> Cette page a été traduite à partir de l'anglais par la communauté. Vous pouvez également contribuer en rejoignant la communauté francophone sur MDN Web Docs.

# Ajouter des fonctionnalités à notre exercice des balles rebondissantes

Dans cet exercice, vous devrez utiliser le jeu des balles rebondissantes de l'article précédent comme base, pour y ajouter de nouvelles fonctionnalitées intéressantes.

| | |
|---|---|
| **Prérequis:** | Avant de vous lancer dans cet exercice, il est fortement conseillé d'avoir vu et compris tous les précédents articles de ce module. |
| **Objectifs:** | Tester votre connaissance du Javascript orienté objet en conception et en pratique. |

## Pour commencer

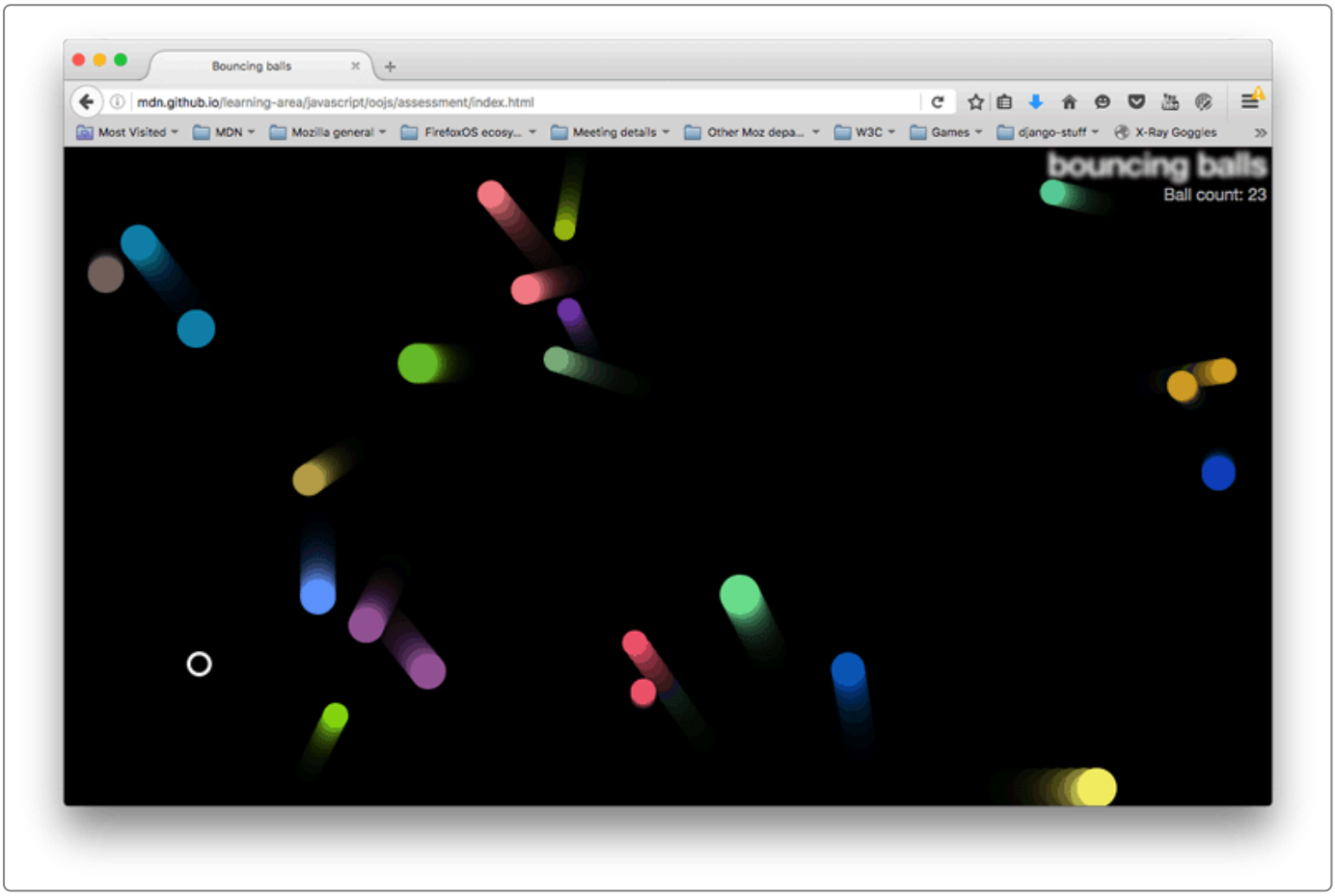
Pour commencer, faite une copie locale de index-finished.html , style.css , et main-finished.js de l'article précédent, dans un nouveau dossier.

> **Note :** Vous pouvez utiliser un site comme JSBin ou Thimble . Vous pouvez copier vos codes HTML, CSS et JavaScript dans l'un d'entre eux. Si celui que vous utilisez ne possède pas de fenêtres séparées pour les différents langages, ajoutez les dans des balises `<script>` / `<style>` dans votre code HTML.

## Le projet en bref

Notre jeu des balles est assez sympa, mais maintenant il s'agit de le rendre plus interactif en y ajoutant un viseur contrôlé par l'utilisateur, qui va détruire une balle s'il la touche. Nous voulons aussi tester votre capacité en programmation orientée objet en créant un object `Shape()` dont le viseur et les balles peuvent hériter. Pour terminer, nous voulons créer un compteur qui permet d'afficher combien de balles il nous reste encore à détruire.

Ce screenshot vous donne une idée du résultat final :




Si vous voulez en savoir plus, regardez l'exemple fini (n'en profitez pas pour récupérer le code source !).

## Vos objectifs

Cette section décrit ce que vous aurez à faire.

### Créons nos nouveaux objets

Pour commencer, modifions le constructeur de l'objet `Ball()` pour qu'il devienne le constructeur de `Shape()` puis créons en un nouveau pour `Ball()` :

1. Le constructeur `Shape()` devra définir les propriétés `x`, `y`, `velX`, et `velY` de la même manière que le constructeur `Ball()` auparavant, mais sans les propriétés `color` et `size`.
2. `Shape()` doit aussi définir une nouvelle propriété `exists`, qui servira à identifier les balles qu'il reste à détruire dans la fenêtre (celles qui n'ont pas encore été détruites). Elle doit retourner un booléen ( `true` / `false` ).
3. Le constructeur `Ball()` doit hériter des propriétés `x`, `y`, `velX`, `velY`, et `exists` du constructeur `Shape()`.
4. `Ball()` doit aussi définir les propriétés `color` et `size`, comme à l'origine.
5. N'oubliez pas de définir le prototype de `Ball()` et son constructeur de manière approprié.

Les méthodes `draw()`, `update()`, et `collisionDetect()` doivent fonctionnées comme avant, sans être modifiées.

Vous devrez ajouter un nouveau paramètre au constructeur `new Ball( ... )` — le paramètre `exists` doit être le 5ème et être égal à `true`.

Vous pouvez recharger la page — tout doit fonctionner comme avant, même après les modifications que vous avez effectuées sur les objets.

## Définition du EvilCircle() (viseur)

Il est temps de vous équipez ! — le `EvilCircle()` ! Dans notre jeu, nous allons créer un viseur mais nous allons nous servir de l'objet `Shape()` pour le définir. Vous voudrez certainement en ajouter un (plusieurs) autre plus tard, qu'un autre joueur ou l'ordinateur pourra contrôler. Vous n'irez probablement pas bien loin avec un seul viseur, mais ce sera suffisant pour le moment !

Le constructeur du `EvilCircle()` doit hériter des propriétés `x`, `y`, `velX`, `velY`, et `exists` de `Shape()`, mais `velX` et `velY` doivent toujours être égales à 20.

Vous devriez utiliser quelque chose comme `Shape.call(this, x, y, 20, 20, exists);`

Le constructeur doit aussi définir ses propres propriétés:

- `color` — `'white'`
- `size` — `10`

Une fois de plus, souvenez-vous de définir vos propriétés héritées en paramètre du constructeur et de définir le prototype et son constructeur de manière appropriée.

## Définir les méthodes du EvilCircle() (viseur)

`EvilCircle()` doit avoir quatre méthodes, comme définie en dessous.

### `draw()`

Cette méthode doit avoir la même fonction que celle de `Ball()` : soit dessiner l'objet dans le canvas. Elle fonctionnera quasiment de la même manière, copiez la fonction `Ball.prototype.draw`. Puis appliquez les modifications suivantes:

- On ne veut pas que le viseur soit plein, mais qu'il ait seulement un contour. Changez `fillStyle` et `fill()` pour `strokeStyle` et `stroke()`.
- On voudrait qu'il soit aussi un peu plus épais, pour être plus facile à voir. Pour ça on doit définir un attribut `lineWidth` (en-US) à ctx après l'appel à la fonction `beginPath()` (avec une valeur de 3).

### `checkBounds()`

Cette méthode à la même fonction que la première partie de `Ball()` `update()` — Savoir si le viseur va hors de l'écran, et l'arrêter si besoin. Une fois encore, copié la méthode `Ball.prototype.update`, mais en effectuant quelques changements:

- Débarrassez-vous des deux dernières lignes — on a pas besoin de connaître la position du viseur à chaque frame, car nous le déplacerons d'une manière différente comme vous pourrez le voir.
- Dans les conditions en `if()`, si la condition retourne true on ne veut pas modifier (update) les propriétés `velX` / `velY` ; mais plutôt changer les valeurs de `x` / `y` de manière à ce que le viseur revienne doucement dans l'écran. Ajouter ou soustraire de manière appropriée la taille ( `size`) du viseur sera suffisant.

## `setControls()`

Cette méthode ajoute un écouteur d'évènement `onkeydown` à l'objet `window` ce qui permettra en enfonçant certaine touche du clavier de déplacer le viseur dans la fenêtre. Insérez le code suivant dans la méthode :

JS

```
var _this = this;
window.onkeydown = function (e) {
  if (e.keyCode === 65) {
    _this.x -= _this.velX;
  } else if (e.keyCode === 68) {
    _this.x += _this.velX;
  } else if (e.keyCode === 87) {
    _this.y -= _this.velY;
  } else if (e.keyCode === 83) {
    _this.y += _this.velY;
  }
};
```

Quand une touche est enfoncée, la propriété keyCode (en-US) de l'objet event est consultée pour savoir quelle touche est enfoncée. Si c'est une des touches spécifiée, alors le viseur ce déplacera à gauche, à droite, en haut ou en bas.

- Pour un point bonus, faite apparaître à quel touche correspond le code de celle que l'utilisateur a enfoncé.
- Pour un second point bonus, pouvez vous nous dire pourquoi nous devons définir `var _this = this;` de cette façon ? Cela à quelque chose à voir avec la portée des fonction.

## `collisionDetect()`

Cette méthode fonctionne d'une manière similaire à `Ball()` `collisionDetect()`, copier celle-ci pour vous en servir comme base. Il y a deux différences:

- Dans la condition extérieure `if`, nous n'avons plus besoin de vérifier si la balle actuellement dans la boucle est celle actuellement surveiller — Parce que ce n'est plus une balle, mais notre viseur ! A la place, on doit tester si la balle visée existe (avec

quelle propriété pourrez vous faire cela?). Si elle n'existe pas, c'est qu'elle a déjà été détruite, on a donc pas besoin de la vérifier encore une fois.

- Dans la condition intérieur `if`, on ne souhaite plus changer un élément de couleur lorsqu'une collision est détéctée — A la place, on veut détruire les balles qui entre en collision avec le viseur (encore une fois, comment pensez-vous faire cela ?).

## Insérer le viseur dans notre programme

Maintenant que nous avons définit notre viseur, on a besoin de le faire apparaître à l'écran. Pour ce faire on doit appliquer quelques modifications à la fonction `loop()`.

- Premièrement, créons une nouvelle instance de l'objet viseur (en spécifiant les paramètres nécessaire), et appelons sa méthode `setControls()`. On doit seulement effectuer ses deux actions une seule fois, pas à chaque itération.
- Au moment où l'on boucle à travers toutes les balles et que l'on appelle les méthodes `draw()`, `update()`, et `collisionDetect()` pour chacune d'entre elle, faite de manière à ce que ces fonctions soit appelées seulement si la balle existe.
- Appellez les méthodes de l'instance du viseur `draw()`, `checkBounds()`, et `collisionDetect()` à chaque itération de la boucle.

## Implémenter le compteur de score

Pour implémenter le compteur de score, suivez les étapes suivantes:

1. Dans votre fichier HTML, ajoutez un élement `<p>` qui contiendra le texte suivant "Ball count: ", juste en dessous de l'élément `<h1>`.
2. Dans votre fichier CSS, ajouter les règlesz suivantes:

   CSS

```
p {
  position: absolute;
  margin: 0;
  top: 35px;
  right: 5px;
  color: #aaa;
}
```

3. Dans votre JavaScript, effectuez les modifications suivante :

- Incrémentez le compteur de balle à chaque fois qu'une balle apparait à l'écran.
- Décrementez le compteur à chaque fois qu'une balle est détruite par le viseur.

## Conseils et astuces

- Cet exercice est un bon challenge. Prenez le temps de faire et de comprendre chaque étape.
- Ce serait une bonne idée de garder une copie de chaque étape lorsque vous arrivez à la faire marcher correctement, pour vous y référrer si vous n'arrivez plus à progresser ensuite.

## Evaluation

Si vous effectuez cette évalutation dans le cadre d'un cours, vous devriez pouvoir fournir votre travail à votre professeur/mentor pour correction. Si vous apprenez par vous même, vous pouvez obtenir la correction sur discussion thread for this exercise , ou sur #mdn IRC channel sur Mozilla IRC . Tout d'abord effectuez cet exercice — vous n'obtiendrez jamais rien en trichant !

This page was last modified on 3 août 2023 by MDN contributors.