TP Bonus : Développement WEB : Javascript
Un peu de graphisme et d'animation : gestion de l'élément « canvas »

Objectifs :

- ✓ Utilisation de code Javascript au sein du développement web
- ✓ Dessiner par programmation : étude de l'élément HTML5 « canvas »

Durée : 2H environ en autonomie quand les TP obligatoires sont terminés

L'élément `canvas` fournit des scripts dépendant de la résolution, qui peuvent être utilisés pour l'affichage de graphiques, de jeux ou de n'importe quelle image. Plus simplement, l'élément `canvas` va nous fournir une zone de dessin et les outils nécessaires pour dessiner.

Description de l'outil `Canvas` (fichier « HTML5_Canvas_Cheat_Sheet.pdf » joint) :

HTML5 Canvas Cheat Sheet v1.1

http://blog.nihilogic.dk/

Canvas element

Attributes

Name	Type	Default
width	*unsigned long*	300
height	*unsigned long*	150

Methods

Return	Name
string	toDataURL([Optional] *string* type, [Variadic] *any* args)
Object	getContext(*string* contextId)

2D Context

Attributes

Name	Type
canvas	*HTMLCanvasObject* [readonly]

Methods

Return	Name
void	save( )
void	restore( )

Transformation

Methods

Return	Name
void	scale(*float* x, *float* y)
void	rotate(*float* angle)
void	translate(*float* x, *float* y)
void	transform(*float* m11, *float* m12, *float* m21, *float* m22, *float* dx, *float* dy)
void	setTransform(*float* m11, *float* m12, *float* m21, *float* m22, *float* dx, *float* dy)

Image drawing

Methods

Return	Name
void	drawImage(*Object* image, *float* dx, *float* dy, [Optional] *float* dw, *float* dh)
	Argument "image" can be of type *HTMLImageElement*, *HTMLCanvasElement* or *HTMLVideoElement*
void	drawImage(*Object* image, *float* sx, *float* sy, *float* sw, *float* sh, *float* dx, *float* dy, *float* dw, *float* dh)

Compositing

Attributes

Name	Type	Default
globalAlpha	*float*	1.0
globalCompositeOperation	*string*	source-over

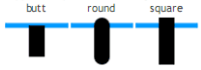
Supports any of the following values:

Line styles

Attributes

Name	Type	Default
lineWidth	*float*	1.0
lineCap	*string*	butt

Supports any of the following values:



lineJoin	*string*	miter

Supports any of the following values:

miterLimit	*float*	10

Colors, styles and shadows

Attributes

Name	Type	Default
strokeStyle	*any*	black
fillStyle	*any*	black
shadowOffsetX	*float*	0.0
shadowOffsetY	*float*	0.0
shadowBlur	*float*	0.0
shadowColor	*string*	transparent black

Methods

Return	Name
CanvasGradient	createLinearGradient(*float* x0, *float* y0, *float* x1, *float* y1)
CanvasGradient	createRadialGradient(*float* x0, *float* y0, *float* r0, *float* x1, *float* y1, *float* r1)
CanvasPattern	createPattern(*Object* image, *string* repetition)
	Argument "image" can be of type *HTMLImageElement*, *HTMLCanvasElement* or *HTMLVideoElement*
	"repetition" supports any of the following values: [repeat (default), repeat-x, repeat-y, no-repeat]

CanvasGradient interface

void	addColorStop(*float* offset, *string* color)

CanvasPattern interface

No attributes or methods.

Paths

Methods

Return	Name
void	beginPath( )
void	closePath( )
void	fill( )
void	stroke( )
void	clip( )
void	moveTo(*float* x, *float* y)
void	lineTo(*float* x, *float* y)
void	quadraticCurveTo(*float* cpx, *float* cpy, *float* x, *float* y)
void	bezierCurveTo(*float* cp1x, *float* cp1y, *float* cp2x, *float* cp2y, *float* x, *float* y)
void	arcTo(*float* x1, *float* y1, *float* x2, *float* y2, *float* radius)
void	arc(*float* x, *float* y, *float* radius, *float* startAngle, *float* endAngle, *boolean* anticlockwise)
void	rect(*float* x, *float* y, *float* w, *float* h)
boolean	isPointInPath(*float* x, *float* y)

Text

Attributes

Name	Type	Default
font	*string*	10px sans-serif
textAlign	*string*	start
	Supports any of the following values: [start, end, left, right, center]	
textBaseline	*string*	alphabetic
	Supports any of the following values: [top, hanging, middle, alphabetic, ideographic, bottom]	

Methods

Return	Name
void	fillText(*string* text, *float* x, *float* y, [Optional] *float* maxWidth)
void	strokeText(*string* text, *float* x, *float* y, [Optional] *float* maxWidth)
TextMetrics	measureText(*string* text)

TextMetrics interface

width	*float*	[readonly]

Rectangles

Methods

Return	Name
void	clearRect(*float* x, *float* y, *float* w, *float* h)
void	fillRect(*float* x, *float* y, *float* w, *float* h)
void	strokeRect(*float* x, *float* y, *float* w, *float* h)

Pixel manipulation

Methods

Return	Name
ImageData	createImageData(*float* sw, *float* sh)
ImageData	createImageData(*ImageData* imagedata)
ImageData	getImageData(*float* sx, *float* sy, *float* sw, *float* sh)
void	putImageData(*ImageData* imagedata, *float* dx, *float* dy, [Optional] *float* dirtyX, *float* dirtyY, *float* dirtyWidth, *float* dirtyHeight)

ImageData interface

width	*unsigned long*	[readonly]
height	*unsigned long*	[readonly]
data	*CanvasPixelArray*	[readonly]

CanvasPixelArray interface

length	*unsigned long*	[readonly]

Source: http://www.whatwg.org/specs/web-apps/current-work/ (2009-05-04)

D'autres liens utiles de description et d'exemples d'utilisation :

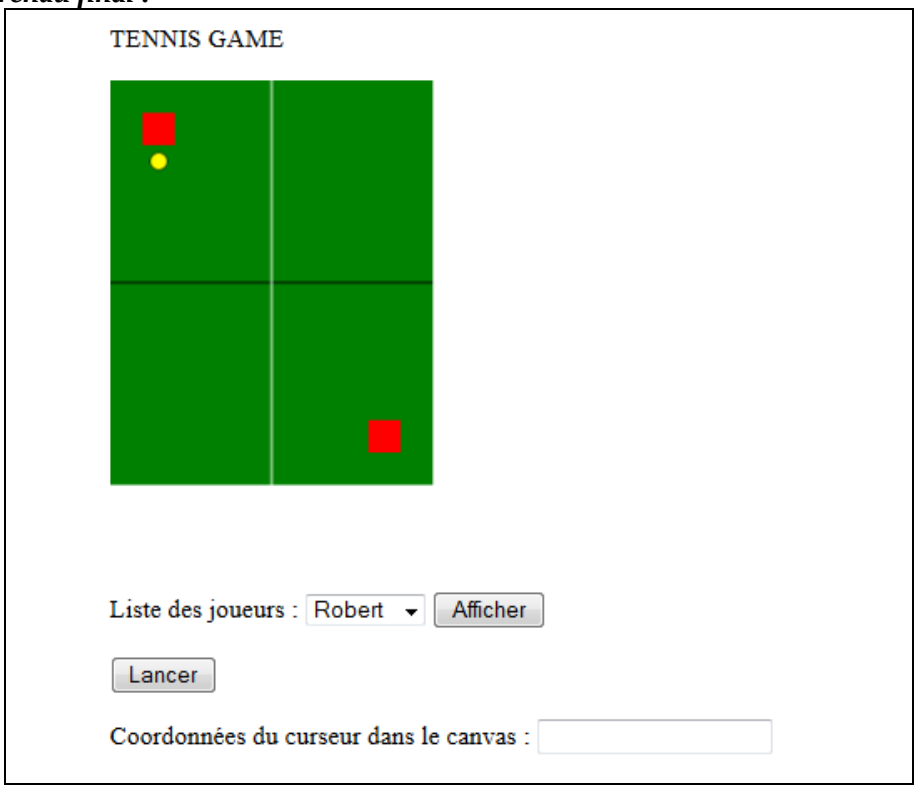
- ✓ http://msdn.microsoft.com/fr-fr/magazine/hh708752.aspx
- ✓ http://www.w3schools.com/html/html5_canvas.asp
- ✓ http://www.html5canvastutorials.com/
- ✓ https://developer.mozilla.org/fr/docs/Web/Guide/Graphics/Dessiner_avec_canvas

Travail à faire :

Nous allons mettre en place un jeu de tennis entre 2 joueurs. La présentation du jeu permettra de :
- dessiner le terrain,
- nommer les joueurs,
- insérer une balle
- et lancer la balle.
- A tout moment nous pourrons vérifier les coordonnées de la balle sur le terrain.

Des interactions vont donc se programmer entre le code HTML et la zone graphique définie par le canvas (dans les 2 sens, du HTML vers le graphique et du graphique vers la zone).

Exemple de rendu final :



Etape 1 : Mise en place de la zone graphique servant de terrain

Votre code HTML sera de la forme ci-dessous avec notamment la section « head » et la section « body » où nous insérerons le code javascript :

```
<html><head>
<meta charset="utf-8">
<title>Exercice TP8 …</title>
</head>
<body>
<p> <label> TENNIS GAME</label> </p>

<canvas id="graphique" width="300" height="300">
 Erreur : votre navigateur ne supporte pas l'élément canvas
</canvas>

<script language="javascript">
   // Fonctions javascript…
</script>
    <!-- code HTML pour zone de texte, label, liste déroulante, boutons…-->
</body>
</html>
```

1. Reproduire ce code HTML, repérez notre objet canvas qui nous servira pour définir notre zone graphique.

2. Pour définir la zone graphique, nous allons déclarer en javascript une variable canvas qui fera le lien avec l'élément HTML canvs précédemment défini puis nous définissons notre zone en 2D appelée ici context :

```
var canvas = document.getElementById("graphique");
var context = canvas.getContext("2d");
```

3. Pour définir notre terrain de tennis, nous allons écrire une fonction javascript qui va dessiner un rectangle vert et délimiter le terrain en son milieu (vous pouvez changer la taille et donc les proportions). Bien comprendre ce que vous faites.

```
function drawField() {
    // dessin du terrain
    // definir un rectangle de couleur verte
    context.fillStyle="green";
    context.fillRect(0,0,200,250);

    // ligne de séparation entre les 2 camps : style et épaisseur
    context.strokeStyle="black";
    context.lineWidth = "2.0";
    context.beginPath(); // création d'un "crayon"
    // point d'origine de la ligne à dessiner
    context.moveTo(0,125);
    // point de destination de la ligne à dessiner
    context.lineTo(200, 125);
    context.stroke(); // pour la voir dessiner
    context.closePath(); // fermeture du "crayon"
}
```

Testez en appelant votre fonction :

```
drawField();
```

4. Compléter votre code de la fonction `drawField` pour ajouter une ligne de milieu horizontale en noir. Vous pouvez aussi ajouter une ligne de service.

5. Testez, votre terrain doit être dessiné.

Etape 2 : Ajout de deux joueurs et de la balle

1. Pour définir un joueur, nous allons le dessiner par un carré rouge. Ecrire une fonction en javascript `drawPlayers()` qui dessine 2 carrés rouges à chaque bout du terrain comme dans l'exemple page 2. Pour dessiner un rectangle (un carré est un rectangle dont les côtés sont égaux !), utiliser les syntaxes :

```
context.fillStyle="red";
context.fillRect(X, Y, taille_long,taille_larg);
```

où X = coordonnées en horizontal et Y = coordonnées à la verticale

Appeler votre fonction pour tester : `drawPlayers() ;`

2. Pour définir la balle, nous allons le dessiner un rond jaune. Il s'agit en fait de définir un arc de cercle de 0 à 360°. Ecrire la fonction suivante qui définit un cercle aux coordonnées x et y, de rayon r, fillColor est la couleur de remplissage et strokeColor la couleur de contour :

```
function drawCircle(x,y,r,fillColor,strokeColor) {
    context.fillStyle=fillColor;
    context.strokeStyle=strokeColor;
    context.beginPath();
    context.arc(x,y,r,0, Math.PI * 2, true);
```

```
    context.closePath();
    context.stroke();
    context.fill();
}
```

3. Appeler cette fonction pour dessiner une balle jaune au contour noir d'un rayon 5 (placer là où vous voulez au départ).

Etape 3 : Interaction HTML vers la Zone Graphique

Pour tester cette interaction de l'HTML vers la zone graphique canvas, je vous propose de créer en HTML une liste déroulante proposant des noms de joueurs. Sur un bouton « afficher », nous écrirons sur le terrain le nom associé au joueur en dessous du carré rouge spécifique.

1. En HTML créer une liste déroulante et un bouton « afficher ».

```
<p>
   <label for="player"> Liste des joueurs : </label>
   <select id="player">
      <option label="joueur 1" value="player1">Robert</option>
      <option label="joueur 2" value="player2">Charles</option>
   </select>
   <input type="button" value="Afficher" onclick="displayPlayer()"/>
</p>
```

2. L'interaction va donc se faire sur le click du bouton en lançant la fonction `displayPlayer()`. Cette fonction va écrire du texte, nous mettons donc en place une fonction `drawText(text, x, y)` qui écrit du texte dans la zone graphique à une place donnée et selon un style précis.

```
function drawText(text, x, y) {
// fonction qui permet d'écrire du texte
  context.fillStyle="black";
  context.font = "12px sans-serif";
  context.textBaseline = "top";
  context.fillText(text, x, y);
}

function displayPlayer() {
// selon le joueur sélectionné dans la liste déroulante : afficher son nom
var player = document.getElementById("player").value; // pour récupérer l'élément sélectionné de la liste HTML
switch (player) {
   case "player1" :
        drawText("Robert",22,45);
        break;
   case "player2" :
        drawText("Charles",160,230);
        break;
   }
}
```

3. Testez et comprenez bien ces deux fonctions.

4. On peut ainsi créer d'autres interactions entre l'HTML et le graphique en récupérant le HTML via l'id (méthode `document.getElementById("id").value`.

   Ajouter un bouton « Service » qui permet sur le click de préciser uniquement le nom du joueur qui est au service. La balle sera mise en face du joueur au service. Par souci de simplicité, on change de joueur au service à chaque nouveau click sur le bouton « Service ».

Etape 4 : Interaction Zone Graphique vers l'HTML

Pour tester cette interaction la zone graphique canvas vers l'HTML, je vous propose de créer en HTML une zone de texte en lecture seule qui va récupérer les coordonnées de la souris à chaque click sur le terrain.

1. En HTML créer une zone de texte avec un label de type :

```
<p>
  <label for="mouse"> Coordonnées du curseur dans le canvas : </label>
  <input id="mouse" type="text" value="" readonly />
</p>
```

2. Nous allons créer une fonction javascript qui nous servira de gestionnaire d'évènements et qui se déclenchez dès qu'un évènement précis se produira (ici un clic de souris : `"mousedown"`)

```
function displayPosition(event) {
// gestionnaire d'évènements
   // position du curseur sur la page
   var posX, posY;
   if ((event.pageX != undefined) && (event.pageY != undefined))
   {
     posX = event.pageX;
     posY = event.pageY;
   }
   else
   {
   // portablilté : si le navigateur ne reconnait pas les propriétés
pagX et pageY comme IE
     posX    =    event.clientX    +    document.body.scrollLeft    +
document.documentElement.scrollLeft;
     posY     =     event.clientY     +     document.body.scrollTop+
document.documentElement.scrollTop;
   }
   // position du curseur dans la zone graphique
   posX -= canvas.offsetLeft;
   posY -= canvas.offsetTop;
   // écriture des coordonnées dans la zone de texte
   document.getElementById("mouse").value = "(" + posX + ", " + posY
+")";
}
```

3. Testez. Il ne devrait rien se passer de nouveau !

4. Il fait désormais ajouter cet évènement sur le clic de la souris à notre gestionnaire d'évènements pour qu'il soit en écoute :

```
canvas.addEventListener("mousedown", displayPosition, false);
```

5. Testez à nouveau. Cela devrait mieux fonctionner ! Bien comprendre où se fait le lien entre la récupération des coordonnées X et Y et l'écriture dans la zone de texte HTML par le biais de `document.getElementById`.

Etape 5 : Un peu d'animation !

Une balle, ça se lance !
Pour commencer on va lancer la balle et la déplacer de quelques pixels à chaque fois pour former un mouvement. A tout moment, nous pourrons stopper ce déplacement.

1. En HTML créer un bouton « Lancer » qui exécutera une fonction `launch` (on met `this` ici en paramètre pour préciser l'objet du bouton qui définit l'appel à la fonction) :

```
<input type="button" value="Lancer" onclick="launch(this)"/>
```

2. La fonction `launch` travaille avec un timer qui toutes les 10 mms ici redessine la zone graphique avec les nouvelles coordonnées de la balle. Le bouton « Lancer » devient alors « Stop » pour arrêter le timer :

```
var timer;
function launch(object) {
   if (object.value == "Lancer") {
      object.value = "Stop";
        timer = setInterval(drawAndMoveBall, 10);
      }
      else {
        object.value = "Lancer";
        clearInterval(timer);
      }
}
```

3. La fonction `drawAndMoveBall` redessine la zone graphique et selon les coordonnées nouvellement calculées place la balle. Il faut préalable mettre les positions, les déplacements voulus et la taille de la zone en variables globales :

```
// positions initiales de la balle
var x = 30;
var y = 50;
// déplacements élémentaires de la balle
var dx = 3;
var dy = 5;
// taille de la zone graphique
var width = 200;
var height = 250;

function drawAndMoveBall() {
  // remise à 0 de la zone de dessin
  context.clearRect(0,0,width, height);
  drawField();
  drawPlayers();
  drawCircle(x,y,5, "yellow", "black");
  // calcul de la prochaine position
  if ((x + dx > width -10) || (x + dx < 0))
  {
     dx = -dx;
  }
  if ((y + dy > height -10) || (y + dy < 0))
  {
     dy = -dy;
  }
   x += dx;
   y += dy;
}
```

4. Ces deux fonctions ne s'inventent pas mais ne sont pas difficiles à comprendre. Bien les comprendre (ou demander de l'aide) et Testez.

5. Pour aller plus loin : modifier les fonctions précédentes pour que la balle s'arrête dès qu'elle touche le fond du terrain opposé et que le service change pour le lancement suivant.

6. Pour aller encore plus loin : prévoir les déplacements des joueurs quand on clique dessus.