Etude HTML Canvas

# Chapitre 1: Démarrer avec html5-canvas

## Qu’est-ce qu’un « canvas »

Html-Canvas :

* Est un élément HTML5.
* Est pris en charge dans la plupart des navigateurs modernes (Internet explorer 9+)
* Est un élément visible et transparent par défaut.
* A une largeur par défaut de 300px et une hauteur de 150px.
* Tout le contenu du canvas doit être ajouté par JavaScript.

## Création et redimensionnement de ton canvas

Il n’est pas nécessaire d’installer des librairies ou un module quelconque car canvas est un élément natif de HTML5. Voici les étapes pour créer un élément canvas :

* Créez un fichier « index.html » et un fichier « index.js » dans un dossier
* Ouvrez dans votre éditeur de code (bloc-notes, notepad++, VScode,…)
* Dans le fichier « index.html », placez un élément <canvas> dans le <body> :

<!doctype html>  
<html>  
<head>  
 <style>  
 \*{  
 margin : 0 ;

}  
 canvas {  
 width : 100% ;  
 height : 100% ;  
 border:1px solid black;   
 }  
 </style>  
</head>  
 <body>  
  
 <!-- add a canvas element using html -->  
 <canvas ></canvas>  
   
 <script src= « index.js »> </script>  
 </body>  
</html>

* Dans le fichier « index.js», récupérez l’élément canvas avec le DOM et appeler con contexte en 2D :

var myCanvas = document.querySelector(«canvas»);   
var ctx = myCanvas.getContext("2d");

## Dessiner des formes primitives

Les formes sont des ensembles des lignes et des courbes pouvant être visiblement dessinées sur le canvas. Une forme est généralement encadrer par la fonction beginPath() qui sert à commencer le dessin et les fonctions stroke() pour dessiner, fill() pour remplir et clip() pour remplir avec une image.

L’unité de mesure est en pixel

### Rectangle :

* Vide : ctx.rect (leftX, topY, width, height); ex : ctx.rect (100, 200, 150, 150); // Carré
* Plein : ctx.fillRect (leftX, topY, width, height); ex : ctx.rect (100, 100, 200, 50) ;

### Arc :

* vide : ctx.arc(centerX, centerY, rayon, startingRadianAngle, endingRadianAngle); // angle en radian  
  ex : ctx.arc(150, 350, 200, 0, Math.PI); // demi-cercle (PI = 180deg)  
   ctx.stroke() ; // la fonction stroke() pour le dessiner un trait
* Plein : ctx. arc (l(centerX, centerY, rayon, startingRadianAngle, endingRadianAngle);

ex : ctx.arc(150, 350, 200, 0, Math.PI \* 2); // cercle ( PI \* 2 = 360deg)   
 ctx.fill() ; // fonction fill() pour remplir

### Ligne

# Spécial : Utiliser canvas dans angular 14

* Créer un projet angular : ng new nomprojet
* Lancer le serveur d’angular : ng serve
* Dans le template appComponent.html, il faut juste mettre une référence dans l’élément canvas sans aucun autre attribut ni style dans « ccs », toute la manipulation doit être effectué dans le contrôleur:

<canvas #monCanvas > </canvas>

* Dans le contrôleur appComponent.ts,
  + utiliser ViewChild pour récupérer l’élément référencé comme dans le DOM et appeler l’objet nativeElement dans l’initiation du composant:

@ViewChild(‘’monCanvas’’, {static :true}) myCanvas : ElementRef ;

Canvas : HTMLCanvasElement ;  
ctx : any ;

ngOnInit() : void {

this.canvas = this.myCanvas.nativeElement ; // récupérer les éléments   
 natifs de canvas  
 this.canvas.style.backgroundColor = "aliceblue" ; // pour mettre le  
 canvas en blanc

this.ctx = this.canvas.getContext("2d"); // récupérer le contexte  
 du canvas

if (this.ctx) {

this.adapteWindow(); // appel des méthodes au initiation appComp

this.drawingRect(this.ctx);

}

}  
 adapteWindow() { // adapter mon style d’écran

this.canvas.height = window.innerHeight - 4; // -4 : pour éviter les  
 barres de scroll

this.canvas.width = window.innerWidth;

}

drawingRect(ctx: CanvasRenderingContext2D) { // typer ‘’ctx’’ pour  
 l’autocomplétion du codage

ctx.fillRect(20, 20, 100, 100) // construire un carré de côté=100px

}

* pour éviter les petits marges à gauche, mettre dans scr/style :

body{

margin: 0;

padding: 0;

background: #000; // du noir en fond

}

* il est préférable de construire chaque objet et ses methodes dans une classe appart et juste instancier(new) dans appComp : ng g class model/ballon  
   const bal = new Ballon(60, 10, 30, 'blue');

bal.drawBall(this.ctx) // methode dans la classe ‘’Ballon’’

* pour animé un element, il faut le mettre dans la fonction fleché (pour que ‘’this’’ de requestAnimationFrame(this.animate) fait référence au classe):

animate = () => { // fonction fléché (pour qui ‘’this’’ soit global)

this.ctx.clearRect(0,0,this.canvas.width, this.canvas.height);

const bal = new Ballon(this.x, 250, 30, 'red');

bal.drawBall(this.ctx)

requestAnimationFrame(this.animate); // boucler sur l’animation   
}

mettre requestAnimationFrame(this.animate) dans l’initiation du appComp mais non pas la fontion animate();  
 ngOnInit(): void {

this.canvas = this.myCanvas.nativeElement;

this.ctx = this.canvas.getContext("2d");

if (this.ctx) {

requestAnimationFrame(this.animate);

}  
 }

* animations de plusieurs particule :
  + créer plusieurs particules : dans ngOnInit(){ drawBalls() }

drawBalls() {

for (let i = 0; i < 200; i++) {

let x = Math.random() \* this.canvas.width; // mettre dans la boucle parce que a  
 chaque

let y = Math.random() \* this.canvas.height; // valeur de i, la valeur de x et y  
 changent.

let dx = Math.random() \* 5;

let dy = Math.random() \* 5;

const bal = new Ballon(x, y, dx, dy, 30, 'red');

bal.drawBall(this.ctx);

this.bals.push(bal); // mettre dans un tableau (bals[])

}

}

* + Boucler sur l’update de chaque particule dans animate :

animate = () => {

this.ctx.clearRect(0, 0, window.innerWidth, window.innerHeight); // ity dia atao

for (let i = 0; i < this.bals.length; i++) { ivelani update() ny clearRect()  
 satria raha mamafa daholo izy  
 this.bals[i].updateBall(this.ctx); rehetra dia iray iany no hita

}  
   
 requestAnimationFrame(this.animate); }

* Format de classe d’objet, avec la mmethode update() :

export class Ballon {  
 centerX: number;  
 centerY: number;  
 dx: number; // vitesse horizontal  
 dy: number; // vitesse vertical  
 rayon: number;  
 couleur: string;

constructor(x: number, y: number, dx: number, dy: number, r: number, clr: string) {  
 this.centerX = x;  
 this.centerY = y;  
 this.dx = dx;  
 this.dy = dy;  
 this.rayon = r;  
 this.couleur = clr; }

drawBall(ctx: CanvasRenderingContext2D) {  
 ctx.beginPath();  
 ctx.fillStyle = this.couleur;  
 ctx.arc(this.centerX, this.centerY, this.rayon, 0, Math.PI \* 2);  
 ctx.fill(); }

updateBall(ctx: CanvasRenderingContext2D) {  
 let r = Math.random() \* 255; // couleur aléatoire pour chaque particule  
 let g = Math.random() \* 255;  
 let b = Math.random() \* 255;

ctx.beginPath(); // tracage de nouvelle particule a chaque update()  
 ctx.fillStyle = this.couleur; // en gardant la même couleur   
 ctx.arc(this.centerX, this.centerY, this.rayon, 0, Math.PI \* 2); // mais la position  
 ctx.fill(); avec centerX = centerX + dx. (y aussi)

if (this.centerX + this.rayon > window.innerWidth   
 || this.centerX - this.rayon < 0) { // si la particule se heurte à gauche et à  
 this.dx = -this.dx; droite, son sens change  
 this.couleur = 'rgb(' + r + ',' + g + ',' + b + ')'; et aussi sa couleur

}   
   
 if (this.centerY + this.rayon > window.innerHeight   
 || this.centerY - this.rayon < 0) {  
 this.dy = -this.dy; // si la particule se heurte en   
 this.couleur = 'rgb(' + r + ',' + g + ',' + b + ')'; } haut et en bas,  
 son senschange et aussi sa couleur

this.centerX += this.dx; // valeur de sa position x à chaque update = x + dx  
 this.centerY += this.dy; // valeur de sa position y à chaque update = y + dy

} }