#8 : Animation

1) Principe de l'animation

❑ Afin de comprendre le principe des animations dans le canvas préparez ce fichier **anim.html** :

```
<!doctype html>
<html lang="fr">
<head>
	<meta charset="utf-8">
	<title>Animation dans le Canvas</title>
	<script src="anim.js"></script>
</head>
<body>
	<h1>Le beau Canvas !</h1>
	<canvas id="myCanvas" width="800" height="600" style="border:1px black solid"></canvas>
</body>
</html>
```

❑ Et voici le fichier javascript pour dessiner l'animation : **anim.js**

```
// Global variables
var canvas;
var context;
var largeurcarre = 20;
var posX = 0;
var posY = 50;

window.onload = function()   // At start
{
	//init our canvas and graphic context
	canvas = document.getElementById("myCanvas");
	context = canvas.getContext("2d");

	// Render the scene
	dessine();

	// launch animation
	setInterval(animate,20);
}

function animate()	// Animation core
{
		posX += 5;  // move the mob !
		dessine();
}

function dessine()	// Draw the scene
{
	// Clear
	context.clearRect(0, 0, canvas.width, canvas.height);

	// Draw Mob
	context.beginPath();
	context.rect(posX, posY, largeurcarre, largeurcarre);
	context.closePath();
	context.fill();
}
```

❑ Lancez cet exemple ! Comme vous pouvez le voir pour créer des animations il faut au démarrage :

1. Initialiser le **canvas** et son **contexte graphique**
2. Dessiner la scène initiale
3. Lancer **setInterval** qui va se charger de lancer la fonction **animate** toutes les 20 ms.

❑ La fonction **animate** se charge toutes les 20 ms de **modifier la position des objets** et de **dessiner la scène**.

❑ Et enfin la fonction **dessine efface la scène** en dessinant un rectangle sur tout le canvas et **dessine les objets à leur nouvelle place**.

En route vers les jeux vidéo !

Problématique :
Les jeux vidéo sont souvent basés sur un principe simple : un objet (ou un personnage) se déplace en interagissant avec son environnement. Quels sont les algorithmes qui permettent de gérer ces déplacements et ces interactions ?

Déroulement de l'activité :
Le but de cette activité est d'arriver progressivement à écrire un script JavaScript, relativement complexe. Ce script devra permettre d'animer une multitude de « balles » à l'écran (les chocs et les rebonds étant gérés).
Des « missions » sont proposées, la réussite d'une mission entraînant le « déblocage » de la mission suivante. Vous travaillez donc à votre rythme, en « débloquant » plus ou moins rapidement les nouvelles missions.

Mission 1 (principes de base : script 1)
Une balle traverse l'écran de gauche à droite (sans rebond sur les parois).
Quelques pistes :
– procédure pour créer un canvas (code HTML)
– procédure pour créer un context (code JavaScript)
– utilisation des méthodes suivantes : clearRect, beginPath, **arc**, closePath et fill
– utilisation de la méthode **setInterval** (boucle d'animation)

Mission 2 (rebond sur les parois : script 2)
La balle continue à se déplacer horizontalement, mais elle doit maintenant rebondir sur les bords.

Mission 3 (déplacement sur x et y : script 3) : La balle doit maintenant se déplacer « en diagonale ».

Mission 4 (animation de plusieurs balles : script 4)
Même animation que précédemment, mais avec plusieurs balles (5 minimum).

Mission 5 (vecteur vitesse initiale aléatoire : script 5)
Les coordonnées d'origine de chaque balle seront aléatoires. Ensuite, la direction initiale et la vitesse de chaque bille sont aussi aléatoires. Indice : **Math.random.**

Mission 6 (chaque balle a sa propre couleur (aléatoire))
Chaque balle doit avoir une couleur différente (et aléatoire)

Mission 7 (déviation (non réaliste) lors d'une collision balle contre balle)
Lors de la collision entre 2 balles, les 2 balles concernées doivent voir leurs trajectoires modifiées. L'idéal serait ici d'utiliser la conservation de la quantité de mouvement et la conservation de l'énergie, mais cela nous ferait entrer dans des considérations très complexes (trigonométrie, projection de vecteur, ...). Nous pourrons donc nous contenter du changement de sens des balles (comme si le choc avait été frontal).
Indices : - réfléchir sur le calcul de la distance entre 2 balles (utilisation du théorème de Pythagore).
- utilisation de **Math.sqrt (racine carrée)**
Il est cependant important de bien comprendre qu'il est indispensable d'utiliser les 2 lois physiques citées ci-dessus si l'on veut simuler des chocs (élastiques) « réalistes »... Pour la mission 9 ?

Mission 8 (détection des collisions entre les balles)
Une variante de la mission précédente, en cas de choc, les 2 balles concernées disparaissent.
Indice : la méthode **splice**.