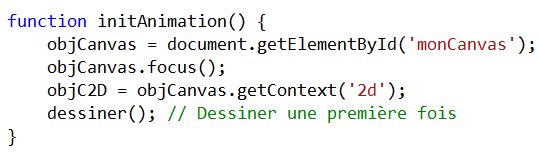
|  |
| --- |
| Laboratoire 4B: L’interactivité |

Théorie

1. L’interactivité : les événements du clavier

Ouvrez la page Web **Théorie 4B-1.htm**.

Cliquez dans le canevas. Pour faire tourner l’objet vers la droite, appuyez sur la touche « **flèche-à-droite** ». Pour faire tourner l’objet vers la gauche, appuyez sur la touche « **flèche-à-gauche** ». Pour arrêter l’animation, appuyez sur la touche « **End** ».

Observez la manière que cela a été programmé.

Tout d’abord, au chargement de la page Web, nous dessinons l’objet une première fois.

Présentement, il n’y a pas d’animation. L’animation va se réaliser lorsque l’utilisateur va relâcher une des touches du clavier (événement ***onkeyup***).



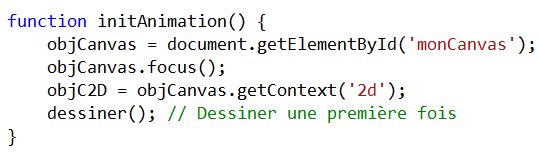
Ici, nous utilisons l’événement ***onkeyup*** mais il existe deux autres événements qui concernent le clavier.

***onkeydown***  : Lorsque l’utilisateur appuie sur une des touches du clavier. Si l’utilisateur maintient la touche appuyée, l’événement ***onkeydown*** est appelé à plusieurs reprises.

***onkeypress***  : Lorsque l’utilisateur appuie sur une des touches du clavier. Si l’utilisateur maintient la touche appuyée, l’événement ***onkeydown*** est appelé à plusieurs reprises.

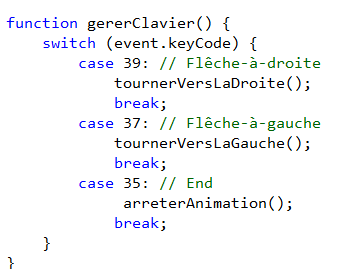
La différence entre les deux (2) événements c’est que ***onkeypress*** ne réagit qu’aux touches qui ont un code de caractère tandis que ***onkeydown*** réagit seulement aux touches qui ont un code de touche.





Observez l’attribut ***tabIndex="1"***. La raison c’est que, par défaut, en **HTML**, les canevas ne peuvent pas recevoir le focus. Pour donner la possibilité à un canevas de recevoir le focus, il faut lui assigner un « **tabIndex** ».

Une des manières de donner le focus à un élément Web c’est de cliquer sur cet élément mais une autre manière de le faire c’est de lui donner le focus de manière automatique (par programmation).

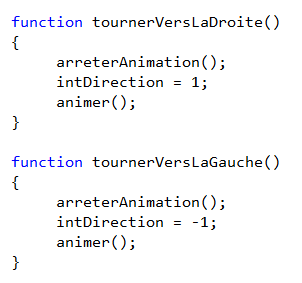
Concernant l’événement ***onkeyup,*** dans le cas qui nous concerne, dès que l’utilisateur relâche une touche du clavier, la fonction **gererClavier** est appelée.

L’objet ***event*** est un objet du système. Cet objet donne des informations sur l’événement qui vient de se produire. Une de ces informations est le code de la touche (.***keyCode***).

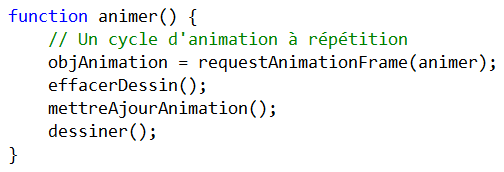
Ici, lorsque l’utilisateur relâche la touche « **flèche-à-droite** » (*keyCode=39*), nous appelons la fonction **tournerVersLaDroite**, lorsque l’utilisateur relâche la touche « **flèche-à-gauche** » (*keyCode=37*), nous appelons la fonction **tournerVersLaGauche** et, lorsque l’utilisateur relâche la touche « **End** » (*keyCode=35*), nous appelons la fonction **arreterAnimation**.

Pour connaître un code de touche particulier et/ou un code de caractère particulier, allez sur la page Web :

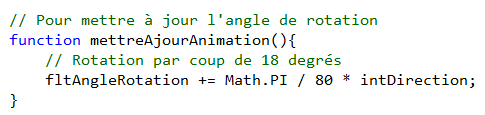
<http://www.west-wind.com/WestwindWebToolkit/samples/Ajax/html5andCss3/keycodechecker.aspx>

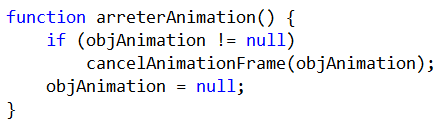
Pour touner l’objet vers la droite ou vers la gauche, nous arrêtons l’animation, nous modifions la direction (1 signifie vers la droite et -1 signifie vers la gauche) puis nous redémarrons l’animation.

La fonction **animer** est simple. Il suffit de créer un cycle d’animation 60 fois par seconde à l’aide d’une requête.



Observez que nous avons placé cette requête d’animation dans la variable **objAnimation**. Cette variable est importante. Elle contient tout ce qui concerne la requête. Elle va être utilisée plus tard pour arrêter l’animation.

Observez également, lorsque nous mettons à jour l’animation, nous multiplions l’angle par la direction. Ainsi, si la direction vaut -1, nous allons soustraire l’angle au lieu de l’additionner. C’est cela qui va donner l’illusion que l’objet tourne vers la gauche.

Pour arrêter l’animation, nous appelons la fonction ***cancelAnimationFrame***. Cette fonction annule la requête d’animation. Elle prend en paramètre la requête en tant que tel.

Maintenant, la question à 100.00$. Pourquoi, lorsqu’on tourne vers la droite ou vers la gauche, faut-il arrêter l’animation avant de changer de direction?

La raison, c’est que la requête d’animation est un objet autonome. Si cette requête n’est pas annulée, elle va poursuivre son exécution et cette requête va créer une nouvelle requête d’animation. Cela va avoir comme conséquence que plusieurs requêtes d’animation vont s’exécuter dans le même laps de temps.

Pour tester cela, mettez en commentaires l’appel à la fonction **arreterAnimation** dans la fonction **tournerVersLaDroite.** Exécutez l’application puis appuyez plusieurs fois sur la flèche-à-droite. L’effet est assez amusant. Vous allez comprendre ce qu’est un effet de superposition.

2. L’interactivité : les événements de la souris

Voici la description de tous les événements de la souris.

***onclick*** : Lorsque l’utilisateur clique sur l’élément.

***ondblclick*** : Lorsque l’utilisateur double-clique sur l’élément.

***onmousedown*** : Lorsque l’utilisateur appuie sur un des boutons de la souris. Si l’utilisateur maintient le bouton appuyé, cet événement est appelé à plusieurs reprises.

***onmouseup*** : Lorsque l’utilisateur relâche un des boutons de la souris.

***onmouseover***: Lorsque la souris pénètre à l’intérieur des frontières d’un élément.

***onmouseout***: Lorsque la souris quitte les frontières d’un élément.

***onmousemove***: Lorsque la souris se déplace à l’intérieur des frontières d’un élément. Cet événement est appelé régulièrement tout au long du déplacement.

***onmousewheel***: Lorsque l’utilisateur tourne la roulette de la souris.

Ouvrez la page Web **Théorie 4B-2.htm**.

Survolez le canevas à l’aide de la souris. Observez que l’objet s’anime.

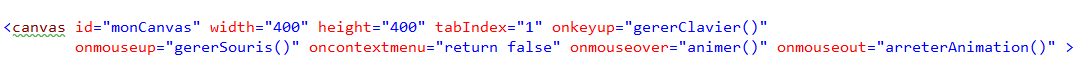
Survoler l’extérieur du canevas à l’aide de la souris. Observez que l’objet ne s’anime plus.

Cliquez sur le canevas à l’aide du bouton gauche. Observez que l’objet tourne vers la gauche.

Cliquez sur le canevas à l’aide du bouton droit. Observez que l’objet tourne vers la droite.

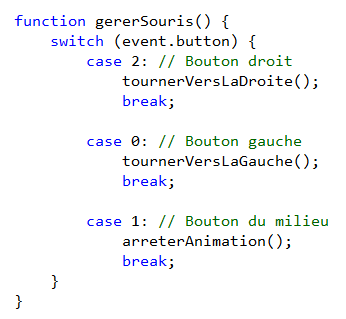
Cliquez sur le canevas à l’aide du bouton du milieu. Observez que l’animation s’arrête.

Observez les événements souris qui ont été programmés ; il y en a trois : ***onmouseup***, ***onmouseover*** et ***onmouseout***.



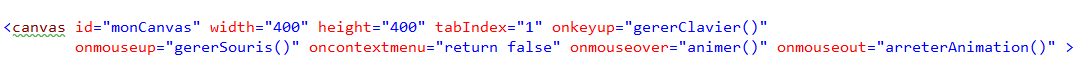
Plus tard, dans ce laboratoire, nous allons programmer des opérations de type « glisser-déposer**»**. Pour l’instant, concentrons-nous sur les événements souris qui ont été programmés.

Ici, lorsque l’utilisateur relâche un des boutons de la souris, la fonction **gererSouris** est appelée.

L’objet ***event*** est un objet du système. Cet objet donne des informations sur l’événement qui vient de se produire. Une de ces informations est le bouton de la souris que l’utilisateur a relâché (***button***). La valeur 0 signifie le bouton gauche, la valeur 1 signifie le bouton du milieu et la valeur 2 signifie le bouton droit.

La programmation des fonctions **tournerVersLaDroite**, **tournerVersLaGauche** et **arreterAnimation** ont déjà été vus dans la section précédente.

Observez que, lorque la souris franchit les frontières du canevas, la fonction **animer** est appelée et, lorsque la souris quitte les frontières du canevas, la fonction **arreterAnimation** est appelée. Ces fonctions ont également été vus dans la section précédente.



Observez également l’événement ***oncontextmenu***. Cet événement se produit lors de l’affichage du menu contextuel. Ici, nous retournons **false**. Cela annule l’affichage du menu contextuel. C’est important de le faire car, si on ne le fait pas, le navigateur va afficher le menu contextuel lorsque l’utilisateur va relâcher le bouton droit de la souris.

3. L’interactivité : le glisser-déposer

Contrairement aux autres événements, les événements du glisser-déposer n’existaient pas en **HTML4**. Les événements du glisser-déposer sont nouveaux en **HTML5**.

Voici la description de tous les événements du glisser-déposer :

***ondragstart***: Lorsqu’une opération de type « glisser-déposer » s’amorce.

***ondragend***: Lorsqu’une opération de type « glisser-déposer » se termine.

***ondrag***: Appelé régulièrement tout au long de l’opération de type « glisser-déposer ».

***ondragenter***: Lorsque la souris pénètre les frontières d’un élément dans lequel l’objet glissé peut être déposé.

***ondragleave***: Lorsque la souris quitte les frontières d’un élément dans lequel l’objet glissé peut être déposé.

***ondragover***: Lorsque la souris se déplace à l’intérieur des frontières d’un élément dans lequel l’objet glissé peut être déposé. Cet événement est appelé régulièrement tout au long du glisser.

***ondrop***: Lorsque l’objet glissé est relâché.

Ouvrez la page Web **Théorie 4B-3.htm** puis, à l’aide de la souris, glissez-déposez le logo du Cégep dans le conteneur du bas.

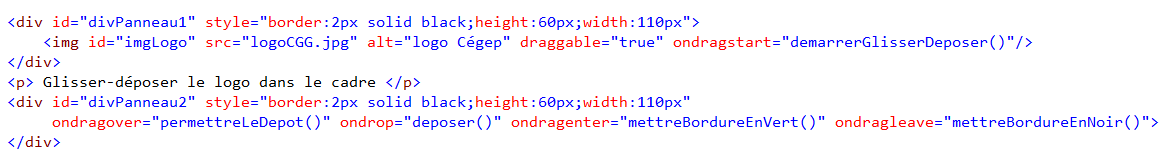
Glisser-déposer

Lorsqu’on programme le glisser-déposer, il faut toujours se poser deux questions :

* Quelle est la source (c’est-à-dire quel élément veut-on déposer)?
* Quelle est la cible (c’est-à-dire où veut-on déposer cet élément)?

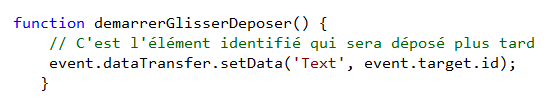
Habituellement, les événements suivants doivent être programmés dans l’élément source : ***ondragstart*** et ***ondragend***. De plus l’élément source doit être « glissable » (attribut ***draggable="true"***); tous les autres événements du glisser-déposer doivent être programmés dans l’élément cible. Par contre, ce n’est pas une règle stricte.

Ici, l’image identifié **imgLogo** est l’élément source et le panneau identifié **divPanneau2** est l’élément cible. Observez que, présentement, l’élément source est situé à l’intérieur du panneau identifié **divPanneau1**.



Le premier événement que l’on doit normalement programmer est l’événement ***ondragstart***.

L’événement ***ondragstart*** est appelé lorsque l’utilisateur appuie sur le bouton gauche de la souris lorsque la souris est située au-dessus de l’élément. Cela amorce le processus du glisser-déposer.

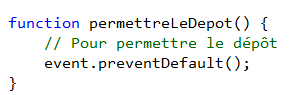
Ici, on copie, dans le presse-papiers de l’événement (***event.dataTransfer***), l’id de l’élément (***even.target.id***) qu’on va déposer plus tard. Par conséquent, dans le cas qui nous concerne, ce qu’il va y avoir dans ce presse-papiers, c’est la chaîne de caractères **'imgLogo'**. Nous allons utiliser cette information plus tard.



A partir de ce moment, une image fantôme de l’élément est automatiquement créée par le système. C’est cette image fantôme qui indique à l’utilisateur ce qui sera déposée plus tard. Ici, l’image fantôme créée par le système est l’image du logo.

Le second événement que l’on doit programmer est l’événement ***ondragover*.** Cet événement représente « le glisser ».

L’événement ***ondragover*** est appelé lorsque l’utilisateur déplace la souris au-dessus de l’élément tout en maintenant le bouton gauche appuyé.

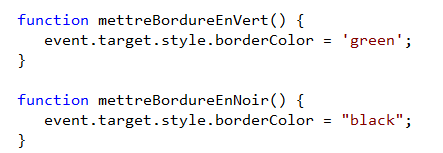
Ici, lorsque l’image fantôme glisse au-dessus du deuxième panneau, l’instruction ***event.preventDefault()*** signifie de ne pas adopter le comportement par défaut. Par défaut, on ne peut rien déposer. Par conséquent, cette instruction fait en sorte que le dépôt est permis. Il est important de programmer cela. Si vous ne le faites pas, vous ne pourrez pas déposer le logo du Cégep.



Pour illustrer cela, mettez en commentaire cette instruction, survolez le deuxième panneau et observez que l’icône de la souris revêt la forme suivante. Cela signifie que vous ne pouvez pas déposer.

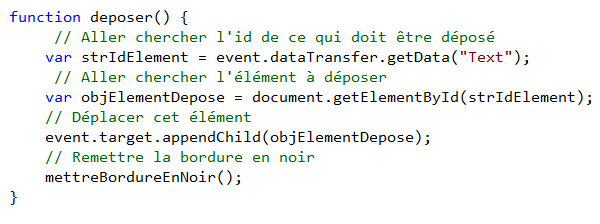
Enlevez le commentaire. Survolez le deuxième panneau et observez que vous pouvez déposer.

Les autres événements qu’il est possible de programmer (mais ce n’est pas obligatoire), ce sont les événements ***ondragenter*** et ***ondragleave***.

L’événement ***ondragenter*** est appelé lorsque la souris franchit les frontières de l’élément et ***ondragleave*** est appelé lorsque la souris quitte les frontières de l’élément.

Ici, lorsque la souris franchit les frontières du deuxième panneau (***event.target***), sa bordure devient verte et lorsque la souris quitte les frontières du deuxième panneau, sa bordure redevient noire.

Il ne nous reste qu’à programmer l’événement ***ondrop*.** Cet événement représente « le déposer ».

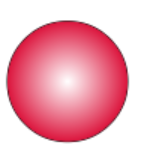
L’événement ***ondrop*** est appelé lorsque l’utilisateur relâche le bouton gauche de la souris lorsque la souris est au-dessus de l’élément.

Cet événement est l’événement le plus important à programmer. Ce que l’on doit programmer dans cet événement, c’est ce qui doit se passer lorsque l’utilisateur dépose l’élément.

Ici, la première chose que l’on fait, c’est d’aller chercher l’id de l’élément qui doit être déposé. Dans le cas qui nous concerne, cet id devrait être la chaîne de caractères **'imgLogo'.**

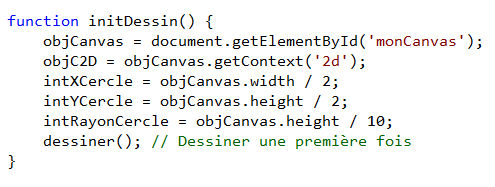
A partir de cet id, on va chercher l’élément à déposer (cet élément est l’image) puis, on dépose l’élément dans le deuxième panneau (***event.target***). La méthode **.*appendChild*** fait en sorte que l’image se déplace à l’intérieur du deuxième panneau.

Finalement, on remet la bordure du deuxième panneau en noir (car, au moment du dépôt, elle est verte).

4. L’interactivité : le glisser-déposer – prise 2

Ouvrez la page Web **Théorie 4B-4.htm** puis, à l’aide de la souris, glissez-déposez le cercle n’importe où à l’intérieur du canevas.

Observez la manière que cela a été programmé.



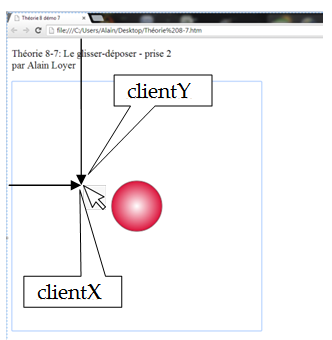
Tout d’abord, au chargement de la page Web, nous dessinons le cercle une première fois.

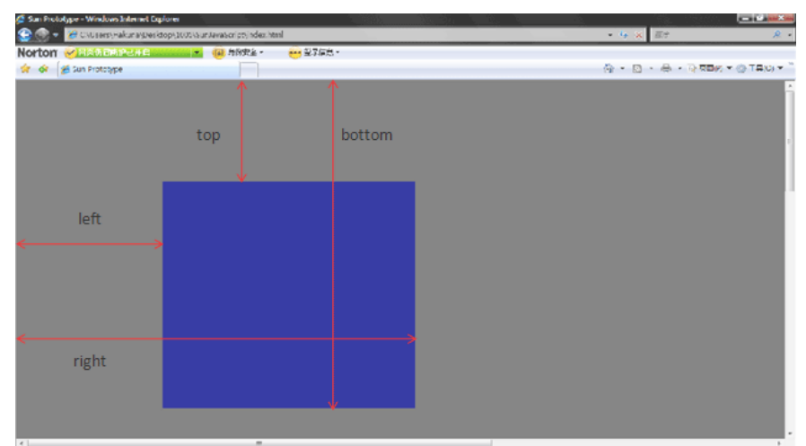
(**intXCercle, intYCercle**) représente la position du cercle (au point de départ, il est situé au centre du canevas) et **intRayonCercle** représente le rayon du cercle (1/10 de la hauteur du canevas).

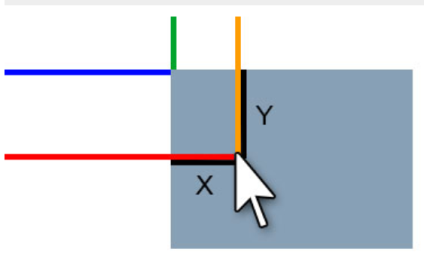
Dans cette application, nous allons déplacer ce cercle à l’intérieur du canevas en utilisant le glisser-déposer.

Ici, nous avons programmé trois événements reliés au glisser-déposer : **ondragstart**, **ondragover** et **ondrop.**

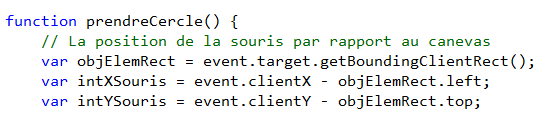


Au démarrage du glisser-déposer (***ondragstart***), il faut que l’utilisateur prenne le cercle à l’aide de la souris mais, pour savoir si la souris est située à l’intérieur du cercle, il faut connaître la position de la souris.

Il est possible de connaître la position de la souris en utilisant les propriétés (.***clientX, .clientY***) de l’événement. Malheureusement, cette position est relative au coin supérieur gauche de l’intérieur de la fenêtre. Cette position n’est pas relative au coin supérieur gauche du canevas.

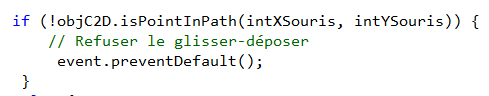
Par contre (et heureusement), il existe une méthode que l’on peut appliquer sur n’importe quel élément conteneur pour connaître les coordonnées de ce conteneur : la méthode *.****getBoundingClientRect***. Cette méthode retourne un objet qui contient la position gauche du conteneur (.***left***), sa position droite (.***right***), sa position supérieure (.***top***) et sa position inférieure (.***bottom***) à l’intérieur de la fenêtre.

Pour connaître la position de la souris par rapport au coin supérieur gauche du canevas, il suffit de soustraire la position gauche du canevas (.***left***) de la position en X de la souris (.***clientX***) ainsi que de soustraire la position supérieure du canevas (.***top***) de la position en Y de la souris (.***clientY***).



C’est ce qui est réalisé dans le code que vous voyez ci-contre.

Maintenant que l’on connaît la position de la souris, il faut vérifier si la souris est à l’intérieur du cercle ou non.

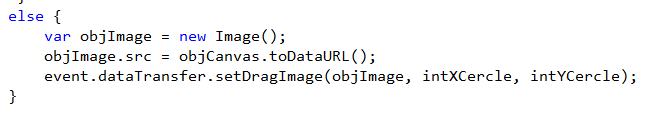
Pour ce faire, nous utilisons la méthode ***.isPointInPath(intX, intY)***. Cette méthode retourne **true** si ***(intX, intY)*** est située à l’intérieur du dernier tracé sinon elle retourne **false**. Ici, il n’y a qu’un seul tracé (le cercle). Donc, nous pouvons utiliser cette méthode.

Dans le cas qui nous concerne, si la position de la souris est située à l’extérieur du cercle, nous refusons le glisser-déposer (par défaut, dans l’événement ***ondragstart***, le glisser-déposer est accepté).

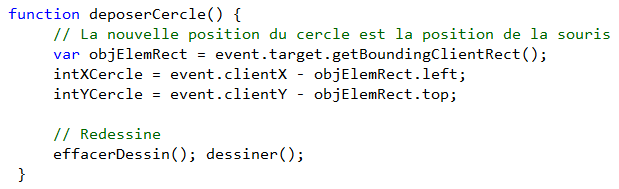
Il est important de prendre note que, lorsque le glisser-déposer est refusé, la souris prend sa forme normale et aucun autre événement qui concerne le glisser-déposer est appelé (même s’il a été programmé).

Par contre, si la souris est située à l’intérieur du cercle, nous devons créer une image fantôme sur le canevas. **Internet Explorer** crée une image fantôme de manière automatique sur un canevas mais **Chrome** ne le fait pas.

Le problème avec les images fantômes, c’est que cela requiert une catégorie d’images bien particulière. Les images acceptées sont les images qui sont situées à l’intérieur d’une balise ***<img />*** ou les images qui ont une adresse URL.

Heureusement il est possible d’aller chercher l’adresse URL d’un canevas en utilisant la méthode *.****toDataURL***.

Observez la méthode ***.setDragImage***. C’est grâce à cette méthode qu’il est possible de créer notre propre image fantôme. Cette méthode prend l’image en paramètre ainsi que la position (x, y) à l’intérieur de l’image sur laquelle la pointe de la flèche de la souris va se situer.



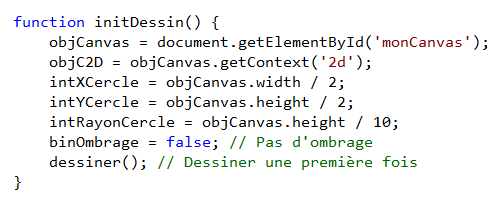
Il ne nous reste plus qu’à déposer le cercle à l’intérieur du canevas (dans l’événement ***ondrop***). Ici, nous allons chercher la position de la souris à l’intérieur du canevas. C’est à cette position que va se dessiner le cercle déposé par l’utilisateur.

5. L’interactivité : le glisser-déposer – prise 3



Ouvrez la page Web **Théorie 4B-5.htm** puis, à l’aide de la souris, glissez-déposez le cercle n’importe où à l’intérieur du canevas.

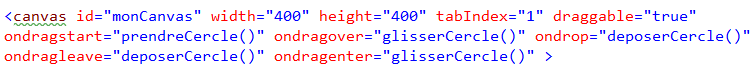
Observez que le glisser affiche un ombrage et qu’il n’y a pas d’image fantôme. Dès que le cercle est déposé, l’ombrage disparaît.

Observez la manière que cela a été programmé.

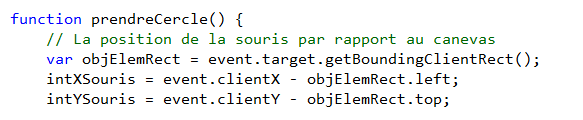
Tout d’abord, au chargement de la page Web, nous dessinons le cercle une première fois sans ombrage (***binOmbrage= false***).

Le reste de cette fonction est identique à celle qui a été programmée précédemment.

Ici, nous avons programmé cinq événements reliés au glisser-déposer : ***ondragstart***, ***ondragover*, *ondrop*, *ondragleave*** et ***ondragenter*.**



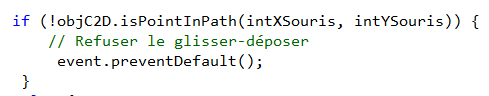
Au démarrage du glisser-déposer, nous appelons la fonction ***prendreCercle***.

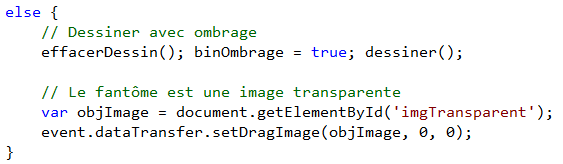


Pour ce faire, nous allons chercher la position de la souris par rapport au canevas.

La grande différence avec le programme précédent, c’est que la position de la souris a été déclarée globalement. Nous allons utiliser cette position plus tard.



Comme dans le programme précédent, si la souris est située à l’extérieur du cercle, nous refusons le glisser-déposer.

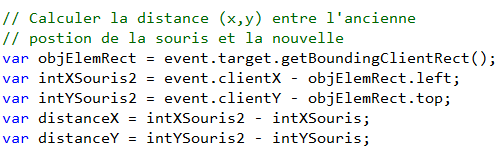
Par contre, si ce n’est pas le cas, nous redessinons le cercle avec ombrage. A cause de l’ombrage, cela va donner l’impression que le cercle a été pris par l’utilisateur.

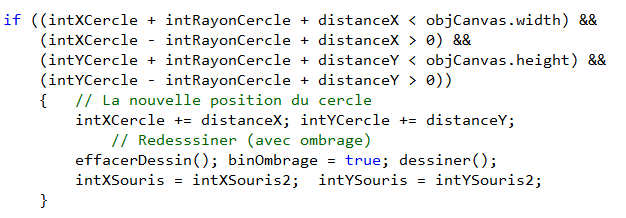
De plus, nous créons notre propre image fantôme.

En fait, l’image fantôme est une image **gif** qui ne contient qu’un seul pixel transparent. Donc le fantôme est complètement transparent. C’est comme s’il n’y avait aucune image fantôme.

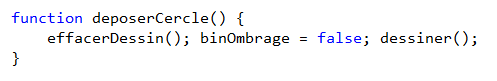


La programmation du glisser est très différent du programme précédent. Ce que nous voulons, c’est déplacer le cercle au fur et à mesure que la souris se déplace et non pas attendre à la fin du glisser-déposer comme c’est le cas dans un glisser-déposer traditionnel.

L’événement ***ondragover*** (qui s’occupe du glisser) est appelé à répétition. Voici la manière que nous avons programmé ce glisser. Tout d’abord, nous allons chercher la position actuelle de la souris (**intXSouris2, intXSouris2**) puis nous calculons la distance (**distanceX, distanceY**) qu’a parcouru la souris lorsque le glisser-déposer a démarré.

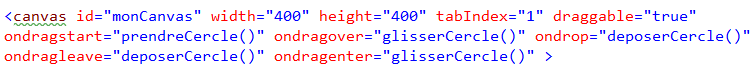
Puis nous additions cette distance à la position du cercle (**intXCercle, intYCercle**). Par conséquent, le cercle va se déplacer de la même distance que la souris s’est déplacée. En bref, le cercle va suivre le déplacement de la souris. Observez que nous ne déplaçons pas le cercle si celui-ci dépasse les frontières du canevas. Nous faisons cela pour que le cercle soit toujours visible à l’intérieur du canevas.

Observez également que nous conservons la nouvelle position de la souris : ***intXSouris = intXSouris2*** et ***intYSouris = intYSouris2.*** N’oubliez pas que l’événement ***ondragover*** est appelé à répétition. Le fait de conserver la position de la souris va servir de point de départ pour le prochain appel à l’événement ***ondragover***.



La programmation de l’événement ***ondrop*** (le déposer) est simple. On redessine le cercle mais cette fois sans ombrage.

Observez, lorsque la souris quitte les frontières du canevas (***ondragleave***), que nous déposons le cercle. De cette manière, si l’utilisateur dépose le cercle à l’extérieur du canevas (en relâchant le bouton gauche de la souris), le cercle va déjà être déposé.



Par contre, si l’utilisateur refranchit la frontière du canevas sans relâcher le bouton gauche de la souris (événement ***ondragenter***), le cercle va continuer de se déplacer comme si de rien ne s’était passé. Vous pouvez tester.

6. Références

<http://www.w3schools.com>

<http://chimera.labs.oreilly.com/books/1234000001654/ch05.html#ball_bounce>

<http://stackoverflow.com/questions/24354115/finding-top-and-left-values-of-div-without-setting-stylesheet-values>

<http://dev.opera.com/articles/better-performance-with-requestanimationframe/>

<https://developer.mozilla.org/fr/docs/Glisser_et_d%C3%A9poser/Op%C3%A9rations_de_glissement>

<http://www.cambiaresearch.com/articles/15/javascript-char-codes-key-codes>

<http://www.script-home.com/javascript-getboundingclientrect-to-obtain-the-position-of-the-elements-on-the-page-code-revised-pp-12.html>