Créez un nouveau dossier pour ce TP : votre compte \ Multimedia \ TP1.

On reprend les mécanismes des TD1 et TD2 pour dessiner un objet SVG dynamique.

Voici le résultat à obtenir, figure 1. Il s'agit d'un afficheur qui peut servir à représenter un niveau, une tension, une force, une température... Les leds s'allument de bas en haut, en fonction d'une valeur fournie en entrée. Le nombre de leds est configurable, les marges également.

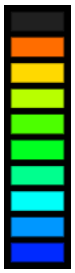

Figure 1: bargraph

Cet afficheur n'est composé que de rectangles, l'encadrement et les leds. La complexité de l'exercice réside dans le positionnement et dimensionnement de ces rectangles.

1. Conception générale

On part du fichier base_gsap.htm (clic droit, enregistrer la cible du lien sous...) accompagné de svgelement.js.

Renommez `base_gsap.htm` en `tp1.htm`.

Dans l'élément `<svg>` supprimez l'attribut `width`, laissez `height`, et rajoutez un attribut `viewBox="0 0 2 10"`. L'attribut `viewBox` ainsi que `height` permet de calculer `width` manquant. Le *bargraph* fera 2 unités de large et 10 de haut, comme dans la figure 1. Évidemment, on peut changer ça à volonté.

1.1. Classe

L'afficheur doit être réalisé sous la forme d'une classe. Chaque instance pourra être insérée dans un élément `<svg>`. Le constructeur doit recevoir l'élément `<svg>` en premier paramètre, nommé `parent`, et également d'autres paramètres de configuration comme la taille et le nombre de leds.

Voici le début :

```
class BarGraph {
  constructor(parent, width, height, nleds) {
    // groupe pour regrouper tout le contenu
    const g = parent.appendElement("g")
    // encadrement en noir
    g.appendElement("rect", {x: 0, y: 0, width, height, fill: "black"})
    ...
  }
}
```

Avec cette classe, il suffira de faire ceci pour afficher un *bargraph* :

```
// ajouter le bargraph
const svg = SVGelement.fromSelector("#dessin")
const bargraph = new BarGraph(svg, 2, 8, 8)

// définir la valeur du bargraph
bargraph.setValue(5)
```

La dernière ligne montre une méthode permettant de changer le nombre de leds allumées. Il faut donc ajouter cette méthode à la classe mais laissez-la vide pour l'instant.

Vous avez remarqué comment le rectangle d'encadrement est défini en JS ? Les attributs `width` et `height` n'ont pas de valeur. C'est une possibilité de JS, si des variables portent ces noms, alors on peut directement en faire des champs dans un objet. Donc au lieu d'écrire `width: width, height: height`, on se contente de `width, height`.

1.2. Aspects graphiques

Le *bargraph* doit pouvoir être entièrement configuré, parce que la *viewBox* peut être quelconque. Pour commencer, il y a la largeur et la hauteur. L'encadrement est donc facile à dessiner, voir le listing ci-dessus.

On voudrait que les leds soient séparées du bord par une marge identique de tous les côtés, et par le même espace entre elles. Cette marge doit être définie par une proportion, entre 0 et 1. Voici l'effet de différentes marges, figure 2.

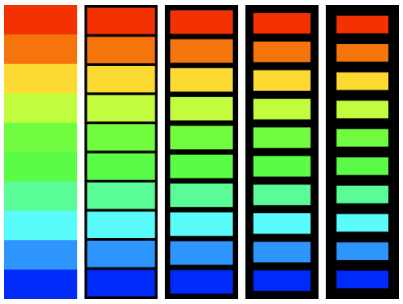

Figure 2: marges 0.0, 0.1, 0.2, 0.3 et 0.4

Donc finalement, on doit tracer des rectangles selon le schéma de la figure 3. Les petites flèches sans nom ont toutes la même taille, m. C'est lié à l'espacement donné indirectement par le paramètre *marge*. Ce paramètre indique une proportion de la hauteur : quand *marge* est nul, les leds sont jointives ; quand *marge* vaut 1, les leds ont une hauteur nulle, et sont donc invisibles.

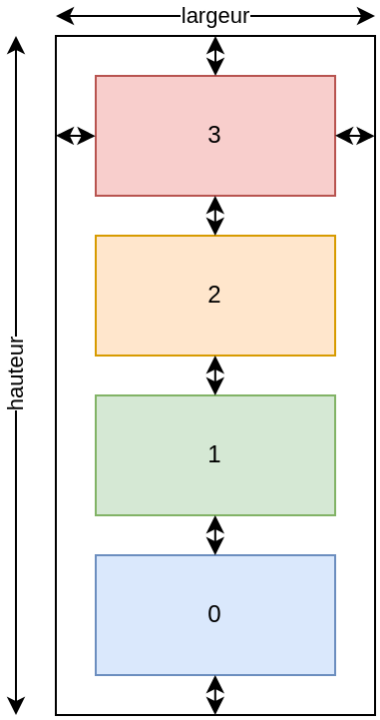

Figure 3: calculs géométriques

Vous devez écrire les équations de placement des rectangles représentant les leds. NB: leur couleur sera donnée ultérieurement. C'est à dire que vous devez écrire les formules mathématiques qui donnent la position du coin haut-gauche, x et y, ainsi que calculer la largeur et la hauteur de chaque led. Remarquez que x, *width* et *height* sont identiques pour toutes les leds ; seul y change.

Indications :

- La hauteur totale $hauteur_{totale}$ est la somme des hauteurs des N_{leds} leds et de $N_{leds}+1$ marges m. Mais tant qu'on ne connaît pas m, on ne peut pas calculer la hauteur des leds.
- La marge m est une fonction de $marge$, de $hauteur_{totale}$ et de N_{leds}. Comment déterminer cette fonction ?
  On a deux cas extrêmes :
  - Quand $marge = 0$, $m = 0$.
  - Et quand $marge = 1$, la hauteur des leds est nulle. Donc m découpe la $hauteur_{totale}$ en $N_{leds}+1$ espaces égaux.
- Donc on se rend compte que c'est une fonction linéaire de $marge$ quand on considère $hauteur_{totale}$ et N_{leds} constants. C'est à dire que $m = k*marge$ avec k constante dépendante de $hauteur_{totale}$ et N_{leds}. Tracez un graphique avec $marge$ en abscisse allant de 0 à 1 et m en ordonnée et déduisez l'expression de m.
- Une fois qu'on connaît m, il est extrêmement facile de calculer x.
- La largeur des leds vaut $largeur_{totale} - 2m$.
- La hauteur des leds est à peine plus compliquée.

Complétez la définition de la classe à partir de ce qui suit, en complétant la boucle qui crée les rectangles des leds. Pour l'instant, ils ont une couleur unique, ici, un vert clair. Et pour l'instant, toutes les leds sont allumées.

```
let width = ...
let height = ...
let x = ...
for (let i=0; i<nleds; i++) {
  let y = ...
  let fill = "Chartreuse"
  let led = g.appendElement("rect", {x, y, width, height, fill})
}
```

Renommez les variables, ou placez ce code dans une fonction afin de distinguer `this.width` et `width`.

1.3. Couleurs des leds

Pour donner une couleur spécifique à chaque led, il suffit de modifier le bout de code précédent comme ceci :

```
for (let i=0; i<nleds; i++) {
  let y = ...
  // couleur HSL à mettre
  let teinte = 220 * i / (nleds-1)
  let fill = `hsl(${teinte} 100% 50%)`
  let led = g.appendElement("rect", {x, y, width, height, fill})
}
```

La teinte est une chaîne au format `hsl(teinte saturation luminosité)`. C'est une autre manière de définir les couleurs qu'avec `rgb(rouge vert bleu)` ou `#RRVVBB`. Dans le système HSL, la teinte est un angle entre 0° et 360° sur la roue des couleurs, figure 4.

Figure 4: roue des couleurs HSV

La saturation entre 0 et 100% indique si la couleur est pure, ou mélangée avec du gris. La luminosité est entre 0% (noir) et 100% (blanc) ; 50% est une bonne valeur pour notre application. Essayez ce sélecteur HSL.

Donc, ici, on choisit une teinte entre 0° (rouge) et 220° (bleu) en fonction du numéro de led. Le calcul est un peu bizarre, parce que le bleu doit être affiché en bas, quand $i = N_{leds} - 1$, et le rouge en haut quand $i = 0$, sauf si on fait un calcul différent avec y.

1.4. Allumage des leds

Il faut commencer à prévoir comment on va mettre à jour les leds allumées par la méthode `setValue` proposée au début. On ne doit pas recréer les rectangles, ça serait trop lent. L'idée est de ne changer que leur couleur. Mais alors, il faut mémoriser les rectangles dans un tableau lors de leur création.

```
class BarGraph {
  constructor(parent, width, height, nleds) {
    // groupe pour regrouper tout le contenu
    const g = parent.appendElement("g")
    // encadrement en noir
    g.appendElement("rect", {x: 0, y: 0, width, height, fill: "black"})
    // créer les leds
    this.leds = []
    const m = ...
    width = ...
    height = ...
    let x = ...
    for (let i=0; i<nleds; i++) {
      let y = ...
      // couleur HSL à mettre
      let fill = ...
      let led = g.appendElement("rect", {x, y, width, height, fill})
      // ajouter cette led à la fin du tableau
      this.leds.push(led)
    }
  }
}
```

Mais alors la question, c'est : dans quel ordre les leds sont-elles dans le tableau ? Il serait pratique que la led bleue tout en bas soit `this.leds[0]` et que les leds suivantes, en remontant, soient dans les cases suivantes, pour les avoir dans l'ordre croissant d'allumage.

Il est donc préférable d'avoir la coordonnée y qui décroit quand i augmente, afin de remplir `this.leds` avec les rectangles allant de bas en haut. Voir les numéros des leds dans la figure 3. Une autre approche, si les leds sont créées de haut en bas, c'est de stocker la nouvelle led au début du tableau avec la méthode `unshift` au lieu de `push`.

Il reste à programmer la méthode `setValue(v)`. Elle doit colorer les leds qui sont en dessous de v et mettre en gris très foncé celles qui sont au dessus. Il faut utiliser la méthode `this.leds[i].setAttribute("fill", fill)`. Ça doit marcher quand le paramètre v est un réel quelconque.

Testez la totalité de ce logiciel : différentes tailles, nombre de leds, marges et affectation des valeurs.

2. Animation

Le but est de faire changer la valeur affichée par le *bargraph*, c'est à dire le nombre de leds. On voudrait une animation régulière, donc on va faire appel à `setInterval` (documentation). Voici une fonction à mettre en dehors de la classe, après la création du *bargraph* :

```
function updateBarGraph() {
  let v = Math.random() * nleds
  bargraph.setValue(v)
}
setInterval(updateBarGraph, 350)
```

On utilise `setInterval` parce qu'on veut une variation régulière, tandis que `gsap.ticker.add` appelle la fonction aussi souvent que possible, donc à un rythme irrégulier.

On voudrait améliorer l'affichage de la valeur, que ça soit progressif et non pas brusque. C'est dans la méthode `BarGraph.setValue`. Cette méthode modifie l'attribut `fill` des rectangles. Au lieu de le faire avec `this.leds[i].setAttribute`, faites-le avec `gsap.to`. Mettez une courte durée, par exemple 0.2.

Pour un effet un peu plus progressif, vous pouvez ajouter un délai dépendant du numéro de led, p. ex. `{fill, delay: 0.05*i, ...}`. Ça va donner l'impression que les leds s'allument progressivement, comme avec un vrai signal analogique.

Par contre, ce délai croissant fait que les leds du haut restent allumées un peu plus longtemps que celles du bas quand la valeur décroît. Vous pourriez faire en sorte de distinguer deux cas : quand on fournit une nouvelle valeur plus grande que la valeur courante, et quand la nouvelle valeur est plus petite. Dans le second cas, il faut que le délai soit inversement proportionnel au numéro de led.

3. Extensions

3.1. Allumage partiel

Actuellement, les leds s'allument en « tout ou rien ». Modifiez la méthode `setValue` pour qu'une valeur non entière conduise à un allumage partiel de la led correspondant à v. Par exemple, si $v = 0.5$ alors la led 0 s'allume à moitié.

Pour faire cela, il faut d'une part extraire la partie fractionnaire de v. C'est avec `v - Math.floor(v)`.

Ensuite, il faut utiliser cette valeur pour modifier la luminosité de la led, en plus de sa couleur :

```
// teinte de la led
let teinte = 220 * (this.nleds-1-i)/(this.nleds-1)
// luminosité par défaut
let lum = 5
// TODO calculer lum 5..50 en fonction de v
```

```
// couleur HSL à mettre
let fill = `hsl(${teinte} 100% ${lum}%)`
```

Quand la luminosité `lum` vaut 50, alors la couleur est vive. Quand la luminosité vaut 5, peu importe la teinte, la couleur est très sombre. Donc vous devez calculer `lum` pour valoir 50 quand le numéro de la led est inférieur à v, pour valoir 5 quand le numéro de la led est supérieur à v, et une valeur intermédiaire entre 5 et 50 quand le numéro de la led est égal à la partie entière de v, et c'est la partie fractionnaire de v qui indique cette valeur intermédiaire. (Interprétez ça à vorte manière).

NB: quand lum dépasse 50, la teinte tend vers du blanc. Donc ici, on veut faire varier lum entre 5 et 50.

3.2. Autres mises en page

Si vous avez le temps, ajoutez un paramètre supplémentaire au constructeur, `orientation` valant soit `vertical`, soit `horizontal` permettant de changer l'axe du *bargraph*.

Au lieu d'avoir des leds rectangulaires, on pourrait avoir des leds rondes. Le rayon serait alors la plus petite des valeurs entre la largeur et la hauteur, et il faudra recentrer les leds.