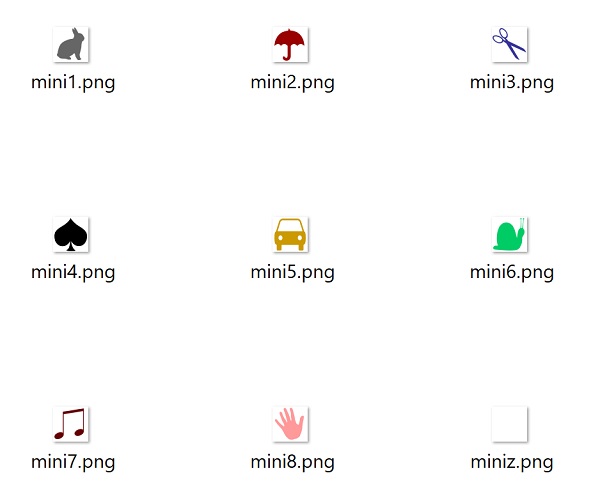
**Jeu du Memory pour le Web**  
  
Nous proposons de créer le **jeu du Memory** pour les pages Web clientes. Ce **jeu** consiste à retrouver les paires d'images correspondantes. Au début de la partie, après un **positionnement aléatoire**, toutes les miniatures restent affichées durant quelques secondes. Puis elles disparaissent. Ce **jeu** fait donc appel à la concentration. Dans ce laps de temps, il s'agit de mémoriser les dessins et leurs emplacements. L'internaute clique sur une première case et dévoile l'image qu'elle masque. Il clique alors sur une seconde case pour afficher son dessin. Si les deux correspondent, l'association est validée. Les images restent affichées. Dans le cas contraire, le score est décrémenté et les dessins disparaissent de nouveau. Il s'agit de reconstruire les huit associations des seize images pour finir le jeu. Le temps imparti est limité à trois minutes.   
  
 

La capture ci-dessus de l'application Web finalisée montre le résultat d'une partie menée avec succès. L'internaute a obtenu le score de 9/10 en 28 secondes. Il n'a commis qu'une seule erreur, pour retrouver toutes les paires d'images.   
  
**Sources et présentation du concept**

* décompresser le fichier 18- jeu-du-memory.rar dans le dossier de votre choix,

Comme vous le constatez, la décompression conduit au fichier de la page Web principale nommé : **jeu-du-memory.html**. Il est accompagné de ses ressources externes dans les sous dossiers locaux. Le **sous dossier styles** par exemple, renferme la feuille de **styles mef.css**. C'est elle qui permet la mise en forme ainsi que la mise en page des différents éléments Html, articulés dans la page Web.

* Double cliquer sur le **sous-dossier mini** pour afficher son contenu,

Vous y notez la présence de miniatures d'images à disposer de façon aléatoire sur la grille du jeu de la page Web, afin de permettre la reconstitution des paires. L'une d'entre elles est une image vierge (miniz.png) qui sera utilisée pour les masquer toutes, après un délai ayant permis de mémoriser les associations à reproduire.   
  
   
  
Les autres portent des noms judicieux pour simplifier leur appel par le **code Javasscript**, à l'aide d'une boucle de traitement. Chaque nom est en effet préfixé de la chaîne statique mini suivi d'un numéro incrémenté.

* A la racine du dossier de décompression, double cliquer sur le fichier de la page Web : **jeu-du-memory-cpt.html**, pour l'afficher dans un navigateur Internet,

Comme nous l'avons dit plus haut, la **structure Html** de la page Web est déjà conçue. Les images sont positionnées aléatoirement à l'aide d'un **code Javascript**, emprunté aux formations précédentes. D'ailleurs si vous enfoncez la **touche F5 du clavier** pour rafraîchir la page, vous remarquez qu'elles se réorganisent différemment. Un bouton Nouvelle partie, non fonctionnel à ce stade, doit permettre de réinitialiser le jeu. Le score juste au-dessus, doit être décrémenté, au fur et à mesure des erreurs commises, pour accompagner l'internaute durant sa partie.

* A la racine du dossier de décompression, cliquer désormais avec le bouton droit de la souris, sur le fichier de la page Web : jeu-du-memory-cpt.html,
* Dans le menu contextuel, choisir de l'ouvrir avec un éditeur tel que le Notepad ++,

Nous accédons ainsi à l'organisation Html fort conventionnelle de la page Web. Entre les lignes 49 et 78 pour un éditeur Notepad, vous notez l'imbrication des calques ayant permis de construire cette grille pour accueillir les images à placer.   
  
*<div class='colonne' id='liste'>  
<div id='grille'>  
<div class='case' id='case0'></div>  
<div class='case' id='case1'></div>  
<div class='case' id='case2'></div>  
<div class='case' id='case3'></div>  
  
<div class='case' id='case4'></div>  
<div class='case' id='case5'></div>  
  
...  
  
<div class='case' id='case14'></div>  
<div class='case' id='case15'></div>  
</div>  
Vous avez trois minutes,<br />soit <span style='color:#CC3300;'>180 secondes :</span><br/><br /><br />  
<span style='font-size:18px;font-weight:normal;'>Votre score:<br />  
<div id='scrore' style='font-size:30px;'><strong>10</strong>/10</div>  
<br /><br />  
<div id='temps' style='font-size:20px;'></div><br />  
<input type='button' class='btn' value='Nouvelle partie'/>  
</span>  
</div>*  
  
Seize cases numérotées de 0 à 15 sont prévues pour accueillir les huit paires d'images. A l'instar des miniatures, elles possèdent toutes un identifiant construit sur une partie fixe (case), suivie d'un numéro incrémenté. L'objectif une fois encore, consiste à faciliter leur énumération par le **code Javascript**, avec une **boucle de traitement**.  
  
Sous les cases, disposées à droite sur la page Internet, vous notez la présence des données de synthèse. Certaines sont encapsulées dans des calques (Elément Div) avec identifiant, pour permettre de les modifier, selon le cours du jeu.  
  
Enfin, tout en bas du **code Html**, figure une **section de script**. Celle-ci propose la fonction permettant le **placement aléatoire** des images. Cette fonction est appelée par son nom si bien que son code s'exécute en même temps que la page se charge.  
*generation();  
  
function generation()  
{  
var nb\_alea; var nb\_img='';  
var test = true; var chaine = '';  
  
for (var i=0;i<16;i++)  
{  
while (test==true)  
{  
nb\_alea = Math.floor(Math.random()\*16) + 1;  
if(chaine.indexOf('-' + nb\_alea + '-')>-1)  
nb\_alea = Math.floor(Math.random()\*16) + 1;  
else  
{  
nb\_img = Math.floor((nb\_alea+1)/2);  
document.getElementById('case' + i).innerHTML = '<img style='cursor:pointer;' id='img' + i + '' src='mini/mini' + nb\_img + '.png' onClick='verifier('img' + i + '','mini' + nb\_img + '')' alt='' />';  
chaine += '-' + nb\_alea + '-';  
test=false;  
}  
}  
test=true;  
}  
}*  
  
Nous avons néanmoins ajouté une adaptation intéressante par rapport aux formations précédentes. 16 positions aléatoires sont à générer (for (var i=0;i<16;i++)). Pourtant, nous ne disposons que de 8 images. Mais chacune d'entre elles doit apparaître deux fois. Nous générons donc ces nombres entre 0 et 15 (nb\_alea =Math.floor(Math.random()\*16) + 1), tout en veillant à ce qu'ils n'aient pas déjà été proposés. Nous prélevons la partie entière du nombre divisé par 2 (nb\_img= Math.floor((nb\_alea+1)/2)). Ce principe est reproduit 16 fois, si bien que pour chacune des huit images, deux positions aléatoires différentes sont proposées. 

**Masquer les images après temporisation**  
Au chargement de la page ou à la création d'une nouvelle partie, les images doivent rester affichées quelques secondes puis disparaître. Comme il est nécessaire de mémoriser plusieurs informations au cours du jeu, nous devons commencer par [déclarer des variables publiques](https://www.bonbache.fr/les-variables-en-vba-excel-premiers-pas-en-developpement-119.html).

* Au tout début de la section du script, avant l'appel de la fonction generation, ajouter les déclarations de variables suivantes :

*var nb\_clics = 0;  
var mini1 = ''; var mini2='';  
var case1 = ''; var case2='';  
var img\_ok = 0; var nb\_erreurs=0;  
var le\_score=0;  
var depart = false; var temps\_debut = new Date().getTime();*  
  
A chaque case cliquée par l'internaute, nous devons savoir s'il s'agit du premier ou du second clic, d'où l'importance de la **variable nb\_clics**initialisée à zéro. Au premier clic, l'image (mini1) et la case (case1) cliquée doivent être mémorisées pour comparaison ultérieure. Au second clic, la nouvelle image (mini2) et sa nouvelle case (case2) doivent être mémorisées pour réaliser la comparaison avec les premières. Les cases ne doivent pas correspondre. En d'autres termes, l'utilisateur ne doit pas cliquer deux fois au même emplacement. Si les images ne correspondent pas, la **variable nb\_erreurs** doit être incrémentée pour restituer le score (le\_score), au fur et à mesure de la partie. Si les 16 paires ont été reconstituées, la partie est finie. C'est la **variable img\_ok** incrémentée de deux unités à chaque nouvelle association validée, qui doit en informer. Le **booléen depart** réglé à **False** est une sécurité qui permettra de comptabiliser ou non, les clics de l'utilisateur, selon le contexte. Enfin, nous prélevons le temps qu'il est au démarrage de la partie. Pour ce faire, nous instancions la **classe Javascript Date** associée à sa **méthode getTime**. Nous mémorisons ainsi l'heure en millisecondes dans la **variable temps\_debut**. La différence avec le même calcul en fin de partie permettra de compter le temps écoulé en secondes.  
  
Pour que les miniatures disparaissent au bout d'un certain temps, le **code Javascript** doit être temporisé avant d'agir. Et comme nous l'avait appris la [formation Javascript pour créer un diaporama automatisé](https://www.bonbache.fr/diaporama-automatise-de-photos-pour-le-web-en-javascript-234.html), c'est la **fonction setTimeout** qui permet de réaliser des actions après un délai. Sa syntaxe est la suivante :  
  
*var attente = setTimeout(function() { Traitements }, delai\_en\_millisecondes);*  
  
Son résultat doit en effet être affecté à une variable. Ses actions doivent être réalisées par une **fonction** qui peut être écrite dans le flux, soit en premier paramètre.

* Après l'appel de la **fonction generation**, créer les bornes de la **fonction setTimeout**, comme suit :

*var attente = setTimeout(function()  
{  
  
},4000);*  
  
Les actions que nous allons inscrire entre les bornes de la fonction s'exécuteront au bout d'un délai de 4000 millisecondes, soit 4 secondes. Il s'agit de parcourir les 16 **balises img** inscrites à la volée par la **fonction generation**, pour remplacer leurs miniatures par l'image vierge.

* Entre les bornes de la fonction de l'instruction setTimeout, ajouter les lignes de code suivantes :

*for (var i=0;i<16;i++)  
{  
document.getElementById('img' + i).src = 'mini/miniz.png';  
}  
depart = true;*  
  
Les contrôles images étant tous préfixés du **nom img** suivi d'un incrément numérique, nous les passons tous en revue du premier (0) au dernier (15), grâce à une [boucle Javascript For](https://www.bonbache.fr/traitements-recursifs-optimises-en-javascript-223.html). Pour chacun d'entre eux, nous chargeons l'image vierge grâce à la **propriété src** d'une **balise Html img**. Comme toujours, c'est la [**méthode getElementById** de l'**objet Javascript document** qui permet de pointer sur un élément Html désigné par son identifiant](https://www.bonbache.fr/interactions-html-et-evenements-javascript-222.html). Ensuite, nous n'oublions de basculer la **variable booléenne à True** afin d'indiquer que la partie peut commencer.

* Enregistrer les modifications (CTRL + S) et basculer sur le navigateur Web,
* Rafraîchir la page Internet à l'aide de la touche F5 du clavier,

Tout d'abord, les images se réorganisent **aléatoirement** grâce à la **fonction generation**. Puis elles disparaissent en effet au bout de 4 secondes.  
  
   
  
La partie peut donc débuter. Mais pour cela, nous devons encore développer le code gérant les clics de l'internaute. L'objectif est de comparer les images désignées.

* Revenir dans l'éditeur Html de la page Web,

Avant de poursuivre le développement, nous proposons de rendre fonctionnel le **bouton Nouvelle partie**. Il lui suffit de rafraîchir la page pour forcer son chargement.

* Dans la balise du bouton Nouvelle partie, en ligne 76 pour un éditeur Notepad, ajouter la gestion de l'événement du clic, comme suit :

*...  
<div id='scrore' style='font-size:30px;'><strong>10</strong>/10</div>  
<br /><br />  
<div id='temps' style='font-size:20px;'></div><br />  
<input type='button' class='btn' value='Nouvelle partie'****onClick='window.location.reload()'****/>  
</span>  
</div>  
...*  
  
Si vous enregistrez les modifications et rafraîchissez la page, vous constatez qu'après un clic sur ce bouton, les images se réorganisent aléatoirement, puis disparaissent au bout de 4 secondes. Nous avions abordé l'**objet window** et sa **propriété location** dans la [formation Javascript pour récupérer les paramètres d'Url](https://www.bonbache.fr/recuperer-les-parametres-d-url-d-une-page-web-en-javascript-258.html). Ils permettent de désigner la page Web en cours dans la fenêtre du navigateur. Ainsi, la **méthode reload** force le rechargement de la page. En conséquence, les variables publiques sont de nouveau déclarées, la **fonction generation** est appelée et la **fonction setTimeout** temporise avant de masquer les miniatures. 

**Comparer les images cliquées par le code Javascript**  
La **fonction generation** crée les **balises Html img** à la volée en même temps qu'elle définit leur emplacement aléatoire. Elle en profite pour leur associer un gestionnaire d'événement (onClick).   
  
*...  
nb\_img = Math.floor((nb\_alea+1)/2);  
document.getElementById('case' + i).innerHTML = '<img style='cursor:pointer;' id='img' + i + '' src='mini/mini' + nb\_img + '.png'****onClick='verifier('img'+ i + '', 'mini' + nb\_img + '')' alt='' />****';  
chaine += '-' + nb\_alea + '-';  
test=false;  
...*  
  
Ainsi, au clic sur chacune des images, la **fonction verifier** est appelée avec deux paramètres. Il s'agit du **nom de l'image** et du **nom de la balise img** hébergeant l'image. L'objectif est de comparer que les noms des miniatures sont bien identiques, dans la mesure où l'internaute n'a pas cliqué sur le même emplacement. Cette fonction n'existe pas, nous devons donc la créer.

* Sous la **fonction generation**, créer la **fonction verifier** comme suit :

*function verifier(limg,source)  
{  
if(depart==true)  
{  
  
}  
}*  
  
Nous lui attribuons les deux paramètres qu'elle doit recevoir, selon l'appel effectué au sein de chaque **balise Img**, après le positionnement aléatoire. Puis, nous débutons par un test qui consiste à avorter la suite du traitement si la **variable booléenne** n'est pas calée à **True**. C'est elle qui détermine si les clics de l'internaute doivent être comptabilisés. Souvenez-vous, dans la **fonction setTimeout**, elle est basculée à True au bout de 4 secondes, soit une fois que toutes les miniatures ont été masquées.  
  
A chaque appel de la **fonction verifier**, il s'agit de comptabiliser le nombre de clics total, grâce à la **variable publique nb\_clics**. S'il s'agit du second, la variable devra être réinitialisée à zéro, pour reprendre le processus de découverte des miniatures. A chaque appel, l'image cliquée doit également être dévoilée, quitte ensuite à être masquée.

* Dans les bornes de l'**instruction conditionnelle If**, ajouter les deux affectations suivantes :

*nb\_clics++;  
document.getElementById(limg).src = 'mini/' + source + '.png';*  
  
Nous incrémentons donc la variable des clics et affichons la miniature en fonction de son nom passé en second paramètre, dans la **balise Html Img** dont l'identifiant est passé en premier paramètre.  
  
Il s'agit maintenant de poursuivre le traitement. Mais selon qu'il s'agit du premier clic ou du second, l'issue est différente. Dans le premier cas, il faut simplement mémoriser le nom de l'image et le nom de son contrôle. Dans le second cas, dans la mesure où le contrôle cliqué n'est pas le même, il faut comparer les images pour savoir si elles sont identiques. Si elles le sont, l'association doit être validée et doit rester affichée. Le cas échéant, les deux miniatures doivent disparaitre après un petit délai. De même, les variables dont nb\_clics, doivent être réinitialisées.

* A la suite du code, ajouter les instructions Javascript suivantes :

*if(nb\_clics == 1)  
{  
mini1=source;  
case1 = limg;  
}  
else  
{  
mini2=source;  
case2 = limg;  
}*  
  
Nous écrivons ce que nous avons traduit juste au-dessus. La **branche Else** de l'instruction conditionnelle est nécessairement plus complexe. Comme nous l'avons évoqué, avant de comparer les images dont les noms sont désormais mémorisés, nous devons nous assurer que l'internaute n'a pas cliqué deux fois de suite sur la même case. Ces noms de contrôles sont stockés dans les **variables case1 et case2**. Il s'agit simplement de les comparer à titre de vérification.

* A la suite du code dans la **branche Else**, ajouter le nouveau test suivant :

*if(case1!=case2)  
{  
  
}  
else  
{  
if(nb\_clics==2) nb\_clics=1;  
}*  
  
Dans le cas où les cases cliquées sont bien différentes, nous devons exécuter un code pour comparer les images correspondantes. Dans le cas contraire en revanche, nous réinitialisons la **variable nb\_clics** pour que l'internaute puisse désigner une autre vignette. Au prochain appel de la fonction, la **variable nb\_clics** vaudra donc 2. Si les cases cliquées sont désormais différentes, la comparaison doit débuter. C'est le **code Javascript** que nous devons désormais développer.  
  
Lorsque les contrôles cliqués sont effectivement différents, deux nouveaux cas sont à envisager. Premièrement, les deux images ne correspondent pas. Dans ces conditions, elles doivent être masquées après un délai, le score doit être décrémenté et les clics réinitialisés.

* Dans la branche If de l'instruction conditionnelle, ajouter les traitements suivants :

*depart=false;  
if(mini1 != mini2)  
{  
var attente = setTimeout(function()  
{  
document.getElementById(case1).src = 'mini/miniz.png';  
document.getElementById(case2).src = 'mini/miniz.png';  
depart=true;  
nb\_clics=0;  
nb\_erreurs ++;  
if(nb\_erreurs<11) le\_score = 10 - nb\_erreurs;  
document.getElementById('scrore').innerHTML ='<strong>' + le\_score + '</strong>/10';  
},1000);  
}  
else  
{*  
  
Nous commençons par réinitialiser la **variable booléenne** afin que les clics soient stoppés temporairement. Puis, nous créons la structure de l'**instruction conditionnelle**. Son critère consiste à vérifier que les deux images sont différentes. Si la condition est validée, les images doivent être masquées, moyennant un délai, défini sur une seconde cette fois. Nous exploitons comme précédemment la **fonction Javascript setTimeout**. Dans ses bornes, nous commençons par remplacer les deux miniatures incriminées par l'image vierge, afin de les masquer à nouveau, attestant que la combinaison est incorrecte. Nous autorisons la partie à reprendre en réaffectant la **variable booléenne** à **True**. Son affectation est bien sûr effective une fois la temporisation terminée. Nous réinitialisons la **variable nb\_clics** afin que le décompte reparte depuis le début. Nous incrémentons la **variable publique nb\_erreurs**, qui cumule toutes les fautes commises. Nous la retranchons aux dix unités de départ pour afficher le score en cours dans le **contrôle score** prévu à cet effet. Comme il s'agit d'un calque, notez l'emploi de sa **propriété innerHTML** afin de pouvoir affecter son contenu avec mise en forme, par les **balises Html**.  
  
Dans le cas contraire (else), nous en déduisons que les deux images correspondent. Et comme nous sommes dans l'instruction conditionnelle ayant vérifié que les contrôles cliqués sont différents, nous devons valider cette association.

* Dans la **branche Else** de l'instruction conditionnelle, ajouter les lignes Javascript suivantes :

*depart=true;  
nb\_clics=0;  
img\_ok += 2;  
if(img\_ok==16)  
{  
var dif\_temps = Math.floor((new Date().getTime() - temps\_debut)/1000);  
document.getElementById('scrore').innerHTML = '<strong>' + le\_score + '</strong>/10';  
document.getElementById('temps').innerHTML = 'Vous avez mis <strong>' + dif\_temps + '</strong> secondes';  
}*  
  
Nous commençons par débloquer la partie en réaffectant la **variable booléenne**. Aucune temporisation n'est nécessaire ici. La combinaison étant validée, la partie peut se poursuivre. Comme précédemment, pour reprendre le fil du jeu, nous réinitialisons également la variable comptant les clics. Puis, nous incrémentons de deux unités la **variable img\_ok**. C'est elle qui indiquera que la partie est finie lorsqu'elle aura atteint les 16 unités, correspondant aux 16 images à retrouver. D'ailleurs l'instruction conditionnelle qui suit vérifie ce critère. Si la partie est terminée, nous prélevons de nouveau le temps à l'instant T, grâce à la **méthode getTime** de la **classe Javascript Date**. Nous lui retranchons les millisecondes de l'heure prélevée en début de partie. Nous divisons le résultat par mille afin de le convertir en secondes. La **méthode floor** de l'**objet Math** arrondit cette valeur à l'entier directement inférieur. Puis, en même temps que nous affichons le score dans son calque, nous inscrivons le temps de jeu dans le **calque d'identifiant temps**.  
  
Il ne nous reste plus qu'à ajouter le contrôle sur le temps. Le délai imparti est de 3 minutes, soit 180 secondes.

* Toujours dans la même branche else, après l'instruction If de la variable img\_ok, ajouter le test suivant :

*if(dif\_temps > 180)  
{  
document.getElementById('temps').innerHTML = 'Le temps imparti est dépassé, vous avez perdu !';  
depart=false;  
}*  
  
Nous exploitons la **variable dif\_temps** dont le résultat a été converti en secondes. Si le délai est dépassé, nous inscrivons un message dédié dans le **calque d'identifiant temps** prévu à cet effet. Et bien sûr, nous basculons la **variable booléenne sur false** afin de bloquer la poursuite de la partie.

* Enregistrer les modifications et basculer sur le navigateur Web,
* Rafraîchir la page Internet à l'aide de la touche F5 du clavier,
* Puis, débuter la partie en cliquant sur les cases afin de retrouver les paires,

Comme vous le remarquez, lorsque deux images consécutives sont identiques, elles sont conservées et le score reste intact. Si vous cliquez deux fois de suite sur la même case, la **variable nb\_clics** étant décrémentée, le **code Javascript** attend un nouveau choix pour valider l'association. Si les deux images ne correspondent pas, elles disparaissent en effet au bout d'une seconde. Le score est mis à jour en temps réel sur la droite du plateau de jeu.   
  
 

A l'issue de la partie, une fois toutes les combinaisons trouvées, la durée de jeu s'affiche en dessous du score, comme l'illustre la capture ci-dessous.   
  
   
  
Lorsque le délai est dépassé, le message que nous avons codé en informe l'internaute, comme l'illustre la capture ci-dessous.   
  
   
  
Cependant à ce stade, l'application laisse l'occasion au joueur de terminer la partie. Pour que cette dernière soit stoppée, le **calcul sur la différence de temps** doit être réalisé quelle que soit la condition traitée dans la **fonction verifier**. Le test sur la **variable dif\_temps** doit lui emboîter le pas dans une partie commune de la fonction. C'est une petite adaptation que vous aurez tout le loisir de réaliser si vous le souhaitez. Car en attendant, nous avons pu vérifier que notre création était tout à fait fonctionnelle. Notre **jeu du Memory** permet bien de reconstruire les paires d'images en contrôlant les clics de l'internaute par le **code Javascript**.