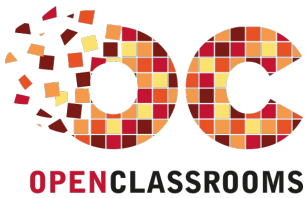
Régler les problèmes de collisions

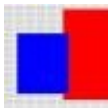
Par isagaw



www.openclassrooms.com

Sommaire

Sommaire 2
Régler les problèmes de collisions 3
Problème avec hitTest 3
Première solution - Retour en arrière 4
Deuxième solution - Repositionnement après collision 5
Troisième solution - Une boucle 5
Correction 6
Partager 7



Régler les problèmes de collisions

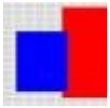
Par isagaw
Mise à jour : 01/01/1970

Bonjour amis Zéros.

Dans ce tuto, je vais vous expliquer en détail comment régler certains problèmes dus aux collisions en Flash.

Je ne vous expliquerai pas comment créer des collisions, vous savez sûrement déjà faire , et si vous ne savez pas, il y a déjà un très bon tuto qui vous l'explique.

Sommaire du tutoriel :



- Problème avec hitTest
- Première solution - Retour en arrière
- Deuxième solution - Repositionnement après collision
- Troisième solution - Une boucle

Problème avec hitTest

Nous allons commencer avec un petit exemple tout simple.
Vous êtes un personnage, et vous décidez de vous arrêter lorsque vous rencontrez un obstacle.

Un petit exemple pour visualiser ça

Vous comprendrez que je ne me suis pas attardé sur le design.

Alors voilà : votre personnage avance vers la droite. Il rencontre un obstacle et s'arrête.

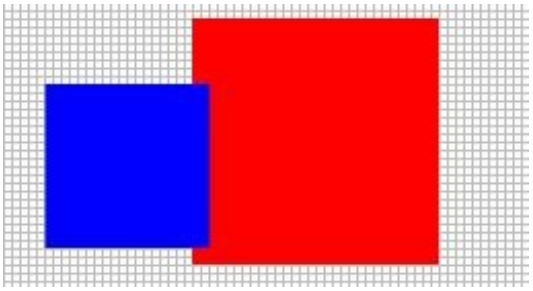
Code : Autre

```
var deplacement = 1;
var vitesse = 5;

_root.personnage.onEnterFrame = function() {
    if(deplacement) {
        if(this.hitTest(_root.obstacle)) {
            deplacement = 0;
        }
        if(Key.isDown(Key.RIGHT) && deplacement) {
            this._x += 1 * vitesse;
        }
    }
};
```

Avez-vous vu le problème ?

Voici un zoom de la collision :



Techniquement, que se passe-t-il ?
Dans mon exemple, vous avez sans doute vu que j'ai initialisé ma variable **vitesse** à 5.
Par conséquent, l'occurrence du clip **personnage** (le petit truc bleu ^^) avance de 5 pixels 12 fois par seconde (cadence par défaut).
On vérifie s'il y a collision.
S'il n'y en a pas, le personnage peut se déplacer vers la droite.
Mais si les deux occurrences sont écartées par un écart de 3 pixels ?
Il n'y a pas collision, on peut donc se déplacer de 5 pixels vers la droite. Et là : superposition.

Voilà le problème. Dans certains cas, ce n'est pas très gênant ; dans d'autres, ça peut le devenir.

Que faire alors pour régler le problème ?

Vous pourrez augmenter la cadence de 40 à 50 images par seconde, et réduire la vitesse à 1, l'occurrence du clip avancera de 1 pixel 40 à 50 fois par seconde, et là il n'y aura plus de superposition, mais c'est un peu brutal comme technique.

Et puis, pensez à ceux qui n'ont pas un ordinateur très puissant (j'ai un P3 à 866 MHz et mon ordi doit avoir mon âge, alors j'y pense continuellement).

Alors que faire ?

Rassurez-vous, il existe plusieurs solutions.
Je vais vous en donner quelques-unes, mais il en existe sûrement beaucoup plus.

Première solution - Retour en arrière

Voici la première solution.
Lorsqu'on détecte la collision, on revient à sa position précédente.

Voir l'exemple

Code : Autre

```
var deplacement = 1;
var vitesse = 4;

_root.personnage.onEnterFrame = function() {
        if(this.hitTest(_root.obstacle)) {
                this._x -= vitesse;
                deplacement = 0;
        }
        else {
                deplacement = 1;
        }
        if(deplacement) {
                if(Key.isDown(Key.RIGHT) && deplacement) {
                        this._x += 1 * vitesse;
                }
        }
};
```

Ce n'est pas génial, comme vous le voyez, le personnage "rebondit" sur l'obstacle.

Bref, on oublie cette solution.

Deuxième solution - Repositionnement après collision

Nous en arrivons à la deuxième solution.
Dès lors qu'une collision est détectée, au lieu de reculer le personnage, on va le repositionner.

Voici un jeu que j'avais commencé qui utilise ce procédé.

Que se passe-t-il ?

Le personnage avance. Dès qu'il rencontre un obstacle, on ne le fait pas reculer, on le repositionne.
Pour cela, on lui donne les coordonnées de l'obstacle auxquelles on soustrait la largeur du personnage (dans le cas de collisions horizontales) ou sa hauteur (dans le cas de collisions verticales).

Voici à quoi devraient ressembler vos commandes de repositionnement :

Code : Autre

```
personnage._y = _root.sol._y - personnage._height;
personnage._x = _root.obstacle._x - personnage._width;
```

Bien sûr, si vous avez suivi, la première ligne repositionnera le personnage au-dessus du sol, la deuxième à gauche d'un obstacle.

Si votre personnage touche un obstacle en se déplaçant vers la gauche, il faudrait faire :

Code : Autre

```
personnage._x = _root.obstacle._x + personnage._width;
```

Encore une fois, cette technique n'est pas parfaite, car si on n'a pas un ordi très puissant ( ), l'animation sera ralentie, et on verrait le personnage "s'enfoncer" dans le sol pour en ressortir aussitôt.

Voir l'exemple

Troisième solution - Une boucle

Voici la troisième et dernière solution qui me paraît la plus appréciable.
Elle va permettre d'arrêter le personnage sans qu'il ne se superpose à l'obstacle.

Partons de cet exemple.

Code : Autre

```
var deplacement = 1;
var vitesse = 1;

_root.personnage.onEnterFrame = function() {
        if(deplacement) {
                if(this.hitTest(_root.obstacle)) {
                        deplacement = 0;
                }
                if(Key.isDown(Key.RIGHT) && deplacement) {
```

```
                        this._x += 1 * vitesse;
                }
        }
};
```

Le personnage avance pixel par pixel, et s'arrêtera donc pile au moment où il touchera l'obstacle. Pas de superposition donc.
Seul problème : le personnage n'avance pas vite...
On pourrait augmenter la valeur de la variable **`vitesse`**, mais à ce moment-là, le personnage risque de se superposer à l'obstacle. On pourrait aussi augmenter la cadence, mais comme je l'ai dit dans l'introduction, ce n'est pas génial comme méthode, et surtout, ça empêche de modifier la vitesse au cours de l'animation (imaginez un jeu où le personnage pourrait marcher ou courir, ça ne serait pas réalisable sans modifier la vitesse).

Que peut-on faire alors ?

On va quand même essayer d'augmenter la vitesse.

Essayons ça :

Code : Autre

```
var deplacement = 1;
var vitesse = 5;

_root.personnage.onEnterFrame = function() {
        if(deplacement) {
                if(this.hitTest(_root.obstacle)) {
                        deplacement = 0;
                }
                if(Key.isDown(Key.RIGHT) && deplacement) {
                        this._x += 1 * vitesse;
                }
        }
};
```

Pas de nouveautés, c'est le code du premier exemple, et ça ne marche pas plus qu'avant.

Il va falloir ruser : on va utiliser une boucle.

En fait, au lieu d'avancer le personnage d'un nombre équivalent de pixels à la valeur de la vitesse, on va l'avancer d'un pixel, mais autant de fois que la valeur de la variable **`vitesse`** nous l'indiquera.
On va utiliser une boucle **`for`**, qui s'exécutera **`vitesse`** fois, dans laquelle nous ferons avancer notre personnage d'un pixel, puis dans cette même boucle on vérifiera si le personnage peut toujours avancer.

Allez, à vos ~~crayons~~ ordis.

Correction

Secret (cliquez pour afficher)

Code : Autre

```
var deplacement = 1;
var vitesse = 5;

_root.personnage.onEnterFrame = function() {
        if(deplacement) {
                for(i=0; i<vitesse; i++) {
                        if(this.hitTest(_root.obstacle)) {
```

```
                                        deplacement = 0;
                                }
                                if(Key.isDown(Key.RIGHT) && deplacement) {
                                        this._x += 1;
                                }
                        }
                }
        };
```

Voir l'exemple

Voilà.
Le code n'est pas bien dur, j'espère que vous avez trouvé.
Je pense que c'est la meilleure solution, parmi celles que je connais tout du moins.
À vous de trouver la vôtre.
Bon : je vous avoue quelque chose, je n'utilise presque pas de **`hitTest`**, pour vérifier les collisions en flash.

QUOI ?????

Ne m'en voulez pas , ce n'est pas de ma faute.
En fait, **`hitTest`** peut être pratique dans certains cas, mais lorsque vous avez beaucoup d'éléments différents, et que vous voulez vérifier les collisions, ce n'est pas facile.
Bon, allez, je vous mets sur la voie.
Lorsque j'ai beaucoup d'éléments, j'utilise le principe de "tuiles".
C'est un peu plus compliqué, mais rien d'impossible, et c'est ce principe qui est à la base de vieux jeux, comme Zelda Link's Awakening, ou Sword of Mana.
Je ne vais pas m'attarder sur le sujet, car ce n'est pas le thème de mon tuto, mais je vous conseille de chercher un peu sur le web comment ça fonctionne précisément, si jamais vous avez la phobie des **`hitTest`**.

Voilà, ce tuto est terminé ; j'espère qu'il vous aura été utile.

Ce tutoriel a été corrigé par les zCorrecteurs.