3 minutes

# Les bases des animations

## Les étapes d'une animation

Les étapes d'une animation sont les suivantes :

1 - On efface tout ou la partie du `canvas` animée.

2 - On dessine.

Si vous utilisez des transformations vous pouvez avant et après l'étape de dessin appeler `save()` et `restore()`.

## Les fonctions pour exécuter les animations

Ces étapes sont répétées à intervalle de temps régulier ou lors d'une interaction utilisateur.

Il existe trois fonctions utilisables :

`setInterval()` lorsqu'il n'y a pas besoin d'interaction utilisateur. Elle permet d'exécuter une fonction tous les x millisecondes.

`setTimeout()` permet d'exécuter une fonction au bout d'une durée spécifiée en millisecondes.

`requestAnimationFrame()` demande au navigateur d'exécuter une fonction de rappel pour exécuter une animation lors du prochain rafraîchissement. Le taux de rafraîchissement, dépend du taux de rafraîchissement de l'écran mais sera en moyenne de 60 fois par seconde. Si plusieurs onglets sont ouverts ou si l'onglet n'est pas visible le taux de rafraîchissement sera abaissé plus ou moins suivant le navigateur.

## Projet : l'horloge

Voici le résultat :
https://codesandbox.io/embed/js-c24-l11-1-5g75d

Il s'agit d'un code un peu complexe et donc n'hésitez pas à y revenir plus tard.

Pour ce cas d'utilisation nous allons choisir le `setInterval()` car l'horloge n'a besoin d'être dessinée qu'une fois par seconde et n'a pas à réagir aux interactions utilisateurs.

**Translation au centre du `canvas`.**

Nous allons modifier l'origine (`0, 0`) du `canvas` pour la mettre en son centre :

```
const rayon = canvas.height / 2;
```

```
ctx.translate(rayon, rayon);
```

Il sera ainsi beaucoup plus simple de dessiner en utilisant des rotations car tous les dessins seront dessinés en utilisant des angles par rapport à ce point.

### Dessin du contour et du fond de l'horloge

Nous allons ensuite dessiner l'horloge, pour le moment c'est simple :

```
function drawFace() {
  ctx.beginPath();
  ctx.arc(0, 0, rayon * 0.9, 0, 2 * Math.PI);
  ctx.fillStyle = "white";
  ctx.fill();
  ctx.strokeStyle = "blue";
  ctx.lineWidth = 4;
  ctx.stroke();
}
```

Nous dessinons simplement un arc de cercle qui fait `0,9 * rayon` du `canvas`. (Oui, le `canvas` est un carré mais parle du rayon du plus grand cercle pouvant être dessiné dans ce carré).

Il s'agit d'un cercle et donc les angles de départ et d'arrivée sont 0 et `2 * Math.PI`.

Nous dessinons le contour en bleu avec `stokeStyle`.

### Dessin des heures

Cela se complexifie nettement, il faut dessiner 12 nombres sur un cercle imaginaire autour du centre de l'horloge. Problème : si nous faisons une rotation le dessins des textes des nombres font également se faire avec l'angle défini et les nombres seront donc de plus en plus penchés pui à l'envers.

```
function drawHours() {
  ctx.font = "40px arial";
  ctx.fillStyle = "black";
  ctx.textBaseline = "middle";
  ctx.textAlign = "center";
  for (let heures = 1; heures < 13; heures++) {
    ctx.save();
    const angle = (heures * Math.PI) / 6;
    ctx.rotate(angle);
    ctx.translate(0, -rayon * 0.75);
    ctx.rotate(-angle);
    ctx.fillText(heures.toString(), 0, 0);
    ctx.restore();
```

```
  }
}
```

La logique est la suivante, pour chaque heure, nous commençons une nouvelle transformation. Nous appelons donc `save()`.

Ensuite, nous calculons l'angle qui est de `x * 2 * Math.PI / 12`. Il faut en effet découper les `2 * Math.PI` radians du cercle en 12 angles égaux, un pour chaque heure.

L'angle est ensuite fonction de l'heure qui va définir la valeur de `x`.

Après une rotation, nous allons déplacer l'origine à `-rayon * 0,75` : autrement dit nous nous plaçons au niveau d'où nous devons dessiner le nombre de l'heure.

Si nous dessinons toute suite, le nombre serait penché, nous faisons donc une rotation inverse. Ensuite nous pouvons dessiner le nombre en `0, 0` car rappelez-vous nous venons de faire une translation et avons donc déplacé l'origine à l'endroit où nous devions dessiner.

Pourquoi faire cela ? Pour ne pas avoir à calculer les coordonnées du nombre en utilisant la trigonométrie (il faudrait utiliser `sinus` et `cosinus` dans ce cas).

Ensuite, nous appelons `restore()` et nous avons de nouveau pour le prochain nombre un angle de 0 et l'origine au centre du `canvas` et de l'horloge.

Cette partie n'est pas évidente, faites des essais pour bien comprendre.

### Dessiner les traits pour les heures et les minutes

Pour dessiner les traits c'est beaucoup plus simple car nous voulons dessiner les lignes en suivant les angles. Autrement dit ils sont penchés mais c'est ce que nous voulons.

```
function drawMinutes() {
  ctx.save();
  ctx.lineWidth = 5;
  ctx.strokeStyle = "black";
  ctx.lineCap = "butt";
  for (let i = 0; i < 60; i++) {
    ctx.beginPath();
    ctx.moveTo(rayon * 0.9 - 1, 0);
    if (i % 5 !== 0) {
      ctx.lineTo(rayon * 0.9 - 1 - 10, 0);
    } else {
      ctx.lineTo(rayon * 0.9 - 1 - 20, 0);
    }
    ctx.stroke();
    ctx.rotate(Math.PI / 30);
  }
  ctx.restore();
```

```
}
```

Ici la logique est simple, il y a 60 traits à tracer pour toutes les minutes dans une heure.

Si `i % 5` cela veut dire que nous sommes en face d'un nombre et que nous sommes à une tranche de 5 minutes. Par convention on y dessine un trait plus long.

Notez que nous effectuons la rotation **après chaque trait**. Comme cela nous dessinons directement le premier trait à l'angle 0, pour 0 minute.

La rotation est de `2 * Math.PI / 60` car nous devons diviser le cercle en 60 angles égaux.

### Dessiner les traits pour les aiguilles

Pour dessiner les aiguilles la logique est légèrement plus complexe.

Nous récupérons l'heure, les minutes et les secondes depuis un objet `Date`.

**Commençons par les heures** :

```
function drawHoursHand(heures, minutes) {
  ctx.save();
  ctx.rotate((heures % 12) * (Math.PI / 6) + (minutes * Math.PI) / 360);
  ctx.lineWidth = 14;
  ctx.beginPath();
  ctx.moveTo(0, -0.3 * rayon);
  ctx.lineTo(0, 0);
  ctx.stroke();
  ctx.restore();
}
```

Seule la rotation est un peu complexe, mais le raisonnement est simple.

Il y a 24 heures, il faut donc "convertir" l'heure en système utilisant 12 heures. Avec un modulo nous avons :

```
24 % 12 = 0;
1 % 12 = 1;
2 % 12 = 2;
...
12 % 12 = 0;
13 % 12 = 1;
…
```

C'est exactement ce que nous voulons. Ensuite il faut nous occuper de l'angle :

```
(heures % 12) * (Math.PI / 6)
```

C'est la même chose que pour les nombres. Cela nous donne l'angle de l'heure 1, 2, 3 etc.

Sauf que les horloges ne fonctionnent pas comme cela : l'aiguille des heures ne passe pas en 1 seconde de l'heure x à l'heure y. Elle avance progressivement entre les deux nombres à une allure lente.

Il faut donc compléter l'angle. Par exemple s'il est 14H45 nous voulons que l'aiguille des heures marque presque 3 et pas 2.

Nous ajoutons donc :

```
(minutes * Math.PI) / (6 * 60)
```

Ce que nous faisons c'est : `(x * 2 * Math.PI) / (12 * 60)`.

En fait nous prenons l'angle des minutes `2 * Math.PI / 60` et ensuite nous devons le diviser par 12 car nous voulons rester dans la tranche entre l'angle de l'heure x et de l'heure x + 1.

En simplifiant nous obtenons bien : `minutes * Math.PI / 360`.

**Même principe pour l'aiguille des minutes** :

```
function drawMinutesHand(minutes, secondes) {
  ctx.save();
  ctx.rotate((minutes * Math.PI) / 30 + (secondes * Math.PI) / 1800);
  ctx.lineWidth = 12;
  ctx.beginPath();
  ctx.moveTo(0, -0.6 * rayon);
  ctx.lineTo(0, 0);
  ctx.stroke();
  ctx.restore();
}
```

Nous commençons par obtenir l'angle de la minute exacte :

`x * 2 * Math.PI / 60` ce qui donne `x * Math.PI / 30`.

Comme nous voulons que l'aiguille se déplace entre la minute présente et la minute suivante progressivement nous ajoutons l'angle des secondes :

C'est le même angle que pour les minutes `x * Math.PI / 30`.

Nous voulons cependant rester entre nos deux minutes et donc nous devons diviser par 60. Nous obtenons bien : `(secondes * Math.PI) / 1800`.

Le dessin de l'aiguille des secondes est très simple, l'angle de rotation est le même que pour le dessins des traits des minutes.

Cet exemple n'est pas évident, c'est normal si vous n'arrivez pas à le reproduire avant quelques tentatives !

## Code de la vidéo

Vous pouvez retrouver le code des vidéos sur Github.