PROGRAMMATION WEB AVANCEE

TP3

Canvas

Introduction

L'élément <canvas> est une zone dans laquelle il va être possible de dessiner au moyen du JavaScript.

- Canvas supporte également un système basique pour créer des animations.
- Le dessin se fait par l'intermédiaire de coordonnées dont l'origine des axes (le point (0,0)) est le coin supérieur gauche du canvas.
- Une fois dessinée, une forme n'est plus manipulable. Il est nécessaire d'effacer le contenu du canvas, puis de redessiner.
- Canvas ne supporte que quelques formes : le rectangle, l'arc de cercle, et les courbes de Bézier quadratiques et cubiques.
- Il est également possible d'insérer des images au sein du canvas, tout comme de créer des dégradés ou d'ajouter du texte.
- L'utilisation des rotations et des translations facilite les calculs des coordonnées et donc la création du dessin.

Exercice1 : initiation

Etape 1 : insertion du canvas :

```
canvas id="canvas" width="150" height="150">

    <p>Désolé, votre navigateur ne supporte pas Canvas. Mettez-vous à jour</p>

</canvas>
```

Etape 2 : accès au canvas

```
var canvas  = document.querySelector('#canvas');

var context = canvas.getContext('2d');
```

Etape 3 : Une fois qu'on a le canvas, il faut accéder à son *contexte*, avec **getContext()**. Il n'y a pour l'instant qu'un seul contexte disponible : la deux dimensions (2D).

Etape 4 : Principe de fonctionnement

Dessiner avec Canvas se fait par le biais de coordonnées. Le coin supérieur gauche du canvas est de coordonnées (0,0). Si on descend ou qu'on va vers la droite, on augmente les valeurs.

On va utiliser les méthodes pour tracer des lignes et des formes géométriques.

Traçons un rectangle de 50 sur 80 pixels :

```
context.fillStyle = "gold";
context.fillRect(0, 0, 50, 80);
```

Dans un premier temps, on choisit une couleur avec **fillStyle**. Puis, avec **fillRect()**, on trace un rectangle. Les deux premiers paramètres sont les coordonnées du sommet supérieur gauche du rectangle que nous voulons tracer. Le troisième paramètre est la largeur du rectangle, et le quatrième est la hauteur. Autrement dit :

fillrect(x, y, largeur, hauteur).

Code source

```
<html>
<head>
<meta charset="utf-8" />
<title>Partie V - Chapitre 3 - Exemple 1</title>
<style>body { background: black; } canvas { background: white; }</style>
</head>

<body>

<canvas id="canvas" width="150" height="150">
<p>Désolé, votre navigateur ne supporte pas Canvas. Mettez-vous à
jour</p>
</canvas>

<script>

window.onload = function() {
var canvas = document.querySelector('#canvas');
var context = canvas.getContext('2d');

context.fillStyle = "hsl(51, 100%, 50%)";
context.fillRect(0, 0, 50, 80);
```

```
};

</script>
</body>
</html>
```

On centre le rectangle. Et on ajoute un carré de 40 pixels d'une couleur semi-transparente :

```
context.fillStyle = "gold";

context.fillRect(50, 35, 50, 80);

context.fillStyle = "rgba(23, 145, 167, 0.5)";

context.fillRect(40, 25, 40, 40);
```

La propriété **fillStyle** peut recevoir diverses valeurs : le nom de la couleur, un code hexadécimal, une valeur RGB, HSL ou HSLA ou, comme ici, une valeur RGBA. Dans le cas d'une valeur RGBA, le quatrième paramètre est l'opacité, définie sur une échelle de 0 à 1, le 0 étant transparent et le 1 opaque.

Etape 5 : Le fond et les contours

Il est possible de créer des formes creuses, avec un contour.

Canvas considère deux types de formes : **fill** et **stroke**.

Une forme **fill** est une forme remplie,

Une forme **stroke** est une forme vide pourvue d'un contour.

Pour créer un rectangle **fill** on utilise **fillRect(),** pour créer un rectangle **stroke** on va utiliser **strokeRect()**

```
context.strokeStyle = "gold";

context.strokeRect(50, 35, 50, 80);

context.fillStyle = "rgba(23, 145, 167, 0.5)";

context.fillRect(40, 25, 40, 40);
```

Il est possible de choisir l'épaisseur du contour à utiliser.

Ça se fait avec la propriété **lineWidth** :

```
context.lineWidth = "5";

context.strokeStyle = "gold";

context.strokeRect(50, 35, 50, 80);

context.fillStyle = "rgba(23, 145, 167, 0.5)";

context.fillRect(40, 25, 40, 40);
```

Etape 6 : Effacer

Une dernière méthode existe en ce qui concerne les rectangles : ***clearRect(x, y, largeur, hauteur).***

Cette méthode agit comme une gomme : elle effacer du canvas les pixels délimités par le rectangle. On lui fournit les coordonnées des quatre sommets. clearRect() est utile pour faire des découpes au sein des formes.

```
context.strokeStyle = "gold";

context.strokeRect(50, 35, 50, 80);

context.fillStyle = "rgba(23, 145, 167, 0.5)";

context.fillRect(40, 25, 40, 40);

context.clearRect(45, 40, 30, 10);
```

Application 1

Créer une page contenant un canvas.

- contenant un **canvas** dont la couleur de fond sera rouge
- afficher dans ce canvas un rectangle de couleur différente de celle du fond.
- donner au rectangle un contour blanc de 10 pixels de large.

Exercice2 : formes géométriques

Les chemins simples

Les chemins vont permettre de créer des lignes et des polygones.

Pour ce faire, plusieurs nouvelles méthodes **: beginPath() et closePath(), moveTo(), lineTo(), stroke() et son équivalent fill().**

La création de chemins se fait par étapes successives. On commence par initier un nouveau chemin avec **beginPath().** Ensuite, avec **moveTo(),** on déplace le « crayon » à l'endroit où on souhaite commencer le tracé : c'est le point de départ du chemin. Puis, on utilise **lineTo()** pour indiquer un deuxième point, un troisième, etc. Une fois tous les points du chemin définis, on applique au choix **stroke()** ou **fill()** :

```
context.strokeStyle = "rgb(23, 145, 167)";

context.beginPath();

context.moveTo(20, 20);  // 1er point

context.lineTo(130, 20); // 2e point

context.lineTo(130, 50); // 3e

context.lineTo(75, 130); // 4e

context.lineTo(20, 50);  // 5e

context.closePath();     // On relie le 5e au 1er

context.stroke();
```

closePath() n'est pas nécessaire ; il termine le chemin en reliant le dernier point au premier. Si on veut une forme fermée, via **stroke()**, c'est assez pratique. Par contre, si on veut remplir la forme avec **fill()**, la forme sera fermée automatiquement, donc **closePath()** est inutile.

Les arcs

En plus des lignes droites, il est possible de tracer des arcs de cercle, avec la méthode arc(x, y, rayon, angleDepart, angleFin, sensInverse). Les angles sont exprimés en radians (oui, rappelez-vous vos cours de trigonométrie !). Avec les arcs, x et y sont les coordonnées du *centre de l'arc*. Les paramètres angleDepart et angleFin définissent les angles de début et de fin de l'arc. Comme dit plus haut, c'est exprimé en radians, et non en degrés.

Pour rappel, pour obtenir des radians il suffit de multiplier les degrés par π divisé par 180: (Math.PI / 180) * degrees.

```
context.beginPath(); // Le cercle extérieur

context.arc(75, 75, 50, 0, Math.PI * 2); // Ici le calcul est simplifié

context.stroke();



context.beginPath(); // La bouche, un arc de cercle

context.arc(75, 75, 40, 0, Math.PI); // Ici aussi

context.fill();

context.beginPath(); // L'œil gauche

context.arc(55, 70, 20, (Math.PI / 180) * 220, (Math.PI / 180) * 320);

context.stroke();

context.beginPath(); // L'œil droit

context.arc(95, 70, 20, (Math.PI / 180) * 220, (Math.PI / 180) * 320);

context.stroke();
```

 Un smiley dessiné avec JavaScript

Pour chaque arc, il est plus facile de commencer un nouveau chemin avec beginPath().

Utilisation de moveTo()

Comme on l'a vu plus haut, **moveTo()** permet de déplacer le « crayon » à l'endroit où l'on souhaite commencer un chemin. Mais cette méthode peut aussi être utilisée pour effectuer des « levées de crayon » au sein d'un même chemin :

```
context.beginPath(); // La bouche, un arc de cercle

context.arc(75, 75, 40, 0, Math.PI);

context.fill();



context.beginPath(); // Le cercle extérieur

context.arc(75, 75, 50, 0, Math.PI * 2);

context.moveTo(41, 58); // L'œil gauche

context.arc(55, 70, 20, (Math.PI / 180) * 220, (Math.PI / 180) * 320);

context.moveTo(81, 58); // L'œil droit

context.arc(95, 70, 20, (Math.PI / 180) * 220, (Math.PI / 180) * 320);

context.stroke();
```

Application 2

Écrire une page HTML 5 :

- contenant un **canvas**
- dessiner dans ce **canvas** un polygone de 6 cotés
- donner une couleur de contour à ce polygone

Application 3

Écrire une page HTML 5 :

- contenant un **canvas**
- dessiner dans ce **canvas** des arcs de cercle

Exercice3 : les images

Les images

Il est possible d'insérer des images au sein d'un canvas. Pour ce faire, on utilisera la méthode **drawImage(image, x, y)**.

Pour qu'une image puisse être utilisée, elle doit au préalable être accessible via un objet Image ou un élément <img />. Il est également possible d'insérer un canvas dans un canvas ! En effet, le canvas que l'on va insérer est considéré comme une image.

Insérons l'âne Zozor au sein du canvas :

```
var zozor = new Image();

zozor.src = 'zozor.png'; // Image de 80x80 pixels

context.drawImage(zozor, 35, 35);
```

Attention aux grandes images : si l'image est trop longue à charger, elle sera affichée de façon saccadée au sein du canvas. Une solution est d'utiliser onload pour déclencher le dessin de l'image une fois qu'elle est chargée :

```
var zozor = new Image();
    zozor.src = 'zozor.png';
    zozor.addEventListener('load', function() {
        context.drawImage(zozor, 35, 35);
    };
```

Zozor est affiché dans le canvas

Mise à l'échelle

drawImage(image, x, y, largeur, hauteur) possède deux paramètres supplémentaires facultatifs : largeur et hauteur, qui permettent de définir la largeur et la hauteur que l'image occupera une fois incrustée dans le canvas. Si la diminution de la taille des images ne pose pas trop de problèmes, évitez toutefois de les agrandir, au risque de voir vos images devenir floues.

```
context.drawImage(zozor, 35, 35, 40, 40);
```

Ici, l'image est réduite de moitié, puisque de base elle fait 80 pixels sur 80 pixels.

Recadrage

Quatre paramètres supplémentaires et optionnels s'ajoutent à drawImage(). Ils permettent de recadrer l'image, c'est-à-dire de prélever une zone rectangulaire au sein de l'image afin de la placer dans le canvas :

```
drawImage(image, sx, sy, sLargeur, sHauteur, dx, dy, dLargeur, dHauteur)
```

Les paramètres commençant par *s* indiquent la *source*, c'est-à-dire l'image, ceux commençant par *d*indiquent la *destination*, autrement dit le canvas :

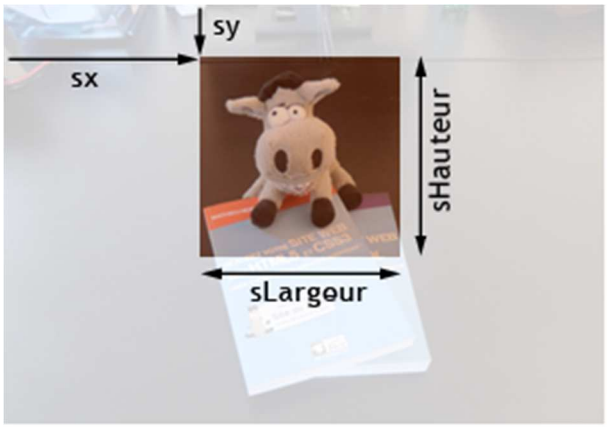


Image Source

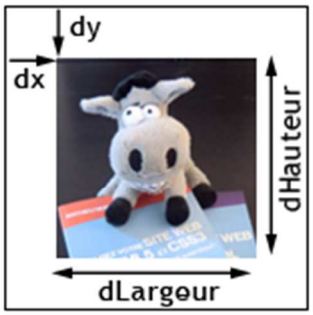


Image Destination

Il est possible de recadrer une image

Toute la difficulté est donc de ne pas s'emmêler les pinceaux dans les paramètres :

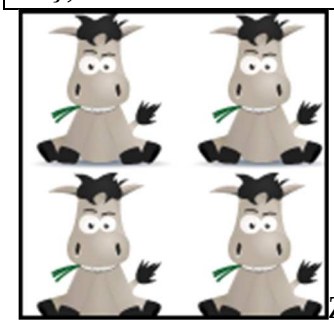
```
var zozor = document.querySelector('#plush');

context.drawImage(zozor, 99, 27, 100, 100, 25, 25, 100, 100);
```

Les patterns

Comment faire se répéter une image pour, par exemple, créer un fond ? C'est possible de faire une double boucle for et d'insérer plusieurs fois la même image.

Mais il y a plus simple : les **patterns**. Un pattern est une image qui se répète comme un papier peint. Pour en créer un, on utilise la méthode **createPattern(image, type).** Le premier argument est l'image à utiliser, et le deuxième est le type de pattern. Différents types existent, mais seul repeatsemble reconnu par la plupart des navigateurs :

```
var zozor = new Image();

    zozor.src = 'zozor.png';

    zozor.addEventListener('load', function() {

        var pattern = context.createPattern(zozor, 'repeat');

        context.fillStyle = pattern;

        context.fillRect(0, 0, 150, 150);

    };
```


Zozor se répète grâce aux patterns

Essayer le code

La façon de procéder est un peu étrange, puisqu'il faut passer le pattern à fillStyle, et ensuite créer un rectangle plein qui recouvre l'entièreté du canvas. En clair, il s'agit de créer un rectangle avec une image qui se répète comme fond.

Vous devez absolument passer par l'événement load, sinon le pattern ne s'affichera pas correctement si l'image n'est pas chargée.

Application 4

Écrire une page HTML 5 :

- contenant un **canvas**
- charger cette image dans le **canvas**
- le canvas aura pour dimension 1306x720 pixels
- placer un cercle rouge semi-transparent de 50 pixels de rayon aux coordonnées 330 (horizontal) et 240 (vertical)
- éffacer la zone rectangulaire de 100x100 pixels placée aux coordonnées 670 (horizontal) et 100 (vertical)

Il faudra faire attention au temps pris par le chargement de l'image, et donc ne commencer le traitement qu'une fois que l'image aura été complètement chargée. Dans le cas contraire on aura un comportement erratique dépendant des conditions de courses des processus chargement/traitement.
L'appel du traitement peut se faire par un handler positionné sur la fin du chargement de l'image par sa méthode **addEventListener('load', *fonction_de_traitement*, false)**.

Exercice4 : le texte

Pour écrire du texte au sein d'un canvas, il y a les méthodes fillText() et strokeText(), secondées par la propriété font, qui permet de définir le style du texte :

```
context.fillStyle = "rgba(23, 145, 167, 1)";

context.fillRect(50, 50, 50, 50);

context.font = "bold 22pt Calibri,Geneva,Arial";

context.fillStyle = "#fff";

context.fillText("js", 78, 92);
```

Les méthodes **fillStyle** et **strokeStyle** sont toujours utilisables, puisque les textes sont considérés comme des formes au même titre que les rectangles ou les arcs.

La propriété font reçoit des informations sur la police à utiliser, à l'exception de la couleur, qui est gérée par **strokeStyle** et **fillStyle**. Dans l'exemple qui va suivre, nous allons utiliser un texte en Calibri, de 22 points et mis en gras.

fillText() reçoit trois paramètres : le texte et les positions x et y*de la ligne d'écriture du texte* :

Un quatrième paramètre peut être ajouté : la largeur maximale que le texte doit utiliser.

Application 5

Écrire un texte rempli de 30px de taille et la police « Arial » à l'aide de fillText() sur le Canvas