|  |
| --- |
| Laboratoire 1: les dessins 2D – partie 1 |

Théorie

1. Les dessins en HTML5

Un des grands avantages du **HTML5,** c’est que **HTML5** donne la possibilité au programmeur de dessiner.

La balise qui donne la possibilité de dessiner est la balise ***<canvas></canvas>***.

arc

Au point de départ, c’est un conteneur vide. Mais il est possible d’utiliser plusieurs fonctions graphiques pour dessiner à l’intérieur de ce canevas et pour créer des animations. Ces animations peuvent être utilisées pour créer des jeux 2D et 3D interactifs.

Il existe plusieurs avantages d’utiliser la balise ***<canvas></canvas>.***

* Cette balise a été standardisée par le *W3C*. Elle fonctionne sur tous les navigateurs qui supportent cette balise.
* Elle s’accompagne très bien avec les autres standards Web (*XHTML*, *CSS*, *Javascript*). Ce n’est donc pas nécessaire d’ajouter un **plug-in** pour visualiser les dessins et les animations (par exemple, pour visualiser des animations *Flash* cela requiert un **plug-in**).
* Elle est performante et accélérée matériellement par la majorité des navigateurs.
* Elle fonctionne sur tous les mobiles.

Le seul vrai inconvénient, c’est que les nombreuses fonctions graphiques de cette balise sont très primitives. Créer un jeu requiert beaucoup de programmation.

Par contre, il existe une multitude de librairies graphiques qui aident le programmeur à développer plus rapidement : **Three**, **RGraph**, **CreateJS**, **EaseJS**, **Construct2**, **PlayCanvas**, **Quintus**, **ImpactJS**, etc.

Dans ce cours, nous n’allons pas utiliser ces librairies.

2. Création et accès au canevas

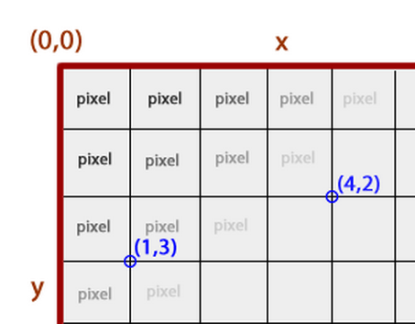
La balise ***<canvas></canvas>*** possède deux attributs qui déterminent sa taille : ***width*** et ***heigth***.

Observez, qu’à l’intérieur de la balise ***<canvas></canvas>,*** il y a du texte. Cela est utile lorsque le navigateur ne supporte pas la balise ***<canvas></canvas>.***

|  |
| --- |
| ***<canvas id="monCanvas" width="350" height="350">***  ***Votre navigateur ne supporte pas la balise canvas***  ***</canvas>*** |

Pour dessiner, il faut utiliser du ***Javascript***.

|  |
| --- |
| ***var objCanvas = document.getElementById('monCanvas');***  ***var objC2D = objCanvas.getContext('2d');*** |



L’objet le plus important est le contexte 2D (***objC2D***). C’est à l’intérieur de ce contexte que sont situées les fonctions de la librairie graphique 2D. Le contexte représente également la surface du dessin.

Prenez note que les coordonnées du contexte sont exprimées en pixels et que le point de référence (0,0) est situé en haut à gauche du contexte.

Prenez note qu’il est possible de dessiner à l’extérieur de cette surface mais ce qui est dessiné est invisible pour l’utilisateur. Par exemple, le point (-1,0) est dessiné à l’extérieur du canevas.

Il n’est pas nécessaire que les coordonnées soient des nombres entiers. Étant donné que chaque point représente un pixel alors chaque coordonnée sera arrondie ou interpolée à l’affichage.

3. Les tracés

La base du 2D, ce sont les tracés. Pour définir un nouveau tracé, on utilise toujours la méthode ***.beginPath()***.

Pour déplacer « le crayon à main levée» au début du tracé, on utilise la méthode ***.moveTo(intX, intY)*** où ***intX*** est la coordonnée sur l’axe des X et où ***intY*** est la coordonnée sur l’axe des Y.

Pour tracer un segment en ligne droite à l’intérieur du tracé, on utilise la méthode ***.lineTo(intX, intY).***

Un tracé peut se terminer par la méthode ***.closePath()***. Cela va faire en sorte que le début va rejoindre la fin. Si on ne veut pas que le début rejoigne la fin, on ne doit pas utiliser la méthode ***.closePath().***

Le tracé se dessine réellement lorsqu’on appelle la méthode ***.stroke()*** pour dessiner le contour du tracé et/ou la méthode ***.fill()*** pour dessiner l’intérieur du tracé.

Voici quelques méthodes et propriétés sur les tracés.

**.beginPath()** : Pour définir un nouveau tracé.

**.moveTo(intX,intY)** : Pour déplacer « le crayon à main levée » au début du tracé.

**.lineTo(intX,intY)** : Pour tracer un segment en ligne droite à l’intérieur du tracé.

**.closePath()** : Lorsqu’on désire que le début et la fin du tracé se rejoignent

**.lineWidth** : La largeur d’un segment (par défaut 1 pixel)

.**lineCap** : Le style de fin de segment (par défaut plat)

**.lineJoin** : Le style de rencontre entre 2 segments

**.setLineDash** : Le style de segment (par défaut plein)

**.strokeStyle** : Le style de couleur pour le contour du tracé (par défaut noir)

**.fillStyle** : Le style de couleur pour le remplissage du tracé (par défaut noir)

**.stroke() :** Pour dessiner le contour du tracé

**.fill() :** Pour dessiner le remplissage du tracé (pour remplir le tracé)

**NOTE IMPORTANTE** : Tous les exemples ***.htm*** sont situés dans le dossier **Théorie 4.**

L’exemple suivant dessine un triangle plein bleu sans contour puis un rectangle plein jaune avec un contour noir.

|  |  |
| --- | --- |
| **Tracé 1.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Le triangle plein sans contour***  ***objC2D.beginPath(); // Définir un nouveau tracé***  ***objC2D.moveTo(50,80); // Le crayon au point (50,80)***  ***objC2D.lineTo(100,230); // Un segment droit vers le point (100,230)***  ***objC2D.lineTo(250,230); // Un segment droit vers le point (250,230)***  ***objC2D.closePath(); // Fermer le tracé***  ***objC2D.fillStyle = 'blue'; // Style de couleur du remplissage***  ***objC2D.fill(); // Remplir le tracé***  ***// Le rectangle plein avec contour***  ***objC2D.beginPath(); // Un autre tracé***  ***objC2D.moveTo(50,250); objC2D.lineTo(50,300);***  ***objC2D.lineTo(250,300); objC2D.lineTo(250,250);***  ***objC2D.closePath(); // Fermer le tracé***  ***objC2D.fillStyle = 'yellow'; // Style de couleur de remplissage***  ***objC2D.strokeStyle = 'black'; // Style de couleur du contour***  ***objC2D.lineWidth = 5; // Largeur de la ligne du contour***  ***objC2D.fill(); // Remplir le tracé***  ***objC2D.stroke(); // Dessiner le contour du tracé*** |  |

L’exemple suivant dessine un segment de droite avec des bouts arrondis puis un rectangle vide avec des coins arrondis.

|  |  |
| --- | --- |
| **Tracé 2.htm** |  |
| ***…***  ***// Un segment de droite avec bouts arrondis***  ***objC2D.beginPath(); // Définir un nouveau tracé***  ***objC2D.moveTo(50, 50); objC2D.lineTo(250, 50);***  ***objC2D.lineCap = 'round'; // Bout arrondi***  ***objC2D.strokeStyle = 'black'; // Couleur du segment***  ***objC2D.lineWidth = 20; // Largeur du segment***  ***objC2D.stroke(); // Dessiner le segment***    ***// Le rectangle vide avec les coins arrondis***  ***objC2D.beginPath(); // Définir un nouveau tracé***  ***objC2D.moveTo(50, 250); objC2D.lineTo(50, 300);***  ***objC2D.lineTo(250, 300); objC2D.lineTo(250, 250);***  ***objC2D.closePath(); // Fermer le tracé***  ***objC2D.strokeStyle = 'black'; // Style de couleur des segments***  ***objC2D.lineWidth = 20; // Largeur des segments***  ***objC2D.lineJoin = 'round' // Coins arrondis***  ***objC2D.stroke(); // Dessiner le contour*** |  |

L’exemple suivant dessine 4 segments de droite :

* Le premier segment est un segment constitué de 5 pixels pleins puis de 5 pixels vides à l’infini.
* Le deuxième segment est un segment constitué de 10 pixels pleins puis de 5 pixels vides à l’infini.
* Le troisième segment est un segment constitué de 10 pixels pleins puis de 5 pixels vides puis de 5 pixels pleins puis de 10 pixels vides à l’infini
* Le quatrième segment est un segment plein

|  |  |
| --- | --- |
| **Tracé 3.htm** |  |
| ***…***  ***// Style: 5 pleins, 5 vides, 5 pleins, 5 vides, ...***  ***objC2D.beginPath(); // Définir un nouveau tracé***  ***objC2D.moveTo(50, 50); objC2D.lineTo(250, 50);***  ***objC2D.setLineDash([5]); // Style de segment***  ***objC2D.stroke(); // Dessiner le segment***    ***// Style: 10 pleins, 5 vides, 10 pleins, 5 vides, ...***  ***objC2D.beginPath(); // Définir un nouveau tracé***  ***objC2D.moveTo(50, 100); objC2D.lineTo(250, 100);***  ***objC2D.setLineDash([10,5]); // Style de segment***  ***objC2D.stroke(); // Dessiner le segment***  ***// Style: 10 pleins, 5 vides, 5 pleins, 10 vides, ...***  ***objC2D.beginPath(); // Définir un nouveau tracé***  ***objC2D.moveTo(50, 150); objC2D.lineTo(250, 150);***  ***objC2D.setLineDash([10,5,5,10]); // Style de segment***  ***objC2D.stroke(); // Dessiner le segment***  ***// Segment plein***  ***objC2D.beginPath(); // Définir un nouveau tracé***  ***objC2D.moveTo(50, 200); objC2D.lineTo(250, 200);***  ***objC2D.setLineDash([]); // Style de segment***  ***objC2D.stroke(); // Dessiner le segment*** |  |

4. Les tracés rectangulaires

Pour définir un tracé rectangulaire, il est possible d’utiliser la méthode **.rect*(intX,intY, intLargeur, intHauteur)*.** Les deux premiers paramètres sont la position ***(intX, intY)*** du rectangle et les deux derniers paramètres sont la taille ***(intLargeur, intHauteur*)** du rectangle.

|  |  |
| --- | --- |
| **Rectangle 1.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***objC2D.beginPath(); // Définir un tracé***  ***objC2D.rect(50,50, 200,100); // Un tracé rectangulaire***  ***objC2D.fillStyle = 'purple'; // Couleur de remplissage***  ***objC2D.fill(); // Remplissage du tracé*** |  |

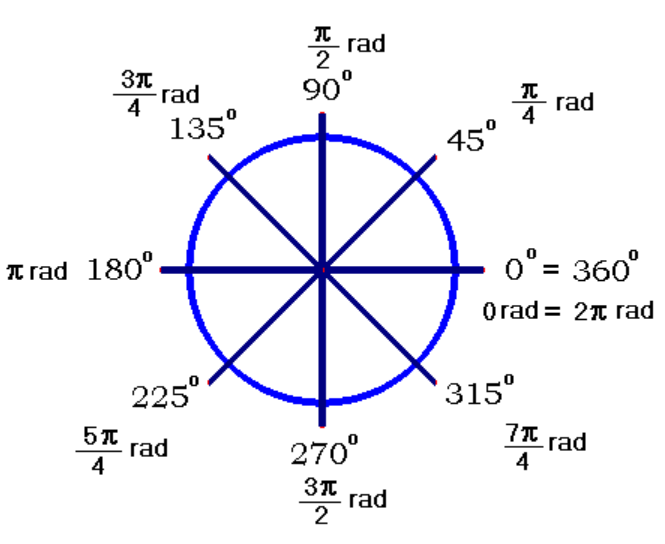
Il est possible de définir un tracé rectangulaire et de le remplir en une seule instruction. Pour ce faire, il faut utiliser la méthode ***.fillRect(intX,intY, intLargeur,intHauteur)***.

|  |  |
| --- | --- |
| **Rectangle 2.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***objC2D.fillStyle = 'purple'; // Couleur de remplissage***  ***objC2D.fillRect(50,50, 200,100); // Remplir le rectangle*** |  |

Prenez note que la méthode ***.strokeRect(intX,intY, intLargeur,intHauteur)*** est similaire à la méthode ***.fillRect(intX,intY, intLargeur,intHauteur)*** sauf qu’elle trace un rectangle vide (seulement le contour).

Finalement, prenez note que la méthode ***.clearRect(intX,intY, intLargeur,intHauteur)*** est similaire à la méthode ***.fillRect(intX,intY, intLargeur,intHauteur)*** sauf qu’elle efface tous les pixels qu’il y a dans le rectangle. Cette méthode est très utilisée lorsqu’on fait de l’animation.

5. Les tracés curvilignes (les courbes)

En 2D, une courbe est un tracé. Le tracé de la courbe doit donc débuter par la méthode ***.beginPath()*** et, possiblement, se terminer par la méthode ***.closePath()*** si on veut que le début et la fin du tracé se rejoignent. Il existe plusieurs types de courbes.

La méthode la plus utilisée est la méthode .***arc(intXCentre, intYCentre, intRayon, fltAngle1, fltAngle2, binSensAntiHoraire)*.** Cette méthode définit un tracé qui a la forme d’un arc de cercle.

***(intXCentre, intYCentre)*** est le centre du cercle; ***intRayon*** est le rayon du cercle; ***(fltAngle1, fltAngle2)*** sont les 2 angles en radians qui déterminent le début et la fin de l’arc par rapport au cercle; ***binSensAntiHoraire*** doit être **true** si l’arc doit se dessiner dans le sens antihoraire.

Prenez note qu’il faut exprimer les angles en radians et non pas en degrés. Pour faire la conversion, il faut appliquer la formule suivante : π radians = 180 degrés ou 1 radian = 180 degrés/π radians.

Voici un arc de cercle de rayon 100 dessiné dans le sens antihoraire.

|  |  |
| --- | --- |
| **Arc 1.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***objC2D.strokeStyle = 'coral';***  ***objC2D.lineWidth = 5;***  ***objC2D.beginPath();***  ***objC2D.arc(150,150,100,0,Math.PI,true);***  ***objC2D.stroke();*** |  |

Voici un arc de cercle de rayon 100 dessiné dans le sens horaire.

|  |  |
| --- | --- |
| **Arc 2.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***objC2D.strokeStyle = 'coral';***  ***objC2D.lineWidth = 5;***  ***objC2D.beginPath();***  ***objC2D.arc(150,150,100,0,Math.PI,false);***  ***objC2D.stroke();*** |  |

Voici un cercle complet avec un contour noir et un remplissage jaune.

|  |  |
| --- | --- |
| **Cercle.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***objC2D.strokeStyle = 'black';***  ***objC2D.fillStyle = 'yellow';***  ***objC2D.lineWidth = 5;***  ***objC2D.beginPath();***  ***objC2D.arc(150,150,100,0,2\*Math.PI,false);***  ***objC2D.stroke();***  ***objC2D.fill();*** |  |

Une autre méthode utilisée est la méthode .***ellipse(intXCentre, intYCentre, intRayonX, intRayonY, fltRotation, fltAngle1, fltAngle2, binSensAntiHoraire)*.** Cette méthode définit un tracé qui a la forme d’une ellipse.

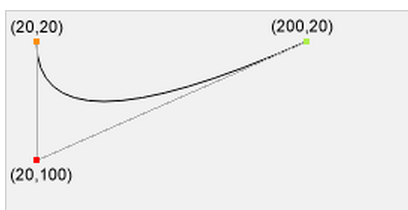
***(intXCentre, intYCentre)*** est le centre de l’ellipse; (***intRayonX, intRayonY)*** sont les rayons horizontal et vertical de l’ellipse; ***fltRotation*** est l’angle de rotation que l’on doit appliquer à l’ellipse avant de le dessiner; ***(fltAngle1, fltAngle2)*** sont les 2 angles en radians qui déterminent le début et la fin du dessin de l’ellipse; ***binSensAntiHoraire*** doit être **true** si l’ellipse doit se dessiner dans le sens antihoraire.

Voici une portion d’ellipse de rayon (100, 50) dessinée dans le sens antihoraire. Observez que l’ellipse n’a subi aucune rotation.

|  |  |
| --- | --- |
| **Ellipse 1.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***objC2D.strokeStyle = 'coral';***  ***objC2D.lineWidth = 5;***  ***objC2D.beginPath();***  ***objC2D.ellipse(150,150, 100,50, 0, 0,Math.PI, true);***  ***objC2D.stroke();*** |  |

Voici une ellipse complète de rayon (100, 50) dessinée dans le sens antihoraire. Observez que l’ellipse a subi une rotation de 45 degrés (π/4 radians).

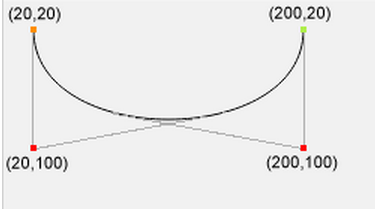
|  |  |
| --- | --- |
| **Ellipse 2.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***objC2D.strokeStyle = 'coral';***  ***objC2D.fillStyle = 'yellow';***  ***objC2D.lineWidth = 5;***  ***objC2D.beginPath();***  ***objC2D.ellipse(150,150, 100,50, Math.PI/4, 0,2 \* Math.PI,true);***  ***objC2D.fill();***  ***objC2D.stroke();*** |  |

Une autre méthode utilisée est la méthode ***.quadraticCurveTo(intXControl, intYControl, intXFin, intYFin)*** pour dessiner une courbe de Bézier quadratique.

Une courbe de Bézier quadratique est formée de deux points d’attache et d’un point de contrôle.

|  |  |
| --- | --- |
| **Bézier quadratique.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***objC2D.strokeStyle = 'black';***  ***objC2D.lineWidth = 5;***  ***objC2D.beginPath();***  ***objC2D.moveTo(20,20); // Point d'attache #1***  ***objC2D.quadraticCurveTo(20,100,200,20); // Point de contrôle et point d’attache #2***  ***objC2D.stroke();*** |  |

Les points d’attache sont faciles à déterminer. Par contre, le point de contrôle est plus difficile à déterminer. Vous pouvez imaginer une courbe de Bézier quadratique comme étant un segment de droite dont les points d’attache sont reliés à une ficelle. En tirant sur la ficelle, le segment de droite se déforme ce qui donne une courbe.

Une autre méthode utilisée est la méthode ***.bezierCurveTo(intXControl1, intYControl1, intXControl2, intYControl2, intXFin, intYFin)*** pour dessiner une courbe de Bézier cubique.

Une courbe de Bézier cubique est formée de deux points d’attache et de deux points de contrôle.

|  |  |
| --- | --- |
| **Bézier cubique.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***objC2D.strokeStyle = 'black';***  ***objC2D.lineWidth = 5;***  ***objC2D.beginPath();***  ***objC2D.moveTo(20,20); // Point d'attache 1***  ***objC2D.bezierCurveTo(20,100,200,100,200,20);***  ***objC2D.stroke();*** |  |

Les courbes de Bézier sont des outils puissants car elles nous donnent la possibilité de dessiner des contours relativement complexes.

Ici, un nuage a été dessiné en utilisant un tracé composé d’une suite de courbes de Bézier cubiques.

|  |  |
| --- | --- |
| **Nuage.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***objC2D.beginPath();***  ***objC2D.moveTo(70, 80);***  ***objC2D.bezierCurveTo(30, 100, 30, 150, 130, 150);***  ***objC2D.bezierCurveTo(150, 180, 220, 180, 240, 150);***  ***objC2D.bezierCurveTo(320, 150, 320, 120, 290, 100);***  ***objC2D.bezierCurveTo(330, 40, 270, 30, 240, 50);***  ***objC2D.bezierCurveTo(220, 5, 150, 20, 150, 50);***  ***objC2D.bezierCurveTo(100, 5, 50, 20, 70, 80);***  ***objC2D.closePath();***  ***objC2D.lineWidth = 5;***  ***objC2D.fillStyle = '#8ED6FF';***  ***objC2D.fill();***  ***objC2D.strokeStyle = 'blue';***  ***objC2D.stroke();*** |  |

6. Remarques sur les couleurs

Jusqu’à date, nous avons affecté les couleurs en utilisant la forme littérale mais, en réalité, il existe au moins trois (3) manières d’affecter une couleur :

* En utilisant un nom de couleur déjà définie (dans sa forme littérale) : ***objC2D.fillStyle = 'yellow'***
* En utilisant le code de la couleur**: *objC2D.fillStyle = '#ffff00'*** (cela correspond à la couleur jaune)
* En utilisant **rgb : *objC2D.fillStyle = 'rgb(255,255,0)'*** (cela correspond à la couleur jaune)

Pour voir les noms des couleurs qui ont déjà été définies, consultez la page web <http://www.w3schools.com/html/html_colornames.asp>.

7. Le dessin d’un texte

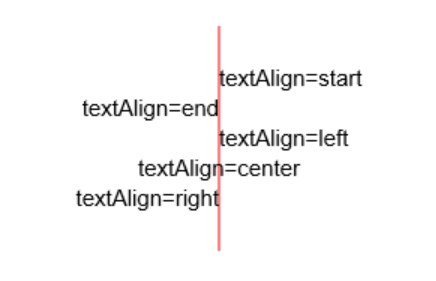
En 2D, il est possible de dessiner du texte. Il existe plusieurs propriétés et méthodes pour dessiner du texte.

***.font :*** Propriété qui détermine la police de caractères utilisée.

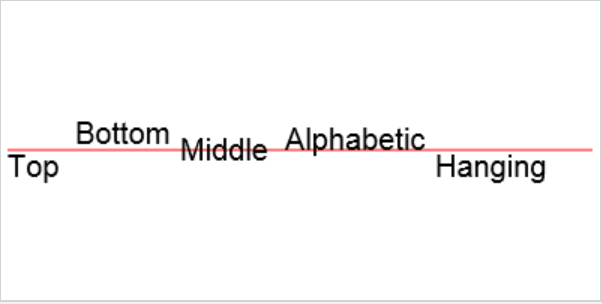
***.fillText(strTexte, intX, intY)*** : Méthode pour dessiner du texte avec la couleur de remplissage à la position ***(intX,intY)***.

***.strokeText(strTexte, intX, intY)*** : Méthode pour dessiner le contour du texte avec la couleur du contour à la position ***(intX,intY)***.

***.textAlign*** : Propriété qui détermine à partir d’où le texte va se dessiner horizontalement.



***.textBaseLine*** : Propriété qui détermine à partir d’où le texte va se dessiner verticalement.

******

.***measureText(strTexte)*** Méthode pour mesurer (en pixels) la largeur du texte qui va être dessiné.

Dans l’exemple suivant, le texte est dessiné à la même position en X. Par contre, l’alignement horizontal est différent.

|  |  |
| --- | --- |
| **Texte.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***var strTexte = 'Allo';***  ***objC2D.fillStyle = 'yellow';***  ***objC2D.strokeStyle = 'black';***  ***objC2D.font = '30pt Verdana';***  ***// Dessiner de la gauche vers la droite (par défaut)***  ***objC2D.textAlign = 'left';***  ***objC2D.fillText(strTexte, objCanvas.width/2,40);***  ***objC2D.strokeText(strTexte, objCanvas.width/2,40);***  ***// Dessiner à partir du centre***  ***objC2D.textAlign = 'center';***  ***objC2D.fillText(strTexte,objCanvas.width/2,80);***  ***objC2D.strokeText(strTexte,objCanvas.width/2,80);***  ***// Dessiner de la droite vers la gauche***  ***objC2D.textAlign = 'right';***  ***objC2D.fillText(strTexte,objCanvas.width/2, 120);***  ***objC2D.strokeText(strTexte,objCanvas.width/2,120);*** |  |

**Attention** : Verticalement, par défaut, le texte se dessine du bas vers le haut (***alphabetic***). Il est possible de modifier ce comportement en utilisant la propriété ***.textBaseLine***. Par exemple, ***objC2D.textBaseline='top'*** va dessiner le texte du haut vers le bas. Par exemple, ***objC2D.textBaseline='middle'*** va dessiner le texte à partir du milieu verticalement.

Il existe une méthode qui détermine la largeur totale du texte en pixels : la méthode .***measureText(strTexte)***. Cela peut être utile à l’occasion de connaître cette largeur.

|  |
| --- |
| ***var objC2D = objCanvas.getContext('2d');***  ***var strTexte = 'Allo';***  ***objC2D.fillStyle = 'yellow';***  ***objC2D.strokeStyle = 'black';***  ***objC2D.font = '30pt Verdana';***  ***var objTailleTexte = objC2D.measureText(strTexte);***  ***var intLargeur = objTailleTexte.width; // La largeur du texte Allo***  ***// On dessine le texte***  ***…*** |

8. Le dessin d’une image

En 2D, il est possible de dessiner une image grâce à la méthode ***.drawImage(objImage, intX, intY, intLargeur, intHauteur)*** où ***objImage*** est l’image, où ***(intX,intY)*** est la position (coin supérieur gauche) de l’image à l’intérieur du canevas et où ***(intLargeur, intHauteur)*** est la taille de l’image dessinée.

En l’absence de la hauteur et de la largeur, l’image est dessinée dans sa taille originale.

L’image n’est pas dessinée de manière automatique. Habituellement, on dessine l’image dans l’événement ***onload*** (c’est-à-dire dès que le chargement de l’image est terminé). De cette façon, on s’assure que l’image est complètement chargée avant de dessiner cette image.

|  |  |
| --- | --- |
| **Image 1.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***var objImage = new Image(); // Créer l’image***  ***objImage.onload = function()***  ***{ // L’image est dessinée lorsque le chargement est terminé***  ***objC2D.drawImage(objImage, 0, 0);***  ***}***  ***objImage.src = 'prof.jpg'; // Le fichier qui contient l’image*** |  |

Le problème ici, c’est que l’image n’est pas complètement dessinée car la taille originale de l’image dépasse la taille du canevas. Une partie de l’image est invisible.

Une manière de résoudre ce problème, c’est de lui donner une taille (ici, 300X300).

|  |  |  |
| --- | --- | --- |
| **Image 2.htm** |  | |
| ***var objC2D = objCanvas.getContext('2d');***  ***var objImage = new Image(); // Créer l’image***  ***objImage.onload = function()***  ***{ // L’image est dessinée lorsque le chargement est terminé***  ***objC2D.drawImage(objImage, 0, 0, 300, 300);***  ***}***  ***objImage.src = 'prof.jpg'; // Le fichier qui contient l’image*** |  | |
| Le problème ici, c’est que l’image n’a pas conservé ses proportions. | | |

Une manière de résoudre ce problème, c’est de lui donner une proportion à partir de la taille originale de l’image (ici 50% de la taille originale de l’image).

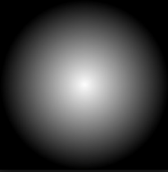
|  |  |  |
| --- | --- | --- |
| **Image 3.htm** |  | |
| ***var objC2D = objCanvas.getContext('2d');***  ***var objImage = new Image(); // Créer l’image***  ***objImage.onload = function()***  ***{ // L’image est dessinée lorsque le chargement est terminé***  ***objC2D.drawImage(objImage, 0, 0, 300, 300,***  ***objImage.width \* 0.50, objImage.height \* 0.50);***  ***}***  ***objImage.src = 'prof.jpg'; // Le fichier qui contient l’image*** |  | |
|  | | |

9. Les styles de couleurs

Pour l’instant, pour les contours et pour le remplissage, nous avons choisi un style de couleur uniforme mais il est possible de choisir d’autres styles de couleurs.

Un de ces styles est le dégradé. Il existe deux (2) types de dégradés : le dégradé linéaire (le long d’un axe) et le dégradé radial (à partir d’un centre).

Pour créer un dégradé linéaire, il faut utiliser la méthode ***.createLinearGradient(intX1,intY1, intX2,intY2)*** où ***(intX1,intY1)*** est le point de départ du dégradé et où ***(intX2,intY2)*** est le point d’arrivée du dégradé.

Pour créer un dégradé radial, il faut utiliser la méthode ***.createRadialGradient(intX1,intY1,intR1, intX2,intY2,intR2)*** où ***(intX1,intY1,intR1)*** est le cercle de départ du dégradé et où ***(intX2,intY2,intR2)*** est le cercle d’arrivée du dégradé.

Par la suite, lorsque le dégradé a été créé, il faut déterminer des couleurs d’arrêt en utilisant la méthode ***.addColorStop(fltPc, strCouleur).***

Par exemple, ici, à 0% du dégradé, la couleur va être bleue, à 50% du dégradé, la couleur va être jaune et à 100% du dégradé la couleur va être rouge.

***objDegrade.addColorStop(0,'blue'); objDegrade.addColorStop(0.5,'yellow'); objDegrade.addColorStop(1,'red');***

Finalement, il faut appliquer le dégradé au remplissage et/ou au contour. Voici un exemple d’un dégradé linéaire.

|  |  |
| --- | --- |
| **Dégradé linéaire.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***var objDegrade=objC2D.createLinearGradient(0,0, 170,170);***  ***objDegrade.addColorStop(0, 'blue');***  ***objDegrade.addColorStop(0.5, 'yellow');***  ***objDegrade.addColorStop(1, 'red');***  ***objC2D.fillStyle=objDegrade;***  ***objC2D.fillRect(0,0,170,170);*** |  |

Voici un exemple d’un dégradé radial dans lequel le cercle de départ est le cercle (85,85,0) et le cercle d’arrivée est le cercle (85,85,85).

|  |  |
| --- | --- |
| **Dégradé radial 1.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***var objDegrade=objC2D.createRadialGradient(85,85,0, 85,85,85);***  ***objDegrade.addColorStop(0,'blue');***  ***objDegrade.addColorStop(0.5, 'yellow');***  ***objDegrade.addColorStop(1, 'red');***  ***objC2D.fillStyle=objDegrade;***  ***objC2D.fillRect(0,0,170,170);*** |  |

Voici un exemple d’un dégradé radial dans lequel le cercle de départ est le cercle (85,85,30) et le cercle d’arrivée est le cercle (85,85,85). Le cercle de départ est plus gros que le précédent.

|  |  |
| --- | --- |
| **Dégradé radial 2.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***var objDegrade=objC2D.createRadialGradient(85,85,30, 85,85,85);***  ***objDegrade.addColorStop(0,'blue');***  ***objDegrade.addColorStop(0.5, 'yellow');***  ***objDegrade.addColorStop(1, 'red');***  ***objC2D.fillStyle=objDegrade;***  ***objC2D.fillRect(0,0,170,170);*** |  |

Voici un exemple d’un dégradé radial dans lequel le cercle de départ est le cercle (85,85,0) et le cercle d’arrivée est le cercle (105,85,85). Le cercle d’arrivée est un peu décalé vers la droite.

|  |  |
| --- | --- |
| **Dégradé radial 2.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***var objDegrade=objC2D.createRadialGradient(85,85,0, 105,85,85);***  ***objDegrade.addColorStop(0,'blue');***  ***objDegrade.addColorStop(0.5, 'yellow');***  ***objDegrade.addColorStop(1, 'red');***  ***objC2D.fillStyle=objDegrade;***  ***objC2D.fillRect(0,0,170,170);*** |  |

Il est important de prendre note que les dégradés s’appliquent à tous les objets qui ont du remplissage et/ou des contours. Voici, par exemple, du texte qui a été rempli à l’aide d’un dégradé linéaire.

|  |  |
| --- | --- |
| **Texte dégradé.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***var strTexte = 'Informatique';***  ***objC2D.font = '60px Arial';***  ***var objTailleTexte = objC2D.measureText(strTexte);***  ***var intLargeurTexte = objTailleTexte.width; // La largeur du texte***  ***var objDegrade=objC2D.createLinearGradient(0,0, intLargeurTexte, 60);***  ***objDegrade.addColorStop(0, 'blue');***  ***objDegrade.addColorStop(0.5, 'yellow');***  ***objDegrade.addColorStop(1, 'red');***  ***objC2D.fillStyle=objDegrade;***  ***objC2D.textAlign = "left";***  ***objC2D.fillText(strTexte, 0, 60);*** |  |

Un autre style est le motif. Un motif est une image que l’on utilise pour dessiner le contour et/ou le remplissage d’un dessin.

La méthode utilisée pour créer un motif est ***.createPattern(objImage,strRépétition)*** où **strRépétition** peut avoir une des valeurs suivantes : ***repeat***, ***repeat-x***, ***repeat-y*** et ***no-repeat***.

Une fois que le motif a été créé, il faut appliquer ce motif au remplissage et/ou au contour.

Voici le motif de la page précédente qui se répète dans un rectangle.

|  |  |
| --- | --- |
| **Motif.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***var objImage = new Image();***  ***objImage.src = 'motif.png'***  ***objImage.onload = function() {***  ***objMotif= objC2D.createPattern(objImage, 'repeat');***  ***objC2D.fillStyle = objMotif;***  ***objC2D.fillRect(0,0,300,300);***  ***}*** |  |

Il est important de prendre note que les motifs s’appliquent à tous les objets qui ont du remplissage et/ou des contours. Voici, par exemple, du texte qui a été rempli à l’aide du motif de la page précédente.

|  |  |
| --- | --- |
| **Texte en motif.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***var objImage = new Image();***  ***objImage.src = 'motif.png'***  ***var strTexte = '420-4P6';***  ***objC2D.font = '80px Arial';***  ***objImage.onload = function() {***  ***objMotif= objC2D.createPattern(objImage, 'repeat');***  ***objC2D.fillStyle = objMotif;***  ***objC2D.textAlign = "left";***  ***objC2D.fillText(strTexte, 0, 80);***  ***}*** |  |

10. Références

<http://dev.w3.org/2006/canvas-api/canvas-2d-api.html>

<http://www.w3schools.com>

<http://www.alsacreations.com/tuto/lire/1484-introduction.html>

<http://cheatsheetworld.com/programming/html5-canvas-cheat-sheet/>

<http://www.html5canvastutorials.com/>