

Animation d'un cheval au galop avec requestAnimationFrame

Compléter les commentaires 1. à 9. du script ci-dessous :

```
1 <body>
2   <h1>Animation - Cheval au galop</h1>
3   
4   <div id="info">
5     Frame affichée : <span id="compteur">0</span> / 4
6   </div>
7   <script>
8     // 1. ..
9     const frames = [
10       "images/cheval_0.png",    // phase appui
11       "images/cheval_1.png",    // envol avant
12       "images/cheval_2.png",    // suspension
13       "images/cheval_3.png",    // réception
14       "images/cheval_4.png",
15     ];
16     // 2. ..
17     const img      = document.getElementById("cheval");
18     const compteur = document.getElementById("compteur");
19
20     let index      = 0;    // indice courant dans le tableau
21     let tick       = 0;    // compteur de callbacks
22
23     const VITESSE = 8;    // changer de frame toutes les 8 fps
24                          // (~60fps/8  7-8 images par seconde)
25     // 3. ...
26     function animer() {
27       tick++;
28       // 4. ...
29       if (tick % VITESSE === 0) {
30         img.src = frames[index];           // mise à jour de l'
image
31         // 5. ...
32         compteur.textContent = index;
33         // 6. ...
34         index = (index + 1) % frames.length;
35       }
36
37       // 8. ...
38       requestAnimationFrame(animer);
39     }
40     // 9. ...
41     requestAnimationFrame(animer);
42   </script>
43
```

```
44 </body>  
45 </html>
```

Partie 2

COURS

Niveau : 1ère NSI

Prérequis : HTML/CSS (mise en page, centrage), JavaScript (tableaux, fonctions, modulo)

Durée estimée : 1h —

2.1 1. Objectif

Réaliser une animation fluide d'un cheval au galop en combinant :

- **HTML/CSS** : afficher et centrer une image à l'écran
- **JavaScript** : gérer une liste d'images et boucler dessus avec le modulo
- **requestAnimationFrame** : synchroniser l'animation avec le rafraîchissement de l'écran

2.2 2. Principe de l'animation par défilement d'images (sprite)

Un cheval au galop peut être animé en affichant successivement plusieurs images (*frames*), chacune représentant une phase du mouvement.

```
1 frame 0 → frame 1 → frame 2 → frame 3 → frame 0 → ...
```

On stocke les chemins vers ces images dans un **tableau JavaScript**, et on affiche l'une après l'autre en boucle.

Partie 3

Les fonctions de callback

En javascript, pour réaliser des appels de fonction à intervalle de temps régulier, il faut utiliser un mécanisme de *callback*, unique solution pour ne pas bloquer une exécution lorsque celle-ci est trop *couteuse*.

Par exemple, le programme suivant devrait établir un décompte de 200 à 0, mais en réalité, celui-ci va entraîner un blocage du système : (**ne pas tester**)

```

1 <html>
2   <div id="bip" class="display"></div>
3 <script >
4   function bip () {
5       document.getElementById("bip").innerHTML = counter + "
6       secondes restantes";
7       bip();
8   }
9   bip();
</script>

```

La pile d'appel est alors :

pile d'appel
main()
bip()
bip()
...

Une fonction de rappel (aussi appelée *callback* en anglais) est une fonction passée dans une autre fonction en tant qu'argument, qui est ensuite invoquée à l'intérieur de la fonction externe pour accomplir une sorte de routine ou d'action, en temps régulier. On utilise ainsi les méthodes `setInterval`, `setTimeout` et `requestAnimationFrame()`

Les méthodes `setTimeout` et `setInterval` sont des méthodes de l'objet `Window`. Ce sont des processus indépendants qui, quand ils sont lancés par une instruction, ne bloquent pas l'affichage du reste de la page ni les actions de l'utilisateur. Elles permettent de placer dans la pile des prochaines fonctions à exécuter une fonction particulière. Javascript exécute cette pile fonction après fonction dans l'ordre de la pile.

- `SetTimeout` indique un délai avant exécution
- `setInterval` déclenche une opération à intervalles réguliers
- `ClearTimeout` interrompt le décompte de `setTimeout` et de `setInterval`

Syntax : `let intervalID = scope.setInterval(function ou bloc de code, [delay (en ms)]);`

Ce compteur de temps est identifié de manière unique : `window.setInterval` renvoie une valeur numérique qui permet cette identification.

Une fonction lambda peut être un argument.

3.0.1 Méthode `requestAnimationFrame()`

C'est aussi une méthode de *callback* pour réaliser une animation en javascript.

La méthode `requestAnimationFrame()` de l'interface `Window` indique au navigateur qu'on souhaite exécuter une animation. Elle demande au navigateur d'appeler une fonction de rappel (callback en anglais) fournie par l'utilisateur avant le prochain rafraîchissement.

Dans cet exemple, issu du site [MDN](#), un élément est animé pour 2 secondes (2000 millisecondes). L'élément se déplace à une vitesse de 0.1px/ms vers la droite. Sa position relative (en pixels CSS) peut donc être calculée en fonction du temps écoulé entre le début de l'animation (en millisecondes) et $0.1 * \text{ecoule}$. La position finale de l'élément est située 200px ($0.1 * 2000$) à droite de sa position initiale.

```
1  const element = document.getElementById("un-element-a-animer");
2  let debut;
3
4  function iteration(chrono) {
5      if (debut === undefined) {
6          debut = chrono;
7      }
8      const ecole = chrono - debut;
9
10     // Math.min() est utilisée ici afin de s'assurer que l'élément
11     // s'arrête
12     // exactement à 200px
13     const compteur = Math.min(0.1 * ecole, 200);
14     element.style.transform = `translateX(${compteur}px)`;
15     if (compteur < 200) {
16         requestAnimationFrame(iteration);
17     }
18 }
19 requestAnimationFrame(iteration);
```

Complément : Animation avec un cheval au galop : [tuto](#)

3.1 3. HTML — Structure de la page

```
1  <!DOCTYPE html>
2  <html lang="fr">
3  <head>
4      <meta charset="UTF-8">
5      <title>Cheval au galop</title>
6      <link rel="stylesheet" href="style.css">
7  </head>
8  <body>
9      
10     <script src="animation.js"></script>
11 </body>
12 </html>
```

Points clés : - Un seul élément `<img>` avec l'identifiant `cheval`. - On ne crée pas plusieurs balises `<img>` : c'est JavaScript qui change la source.

3.2 4. CSS — Centrer l'image à l'écran

```

1 body {
2   margin: 0;
3   height: 100vh;
4   display: flex;
5   justify-content: center;
6   align-items: center;
7   background-color: #87CEEB; /* ciel bleu */
8 }
9
10 #cheval {
11   width: 300px;
12 }

```

Rappel Flexbox :

Propriété	Rôle
display: flex	Active le mode Flexbox sur le <body>
justify-content: center	Centre horizontalement
align-items: center	Centre verticalement
height: 100vh	Le body occupe toute la hauteur de la fenêtre

3.3 5. JavaScript — L'animation

3.3.1 5.1 Le tableau des images

```

1 const frames = [
2   "cheval_0.png",
3   "cheval_1.png",
4   "cheval_2.png",
5   "cheval_3.png",
6   "cheval_4.png"
7 ];

```

On regroupe tous les chemins dans un tableau. Modifier l'animation revient simplement à modifier ce tableau.

3.3.2 5.2 Le modulo pour boucler

Pour parcourir le tableau en boucle infinie, on utilise l'opérateur **modulo (%)** :

```

1 let index = 0;
2
3 // À chaque étape :
4 index = (index + 1) % frames.length;

```

index	(index + 1) % 5
0	1
1	2
4	0 ← repart au début

Propriété clé : $n \% \text{longueur}$ reste toujours dans $[0, \text{longueur} - 1]$.

3.3.3 5.3 requestAnimationFrame

`requestAnimationFrame(callback)` demande au navigateur d'appeler `callback` avant le prochain affichage à l'écran (environ 60 fois par seconde).

C'est préférable à `setInterval` car : - synchronisé avec l'écran (pas de saccades) - mis en pause automatiquement si l'onglet est inactif (économie de ressources)

3.3.4 5.4 Contrôler la vitesse avec un compteur

60 appels par seconde, c'est trop rapide pour une animation de cheval. On n'avance dans le tableau qu'une fois tous les N appels :

```

1  const img      = document.getElementById("cheval");
2  const frames = ["cheval_0.png", "cheval_1.png", "cheval_2.png",
3                  "cheval_3.png", "cheval_4.png", "cheval_5.png"];
4
5  let index      = 0;
6  let compteur   = 0;
7  const VITESSE = 8; // changer une frame toutes les 8 images écran
8
9  function animer() {
10     compteur++;
11
12     if (compteur % VITESSE === 0) {
13         img.src = frames[index];
14         index = (index + 1) % frames.length;
15     }
16
17     requestAnimationFrame(animer); // relance la boucle
18 }
19
20 requestAnimationFrame(animer); // démarre l'animation

```

3.4 6. Schéma de fonctionnement

```

1  requestAnimationFrame(animer)

```

```

2
3
4  animer() appelée
5
6  compteur++
7
8  compteur % VITESSE === 0 ?
9
10 OUI    img.src = frames[index]
11        index = (index+1) % frames.length
12
13        requestAnimationFrame(animer)
14
15        (boucle infinie)

```

3.5 7. Code complet (en un seul fichier)

Voir le fichier `cheval_animation.html` fourni avec ce cours.

3.6 8. Pour aller plus loin

- **Déplacement horizontal** : modifier la propriété CSS `left` ou `transform: translateX(...)` à chaque frame pour faire traverser le cheval d'un bord à l'autre.
 - **Plusieurs chevaux** : créer plusieurs éléments `<img>` avec des décalages de phase différents (index de départ différent).
 - **Sprites CSS** : regrouper toutes les frames dans une seule image et utiliser `background-position` pour les afficher.
-

3.7 9. Résumé des notions mobilisées

Notion	Utilisation
Tableau JS	Stocker les références des images
Modulo %	Boucler sur le tableau indéfiniment
<code>document.getElementById</code>	Accéder à l'élément <code></code>
<code>requestAnimationFrame</code>	Synchroniser l'animation avec l'écran
Flexbox CSS	Centrer l'image dans la page
<code>img.src</code>	Changer dynamiquement l'image affichée