Leçon 7.1 – HTML5 : dessin, interactivité

s1 ----------

Dans cette partie, nous allons voir comment réaliser une animation interactive simple avec HTML5. Nous allons pour ça reprendre le concept d'événement JavaScript et l'adapter au contexte du canvas en prenant deux exemples : le cas d'une interaction clavier et le cas d'un clic souris.

s2 ----------

Comment associer animation et interaction ? Prenons un exemple.

Une des utilisations les plus courantes du clavier consiste à utiliser les touches de flèches pour diriger un objet en mouvement. L'exemple qui suit reprend une animation précédente consistant à déplacer un rectangle du haut vers le bas et à l'adapter pour prendre en compte une interaction clavier.

Remémorons-nous le code de l'animation. On identifie aisément

- la partie de code définissant le canvas,
- la partie de code permettant de lancer une fonction de dessin à intervalles réguliers,
- la partie de code correspondant à cette fonction de dessin au sein de laquelle on identifie la méthode translate qui à chaque itération déplace le contexte d'une quantité 0 suivant x et 2 suivant y. Soit, pour l'instant donc, une translation sur un axe vertical et de haut en bas.

s3 ----------

C'est précisément sur ce déplacement suivant x que l'on va jouer. On remplace ce zéro par une variable que l'on appelle X que l'on initialise à zéro tout au début du code.

Ensuite on installe deux gestionnaires : un pour l'événement keydown, un autre pour l'événement keyup.

L’événement keydown permet d'affecter à X une valeur 4 ou -4 selon que l'on appuie sur la touche droite ou gauche, l'événement keyup permet de réinitialiser X pour que la chute se fasse verticalement.

Je vous propose de tester ce code.

s4 ----------

Dans l'exemple précédent, une interactivité clavier se traduisait par un changement de paramètre dans les méthodes de dessin associées au canvas. Voyons sur un autre exemple comment introduire une interactivité souris sur le canvas lui-même.

Le canvas étant un objet de la page HTML, on peut lui associer de la même façon que précédemment un gestionnaire.

Mais supposons que le lieu du canvas où le clic se produit ait une importance. Supposons, si l'on reprend l'exercice des cercles qui se déplacent de haut en bas, que l'on souhaite faire disparaître le cercle sur lequel on clique.

Rappelons que le canvas est une image constituée de pixels. L'objet cercle n'existe pas en tant que tel. En d'autres termes, on ne peut pas mettre en oeuvre un gestionnaire sur un cercle, mais seulement sur le canvas tout entier.

La solution est la suivante.

On se rappelle que le contenu du canvas est redessiné à intervalles réguliers

On installe un gestionnaire sur le canvas.

- Si un clic bas sur produit sur le canvas, on détermine la position de ce clic / canvas ;
- on teste si on clique à l'endroit où un cercle est dessiné (pour que ceci soit possible, il faut à chaque instant disposer d'un tableau des coordonnées du cercle) ;
- si c'est le cas alors lors de l'itération suivante, on va redessiner le canvas sans le cercle en question.

La difficulté est maintenant de déterminer la position du clic / canvas.

s5 ----------

Voyons comment mettre tout ça en oeuvre.

Premièrement : installer un gestionnaire de clic sur le canvas. Nous avons déjà fait plusieurs fois. Ici c'est la syntaxe classique. Si vous préférez utiliser jQuery, vous trouverez facilement la syntaxe événementielle associée sur le web.

2 : Initialiser un tableau avec les coordonnées des cercles. Pour chaque cercle, on définit ainsi les positions selon x et selon selon y de son centre et son rayon ; la signification de la quatrième valeur sera donnée tout à l'heure.

3 : Changer les arguments de dessineBalle. Deux arguments : l'indice du cercle dans le tableau (0

pour le premier cercle, 1 pour le deuxième, etc.) et la vitesse de la balle.

4 : Modifier dessineBalle pour prendre en compte ce changement ; en profiter pour rajouter une ligne qui modifie à chaque itération la coordonnée suivant y du cercle que l'on dessine.

À ce stade, si l'on teste, on a un fonctionnement identique à l'animation précédente. Il nous reste à exploiter le clic et à donc définir la fonction clicCanvas donnée plus haut comme argument du gestionnaire de clic sur la canvas.

Cinquième étape : calculer la position de la souris dans le document. Les coordonnées de la souris par rapport à la page HTML sont les attributs pageX et pageY de l'événement qui vient de se produire. Facile.

6 : Trouver les coordonnées du canvas dans la page est ici simple : les attributs offsetLeft et offsetTop d'un élément sont ses positions selon x et y par rapport à son conteneur. Ici le conteneur du canvas est la page.

Une fois ces coordonnées déterminées il est facile de déterminer la position du clic / canvas.

7 : En bouclant sur le tableau des cercles, on teste si l'un des cercles est concerné. Si c'est le cas, on met le dernier élément du tableau à 0.

À quoi nous sert ce quatrième élément du tableau des cercles ?

C'est l'étape n° 8 : il suffit au moment de dessiner un cercle de vérifier que cet élément est à 1 !

Je vous propose de tester cela.

s6 ----------

L'exercice que nous proposons ensuite consiste à reprendre le code écrit dans le chapitre précédent pour déplacer un ensemble de balles avec des caractéristiques aléatoires (taille, vitesse, position) et d'y rajouter la possibilité de déplacer un mobile en translation de gauche à droite tout en bas du canvas.

Le joueur devra faire en sorte d'éviter ces balles sinon, le jeu s'arrête. Un test de collision devra être effectué à chaque itération.

Attention ici : on se rappelle qu'après chaque dessin, le contexte est restauré à sa position initiale.

Les déplacements du mobile devront donc être effectués par rapport à cette position initiale.

Bon travail !

s7 ----------

Il nous reste une dernière fonctionnalité à introduire dans ce module : le glisser-déposer. C'est le thème de la prochaine leçon.

À tout de suite.