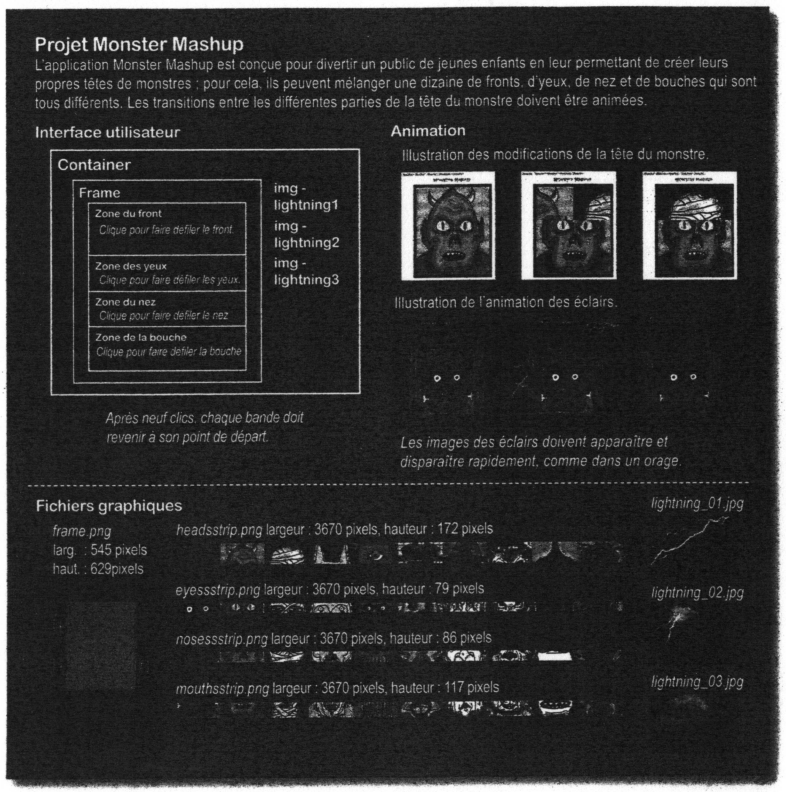
Projet Monster Mashup

Voici la maquette fournie par le chef de projet Web ainsi que les fichiers graphiques de l'application.

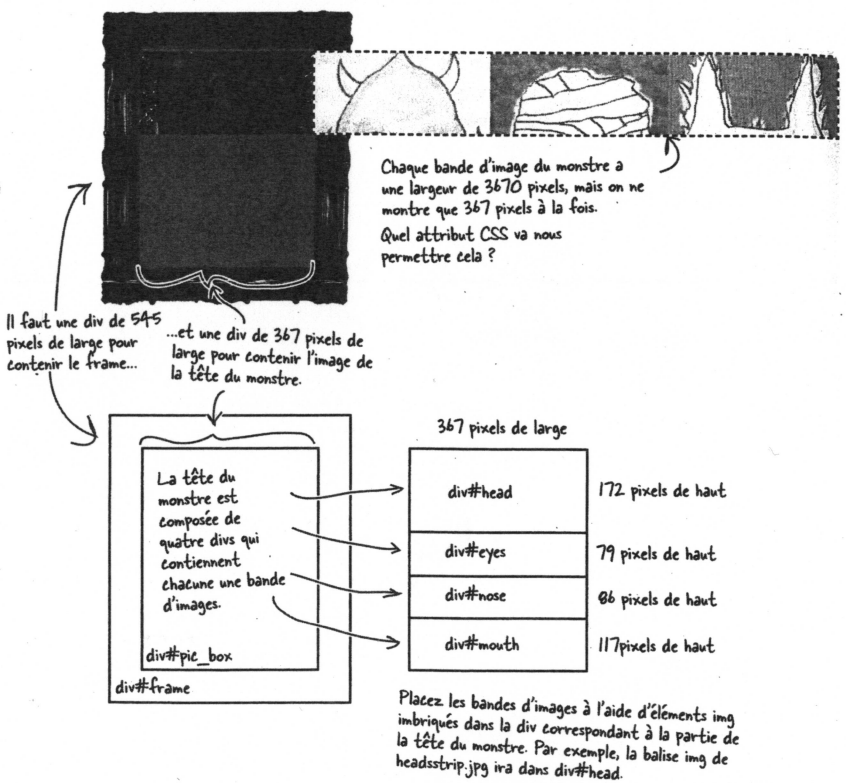


Vous avez de nombreux détails sur ce projet ainsi que les fichiers graphiques dont vous avez besoin, mais le graphiste n'a pas écrit le moindre code HTML ou CSS. C'est par là que vous allez commencer. De quoi avez-vous besoin ?

Sommaire

Monster Mashup a besoin de mise en page

Nous avons déjà beaucoup insisté sur l'importance de la conception de la structure et des styles avant d'écrire la moindre ligne de code jQuery. Ceci est encore plus vrai dans le cas présent car si vous ne soignez pas la mise en page et le positionnement, vos effets risquent de tomber à plat, *rapidement*. Il n'y a rien de pire que de contempler son code jQuery et de se demander pourquoi le navigateur ne fait pas ce qu'on attend de lui. C'est une bonne idée de rassembler vos idées et de réfléchir à ce qui va se passer à l'écran.



Dans chaque ligne vide des fichiers HTML et CSS, écrivez les CSS ID, propriétés ou paramètres qui faciliteront la mise en page et le positionnement de l'application Monster Mashup. En cas de doute, reportez-vous aux deux pages précédentes pour vous guider. Nous avons commencé le travail.

```
body>
<header id="top"><img src="images/Monster_Mashup.png" />
<p>Make your own monster face by clicking on the picture.</p></header>

<div id="frame">
   <div id="pic_box">
       <div id=.....class="face"><img src="images/headsstrip.jpg"></div>
       <div id=.....class="face"><img src="images/eyesstrip.jpg"></div>
       <div id=.....class="face"><img src="images/nosesstrip.jpg"></div>
       <div id=.....class="face"><img src="images/mouthsstrip.jpg"></div>
   </div>
</div>
  <script type="text/javascript" src="scripts/jquery-1.6.2.min.js"></script>
  <script type="text/javascript" src="scripts/my_scripts.js"></script>
</body>
```

index.html

```
#frame {
  position:.....
  left:100px;
  top:100px;
  width:545px;
  height:629px;
  background-image:url(images/frame.png);
  z-index: 2;
  overflow:.....
}

#pic_box{
  position: relative;
  left:91px;
  top:84px;
  height:460px;
  z-index: 1;
  overflow:.....
}
```

```
.face{
  position:.....
  left:0px;
  top:0px;
  z-index: 0;
}

#head{
  height:172px;
}

#eyes{
}

#nose{
}

#mouth{
}
```

my_style.css

Encore un peu de structure et de style

Vous trouverez ci-dessous les modifications de structure des fichiers HTML et CSS. Ajoutez ce code à vos fichiers *index.html* et *my_style.css*. Vous pouvez récupérer les fichiers des images à partir de *www.thinkjquery.com/chapter05*.

À faire !

Ajoutez un container et imbriquez à l'intérieur les images des éclairs.

```
<div id="container">
  <img class="lightning" id="lightning1" src="images/lightning-01.jpg" />
  <img class="lightning" id="lightning2" src="images/lightning-02.jpg" />
  <img class="lightning" id="lightning3" src="images/lightning-03.jpg" />
   <div id="frame">
       <div id="pic_box">
          <div id="head" class="face"><img src="images/headsstrip.jpg"></div>
          <div id="eyes" class="face"><img src="images/eyesstrip.jpg"></div>
          <div id="nose" class="face"><img src="images/nosesstrip.jpg"></div>
          <div id="mouth" class="face"><img src="images/mouthsstrip.jpg"></div>
       </div>
   </div>
</div>
```

index.html

```
#container{
    position:absolute;
    left:0px;
    top:0px;
    z-index: 0;
}

.lightning{
    display:none;
    position:absolute;
    left:0px;
    top:0px;
    z-index: 0;
}

body{
    background-color:#000000;
}
p{
    color:#33FF66;
    font-family: Tahoma, Verdana, Arial,
Helvetica, sans-serif;
    font-size:12px;
}
#text_top {
    position:relative;
    z-index: 4;
}
```

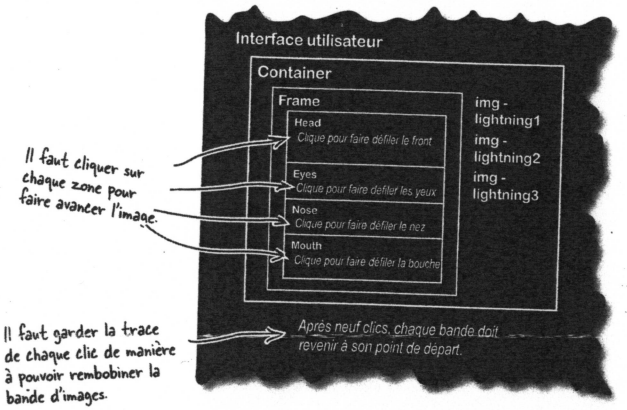
On veut que les images des éclairs soient au départ invisibles.

Quand on veut animer les éléments, on a besoin que leur propriété position soit réglée à absolute, fixed ou relative.

my_style.css

Rendre l'interface cliquable

Maintenant que vous avez défini l'aspect visuel de l'appli Monster Mashup, il faut créer le reste de la section Interface utilisateur de la maquette. Cette partie concerne essentiellement les clics qui vont déclencher les actions et nous avons déjà abordé ce sujet au cours des quatre premiers chapitres. Créer tout cela ne devrait donc être qu'une promenade de santé.



Il n'y a pas de questions idiotes

Q : Je suis un peu rouillé sur le positionnement CSS. Pourquoi est-il nécessaire pour les effets jQuery ?

R : `position` est une propriété CSS qui contrôle la manière et l'endroit où les éléments sont placés par le moteur de rendu du navigateur. jQuery accomplit beaucoup d'effets grâce à la propriété CSS `position`. Si votre mémoire a besoin d'être rafraîchie, consultez cet excellent article :

http://developer.mozilla.org/en/CSS/position#Relative_positioning

Q : Pourquoi faut-il définir la propriété CSS position à `absolute`, `fixed` ou `relative` quand on veut animer des éléments ?

R : Si on laisse la propriété CSS `position` à sa valeur par défaut (`static`), alors on ne peut pas appliquer le positionnement top, right, left ou bottom. Pour la fonction `animate`, il faut être en mesure de définir ces positions, et `static` n'autorise pas cela, ce qui n'est pas le cas des autres paramètres de position (`absolute`, `fixed` et `relative`).

Q : Vous avez évoqué le moteur de rendu du navigateur. C'est quoi au juste ?

R : Le moteur de rendu du navigateur est la partie centrale du navigateur qui interprète le code HTML et CSS et l'affiche dans la fenêtre de visualisation. Google Chrome et Safari utilisent le moteur de rendu Webkit. Firefox utilise le moteur Gecko et Microsoft Internet Explorer utilise un moteur qui s'appelle Trident.

Jeu du magnet

Placez les aimants de code dans le bon ordre pour rendre cliquable l'élément `div#head`. Assurez-vous de la séquence correcte des variables et des instructions conditionnelles de telle sorte que l'on puisse détecter le neuvième clic.

```
headclix
});
}
if (headclix
= 0;
else {
headclix
= 0;
}
});
+= 1;
$(document).ready(function(){
< 9){
var headclix
$("#head").click(function(){
```

Au travail !

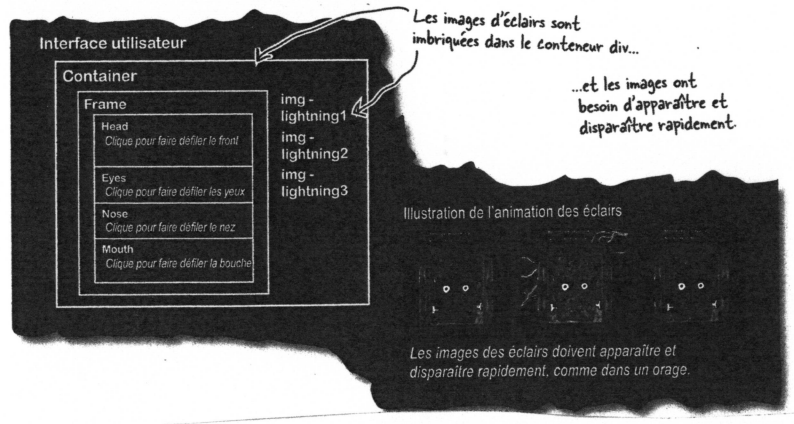
Complétez le script jQuery ci-dessous pour rendre cliquables les éléments eyes, nose et mouth. Nous ajouterons dans un moment des fonctionnalités à chaque fonction click. Assurez-vous de placer les variables et les instructions conditionnelles dans le bon ordre de manière à détecter le neuvième clic.

```
$(document).ready(function(){

        $("#head").click(function(){
             if (headclix < 9){
                    headclix += 1;
                    }
                    else{
                    headclix = 0;
                    }
        });









});
```

my_scripts.js

Créer l'effet des éclairs

Il faut ensuite s'occuper de l'effet des éclairs. Reprenons ce qui est indiqué sur la maquette avant de tenter de faire fonctionner l'effet.



Comment jQuery anime les éléments ?

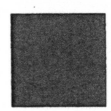
Quand le navigateur charge un fichier CSS, il définit les propriétés visuelles des éléments de la page. En utilisant les effets intégrés de jQuery, l'interpréteur JavaScript modifie ces propriétés CSS et anime le changement sous vos yeux. Mais ce n'est pas de la magie… il ne s'agit que de propriétés CSS. Révisons un peu ce que nous avons déjà vu.

hide, show et toggle changent la propriété CSS display

hide

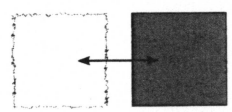
L'interpréteur JS change la propriété CSS `display` de l'élément sélectionné à `none` et le supprime de l'affichage.

show



L'interpréteur JS change la propriété CSS `display` de l'élément sélectionné pour qu'il devienne visible.

toggle



Si un élément est masqué, l'interpréteur JS va l'afficher et vice versa.

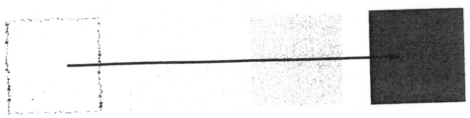
Les effets jQuery modifient les propriétés CSS à la volée, ce qui modifie l'affichage de la page sous vos yeux.

COLLE

`hide`, `show` et `toggle` concernent la propriété `display`. Mais cette fois, nous devons faire glisser les parties de la tête tout en faisant apparaître et disparaître les éclairs. Quelles propriétés CSS pensez-vous que jQuery modifie avec fade et slide ?

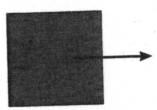
Les effets Fade animent la propriété CSS opacity

fadeIn



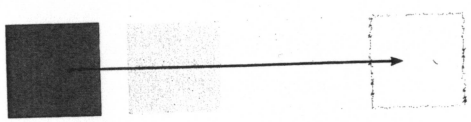
Avec `fadeIn`, l'interpréteur JavaScript modifie la propriété CSS `opacity` de l'élément sélectionné avec une valeur allant de 0 à 100.

fadeTo



`fadeTo` permet d'animer l'élément sélectionné avec une valeur de pourcentage spécifique.

fadeOut

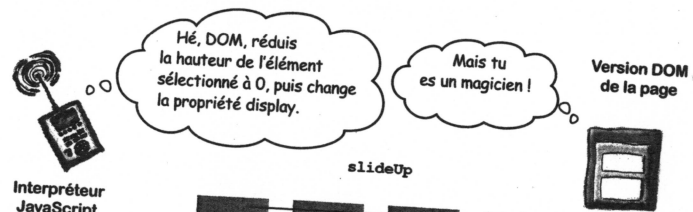


Avec `fadeOut`, l'interpréteur JavaScript modifie la propriété CSS `opacity` de l'élément sélectionné avec une valeur allant de 0 à 100, mais il conserve l'espace de cet élément sur la page.

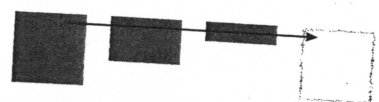
Truc de geek

La propriété CSS `opacity` ne fonctionne pas à l'identique sur tous les navigateurs. Heureusement, jQuery s'occupe de cela à notre place. Et franchement, vous n'avez pas besoin d'en savoir plus !

Glisser n'est qu'une question de hauteur

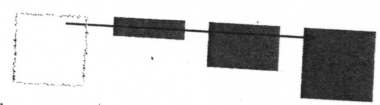


slideUp



L'interpréteur JavaScript dit au DOM de modifier la propriété CSS height des éléments sélectionnés à 0 puis définit la propriété `display` à `none`. Au fond, il s'agit de masquer et de glisser.

slideDown



L'interpréteur JavaScript fait apparaître les éléments sélectionnés en animant leur hauteur de 0 à la valeur spécifiée dans le style CSS.

slideToggle

L'interpréteur JavaScript vérifie la hauteur de l'image (pleine ou 0) et réalise un effet de slide en fonction de ce qu'il trouve. Si l'élément a une hauteur égale à 0, l'interpréteur JavaScript glisse vers le bas. Si l'élément a sa pleine hauteur, l'interpréteur JavaScript glisse vers le haut.

Au travail !

Quels effets jQuery prêts à l'emploi vont fonctionner pour l'application Monster Mashup ? Pour chaque groupe d'effets, expliquez s'ils vont nous être utiles et les raisons pour lesquelles vous les avez choisis ou ignorés.

Effet	Utilité ?	Pourquoi ?
Show/Hide		
Slides		
Fades		

Combiner des effets avec des méthodes de chaînage

Les éclairs vont devoir apparaître et disparaître de manière répétée. Au lieu d'écrire ces effets séparément, on peut utiliser un chaînage que l'on a étudié brièvement au chapitre 4 quand on a eu besoin de parcourir le DOM. Les méthodes de chaînage sont une fonctionnalité de jQuery qui relie des méthodes à exécuter sur un ensemble d'éléments. Elles vont faciliter l'écriture des effets d'éclairs.

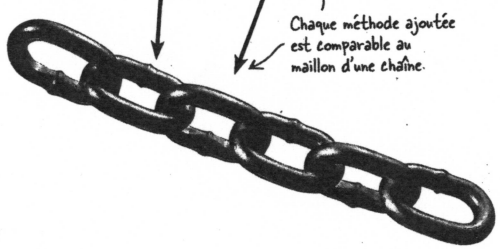
L'affichage des éléments passera du statut masqué à visible avec une totale opacité...

...puis il repassera à une transparence totale.

Si vous n'écrivez pas une durée entre parenthèses, la durée normale de l'effet sera de 400 millisecondes soit 0,4 seconde.

```
$("#lightning1").fadeIn().fadeOut();
```

Chaque méthode ajoutée est comparable au maillon d'une chaîne.



Exercice

Écrivez la ligne de code jQuery qui accomplit chacune des étapes ci-dessous.

1. Faire apparaître l'élément `#lightning1` pendant un quart de seconde.

.....

2. Chaîner un autre effet qui fait disparaître l'élément `#lightning1` un quart de seconde.

.....

Temporisation d'une fonction

On a maintenant l'effet d'éclair qui peut apparaître et disparaître, mais le cahier des charges précise que les éclairs doivent flasher en continu. Dans un véritable orage, il y a un intervalle de temps entre les éclairs et il nous faut donc trouver un moyen de *répéter* le phénomène.

Qu'avons-nous fait dans les chapitres précédents pour répéter une tâche ? Nous avons utilisé des fonctions ! Au chapitre 3, nous les avons employées pour réaliser un fonction click réutilisable et un générateur aléatoire ; à présent, nous pouvons utiliser des fonctions pour faire apparaître les éclairs, attendre un peu, puis refaire la même chose à un intervalle donné. Cela procurera un effet d'orage continu pour l'appli Monster Mashup. Jetons un coup d'œil à la fonction qui va accomplir cela.

Hé, JavaScript, crée une nouvelle fonction.

C'est le nom que l'on va utiliser pour appeler la fonction.

C'est le paramètre de timing (une variable nommée t). On le place là et on l'utilise plus bas.

```
function lightning_one(t){
    $("#lightning1").fadeIn(250).fadeOut(250);
    setTimeout("lightning_one()",t);
};
```

Cette ligne contient le code des effets jQuery.

La méthode setTimeout dit à l'interpréteur JS d'exécuter une fonction puis attend un moment avant de l'exécuter à nouveau.

On voit la puissance de JavaScript. On dit à l'interpréteur JS que l'on veut que la fonction s'appelle elle-même indéfiniment.

C'est la durée d'interruption, en millisecondes, comme la durée des effets que l'on a étudiés quelques pages plus haut.

En seulement trois lignes de code, on a une fonction temporisée d'éclairs pour la première image. On va maintenant essayer d'écrire les fonctions pour les deux autres images.

Jeu du magnet

Placez les aimants de code dans le bon ordre pour créer les fonctions temporisées d'éclairs des deux autres éléments.

```
function   lightning_two   (t){

};

function   lightning_three   (t){

};

.fadeIn(250)   t);   "lightning_two()",   .fadeOut(250);
$("#lightning2")   setTimeout(   .fadeIn(250)   "lightning_three()",
.fadeOut(250);   setTimeout(   t);
$("#lightning3")
```


Ajout de la fonction lightning au script

En utilisant le code de l'exercice de la page précédente, mettez à jour le script de l'appli Monster Mashup.

À faire !

Ces lignes appellent les fonctions qui sont définies en gras tout en bas du code.

```
$(document).ready(function(){
    var headclix = 0, eyeclix = 0, noseclix = 0, mouthclix = 0;
    lightning_one(4000);
    lightning_two(5000);
    lightning_three(7000);

    $("#head").click(function(){
        if (headclix < 9){headclix+=1;}
        else{headclix = 0;}
    });

    $("#eyes").click(function(){
        if (eyeclix < 9){eyeclix+=1;}
        else{eyeclix = 0;}
    });

    $("#nose").click(function(){
        if (noseclix < 9){noseclix+=1;}
        else{noseclix = 0;}
    });

    $("#mouth").click(function(){
        if (mouthclix < 9){mouthclix+=1;}
        else{mouthclix = 0;}
    });

});//end doc.onready function

function lightning_one(t){
    $("#container #lightning1").fadeIn(250).fadeOut(250);
    setTimeout("lightning_one()",t);
};
function lightning_two(t){
    $("#container #lightning2").fadeIn("fast").fadeOut("fast");
    setTimeout("lightning_two()",t);
};
function lightning_three(t){
    $("#container #lightning3").fadeIn("fast").fadeOut("fast");
    setTimeout("lightning_three()",t);
};
```

Les nombres entre parenthèses sont les paramètres en millisecondes qui sont passés à la méthode setTimeout. Ils permettent d'alterner les éclairs.

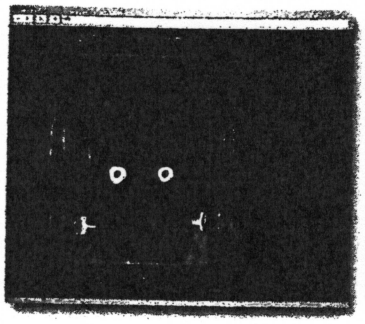
Nous avons supprimé certains sauts de ligne pour économiser l'espace sur cette page. Ne vous inquiétez pas si les sauts de ligne de votre script sont différents.

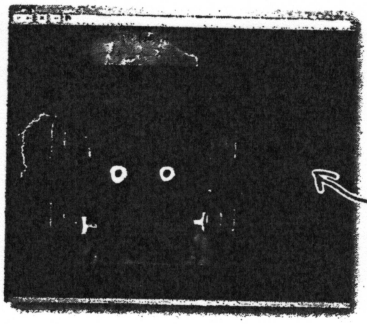
Définitions des fonctions lightning.

my_scripts.js

Banc d'essai

Ouvrez la page dans un navigateur pour voir si l'effet d'orage fonctionne.





L'effet d'éclair est réalisé en le combinant avec la méthode JavaScript setTimeout.

Les éclairs qui apparaissent et disparaissent rapidement à différents intervalles simulent un orage.

Nous avons pour l'instant réalisé les fonctions click et les effets d'orage pour les trois images. Examinons la maquette pour voir ce qu'il nous reste à faire.

Projet Monster Mashup

L'application Monster Mashup est conçue pour divertir un public de jeunes enfants en leur permettant de créer leurs propres têtes de monstres ; pour cela, ils peuvent mélanger une dizaine de fronts, d'yeux, de nez et de bouches qui sont tous différents. Les transitions entre les différentes parties de la tête du monstre doivent être animées..

Animation

Illustration des modifications de la tête du monstre.

Cet extrait de la maquette est la dernière partie du projet à réaliser.

Il nous faut donc maintenant faire des glissements sur la gauche et il n'y a pas d'effet prévu pour cela. Quelle autre méthode utiliser ?

Les effets prêts à l'emploi sont très pratiques, mais ils ne permettent pas tout.

C'est le moment de réaliser un effet *personnalisé* qui va faire glisser vers la gauche les différentes parties de la tête du monstre.

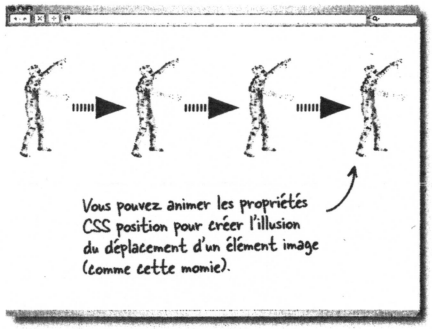
Effets personnalisés d'animation

jQuery n'a donc pas de méthode `slideRight` ou `slideLeft`, et c'est pourtant ce dont nous avons besoin à cette étape du projet. Est-ce que cela signifie pour autant la mort du projet Monster Mashup ?

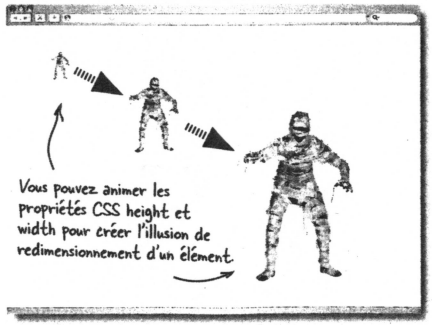
N'ayez crainte ! jQuery propose la méthode `animate` pour créer vos propres effets. Avec `animate`, vous pouvez créer des animations personnalisées qui complètent les effets prêts à l'emploi. La méthode `animate` permet d'animer les propriétés CSS des éléments sélectionnés et elle autorise aussi l'animation de plusieurs propriétés en même temps.

Regardons ce que l'on peut faire avec la méthode `animate`.

Effets de déplacement



Effets de redimensionnement



COLLE

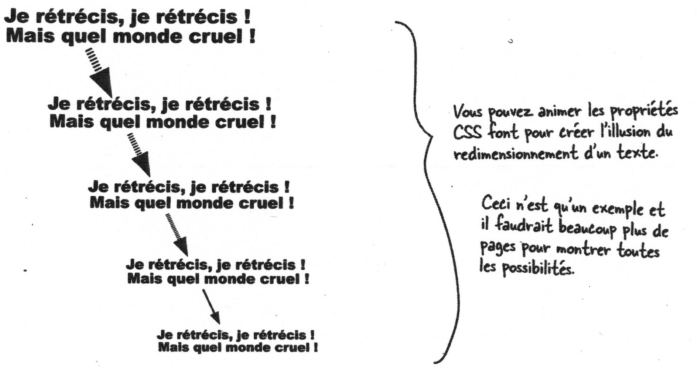
Quelle propriété CSS faut-il animer pour faire glisser à gauche les parties de la tête du monstre à chaque clic ?

Ce qui peut être animé et ce qui ne le peut pas

Avec la méthode `animate`, on peut aussi modifier dynamiquement les propriétés font pour créer des effets textuels. On peut également animer plusieurs propriétés CSS en un seul appel, ce qui offre de nombreuses possibilités aux applications.

Même si la méthode `animate` est géniale, elle a quelques limitations. En coulisse, l'animation utilise beaucoup de calculs (heureusement, on n'a pas à s'en soucier), si bien que l'on ne peut travailler qu'avec des propriétés CSS qui ont des paramètres *numériques*. En connaissant ces limites, vous pouvez exercer votre créativité dans toutes les directions.

Effets textuels



Attention !

La méthode `animate` ne fonctionne qu'avec les propriétés CSS qui utilisent des paramètres numériques.

- `borders`, `margin`, `padding`
- element `height`, `min-height`, and `max-height`
- element `width`, `min-width`, and `max-width`
- `font size`
- `bottom`, `left`, `right`, and `top position`
- `background position`
- `letter spacing`, `word spacing`
- `text indent`
- `line height`

La méthode animate à la loupe

A priori, `animate` fonctionne comme les autres méthodes que vous avez déjà étudiées.

Sélectionne les éléments que l'on veut animer.

Appelle la méthode animate.

Le premier paramètre d'animate permet de sélectionner la propriété CSS à animer.

Le second paramètre est la durée en millisecondes qui contrôle la longueur de l'animation.

```
$("#my_div").animate({left:"100px"},500);
```

Dans cet exemple, on anime la propriété CSS left...

...que l'on définit à 100 px.

Le premier argument est obligatoire, mais le deuxième est optionnel.

Mais l'une des fonctionnalités les plus puissantes d'`animate` est la capacité de modifier plusieurs propriétés des éléments sélectionnés en même temps.

```
$("#my_div").animate({
    opacity: 0,
    width: "200",
    height: "800"
}, 5000);
```

Dans cet exemple, on anime les propriétés opacity et size simultanément.

Attention !

Les paramètres des propriétés CSS doivent être définis en utilisant le standard DOM, et non pas le standard CSS.

GYM DU CERVEAU

À votre avis, que se passe-t-il en coulisse dans le navigateur pour que la méthode `animate` puisse modifier l'affichage sous vos yeux ?

Modifications temporisées des styles

Les effets visuels et les animations que l'on voit au cinéma utilisent l'illusion du déplacement. Les techniciens des effets spéciaux prennent une séquence d'images et les affichent **une par une à une vitesse spécifique** pour accomplir cette illusion (comme dans ces livrets dont on fait tourner les pages rapidement pour créer l'illusion du mouvement).

Il se produit la même chose avec l'écran du navigateur, sauf que l'on ne travaille pas avec une série d'images. Au lieu de cela, l'interpréteur JavaScript **exécute de manière répétée une fonction qui modifie le style de l'élément animé**. Le navigateur trace (ou *repeint*) ces modifications à l'écran. L'utilisateur a l'illusion du déplacement ou de la modification d'un élément au fur et à mesure de la modification de son style.

1. Quand animate s'exécute, l'interpréteur JavaScript définit un compteur pour la durée de l'animation.

Interpréteur JavaScript

Modifie la propriété CSS left à 500px pendant 400 millisecondes.

Si vous ne spécifiez pas le paramètre durée de la méthode animate, sa valeur par défaut est de 400 millisecondes.

2. L'interpréteur JavaScript dit au moteur de rendu du navigateur de modifier les propriétés CSS spécifiées dans les paramètres de la méthode animate. Le moteur de rendu du navigateur affiche une représentation visuelle de ces propriétés CSS à l'écran.

interpréteur JavaScript

Je veux planifier un réaffichage de l'écran, SVP.

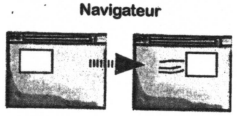
On dirait que je peux régler ça en quelques millisecondes.

Moteur de rendu du navigateur

3. L'interpréteur JavaScript appelle de manière répétée la fonction qui change la propriété CSS de l'élément jusqu'à ce que le temps défini à l'étape 1 soit écoulé. Chaque fois que la fonction s'exécute, la modification est affichée à l'écran.

4. Le visiteur a l'illusion du mouvement au fur et à mesure que le navigateur affiche les modifications de l'élément.

Navigateur



Qui Fait Quoi ?

Faites correspondre chaque extrait de code d'animation personnalisée avec ce qui est réalisé lors de son exécution.

```
$("#my_div").animate({top: "150px"}, "slow")
```

Modifie simultanément les marges gauche et droite de tous les paragraphes.

```
$("p").animate({
      marginLeft:"150px",
      marginRight:"150px"
});
```

Change à 0 la position droite de `#my_div` en une demi-seconde.

```
$("#my_div").animate({width: "30%"}, 250)
```

Anime l'espace entre les lettres de tous les paragraphes avec une durée par défaut de 400 millisecondes.

```
$("#my_div").animate({right: "0"}, 500)
```

Modifie les propriétés padding et width de `#my_div` en même temps.

```
$("p").animate({letterSpacing:"15px"});
```

Modifie la propriété top de `#my_div` lentement.

```
$("#my_div").animate({
      padding: "200px",
      width: "30%"
}, "slow")
```

Modifie rapidement la hauteur de toutes les images.

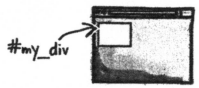
```
$("img").animate({height: "20px"}, "fast")
```

Modifie la propriété width de `#my_div` en un quart de seconde.

De quel point de vue se place-t-on *exactement* ?

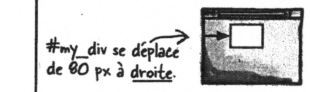
Il est important de noter que la méthode `animate` modifie l'état *actuel* de la propriété CSS spécifiée dans le *premier paramètre*. Pour rendre effective votre animation personnalisée, vous devez bien réfléchir à ce que vous avez actuellement défini dans votre CSS. Dans l'exemple précédent, nous avons modifié la position gauche de `#my_div` à 100px. Le résultat à l'écran dépend entièrement de la valeur actuelle de la propriété CSS `left` de `#my_div`.

État actuel de la propriété CSS




```
#my_div{
  left: 20px;
}
```

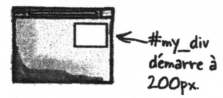
Propriété CSS modifiée




```
$("#my_div").animate({left:"100px"});
```

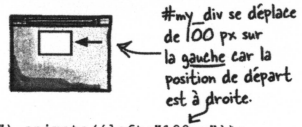
Si la propriété actuelle a une valeur différente, le résultat sera différent.

État actuel de la propriété CSS




```
#my_div{
  left: 200px;
}
```

Propriété CSS modifiée




```
$("#my_div").animate({left:"100px"});
```

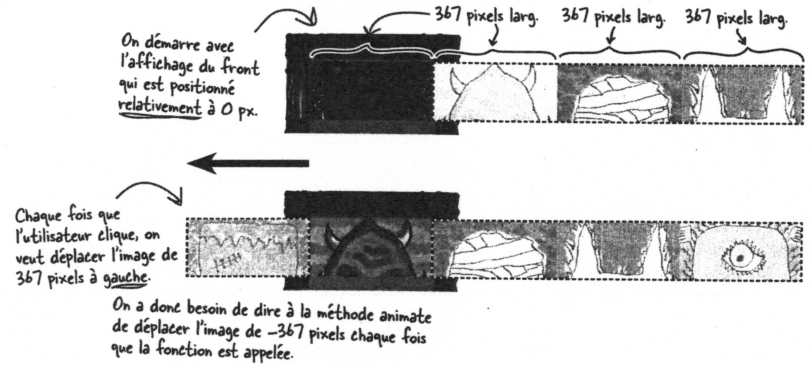
C'est bien joli tout ça, mais comment l'utiliser dans l'appli Monster Mashup ?

Tout est relatif.

Pour que les parties des têtes de l'application Monster Mashup se déplacent comme on le souhaite, nous devons connaître leurs *positions actuelles* et les modifier *relativement à* leurs positions changées la dernière fois par `animate`.

Mouvement absolu versus mouvement relatif

Rappelez-vous que l'on a imbriqué les bandes d'images que l'on veut afficher à l'intérieur d'une `div` ayant l'ID `#pic_box`. La propriété `left` de `div#pic_box` est définie à 91px dans la CSS actuelle. Pour accomplir l'effet désiré de glissement vers la gauche, réfléchissons à la manière de déplacer les bandes d'images.



Pensez à l'exemple d'animation absolue de la page précédente.

Cela dit à la méthode animate de définir la position gauche de #my_div à exactement 100 pixels.

```
$("#my_div").animate({left:"100px"});
```

Mais comment dire que l'on veut déplacer un élément de –367 pixels chaque fois que la méthode animate est appelée ?

```
$("#head").animate({left:"???"});
```

Animation relative = déplace un élément de x pixels à chaque fois

Avec une animation *absolue*, on déplace un élément vers une position *absolue* sur la grille visuelle. Avec une animation *relative*, on déplace l'élément par rapport à l'emplacement où il se trouvait la *dernière fois* qu'une animation l'a déplacé.

Mais comment déplacer un élément relativement avec la méthode `animate` ?

On se déplace relativement avec des combinaisons d'opérateurs

Il y a des opérateurs spéciaux JavaScript qui déplacent les éléments de la même distance chaque fois que la méthode `animate` est appelée. On les appelle *opérateurs d'assignation* car ils sont normalement utilisés pour assigner une valeur à une variable (la variable ajoute la nouvelle valeur à sa valeur actuelle). En fait, c'est plus simple qu'il n'y paraît.

Le signe égal est un opérateur d'assignation.

Quand on combine des opérateurs arithmétiques avec le signe égal, on obtient des raccourcis pratiques.

```
a = 20
a += 30
a -= 10
```

L'opérateur = assigne la valeur 20 à la variable a.

Le signe plus combiné avec l'opérateur égal est un raccourci de "a = a + 30".

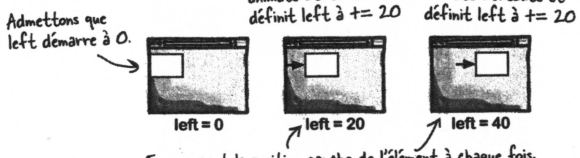
Ici, le signe moins combiné avec l'opérateur égal est un raccourci de "a = a – 10".

Ces combinaisons d'opérateurs facilitent la création d'une animation relative en permettant de définir une valeur à partir de sa valeur actuelle *plus ou moins* un nombre de pixels.

Cela déplace l'élément ayant l'ID box de 20 pixels chaque fois que la méthode animate est appelée.

```
$("#box").animate({left:"+=20"});
```

Voici ce qui arrive à `#box` à chaque appel de la méthode `animate` ci-dessus.



Point clés

Il y a d'autres combinaisons d'opérateurs :

- `a *= 5` est le raccourci de multiplie 5 par la valeur actuelle de `a` et assigne cette valeur à `a`.
- `a /= 2` est le raccourci de divise la valeur actuelle de `a` par 2 et assigne cette valeur à `a`.

Écrivez la ligne de code jQuery qui va accomplir chaque étape ci-dessous.

1. Déplace l'élément `#head` de 367 pixels vers la gauche à chaque appel de `animate` pendant une durée d'une demi-seconde.

   .....

2. Replace l'élément `#head` à sa position originale (`left:0px`) pendant une demi-seconde.

   .....

Ajout des fonctions animate au script

À faire !

En utilisant le code assemblé dans l'exercice de la page précédente, mettez à jour le script de l'application Monster Mashup.

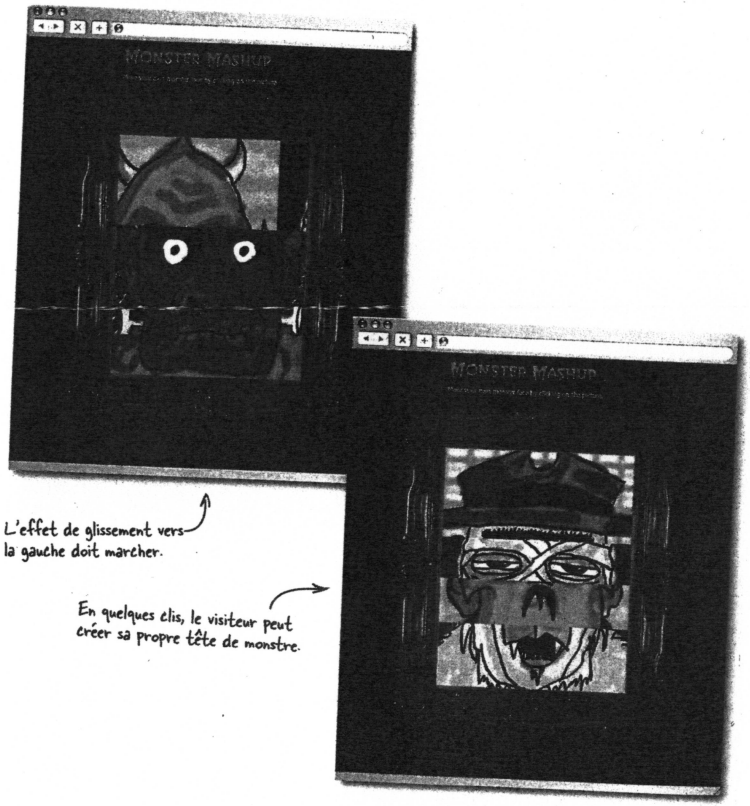
```
$("#head").click(function(){
      if (headclix < 9){
            $(this).animate({left:"-=367px"},500);
            headclix+=1;
      }
      else{
            $(this).animate({left:"0px"},500);
            headclix = 0;
      }
});
$("#eyes").click(function(){
      if (eyeclix < 9){
            $(this).animate({left:"-=367px"},500);
            eyeclix+=1;
      }
      else{
            $(this).animate({left:"0px"},500);
            eyeclix = 0;
      }
});

$("#nose").click(function(){
      if (noseclix < 9){
            $(this).animate({left:"-=367px"},500);
            noseclix+=1;
      }
      else{
            $(this).animate({left:"0px"},500);
            noseclix = 0;
      }
});

$("#mouth").click(function(){
      if (mouthclix < 9){
            $(this).animate({left:"-=367px"},500);
            mouthclix+=1;
      }
      else{
            $(this).animate({left:"0px"},500);
            mouthclix = 0;
      }
```

On peut ici utiliser le mot-clé "this" car on est à l'intérieur de la fonction pour l'élément sur lequel on a cliqué.

my_scripts.js

Banc d'essai

Ouvrez la page dans un navigateur pour vous assurer que tout fonctionne.

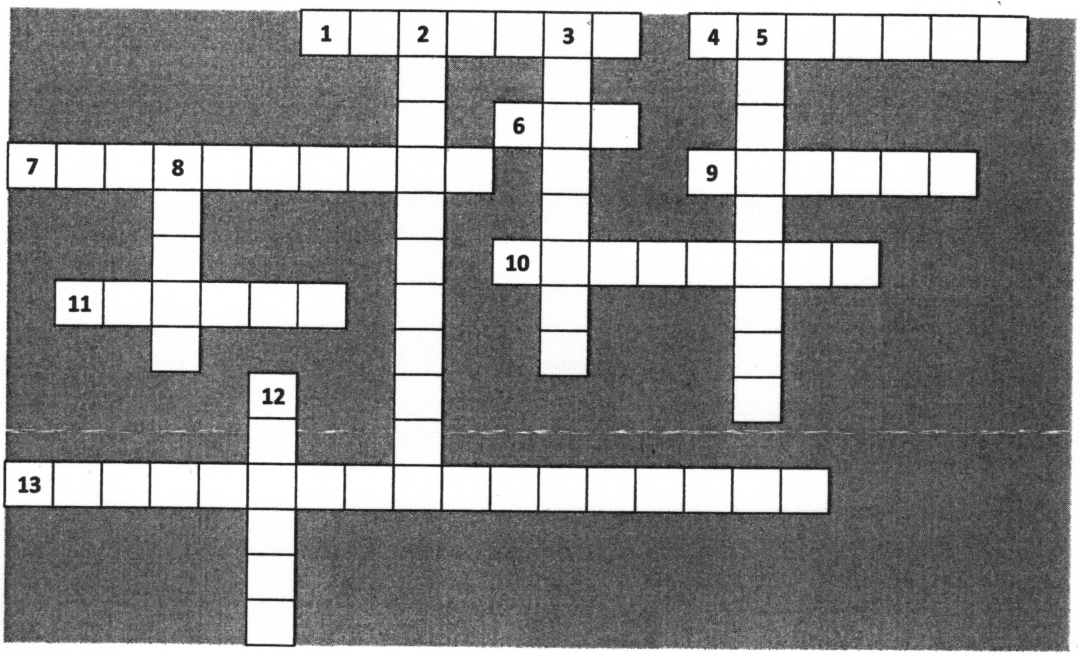


L'effet de glissement vers la gauche doit marcher.

En quelques clics, le visiteur peut créer sa propre tête de monstre.

jQuery-croisés

C'est le moment de vous installer confortablement et de solliciter votre hémisphère gauche. Toutes les définitions de ce mots-croisés font référence à des mots étudiés dans ce chapitre.



Horizontalement

1 hide, show et toggle animent cette propriété CSS.
4 jQuery offre cette méthode pour créer des effets personnalisés.
6 Les effets jQuery utilisent la manipulation des propriétés _____ à la volée.
7 _____ = 1 000 millisecondes.
9 On peut utiliser les propriétés CSS _____ et width pour créer l'illusion de redimensionnement d'un élément.
10 Paramètre qui contrôle le temps que prend un effet pour s'achever.
11 Méthode d'effet qui permet d'animer l'élément sélectionné en spécifiant l'opacité.
13 Fonctionnalité jQuery qui permet de lier des méthodes à exécuter sur un ensemble d'éléments.

Verticalement

2 Méthode d'effet qui fonctionne de la manière suivante :
si l'élément sélectionné a une hauteur de 0, l'interpréteur JS le fait glisser vers le bas ;
si l'élément a sa pleine hauteur, l'interpréteur JS le fait glisser vers le haut.
3 Quand on veut animer un élément, on doit définir sa position à _____, fixed ou relative.
5 La méthode animate ne fonctionne que sur les propriétés CSS qui ont une valeur _____
8 Effet à utiliser quand on veut animer la propriété height d'un élément.
12 Quand on exécute cet effet jQuery, l'interpréteur JS modifie la propriété CSS opacity de l'élément sélectionné de 0 à 100.