

Source Alsacréation

Canvas






Depuis l'arrivée de HTML5 et de ses nouveaux éléments, les sites et applications web peuvent profiter de grandes avancées pour rendre ce média attractif : audio, vidéo et Canvas pour disposer de fonctionnalités de dessin en 2D et 3D dans le navigateur. L'élément qui nous intéresse ici est **<canvas>**. Il s'agit d'un espace de **pixels** initialement transparents, armés de **JavaScript** pour réaliser un bon nombre de fonctions graphiques, partant du simple tracé de courbe pour aller jusqu'aux animations et jeux vidéo. Le nombre de démonstrations existant aujourd'hui sur le web est impressionnant, il serait difficile de toutes les lister ici pour en donner un avant-goût complet des possibilités.

L'inconvénient encore actuel de Canvas est que ses usages sont si variés et modulaires qu'il n'existe pas d'outil miracle ou d'IDE (Environnement de Développement Intégré) pour produire le code nécessaire sans devoir toucher directement au code JavaScript.

Qu'en est-il de SVG ?

Attention, il ne faut pas confondre Canvas, qui est une surface de dessin *bitmap* pilotable en JavaScript spécifique à HTML, et [SVG](#), qui est un format de fichier pour le dessin vectoriel, décrit en XML. Bien souvent, on néglige de penser à SVG comme alternative à Flash, alors que celui-ci serait plus approprié (aussi conçu pour être vectoriel, animé, léger et modulaire).

Canvas est supporté à l'heure actuelle par tous les navigateurs modernes.

Navigateurs	Versions
	Firefox 2.0
	Chrome 4.0
	Internet Explorer 9
	Opera 9.0
	Safari 3.1

Création du canvas et principes généraux

Canvas étant un nouvel élément, il ne déroge pas à une structure HTML classique, avec des dimensions précisées par les attributs **width** et **height** (ou via CSS).

```
<canvas id="mon_canvas" width="350" height="350">
```

Texte alternatif pour les navigateurs ne supportant pas Canvas.

```
</canvas>
```

À partir de ce moment, tout se passe du côté de JavaScript, qui va se servir de cet élément HTML pour accéder à la surface de dessin. Pour ceci, deux fonctions sont appelées :

- **getElementById()** qui va permettre d'aller chercher et cibler l'élément **<canvas>** identifié par son attribut **id** unique (ici **mon_canvas**),
- puis la méthode **getContext()** de l'élément ainsi récupéré pour savoir dans quel contexte de dessin (2D ou 3D) le script va pouvoir agir, et de quelles fonctions il pourra disposer. Le contexte sera l'élément central de gestion de **Canvas**.

```
<script type="text/javascript">
```

```
var c = document.getElementById("mon_canvas");
```

```
var ctx = c.getContext("2d");
```

```
// Le reste du script ici...
```

```
</script>
```

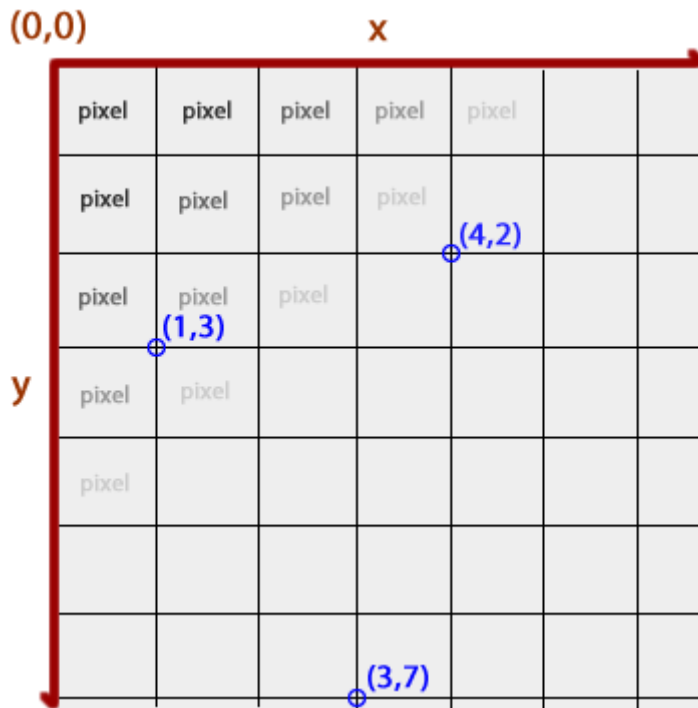
Tous les exemples suivants dans ce tutoriel feront appel à cette structure (élément Canvas + élément script + appel à `getElementById` + `getContext`) qui ne sera pas précisée à chaque fois.

Après cette étape préliminaire de mise en place, il faut se plonger dans l'ensemble des méthodes de dessin 2D.

Celles-ci vont toutes exploiter le même système de coordonnées :

- Le point de référence (0,0) est situé en haut à gauche
- L'axe horizontal (x) est défini par la première coordonnée
- L'axe vertical (y) est défini par la seconde coordonnée
- Ces valeurs correspondent à la grille *entourant* les pixels, et non pas aux pixels eux-mêmes

Par exemple le point de coordonnées (4,2) sera situé 4 pixels à droite du coin supérieur gauche, et 2 pixels en-dessous.



Si l'on trace un polygone entre les 3 points présents sur ce schéma, on obtiendra un triangle.

Ce tutoriel concerne uniquement la version 2D de **Canvas**, car la manipulation de la 3e dimension fait appel à des compétences radicalement différentes et son support est plus limité. Pour utiliser **Canvas** en 3D, le nom du contexte est "webgl", mais change ensuite du tout au tout les fonctions disponibles et les méthodes de tracé

Tracés et chemins

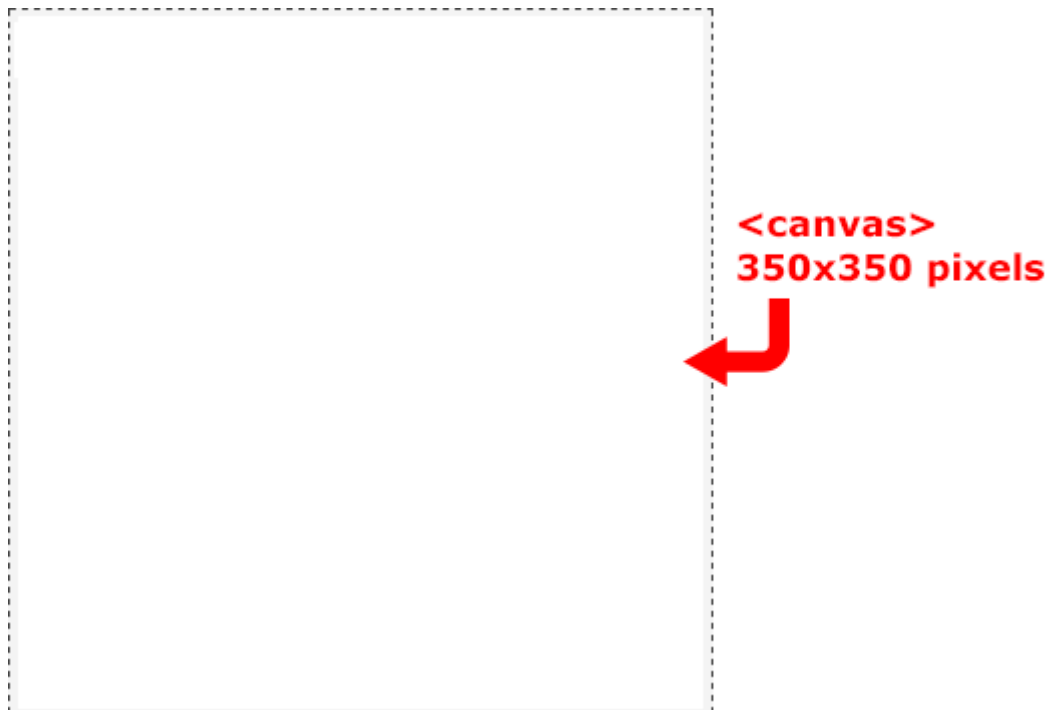
Une fois les bases du dessin établies, les premières fonctions les plus abordables sont celles qui vont tracer des formes géométriques ou des chemins. Un tracé peut se dérouler en plusieurs étapes: initialisation, point de départ puis point d'arrivée, clôture, affichage du contour et/ou du remplissage.

Lignes, tracés, chemins

Un tracé est d'abord initialisé par la méthode **beginPath()**. Le point de référence de début du tracé est désigné avec **moveTo(x,y)**. Il s'agit en quelque sorte de décider à partir de quel emplacement le pinceau va être posé. Puis vient le tracé de la ligne elle-même avec la méthode **lineTo(x,y)** qui va ajouter un segment au chemin qui fut débuté par **beginPath()**. On peut ajouter autant de segments que l'on veut, puis éventuellement "fermer" la forme pour revenir automatiquement au point de départ avec **closePath()**.

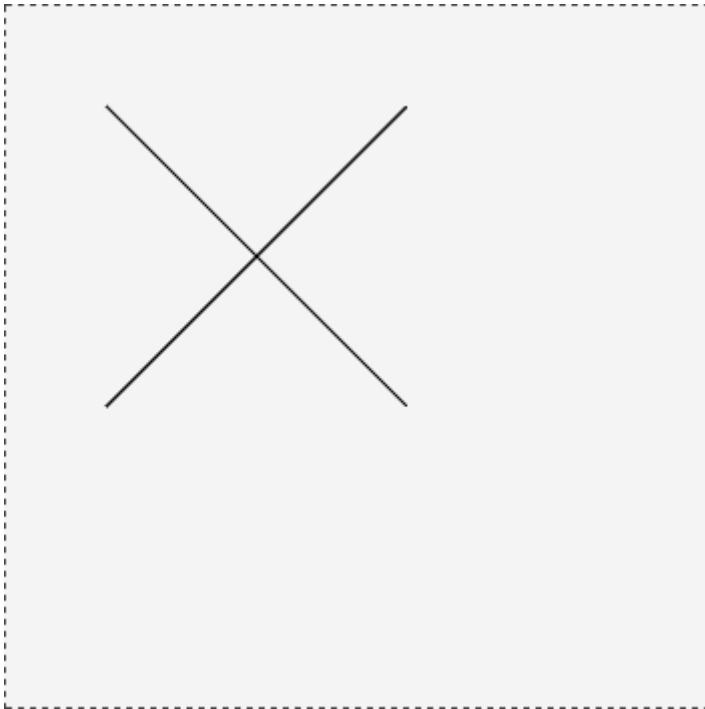
```
var ctx = c.getContext("2d");
ctx.beginPath();      // Début du chemin
ctx.moveTo(50,50);     // Le tracé part du point 50,50
ctx.lineTo(200,200);   // Un segment est ajouté vers 200,200
ctx.moveTo(200,50);    // Puis on saute jusqu'à 200,50
ctx.lineTo(50,200);    // Puis on trace jusqu'à 50,200
ctx.closePath();       // Fermeture du chemin (facultative)
```

Voilà qui est fait le tracé est en place pourtant rien ne change à l'affichage et c'est bien normal, car il n'est mentionné nulle part de couleur, de style de trait ou de remplissage, et si l'on veut plutôt une forme pleine, ou une forme dont on n'afficherait que le contour.

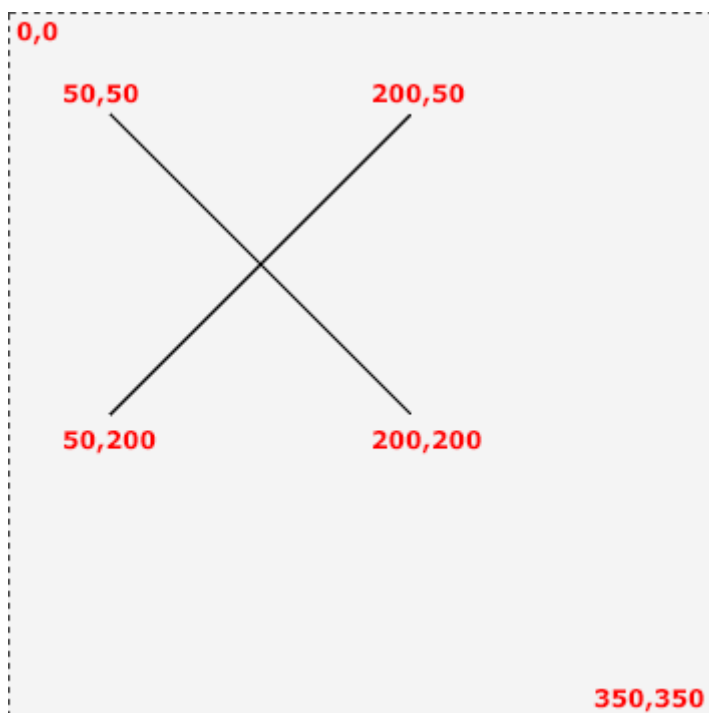


La forme n'apparaît qu'une fois appelée l'une des deux méthodes **fill()** pour remplir et **stroke()** pour le contour. Dans l'exemple courant, il sera utile de visualiser uniquement le contour du tracé.

```
ctx.stroke();
```



Voici le résultat avec pour rappel les coordonnées des points utilisés :



Styles de contour et de remplissage

Pour moduler les styles des couleurs de contour et de remplissage dont dépendent **fill()** et **stroke()**, il faut agir sur des **propriétés du contexte** de dessin qui sont **fillStyle** et **strokeStyle**. Les valeurs acceptées sont tous les codes couleurs reconnus par le navigateur, par exemple comme en CSS : par nom (red, black), par code

hexadécimal (#f00, #000000), par code rgb, etc. On remarque qu'il s'agit ici bien de s'adresser à une propriété JavaScript et non pas de déclencher une fonction (il n'y a pas de parenthèses à la fin de la ligne, mais bien une affectation grâce au signe égal). Ces propriétés peuvent aussi accepter des dégradés ou des motifs, qui seront abordés plus tard.

La propriété `lineWidth` affecte l'épaisseur du trait, qui est de 1 pixel par défaut.

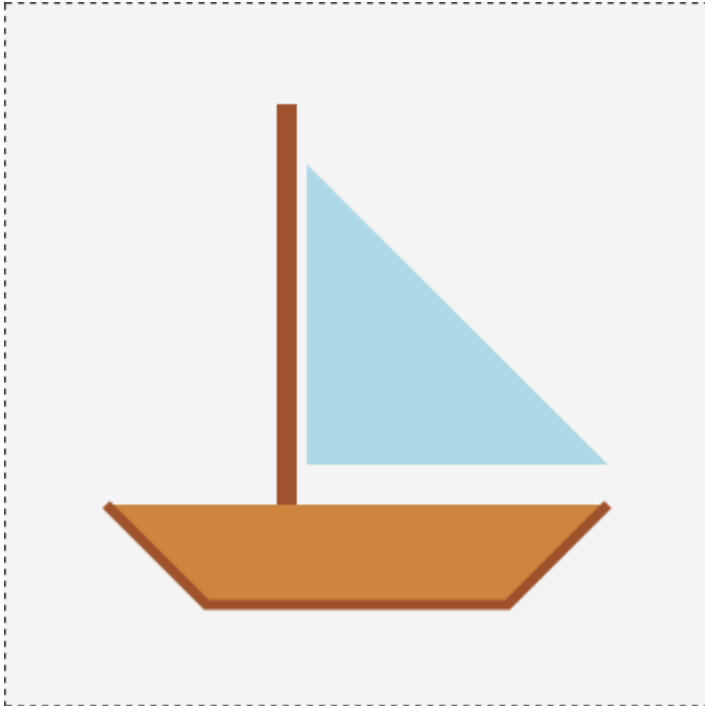
```
var c = document.getElementById('mon_canvas');
var ctx = c.getContext("2d");

// Voile du bateau
ctx.beginPath();      // Début du chemin
ctx.moveTo(150,80);   // Le tracé part du point 150,80
ctx.lineTo(300,230);  // Un segment est ajouté vers 300,230
ctx.lineTo(150,230);  // Un segment est ajouté vers 150,230
ctx.closePath();      // Fermeture du chemin
ctx.fillStyle = "lightblue"; // Définition de la couleur de remplissage
ctx.fill();           // Remplissage du dernier chemin tracé

// Coque du bateau
ctx.beginPath();      // Début d'un autre chemin
ctx.moveTo(50,250);
ctx.lineTo(100,300);
ctx.lineTo(250,300);
ctx.lineTo(300,250);
ctx.fillStyle = "peru";
ctx.strokeStyle = "sienna"; // Définition de la couleur de contour
ctx.lineWidth = 5;         // Définition de la largeur de ligne
ctx.fill();               // Application du remplissage
ctx.stroke();              // Application du contour

// Mât
ctx.beginPath();
ctx.moveTo(140,50);
ctx.lineTo(140,250);
ctx.lineWidth = 10;
ctx.stroke();
```

Le tracé de ces 3 formes avec des contours et remplissages variés produit un simple bateau.

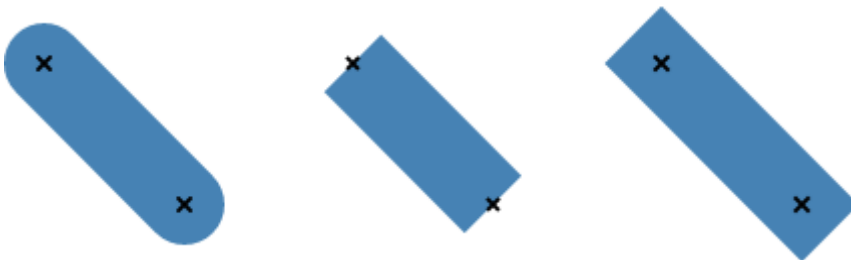


(Ceci est une capture d'écran PNG de la démonstration)

Styles de ligne

Outre la propriété **lineWidth** évoquée précédemment pour la largeur de ligne, existent **lineCap** pour le style de fin de ligne :

- *round* : pour faire un arrondi au delà du bout du tracé,
- *butt* : tracé "coupé" carré,
- *square* : style carré (dépassé du point d'arrivée)



Formes

Outre les fonctions de tracé de chemin, il existe des méthodes orientées vers d'autres formes.

Carré et rectangle

Pour tracer un rectangle en donnant sa *hauteur*, sa *largeur* et son point de départ pour les coordonnées *startx*, *starty*, la fonction **fillRect()** est indiquée. Un rectangle de dimensions horizontale et verticale égales produira un carré.

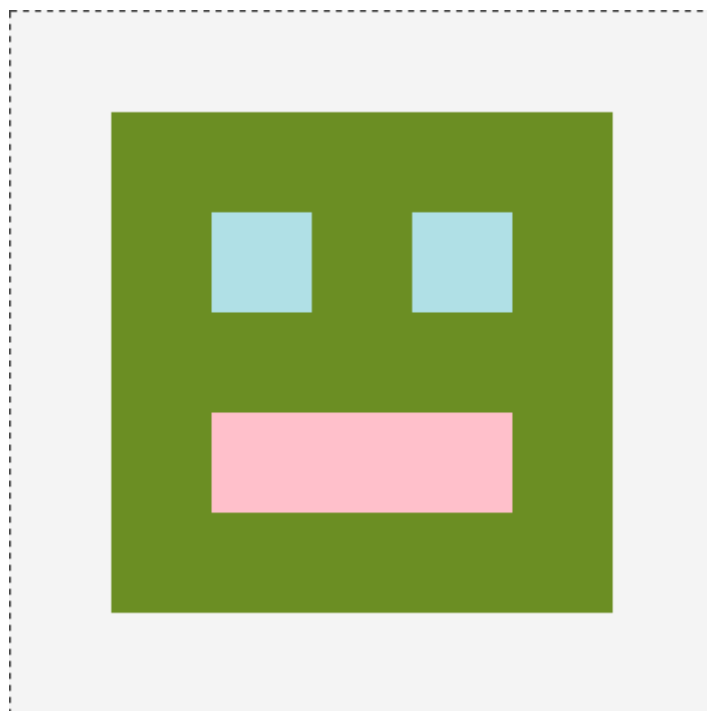
```
fillRect(startx, starty, hauteur, largeur);

var c = document.getElementById( "mon_canvas" );
var ctx = c.getContext("2d");

// Fond
ctx.fillStyle = "olivedrab";
ctx.fillRect(50,50,250,250);

// Bouche
ctx.fillStyle = "pink";
ctx.fillRect(100,200,150,50);

// Yeux
ctx.fillStyle = "powderblue";
ctx.fillRect(100,100,50,50);
ctx.fillRect(200,100,50,50);
```



Effacer une portion de surface

Pour finir cette partie sur les formes voici une petite fonction bien utile qui efface tous les pixels tracés jusqu'à présent, sur la totalité ou une partie du **canvas**.

clearRect(startx, starty, hauteur, largeur)

Ses paramètres sont identiques à **fillRect()** mais elle "gommera" tout le contenu du rectangle ainsi défini.

Arcs, cercles et courbes

Arc de cercle

Les traits droits c'est bien, mais pouvoir faire des cercles ou des arcs de cercle c'est très pratique. Deux méthodes existent : **arcTo()** et surtout **arc()**. Le prototype de cette dernière fonction définit les coordonnées centrales de l'arc, son rayon (toujours en pixels), l'angle de début et de fin, et enfin dans quel sens le pinceau va tourner grâce à un booléen.

arc(x, y, radius, startAngle, endAngle, sensAntiHoraire)

Sachant que l'on est dans une configuration trigonométrique les angles sont définis en radians avec **Math.PI** (un tour complet de cercle = $2 * \text{Math.PI}$) et le sens de rotation est contraire aux aiguilles d'une montre lorsqu'il vaut **true**.

```
var c = document.getElementById( "mon_canvas" );
var ctx = c.getContext("2d");
ctx.lineWidth = 5;
```

```
// Visage
```

```
ctx.beginPath();
ctx.arc(150,150,100,0,Math.PI*2,true);
ctx.strokeStyle = "coral";
ctx.fillStyle = "bisque";
ctx.fill();
ctx.stroke();
```

```
// Bouche
```

```
ctx.beginPath();
ctx.arc(150,150,60,0,Math.PI,false);
ctx.strokeStyle = "red";
```

```

ctx.stroke();
// Yeux
ctx.beginPath();
ctx.strokeStyle = "#369";
ctx.fillStyle="#c00";
ctx.arc(180,130,15,0,Math.PI*2,false);
ctx.stroke();
ctx.beginPath();
ctx.arc(120,130,15,0,Math.PI*2,false);
ctx.stroke();

```



Conclusion:

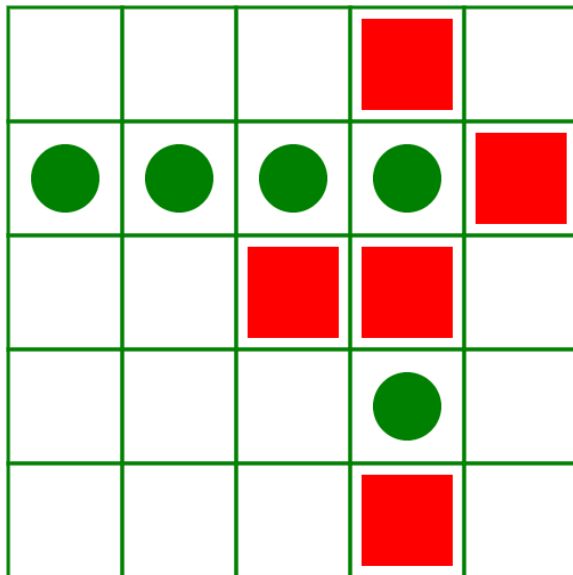
Ce petit tutoriel avait pour objectif de vous montrer rapidement comment utiliser l'élément HTML nommé **canvas** conjointement au javascript. Vous trouverez sur web de nombreux tutoriaux pour compléter cette formation. Il est important de savoir qu'il aussi possible de faire de la 3D grâce à cet élément. Pour ceux que cela intéresse, je vous conseille de chercher des informations sur le standard WebGL et la bibliothèque Three.js.

Si vous voulez aller plus loin :

[Utilisez la balise canvas - Utilisez HTML5 pour l'interface utilisateur - OpenClassrooms](#)

https://threejs.org/examples/#webgl_animation_cloth

TP : Réalisation d'un jeu de morpion 5x5. Pour gagner il faudra aligner quatre carrés ou quatre ronds.



Jeu du MORPION

Joueur 1: Score: <<<<<

Joueur 2: Score: GAGNE

Principe:

1° Pour jouer, il faudra impérativement que les noms des deux joueurs soient saisis.

2° Les joueurs jouent alternativement. Le symbole <<<< indique qui doit jouer.

3° Si un joueur aligne 4 ronds ou 4 carrés, un message **Gagné** apparaît à côté de son nom, son score augmente de 1, le jeu est bloqué et deux boutons apparaissent **'Rejouer'** et **"Nouvelle partie"**.

4° En cliquant sur **Rejouer** les deux même joueurs peuvent refaire une partie (le jeu précédent est effacé et les deux boutons sont masqués)

5° En cliquant sur **Nouvelle partie** les noms et les scores sont effacés. Les deux boutons sont masqués.

6° Chaque fois qu'un score évolue, les noms et les scores sont enregistrés en **localStorage**.

7° Au démarrage d'une partie dans une nouvelle fenêtre du navigateur, si deux noms et deux scores sont enregistrés dans le **localStorage**, ils sont alors affichés avec les deux boutons **'Rejouer'** et **"Nouvelle partie"**.

8° Lorsque 25 coups ont été joués sans réussir à aligner 4 ronds ou 4 carrés, le message **"Partie nulle"** s'affiche, la partie est bloquée et deux boutons **'Rejouer'** et **"Nouvelle partie"** sont rendus visibles.