

PROGRAMMATION WEB AVANCEE

TP4

CANVAS

Application 1 :

En utilisant les concepts déjà vus, créer l'image ci-dessous



Exercice 1 : alpha, scale, rotate, translate

Alpha :

Pour faire un dessin en transparence, on modifie la propriété *globalAlpha* du contexte

context.globalAlpha = 0.5;

Exemple :

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="300" height="150" style="border:1px solid #d3d3d3;">
Votre navigateur ne supporte pas les canvas  HTML5</canvas>
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.fillStyle = "red";
ctx.fillRect(20, 20, 75, 50);
```

```
ctx.globalAlpha = 0.2;
ctx.fillStyle = "blue";
ctx.fillRect(50, 50, 75, 50);
ctx.fillStyle = "green";
ctx.fillRect(80, 80, 75, 50);
</script>
</body>
</html>
```

La translation

Pour effectuer une translation, on utilise la méthode :

context.translate(translateX, translateY);

Où translateX est le déplacement sur l'axe des x (en pixels) et translateY la même chose mais sur l'axe des y.

Exemple :

```
<!DOCTYPE html>
<html>
<head>
<title>HTML5 Canvas - translate</title>
</head>
<body>
<canvas id="DemoCanvas" width="300" height="400"></canvas>
<script>
var canvas = document.getElementById("DemoCanvas");
if (canvas.getContext)
{
var ctx = canvas.getContext('2d');
ctx.beginPath();
ctx.lineWidth = "3";
ctx.strokeStyle = "blue";
ctx.strokeRect(60, 60, 160, 160);
//déplacement de (0,0) à (50,50)
```

```
ctx.translate(60, 60);
ctx.strokeStyle = "red";
ctx.strokeRect(60, 60, 160, 160);
ctx.stroke();
}
</script>
</body>
</html>
```

Le scale :

Pour effectuer un changement d'échelle, on utilise la méthode :

context.scale(scaleX, scaleY);

Où *scaleX* est l'échelle sur l'axe des x que vous souhaitez obtenir et *scaleY* la même chose mais sur l'axe des y.

Exemple :

```
<!DOCTYPE html>

<html>

<body>

<canvas id="myCanvas" width="300" height="150" style="border:1px solid
#d3d3d3;">

Votre navigateur ne supporte pas les canvas HTML5.</canvas>

<script>

var c = document.getElementById("myCanvas");

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

ctx.strokeRect(5, 5, 25, 15);

ctx.scale(2, 2);

ctx.strokeRect(5, 5, 25, 15);

</script>
```

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

La rotation :

Rappel :

L'unité de mesure employée pour une rotation en programmation graphique est le radian.

La formule de conversion degrés/radians : $\text{angle_radians} = \text{angle_degré} * (\text{Math.PI} / 180)$;

La méthode qui vous permettra d'appliquer une rotation à des objets :

context.rotate(angle_radian);

Exemple :

```
<!DOCTYPE html>
<html>
<body>
<canvas id="myCanvas" width="300" height="150" style="border:1px solid #d3d3d3;">
Votre navigateur ne supporte pas les canvas the HTML5.</canvas>
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.rotate(20 * Math.PI / 180);
ctx.fillRect(50, 20, 100, 50);
</script>
</body>
</html>
```

Cumul des transformations, sauvegarde et restauration du contexte

L'objet context utilise une **matrice** pour représenter et stocker le résultat de toutes les transformations qu'on lui applique.

Remarque 1 : Les transformations que l'on applique à une matrice se cumulent et l'ordre dans lequel on les exécute influe sur le résultat obtenu.

Exemple 1 :

```
<!DOCTYPE html>

<html>

<body>

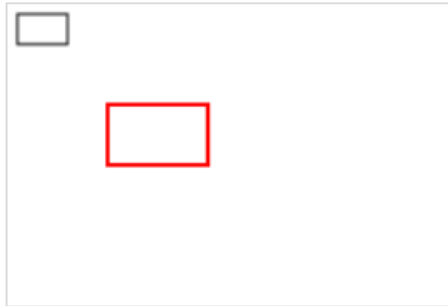
<canvas id="myCanvas" width="300" height="150" style="border:1px solid #d3d3d3;">
Your browser does not support the HTML5 canvas tag.</canvas>

<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.strokeRect(5, 5, 25, 15);
//scale puis translate
ctx.scale(2, 2);
ctx.translate(20, 20);
ctx.strokeStyle = "red";
ctx.strokeRect(5, 5, 25, 15);
ctx.stroke();
</script>

</body>

</html>
```

Résultat à l'écran :



Exemple 2

```
<!DOCTYPE html>

<html>

<body>

<canvas id="myCanvas" width="300" height="150" style="border:1px solid #d3d3d3;">
Your browser does not support the HTML5 canvas tag.</canvas>

<script>

var c = document.getElementById("myCanvas");

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

ctx.strokeRect(5, 5, 25, 15);

//translate puis scale

ctx.translate(20, 20);

ctx.scale(2, 2);

ctx.strokeStyle = "red";

ctx.strokeRect(5, 5, 25, 15);

ctx.stroke();
```

```
</script></body>

</html>
```

Résultat à l'écran :



On peut voir que le résultat obtenu à l'écran est différent suivant que l'on applique **scale avant** ou **après** la translation

Remarque 2 : Si je veux définir une échelle à 0,5 après avoir l'avoir définie à 2, le code suivant ne fonctionne pas :

```
context.scale( 2, 2 ); // l'échelle est à 2
context.scale( 0.5, 0.5 );
// ici l'échelle ne vaut pas 0.5 MAIS 1 car j'ai MULTIPLIE la valeur
// de l'échelle courante par 0.5, donc le résultat est 1, pour avoir une
// valeur d'échelle à 2, il aurait
// fallu que j'applique un scale de 0,25
```

Le problème, c'est que je ne suis pas forcément au courant de l'état actuel de ma matrice au moment où je l'utilise, donc cela peut me poser pas mal de problèmes pour obtenir l'état désiré.

Solution : la sauvegarde du contexte ! En effet, il est possible de stocker en mémoire l'état du contexte et de le restaurer par la suite. Cela fonctionne avec les méthodes suivantes :

context.save()

context.restore()

La méthode **context.save()** permet de sauvegarder l'état actuel du contexte, la méthode **context.restore()** permet de restituer l'état du dernier contexte sauvegardé, c'est-à-dire que les données de transformations de la matrice ainsi que les données de dessins etc. seront exactement les mêmes que lors du dernier appel à context.save().

Ces méthodes fonctionnent un peu à la manière d'une pile d'assiettes, c'est-à-dire que le dernier contexte sauvegardé ira « au-dessus » de la pile et donc, lors du prochain appel à context.restore() ce sera cette dernière « assiette » qui sera restituée.

Exemple :

```
<!DOCTYPE HTML>
<html>
  <head>

    <style>
      #test {
        width: 100px;
        height:100px;
        margin: 0px auto;
      }
    </style>
  </head>
  <body>
    <canvas id="mycanvas">Votre navigateur ne supporte ps les cnvas HTML5</canvas>
    <script type="text/javascript">
      var canvas = document.getElementById('mycanvas');
      var ctx = canvas.getContext('2d');
      ctx.fillStyle = 'red';
      ctx.fillRect(0,0,150,150);
      // Save the state
      ctx.save();
      // Make changes to the settings
      ctx.fillStyle = '#66FFFF'
      ctx.fillRect( 15,15,120,120);
      // Save the current state
      ctx.save();
```



```
// Make the new changes to the settings
ctx.fillStyle = '#993333'
ctx.globalAlpha = 0.5;
ctx.fillRect(30,30,90,90);

// Restore previous state
ctx.restore();

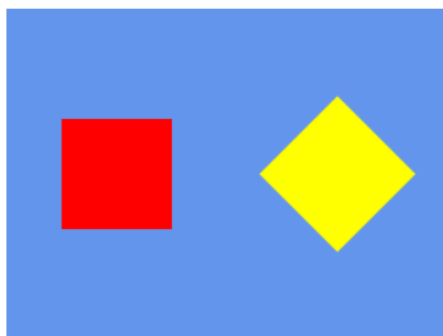
// Draw a rectangle with restored settings
ctx.fillRect(45,45,60,60);

// Restore original state
ctx.restore();

// Draw a rectangle with restored settings
ctx.fillRect(50,50,45,45);
</script>
</body>
</html>
```

Application 2

Créer l'application ci-dessous (utiliser save et restore)



Application 3

Créer l'application ci-dessous en utilisant save et restore

Ajouter en dessous les mêmes carrées avec une rotation



Exercice 3 : ombrage, composition, masque

Ombrage

Des effets d'ombrage sont prévus par Canvas, ce qui est une bonne nouvelle en soi, car les calculer pixel par pixel n'est pas une mince affaire. Quatre propriétés contrôlent le rendu de l'ombrage à partir de sa source.

Propriété	Rôle	Valeurs
<code>shadowOffsetX</code>	Étendue de l'ombrage sur l'axe horizontal (X)	Nombre entier, positif, négatif ou nul
<code>shadowOffsetY</code>	Étendue de l'ombrage sur l'axe vertical (Y)	Nombre entier, positif, négatif ou nul
<code>shadowBlur</code>	Valeur du flou	Nombre entier positif ou nul
<code>shadowColor</code>	Couleur de l'ombrage	Code couleur ou RGBA

Exemple

```
<!DOCTYPE HTML>

<html>

  <head>

    <style>

      canvas {

        border: 1px solid #2C2C2B; /* 1px border around canvas */

      }

    </style>

  </head>
```

```
<body>

  <canvas id="Canvas" width="600" height="250"></canvas>

<script>

  var canvas = document.getElementById('Canvas');

  var ctx = canvas.getContext('2d');

  // Configuration des ombrages pour le rectangle

  ctx.shadowOffsetX = 0;

  ctx.shadowOffsetY = 0;

  ctx.shadowBlur = 15;

  ctx.shadowColor = 'rgba(204,69,228,1)';

  // Tracé d'un rectangle

  ctx.fillStyle = '#fff';

  ctx.fillRect(10,10,150,150);

  // Tracé d'un autre rectangle

  ctx.fillStyle = '#9f0c9d';

  ctx.fillRect(50,50,70,70);

  // Configuration des ombrages pour le polygone

  ctx.shadowOffsetX = 0;

  ctx.shadowOffsetY = 20;

  ctx.shadowBlur = 10;

  ctx.shadowColor = 'rgba(0, 0, 0, 0.2)';

  // Tracés

  ctx.strokeStyle = '#2e6996';

  ctx.lineWidth = 5;

  ctx.fillStyle = '#b2d7f3';
```

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

Application 4

Créer le canvas ci-dessou :



Composition

La propriété ***globalCompositeOperation*** régit la façon dont sont menées les opérations de composition sur le canvas (définit ou renvoie comment une nouvelle image est dessinée sur une image existante), pour les tracés, les formes et les images, en plus de la transparence. Par défaut, la valeur ***source-over*** produit un effet de superposition, c'est-à-dire que la dernière forme tracée recouvre de façon opaque (hors modification de ***globalAlpha***) le contenu déjà présent. C'est l'effet que l'on retrouve de façon logique dans la plupart des programmes de dessin.

Cette propriété peut prendre plusieurs valeurs dont les résultats sont illustrés à l'aide du graphique ci-dessous (source: <http://www.html5canvastutorials.com/advanced/html5-canvas-global-composite-operations-tutorial/>) :



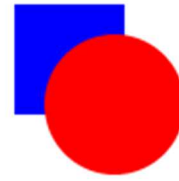
source-atop



source-in



source-out



source-over



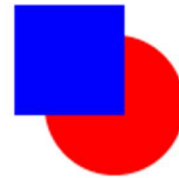
destination-atop



destination-in



destination-out



destination-over



lighter



darker



xor



copy

Exemple composition

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<canvas id="myCanvas" width="300" height="150" style="border:1px solid #d3d3d3;">
```

```
Your browser does not support the HTML5 canvas tag.</canvas>
```

```
<script>
```

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

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

```
ctx.fillStyle = "red";
```

```
ctx.fillRect(20, 20, 75, 50);
```

```
ctx.fillStyle = "blue";
ctx.globalCompositeOperation = "source-over";
ctx.fillRect(50, 50, 75, 50);
ctx.fillStyle = "red";
ctx.fillRect(150, 20, 75, 50);
ctx.fillStyle = "blue";
ctx.globalCompositeOperation = "destination-over";
ctx.fillRect(180, 50, 75, 50);
</script>
</body>
</html>
```

Masque

Un masque dissimule une région de la zone d’affichage, tout en révélant l’autre région.

La méthode utilisée est : ***clip()*** (Utilise le chemin courant comme masque.)

Remarque : la propriété ***globalCompositeOperation*** de l'objet context permet également de créer des masques

Exemple :

```
<!DOCTYPE html>

<html>

<body>

<canvas id="dessin" width="333" height="500"></canvas>

<script>

var moncanvas = document.getElementById('dessin');

var ctx = moncanvas.getContext('2d');

var img = new Image();
```

```
img.src = 'photo.jpg';

// Cercle de coordonnées 100,100

ctx.beginPath();

ctx.arc(100, 100, 150, 0, Math.PI * 2, false);

ctx.closePath();

// Masque sur le chemin courant

ctx.clip();

// Cercle de coordonnées 200,200

ctx.beginPath();

ctx.arc(200, 200, 150, 0, Math.PI * 2, false);

ctx.closePath();

// Masque sur le chemin courant

ctx.clip();

img.onload = function() {

// Dessin de l'image

ctx.drawImage(img, 0, 0);

}

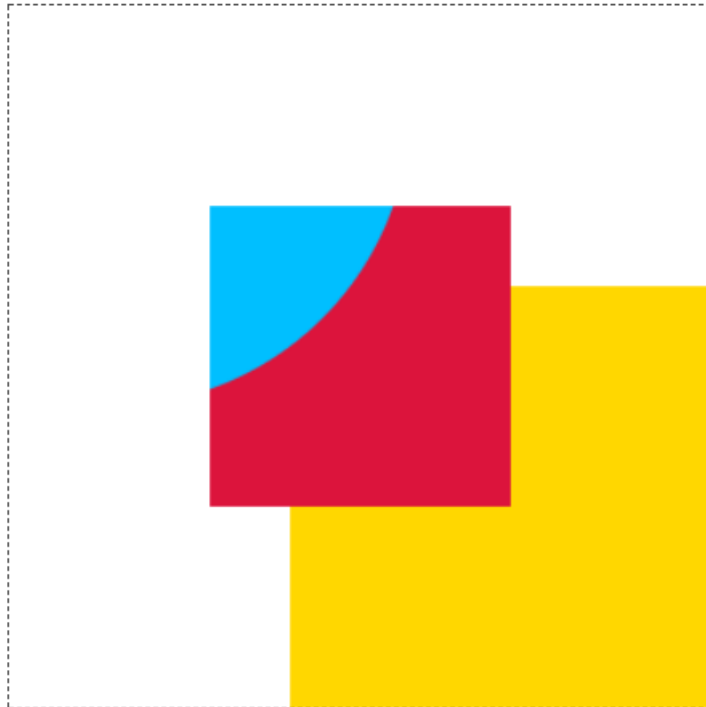
</script>

</body>

</html>
```

Application 5

Créer le canvas ci-dessous :



Application 6

Créer un masque circulaire sur une image

Exercice 3 : Contrôle clavier et souris

En tant qu'élément HTML à part entière, le canvas bénéficie de tous les événements DOM (clavier, souris, navigateur) pouvant survenir et de toutes les fonctions JavaScript afférentes.³

Souris

Le contrôle à la souris est disponible avec les événements **click**, **dblclick**, **mousedown**, **mouseup**, **mousemove**, **mouseenter** et **mouseleave**. Déclarer des gestionnaires pour chacun d'entre eux est un premier pas, compléter la fonction en est un autre. Celle-ci regroupe les instructions nécessaires à l'ajout de tracés, d'images, de formes ou au rafraîchissement global de la zone de dessin.

Exemple :

Le code source suivant crée une ardoise dans le navigateur, équipée d'une palette pour les changements de couleur.


```
<!doctype html>
<html lang="fr">
<head>
<title>HTML5 : Canvas</title>
<meta charset="utf-8">
<style>
body {
background:#eee;
text-align:center;
padding-top:10%;
}
.palette span {
display:inline-block;
width:40px;
height:40px;
cursor:pointer;
border:2px solid transparent;
border-radius:4px;
}
.palette span:hover {
border-color:white;
}
canvas {
cursor:crosshair;
border:5px solid #666;
background:white;
border-radius:4px;
box-shadow:0px 0px 20px #666;
margin-top:20px;
}
</style>
</head>
<body>

<!-- Palette de couleurs _ -->
<div class="palette">
<span onclick="modifierCouleur('#206BA4');" style="background:#206BA4"></span>
<span onclick="modifierCouleur('#54A4DE');" style="background:#54A4DE"></span>
<span onclick="modifierCouleur('#BBD9EE');" style="background:#BBD9EE"></span>
<span onclick="modifierCouleur('#BEDF5D');" style="background:#BEDF5D"></span>
<span onclick="modifierCouleur('#D6EB9A');" style="background:#D6EB9A"></span>
<span onclick="modifierCouleur('#FF9834');" style="background:#FF9834"></span>
<span onclick="modifierCouleur('#FFBF80');" style="background:#FFBF80"></span>
<span onclick="modifierCouleur('#F6E896');" style="background:#F6E896"></span>
<span onclick="modifierCouleur('#b07d42');" style="background:#b07d42"></span>
<span onclick="modifierCouleur('#FF5349');" style="background:#FF5349"></span>
```

```

</div>
<!-- Canvas -->
<canvas id="dessin" width="480" height="360"></canvas>
<script>
var moncanvas = document.getElementById('dessin');
var ctx = moncanvas.getContext('2d');
var en_dessin = false;
// Propriétés graphiques par défaut _
ctx.strokeStyle = "black";
ctx.lineWidth = 2;
// Bouton de souris activé _
moncanvas.onmousedown = function(e) {
// Dessin activé
en_dessin = true;
// Repositionnement du début du tracé
ctx.moveTo(e.offsetX,e.offsetY);
};
// Mouvement de souris _
moncanvas.onmousemove = function(e) {
if(en_dessin) dessiner(e.offsetX,e.offsetY);
};
// Bouton de souris relâché _
moncanvas.onmouseup = function(e) {
// Dessin désactivé
en_dessin = false;
};
// Ajoute un segment au tracé _
function dessiner(x,y) {
ctx.lineTo(x,y);
ctx.stroke();
}

// Modification de la couleur du contexte _
function modifierCouleur(codeCouleur) {
if(codeCouleur) ctx.strokeStyle = codeCouleur;
}
</script>
</body>
</html>

```

Clavier

La gestion du clavier est analogue, avec l'interception de l'événement *keypress*, *keyup* ou *keydown*. La valeur de la touche enfoncée correspond à un code numérique stocké dans la propriété *event.keyCode*.

Exemple

L'ardoise de dessin peut être contrôlée uniquement au clavier pour la transformer en un Télécran

```
<!doctype html>
<html lang="fr">
<head>
<title>HTML5 : Canvas</title>
<meta charset="utf-8">
<style>
body {
background:#eee;
text-align:center;
padding-top:10%;
}
canvas {
border:5px solid #666;
background:white;
border-radius:4px;
box-shadow:0px 0px 20px #666;
}
</style>
</head>
<body>
<canvas id="dessin" width="480" height="360"></canvas>
<script>
var moncanvas = document.getElementById('dessin');
var ctx = moncanvas.getContext('2d');
ctx.strokeStyle = "black";
ctx.lineWidth = 5;
ctx.lineCap = "round";
// Calcul du centre
var x = moncanvas.width/2;
```

```
var y = moncanvas.height/2;
// Position de départ (au centre)
ctx.moveTo(x,y);
// Gestionnaire d'événement keydown
if(document.body.onkeydown) {
document.body.onkeydown = dessiner;
} else if(document) {
document.onkeydown = dessiner;
}
// Déplacement du "pinceau"
function dessiner(event) {
switch(event.keyCode) {
case 38: // Haut
event.preventDefault();
if(y>=5) y-=5;
break;
case 40: // Bas
event.preventDefault();
if(y<moncanvas.height) y+=5;
break;
case 39: // Droite
event.preventDefault();
if(x<moncanvas.width) x+=5;
break;
case 37: // Gauche
event.preventDefault();
if(x>=5) x-=5;
break;
}
// Tracé d'après le décalage effectué
```

```
ctx.lineTo(x,y);  
ctx.stroke();  
};  
</script>  
</body>  
</html>
```

Exercice 4 : animation et jeux

Animation

La gestion des animations avec Canvas est quasi inexistante ! En effet, Canvas ne propose rien pour animer les formes, les déplacer, les modifier... Pour arriver à créer une animation avec Canvas, il faut :

1. Dessiner une image ;
2. Effacer tout ;
3. Redessiner une image, légèrement modifiée ;
4. Effacer tout ;
5. Redessiner une image, légèrement modifiée ;
6. Et ainsi de suite...

En clair, il suffit d'appeler une fonction qui, toutes les x secondes, va redessiner le canvas. Il est également possible d'exécuter des fonctions à la demande de l'utilisateur, mais ça, c'est assez simple.

« framerate »

« Framerate » est un mot anglais pour évoquer le nombre d'images affichées par seconde. Les standards actuels définissent que chaque animation est censée, en théorie, afficher un framerate de 60 images par seconde pour paraître fluide pour l'œil humain. Parfois, ces 60 images peuvent ne pas être toutes affichées en une seconde à cause d'un manque de performances, on appelle cela une baisse de framerate et cela est généralement perçu par l'œil humain comme étant un ralenti saccadé. Ce problème est peu appréciable et est malheureusement trop fréquent avec les fonctions *setTimeout()* et *setInterval()*, qui n'ont pas été conçues à l'origine pour ce genre d'utilisations...

Une solution à ce problème a été développée : *requestAnimationFrame()*. À chacune de ses exécutions, cette fonction va déterminer à quel moment elle doit se redéclencher de manière à garder un framerate de 60 images par seconde. En clair, elle s'exécute de manière à afficher quelque chose de fluide.

Exemple 1 : Animation avec setInterval()

```
<!DOCTYPE html>
<html>
<body>
<canvas id="dessin" width="640" height="480"></canvas>
<script>
var moncanvas = document.getElementById('dessin');
var ctx = moncanvas.getContext('2d');
ctx.lineWidth = 2;
ctx.fillStyle = "rgba(206,0,0,255)";
// Positionnement au centre
ctx.translate(moncanvas.width/2,moncanvas.height/2);
var i = 0;
function dessiner() {
ctx.translate(4,1);
ctx.rotate(0.2);
ctx.fillRect(i,0,20,20);
i++;
if(i>400) clearInterval(inter);
ctx.fillStyle = "rgba(206,0,"+i+",255)";
}
var inter = setInterval(dessiner,10);
</script></body>
</html>
```

La fonction dessiner() est appelée toutes les 10 millisecondes soit en théorie 100 images par seconde et se base sur l'incrément de la variable i pour modifier le contexte graphique (translation, rotation et couleur de remplissage). La méthode clearInterval() arrête l'animation une fois 400 boucles accomplies.

Exemple 2 : Animation avec requestAnimationFrame

```
<!DOCTYPE html>
<html>
<body>

<canvas id="dessin" width="640" height="480"></canvas>
<script>
var moncanvas = document.getElementById('dessin');
var ctx = moncanvas.getContext('2d');
ctx.lineWidth = 5;
ctx.strokeStyle = "rgba(206,0,0,255)";
ctx.shadowOffsetX = 0;
ctx.shadowOffsetY = 0;
ctx.shadowBlur = 15;
```

```

ctx.shadowColor = '#000';
ctx.translate(moncanvas.width/2,moncanvas.height/2);
var i = 0;
// requestAnimationFrame shim (par Paul Irish)
window.requestAnimFrame = (function(){
return window.requestAnimationFrame ||
window.webkitRequestAnimationFrame ||
window.mozRequestAnimationFrame ||
window.oRequestAnimationFrame ||
window.msRequestAnimationFrame ||
function(callback,element) {
window.setTimeout(callback,1000/60);
};
})();
// Fonction de dessin
function dessiner() {
ctx.translate(3,1);
ctx.rotate(0.2);
ctx.beginPath();
ctx.moveTo(0,i);
ctx.lineTo(i+1,i);
ctx.lineTo(i,i+10);
ctx.closePath();
ctx.stroke();
ctx.strokeStyle = "rgba(206,"+i+",0,255)";
i++;
// Nouvelle itération
requestAnimFrame(dessiner);
}
// Premier appel de la fonction de dessin
dessiner();
</script>
</body>
</html>

```

Une fois exécutée, la fonction ***dessiner()*** demande explicitement d’être appelée une nouvelle fois grâce à ***requestAnimationFrame()***. C’est ainsi que l’animation perdure, tant que le navigateur lui donne la main.

Exemple 3

```

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></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.requestAnimationFrame = (function(){  
return window.requestAnimationFrame || // La forme standardisée
```

```
function(callback){ // Pour les mauvais  
window.setTimeout(callback, 1000 / 60);  
};  
})();
```

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

```
function draw(angle) {  
context.save();  
context.clearRect(0, 0, 150, 150);  
context.translate(75,75);
```

```
context.fillStyle = "teal";  
context.rotate((Math.PI / 180) * (45 + angle));  
context.fillRect(0, 0, 50, 50);
```



```
context.fillStyle = "orange";
context.rotate(Math.PI / 2);
context.fillRect(0, 0, 50, 50);

context.fillStyle = "teal";
context.rotate(Math.PI / 2);
context.fillRect(0, 0, 50, 50);

context.fillStyle = "orange";
context.rotate(Math.PI / 2);
context.fillRect(0, 0, 50, 50);

context.restore();

angle = angle + 2;

if (angle >= 360) angle = 0;

window.requestAnimationFrame(function() { draw(angle) });
}

draw(0);
};

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

Jeux :

Les jeux créés avec Canvas ne sont qu'un savant mélange de toutes ces fonctionnalités,

un choix approprié de graphismes et une manipulation de données stockées en tableaux et objets variés. Toutes les briques de base sont disponibles pour assurer un développement complet : images, formes, sprites, événements clavier et souris, voire vidéo et son.

De nombreux frameworks existent pour faciliter la conception d'interfaces et leur positionnement, la gestion des déplacements, des collisions et des contrôles, les transformations d'éléments graphiques ou d'environnements de jeu.

Exemples jeux

Superposition canvas :

http://darchevillepatrick.info/canvas/canvas_superposes.htm

Animation cercle

http://darchevillepatrick.info/canvas/canvas_animation_rond.htm

Site à voir : création jeu

<http://www.lafermeduweb.net/billet/-tutorial-cree-un-jeu-web-avec-du-html-5-canvas-et-javascript-390.html>

Exercice 5 : vidéo et audio

Ce qui est merveilleux avec les technologies ouvertes et interopérables c'est qu'elles se connaissent intimement et savent profiter de leurs avantages mutuels.

Intégrer une vidéo `<video>` HTML 5 dans `<canvas>` est un jeu d'enfant qui consiste à utiliser la méthode ***drawImage()***, non plus avec un objet image, mais avec la vidéo en question. C'est cette fonction qui réplique le contenu graphique dans le canvas.

Exemple 1 : video dans canvas

```
<!doctype html>
<html lang="fr">
<head>
<title>HTML5 : Canvas + Video</title>
<meta charset="utf-8">
<style>
canvas {
border:1px dashed #ccc;
}
video {
border:5px solid #0094e9;
}
</style>
</head>
<body>
<!-- Élément video -->
<video id="mavideo" width="720" height="406">
```

```

<source src="bunny.ogv" type="video/ogg">

</video>
<!-- 2 Canvas -->
<div>
<canvas id="dessin1" width="360" height="203"></canvas>
<canvas id="dessin2" width="360" height="203"></canvas>
</div>
<!-- Contrôles -->
<input type="button" value="Lecture Video" onclick="video.play();">
<input type="button" value="Stop Video" onclick="video.pause();">
<input type="button" value="Stop Canvas"
onclick="clearInterval(inter);">
<script>
var moncanvas1 = document.getElementById('dessin1');
var ctx1 = moncanvas1.getContext('2d');
var moncanvas2 = document.getElementById('dessin2');
var ctx2 = moncanvas2.getContext('2d');
var video = document.getElementById('mavideo');
// Propriétés générales
ctx1.scale(0.5,0.5);
ctx2.translate(-video.width/4,-video.height/4);
// Un intervalle de temps régulier
if(video) var inter = setInterval(dessinVideo, 40);
// Fonction exécutée périodiquement
function dessinVideo() {
if(!isNaN(video.duration)) {
// Dessin de la vidéo dans les deux contextes
ctx1.drawImage(video, 0, 0);
ctx2.drawImage(video, 0, 0, video.width, video.height);
}
}
</script>
</body>
</html>

```

Le document est composé d'un élément **<video>** et de deux **<canvas>** qui possèdent la moitié des dimensions de la vidéo originale. Des contrôles _ activent ou désactivent la lecture et la copie de Vidéo à Canvas. Des transformations sont appliquées aux contextes, respectivement un redimensionnement de moitié et une translation pour centrer la vue.

La vidéo étant composée de multiples images se succédant, il est nécessaire de faire appel à **drawImage()** à de courts intervalles réguliers, grâce à la fonction **setInterval()** de JavaScript _ qui reçoit en arguments la fonction appelée **dessinVideo()** et le délai entre chaque appel, en millisecondes.

Chaque déclenchement de fonction recopie le contenu dans chacun des contextes avec **drawImage()**, en appliquant les transformations. Le premier appel pour **ctx1** est effectué en précisant le point de référence pour le dessin de l'image (0,0), le deuxième pour **ctx2** mentionne la zone (totale) à recopier. Dans le cas présent, ils sont équivalents.

En indiquant des coordonnées différentes, il est envisageable de ne placer dans `<canvas>` qu'une sous-partie du rendu vidéo. Tous les autres traitements et effets sont possibles en temps réel.

Concernant l'audio, l'API est disponible de la même façon (hormis l'absence de rendu visuel), et contrôlable avec JavaScript.

Exemple 2 Déclencher un effet sonore :

```
var effet1 = new Audio('effet1.oga');  
effet1.play();
```

Que ce soit pour déclencher des sons dans un jeu ou prévoir une musique de fond durant une animation, les méthodes équipant `<audio>` et son API sont au service de `<canvas>` et des événements pouvant survenir

Application 7 (note : 12 points pour l'originalité et 8 pour le code)

A votre imagination : créez votre propre effet sonore et vidéo en utilisant les canvas

Exercice 6 : Editeur de dessin

On explore le fonctionnement d'un canvas (html5) : comme y dessiner et comment y capturer un clic de souris.

1. Créer le canvas en html en spécifiant ses dimensions.
2. Écrire une fonction initialisations() qui récupère le *contexte 2d* du canvas et associer cette fonction à l'événement onload du document.
3. Définir une fonction surclic(e) qui lance une alerte affichant les coordonnées du clic contenu dans e. Compléter la fonction initialisations() pour associer un clic sur le canvas à l'appel de la fonction surclic(e).
4. Remplacer le lancement de l'alerte par l'affichage d'un carré à l'endroit du clic. Régler la taille de ce carré.
5. Ajouter des boutons, chaque bouton permettant de choisir une couleur différente.
6. Ajouter deux boutons pour choisir la forme de la prochaine figure, soit un carré soit un disque.
7. Ajouter un bouton qui va lancer la composition aléatoire d'un dessin, par exemple en produisant une figure toutes les demi-secondes, figure dont la forme et la couleur sont choisies au hasard. Le dessin prend fin après 100 figures ou par un clic sur un dernier bouton.

Exercice 7

1. Afficher un carré dans un *canvas*.

2. Permettre son déplacement avec les touches du clavier en veillant à ne pas dépasser les bords du *canvas*.
3. Ajouter un bouton pour lancer le déplacement aléatoire du carré, un autre bouton pour l'arrêter.
4. Comment faire disparaître la trace pour seulement déplacer un carré ?

Exercice 8

Une cible apparaît à une position aléatoire, le joueur a quelques instants pour cliquer dessus.

1. Définir l'aspect du jeu à l'aide d'un *canvas* html5.
2. Faire apparaître un carré en guise de cible, à une position aléatoire du *canvas*. La cible doit rester apparente deux secondes, puis disparaître. Une autre cible est alors générée aléatoirement.

La production de cibles doit débiter dès le chargement de la page.

3. Capturer les clics de l'utilisateur.
Tester si le clic a touché la cible. Si c'est le cas, effacer la cible et en produire une autre.
4. Lors d'un clic, afficher un message indiquant si la cible est touchée ou non. Afficher également le nombre de cibles détruites.
5. Ajouter un bouton pour faire une pause dans la partie et un autre pour démarrer une nouvelle partie.
6. (bonus) Faire varier taille, forme et couleur des cibles.

Exercice 9

Dans ce jeu, il s'agit de trouver un mot dont les lettres ont été éparpillées sur un *canvas*.

1. Définir en html un *canvas*, par exemple un carré de 400 pixels de côté. Prévoir également une zone dédiée au dialogue entre le jeu et le joueur.
2. En JavaScript, définir une variable contenant un mot, et une autre variable contenant la définition de ce mot.
3. Placer chaque lettre du mot à une position du *canvas* choisie aléatoirement. On pourra aussi choisir aléatoirement, pour chaque lettre, sa police, sa taille et sa couleur.
4. Ajouter en html un champ texte qui permet au joueur de saisir une proposition. Y associer une procédure JavaScript qui confronte cette proposition à la solution et affiche un message selon que le joueur a trouvé ou non.
5. Ajouter un bouton « indice » qui, lorsque l'on clique dessus, affiche la définition du mot. Et un autre bouton « solution » qui provoquera l'affichage du mot mystère.
6. Choisir au hasard le mot à trouver dans une liste préalablement définie (pour chaque mot de cette liste on dispose aussi de sa définition). Ajouter un bouton qui permet de commencer une nouvelle partie avec un nouveau mot.

7. Faire varier régulièrement l'affichage des lettres. Ajouter un bouton pour suspendre cette animation, un autre pour la reprendre. Imaginer enfin une animation graphique particulière pour le moment où le mot mystère est découvert.

Frame work js canvas :

<https://openclassrooms.com/courses/piloter-canvas-avec-kineticjs>