1. Animation d'un titre

Créez un nouveau dossier pour ce TD : votre compte / Multimedia / TD2.

Téléchargez td2_exo1.htm et svgelement.js dans ce dossier. C'est la base pour cet exercice.

1.1. Étude de `td2_exo1.htm`

On commence par étudier rapidement `td2_exo1.htm`. Dans le `<body>`, il y a un dessin SVG comprenant un texte. Certaines de ses propriétés sont définies dans les styles du document. Il y a ensuite un bandeau de boutons qui appellent des fonctions JS, chacune devant définir une animation différente.

Dans la partie `<script>`, on commence par affecter une variable avec l'élément texte. Ensuite on trouve les fonctions associées aux boutons. On a une fonction `reset()` pour annuler une animation et tout remettre en état.

La première animation est entièrement programmée et montre le principe pour les autres. Elle utilise la bibliothèque GSAP *GreenSock Animation Platform*, voir cette réponse pour l'origine du nom. La mise en place de l'animation est en deux parties :

a. Il y a une initialisation de la situation initiale avec un appel à `reset()`, suivi d'un appel à `gsap.set(élément, paramètres)` pour définir une valeur particulière pour l'attribut qu'on va modifier durant l'animation. Dans le cas de `animation1`, on change la position y du texte.
b. Ensuite, il y a la cible de l'animation avec `gsap.to(élément, paramètres)`.

Dans `animation1()`, on fait partir le texte de $y = -60$ et on l'amène à $y = 0$, au centre du dessin. Ce sont des coordonnées relatives à une transformation qui est mise en place sur le texte, par l'attribut `transform-origin`.

Après avoir essayé cette animation, modifiez le paramètre `ease`, essayez différentes valeurs que vous trouverez dans la documentation. Notez qu'on doit écrire *type.bords* et que *type* vaut `power1`, `back`, etc. et *bords* vaut `in`, `out` ou `inOut`. Essayez différentes valeurs, gardez celle qui vous convient.

Il reste une dernière chose à comprendre. Dans les deux méthodes, `gsap.set` et `gsap.to`, il y a des lignes commentées. Elles encapsulent la valeur à changer dans un `attr: {...}`. Voici pourquoi :

a. D'abord utilisez l'inspecteur pour voir ce qui se passe avec la balise `<text>`. Quand vous lancez l'animation, vous devez voir des changements sur un attribut `transform`, une matrice. Ça bouge vite mais on le voit bien. En fait, gsap ne modifie pas du tout l'attribut `y` !
b. Commentez les lignes `y:` dans les deux appels et décommentez les deux groupes de lignes `attr: { ... },`. Surveillez ce qui se passe. Maintenant, on voit l'attribut y qui est animé.

La raison est que le paramètre `y` est un raccourci vers une transformation matricielle qui s'applique à la position définie par les attributs `x` et `y` de l'élément. Et c'est pour éviter cette transformation, et vraiment modifier l'attribut qu'on utilise `attr: {...}`.

1.2. Changement d'opacité

Dans `animation2()`, on va jouer sur l'opacité du titre. Il doit commencer par être invisible et devenir pleinement opaque à la fin. C'est l'attribut `opacity` qu'il faut faire passer de 0 à 1 ; 0 = transparent, 1 = opaque.

Lancez l'animation. On doit voir le titre disparaître d'un coup (initialisation) puis réapparaître lentement (animation).

Inspectez ce qui se passe pour l'élément `<text>`. Au lieu de modifier directement l'attribut `opacity`, GSAP modifie l'attribut `style`. Encapsulez les changements dans un paramètre `attr: { "opacity": 0 },` pour `gsap.set` et `gsap.to` et constatez que c'est maintenant l'attribut qui est animé.

1.3. Changement de couleur

La fonction `animation3()` va modifier la couleur, `fill`. Dans l'initialisation, la fonction `gsap.set`, mettre `fill: 'red'` et dans l'animation, mettre pareil mais avec `green`. GSAP sait interpoler entre deux couleurs.

Vous verrez avec l'inspecteur que l'interpolation passe par une définition en `rgba(...)`, et comme précédemment, c'est le style qui est concerné.

On peut remarquer que l'animation fait passer par un jaune foncé, ce qui n'est pas très agréable. C'est parce que les couleurs sont représentées par des composantes RGB *red green blue* variant entre 0 et 255. Or le rouge est représenté par (255,0,0) et le vert par (0,255,0). Et donc l'interpolation entre ces deux triplets fait passer par (127,127,0), du jaune foncé.

Il y a d'autres considérations, comme le fait que l'œil est moins sensible au rouge qu'au vert, et encore moins au bleu. Donc du rouge moyen (127,0,0) paraît moins lumineux que du vert moyen (0,127,0). Regardez les rampes colorées sur cette page. La rampe verte paraît plus lumineuse que les deux autres, mais ça dépend de l'étalonnage de votre écran : température de couleur et gamma (demandez à l'enseignant des précisions si vous voulez en savoir plus).

On voudrait animer du rouge au vert mais en restant dans des couleurs vives. Il faut utiliser une autre représentation des couleurs, appelée HSL *hue saturation lightness*, voir ces explications. On vous propose d'animer entre `hsl(0, 100%, 50%)` et `hsl(150, 100%, 40%)`.

1.4. Changement de taille de police

Le but de `animation4()` est d'animer l'attribut `style` de `"font: 0px sans-serif;"` à `"font: 24px sans-serif;"`. Vérifiez avec l'inspecteur si c'est le style ou un attribut qui est mis à jour.

1.5. Changement d'espacement des lettres (optionnel)

Le but de `animation5()` est d'animer `letter-spacing` de 20px à 0px. C'est une valeur de l'attribut `style`. Il y a deux problèmes :

- Il faudra encapsuler dans un `attr: { style: "...;" },` parce qu'il ne faut pas toucher aux autres styles.

- Ça ne suffit pas. Il va y avoir un problème de police de caractères. La police voulue est `24px sans-serif`. Il faut le spécifier.

C'est à dire qu'il faut mettre ceci dans les deux appels à `gsap`, en changeant NN :

```
attr: {
  style: "font: 24px sans-serif; letter-spacing: NNpx;"
}
```

Mais il y a mieux. On peut écrire les propriétés directement :

```
font: '24px sans-serif',
'letter-spacing': 'NNpx',
```

En JS, on doit mettre des quotes ou double quotes autour des noms d'attributs contenant un tiret. Ou alors on peut les écrire en CamelCase comme ceci :

```
font: '24px sans-serif',
letterSpacing: 'NNpx',
```

JavaScript reconnait le CamelCase et le transforme automatiquement en snake-case avec des tirets.

2. Animation d'une voiture

Téléchargez td2_exo2.htm et Green_Car.svg dans le dossier du TD.

2.1. Étude de `td2_exo2.htm`

Le document ne contient pas directement d'image SVG. Une voiture est sous forme de balise `<img>` dans un grand `<div>` représentant le terrain.

Comme précédemment, une fonction `reset()` remet tout en place, et il y a trois fonctions pour animer la voiture.

2.2. Combiner trois mouvements

La première, `animation1()` montre comment faire un mouvement très simple.

On voudrait y ajouter une rotation de 90° après le déplacement, puis un mouvement vertical. (On admet que cette voiture peut tourner sur place). Les instructions sont écrites, décommentez-les et essayez.

Le problème, c'est que toutes les animations démarrent en même temps au lieu de l'une après l'autre. Il y a deux manières de résoudre le problème :

- soit en rajoutant des délais cumulatifs, c'est à faire dans `animation1()`
- soit en utilisant une *timeline*, dans `animation2()` voir plus bas.

Il suffit de rajouter un attribut `delay: N` aux 2e et 3e animations, juste après `duration`. La valeur du délai est le temps cumulé, en secondes, où doit commencer l'animation considérée, donc 6 et 6+1.

Le défaut de cette méthode est qu'il faut recalculer tous les délais suivants si on change l'une des durées.

2.3. Timeline

La fonction `animation2()` montre la meilleure solution. On place les différents mouvements dans une séquence et on définit exactement quand ils doivent se produire. Par défaut, les événements ajoutés ont lieu les uns après les autres.

Décommentez les lignes et essayez. Modifiez l'une des durées pour voir si la séquence est toujours bonne.

On peut améliorer un peu la qualité de l'animation en décalant les mouvements 2 et 3. Pour cela, on ajoute un 3e paramètre, appelé position parameter : `TL.to(element, {cible..., duration...}, position)`.

Par exemple, ajoutez la chaîne `">-0.5"` aux mouvements 2 et 3 pour les faire démarrer légèrement avant le premier. `>` signifie après le précédent mouvement et `-0.5` met 1/2 seconde en avance. Ça ne permet pas de faire prendre un virage à la voiture, mais

2.4. Suivi d'un chemin

L'équipe GSAP propose une extension gratuite appelée MotionPath. Elle permet de faire bouger un élément du DOM (ou d'un SVG) le long d'une trajectoire prédéfinie.

La fonction `animation3()` montre comment faire. C'est toujours `gsap.to(element, {cible..., duration...})` qui définit le mouvement, mais la cible est maintenant un objet `motionPath` qui spécifie la trajectoire à suivre. La documentation explique le principe.

Dans l'état actuel de `animation3()`, la voiture va en ligne droite. Vous allez ajouter différentes choses.

- Ajoutez d'autres points de passage dans le tableau `path`. Ce sont des objets `{x: abscisse, y: ordonnée}`. Mettez-en deux autres, au tiers et au 2/3 de la distance et plus ou moins haut, mais sans exagérer.
- Il serait bien que la voiture s'oriente selon la trajectoire. Pour cela, il faut ajouter l'option `autoRotate: true`.
- Pour que la voiture commence et finisse bien orientée, il faut ajouter un premier point où seul x est augmenté, par exemple `{x: 100, y: 200}` et un autre avant-dernier, par exemple `{x: xmax()-100, y: 200}`. Ainsi la trajectoire redevient horizontale.

Le mieux pour dessiner un chemin est d'utiliser des *path SVG* comme dans le TD1, plus exactement, la chaîne qu'on place dans l'attribut `d` d'un `<path>`. On peut alors utiliser la directive `C` pour définir des courbes cubiques, voir la doc.