1. Animation d'un titre

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

Téléchargez td2_exo1.htm. C'est la base pour cet exercice. Il utilise le script svgelement.js du TD1.

1.1. Étude de `td2_exo1.htm`

On commence par étudier rapidement `td2_exo1.htm`. Il y a un dessin SVG comprenant un texte, et une définition de style et d'un dégradé (*gradient*) qui servira pour la dernière question. Le texte et le gradient sont mis dans des variables. Il y a ensuite un bandeau de 6 boutons qui déclenchent des fonctions JS, chacune devant définir une animation différente.

La première animation fonctionne et montre le principe pour les autres. C'est en deux parties :

a. Il y a une initialisation de la situation initiale avec un appel à `reset()`. Cette fonction remet les attributs à des valeurs communes. Et il y a 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 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 (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. C'est assez rapide, 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 `fill-opacity` qu'il faut faire passer de 0 à 1.

Il y a un premier piège, c'est que `fill-opacity: 0,` n'est pas du bon JSON. Il faut entourer le nom de l'attribut avec des guillemets : `"fill-opacity": 0,`.

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 `fill-opacity`, c'est dans `style` qu'il y a un changement. Encapsulez les changements dans un paramètre `attr: { "fill-opacity": 0 },` pour `gsap.set` et `gsap.to` et constatez que c'est maintenant l'attribut qui est altéré.

1.3. Changement de couleur

La fonction `animation3` va modifier la couleur, `fill`. Dans l'initialisation, mettre `white` et dans l'animation, mettre `black`. GSAP sait interpoler entre ces deux couleurs. Encore une fois, il faut encapsuler les changements dans un `attr: {}`.

Vous verrez avec l'inspecteur que l'interpolation passe par une définition en `rgba(...)`.

Au lieu de passer de `white` à `black`, vous pouvez essayer `hsl(90, 100%, 100%)` à `hsl(270, 50%, 0%)` avec `ease: "power3.in"`.

1.4. Changement de taille de police

Le but de `animation4` est d'animer de `"font: 0px sans-serif;"` à `"font: 24px sans-serif;"`. C'est l'attribut `style` qui est concerné et il n'y a pas besoin d'encapsuler dans `attr: {...},`. Vérifiez le mécanisme interne avec l'inspecteur.

1.5. Changement d'espacement des lettres

Le but de `animation5` est d'animer `letter-spacing` de 20 à 0. Mêmes vérifications et remèdes que précédemment.

1.6. Animation d'un dégradé

La dernière fonction va être un peu plus longue que les autres. Il s'agit de changer le mode de remplissage du texte, un dégradé. Le début de l'initialisation est en place. Il reste à définir des couleurs pour les points de contrôle.

Un dégradé SVG est défini par des points de contrôle. Chaque point définit une couleur à un certain emplacement 0%..100% du dégradé. Par exemple, au début on veut du vert, donc on écrit `<stop stop-color="#00FF00" offset="0%"/>`. Si on veut du cyan au milieu, on écrit `<stop stop-color="#00FFFF" offset="50%"/>`. Et si on veut que ça se finisse par du rouge, on mettra `<stop stop-color="#FF0000" offset="100%"/>`. Les couleurs intermédiaires sont interpolées.

Ici, le dégradé `<linearGradient x1="0" x2="100%" y1="0" y2="0">` est défini horizontalement, car `y1` = `y2`. On l'applique au texte par l'attribut `fill="url(#gradient)"`.

Vous allez définir les couleurs initiales et finales des 3 points de contrôle. Si vous n'avez aucune idée, vous pouvez commencer par du blanc partout et finir par du bleu, gris clair et rouge.

2. Gare de triage

Téléchargez td2_exo2.htm. Créez un sous-dossier `img` et téléchargez LocomotiveD.svg et LocomotiveG.svg dans ce sous-dossier.

2.1. Analyse de l'existant

Étudiez le projet proposé :

- Quelle est la structure SVG qui est construite ? Les rails sont des instances de la classe `Rail`. On lui donne deux points, x_1, y_1 de départ et x_2, y_2 d'arrivée, avec $x_1 > x_2$ (sinon les locos avanceront tête en bas). Le dessin fait appel à deux `<path>`. Celui de l'extérieur dessine en pointillés, l'autre dessine la voie qui sera suivie par les locomotives. Le calcul du chemin, l'attribut `d` est vraiment compliqué, ne vous en occupez pas.
- Comment se fait l'animation de la locomotive ? La méthode `Rail.moveAlong` permet de construire le mouvement très facilement. Elle fait appel au plugin MotionPath.

Alors premièrement, il manque une propriété pour que les locomotives soient bien alignées avec les rails, `alignOrigin`. Allez voir sa documentation. Il faut faire de sorte que le point servant d'alignement soit au centre-bas des locomotives.

Il manque également le calcul de la longueur des rails. Voir la fin du constructeur. Lisez le cours pour savoir comment on calcule la longueur d'un chemin.

2.2. Travail à faire

Dessiner une autre voie pour arriver au schéma suivant. Les graduations sont de 10 en 10 et sont créées par la classe `Grille`.

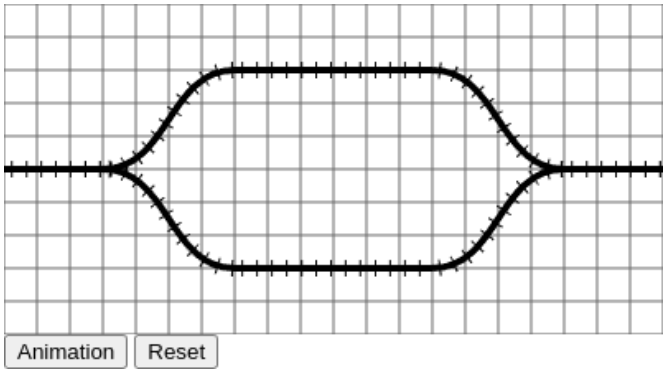

Figure 1: rails à reproduire

Faites en sorte que `locomotiveG` (rouge) parte de la droite et aille à gauche, à peu près en même temps que `locomotiveD` (verte), mais en passant par les rails du bas : chacun sur sa voie. Vous pourrez faire varier la vitesse des locomotives.

Pour cela, il faut définir une *timeline* pour chaque locomotive. Il y a déjà `TLD`, celle de `locomotiveD`. Il y aura `TLG`, celle de `locomotiveG`. La *timeline* `TLG` doit être temporellement inversée pour que la loco rouge fasse le trajet en sens inverse. Utilisez la méthode `TLG.reverse()` *après* l'avoir créée et remplie.

Il faudra ajouter les deux *timelines* à la *timeline* globale, `TL`. Cherchez dans le CM1 comment ajouter une animation et faire qu'elle démarre en même temps. Profitez-en pour regarder la documentation de la méthode add.