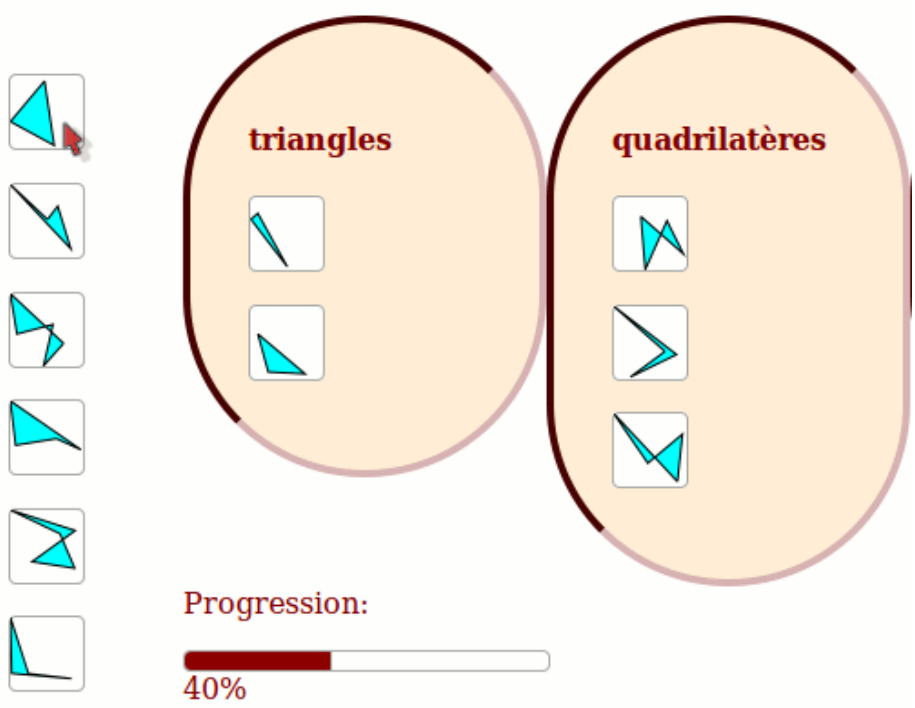
```
<script type="text/coffeescript">
###
this source is free, under MIT license
(http://opensource.org/licenses/MIT)
authors: Alain Busser, Florian Tobé
###
$ ->
    $( ".ensemble" ).sortable
        connectWith: ".ensemble"
        stop: (event, ui) ->
            n=0
            $("#negs div").each (x) ->
                if eval($(this).text())<0 then n++
            $("#poss div").each (x) ->
                if eval($(this).text())>0 then n++
            $("#triche").progressbar
                value: 10*n
            $(".label").text(10*n+'%')
    $( ".nombre" ).addClass( "ui-widget ui-widget-content ui-helper-clearfix ui-corner-all" )
    $( ".ensemble" ).disableSelection()

</script>
```

Voici d'autres exemples d'exerciciels créés par une méthode analogue :

Autre activité de classement

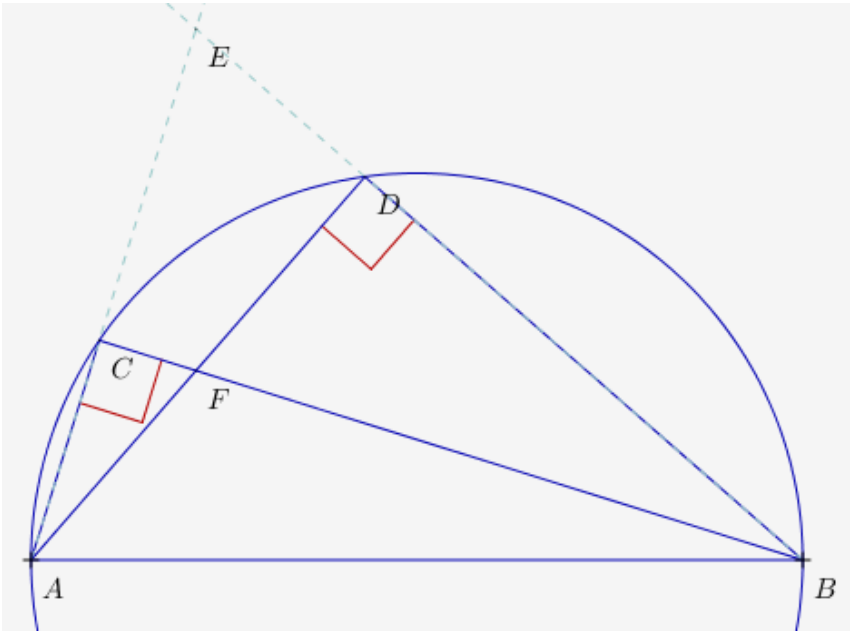
La difficulté ici, est de trouver un moyen de dessiner un polygone [10] dont aucun côté ne soit trop court pour être invisible. L'algorithme choisi ici évite une bonne partie de ces polygones difficiles à identifier, mais la barre de progression est parfois nécessaire tout de même. Le fichier est téléchargeable en bas d'article, sous le nom *polygones.html*.



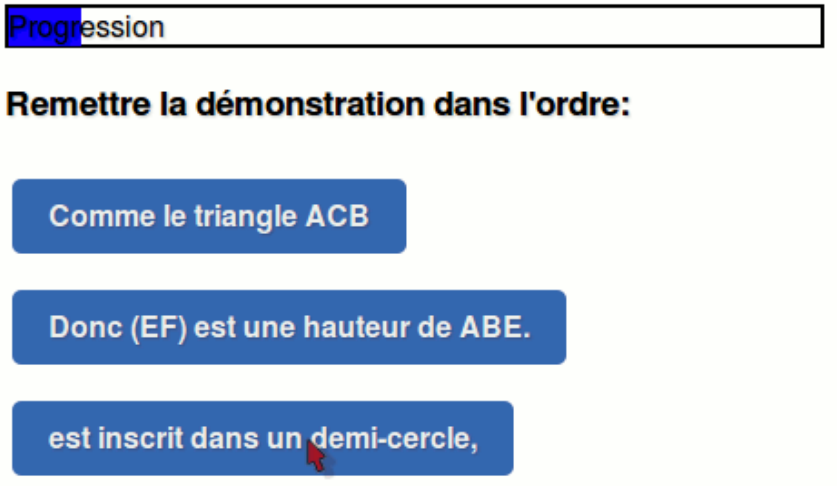
Des activités similaires peuvent être menées en biologie (classer des animaux ou des plantes...), en grammaire (classer des mots), etc.

++++**Activité de tri**

Ici, l'exercice est une démonstration mathématique à remettre dans l'ordre : Il permet donc à un élève de travailler sur du texte sans avoir à écrire. En l'occurence, on demande, avec la figure suivante :



de prouver que les droites (EF) et (AB) sont perpendiculaires ; plus précisément, de remettre la démonstration dans l'ordre :



La figure est téléchargeable en bas d'article sous le nom *démo-div.html*.

On peut aussi concevoir des démonstrations à trous. Et qu'il s'agisse de permuter ou compléter, de tels exercices peuvent aussi servir en grammaire...

On peut aussi imaginer qu'une solution d'équation soit à placer dans l'équation...

Et on peut aussi, en algorithmique, demander de remettre dans l'ordre des lignes de JavaScript pour que le résultat fasse ce qu'on attend de lui.

Il est probable que dans un avenir proche, quelqu'un réalise un portage de Scratch en webApp, à l'aide de ces techniques.

---

[1] on remarque comment l'idée de ramener des nombres SDF dans leur maison, évite de parler d'ensembles de nombres, mais on est bien en train de faire de la théorie des ensembles, ou plus précisément un diagramme de Venn.

[2] le fichier a été développé sur ordinateur (Ubuntu 10.04, Firefox 20.0 avec l'inspecteur de DOM et la console d'erreur JavaScript) et la tablette Android n'a servi qu'à tester.

[3] ici seules la barre de progrès et la possibilité de trier les éléments d'un ensemble à la souris ou au pointeur seront utilisées ; c'est déjà pas mal

[4] comme Haskell, camL ou R

[5] triable en français : C'est moi qui ne suis pas toujours sortable...

[6] répondant au doux nom de Document Object Model...

[7] parce que la barre de progression a été conçue pour afficher un pourcentage

[8] Dans le cas présent, il y a 5 nombres négatifs et autant de nombres positifs ; cela n'est nullement nécessaire.

[9] On peut donc faire des exercices d'algorithmique, où on peut classer des "snippets" de JavaScript, selon la nature des résultats qu'ils renvoient...

[10] avec la balise canvas (HTML) en l'occurence

