1. Cadre de travail

Créez un nouveau dossier pour ce TD, p. ex. : `$HOME/Cours/Multimedia/TD1`.

Pour tous les TD et TP, on utilisera Visual Studio Code. Il y a un *plugin* à installer, « Live Preview » de Microsoft, voir sa documentation. Il permet d'afficher un document HTML dynamique en temps réel. C'est le petit bouton entouré en vert dans la figure 1.

Figure 1: Bouton Show Preview

Par défaut, il est configuré pour rafraîchir l'affichage à chaque changement dans un fichier. Il est souvent préférable de rafraîchir seulement quand on enregistre un fichier. Modifiez les préférences du *plugin* (*extension settings*), item *Auto Refresh Preview* valeur *On changes to Saved Files*.

On peut aussi afficher la console de mise au point ; il faut ouvrir la vue « Output » (ou « Sortie ») et choisir `Embedded Live Preview Console` (ou `Console Préversion en direct incorporée`) dans la case déroulante de l'entête du terminal. Ensuite, il faut arranger les vues pour que la console soit en bas de la fenêtre.

Pour vraiment déboguer, il faut plutôt ouvrir le *preview* dans un navigateur. C'est avec le menu hamburger en haut à droite dans la *preview*, choisir *open in browser*. Là vous pourrez inspecter et utiliser la console. Le projet est automatiquement mis à jour dans le navigateur à chaque changement dans VSC.

2. Dessins manuels simples

On part de ce fichier HTML servant de modèle de base :

```
<!DOCTYPE html>
<html>
<head>
  <!-- voir https://greensock.com/docs/v3/Installation#installer -->
  <script src="https://cdn.jsdelivr.net/npm/gsap@3.12.5/dist/gsap.min.js"></script>
</head>
<body>
  <svg xmlns="http://www.w3.org/2000/svg" width="200" height="200" id="dessin">
  </svg>

  <script>
    // ici, il y aura les scripts JS pour créer, modifier et animer l'image SVG
```

```
  </script>
</body>
</html>
```

Téléchargez-le et renommez-le en `td1_exo1.htm`. Ouvrez-le avec Visual Studio Code et affichez la prévisualisation temps réel.

2.1. Dimensions et zone visible

Commencez par ajouter ceci à l'intérieur de la balise `<svg>` :

```
<line x1="10" y1="10" x2="190" y2="190" stroke="red" stroke-width="4"/>
<line x1="190" y1="10" x2="10" y2="190" stroke="green" stroke-width="5"/>
```

Quelle ligne apparaît au dessus de l'autre ? Intervertissez ces deux lignes et constatez que l'ordre de dessin est important.

Modifier les attributs `width` et `height` de l'image SVG pour arriver à 960x540 pixels. Quelles sont les proportions de cette image, c'est à dire le rapport *largeur/hauteur* ramené à un quotient d'entiers noté *a:b* ? Quelle serait la hauteur à donner si on voulait une proportion 4:3 en gardant la même largeur ?

Définir le système d'unités pour aller de 0 à 16 en x et de 0 à 9 en y. Indication : il faut définir l'attribut `viewBox`. Maintenant, l'image peut avoir la taille que vous voulez, vous n'aurez rien à changer aux coordonnées des éléments graphiques.

Modifiez les coordonnées des lignes pour les voir entièrement dans la nouvelle viewBox :

```
<line x1="1" y1="1" x2="15" y2="8" stroke="red" stroke-width="1"/>
<line x1="15" y1="1" x2="1" y2="8" stroke="green" stroke-width="1.25"/>
```

Remplir l'image avec un rectangle bleu clair (`LightCyan` ou `#DFF`) placé en arrière-plan. Vérifier qu'il remplit totalement l'image en altérant temporairement la `viewBox`.

Enlevez ou commentez les tracés des deux lignes.

2.2. Dessins simples

Dupliquez `td1_exo1.htm` et appelez-le `td1_exo2.htm`.

Dessiner la scène suivante, figure 2, mais sans la grille.

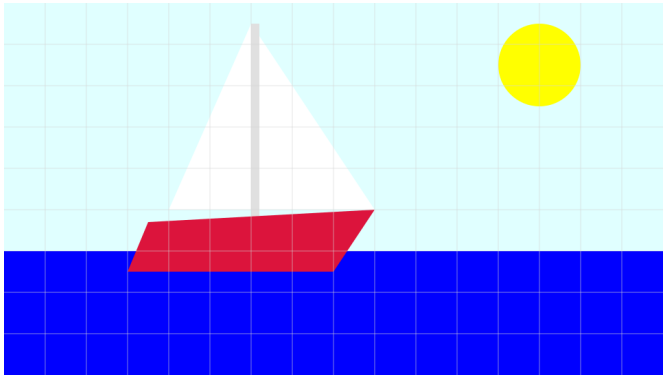

Figure 2: image à reproduire

Conseils : le dessin s'appuie sur le quadrillage 0..16 de gauche à droite, 0..9 de haut en bas. Vous pouvez définir les positions et les couleurs à votre goût.

1. Commencez par la mer. C'est un rectangle bleu qui fait toute la largeur et commençant à la hauteur 6 et allant jusqu'à 9.
2. Ajoutez le soleil. C'est un cercle de rayon 1 placé par exemple en $(13, 1.5)$.
3. Le mât est un rectangle gris clair allant de $(6, 0.5)$ à $(6.2, 5.5)$.
4. La voile est un triangle blanc, mais en SVG les triangles n'existent pas en tant que primitive, on doit utiliser un *path*. Les points de passage sont $(9, 5)$, $(6, 0.5)$ et $(4, 5)$. Pensez à refermer le tracé. Pour faire joli, vous pouvez ajouter les attributs `stroke="black" stroke-width="0.02"`.
5. Le problème, c'est que le mât est masqué par la voile. Réparez cela.
6. La coque est également un *path* $(9, 5)$, $(3.5, 5.3)$, $(3, 6.5)$ et $(8, 6.5)$.

3. Classes pour dessiner

L'objectif est maintenant de dessiner par programmation, afin de pouvoir animer cette image ultérieurement. L'élément `<svg>` sera vide initialement, ou ne contiendra que des éléments statiques, et tous les éléments changeants seront ajoutés ou manipulés par programme.

Dupliquez `td1_exo2.htm` et appelez-le `td1_exo3.htm`.

3.1. Classes pour des éléments graphiques

Voici comment on se propose de travailler :

```
1: const svg = SVGelement.fromSelector("#dessin")
2: const chose = new Chose(svg)

3: class Chose {
4:   constructor(parent) {
5:     // groupe pour englober la chose
6:     const g = parent.appendElement("g")
7:     // bordure autour de la chose
8:     g.appendElement("rect", {x: 1, y: 2, width: 3, height: 4, fill: "Gold"})
```

```
 9:  }
10: }
```

La classe `SVGelement` a pour but de faciliter la création et la modification d'images SVG. Elle n'a pas de constructeur, mais à la place, deux méthodes statiques : `fromSelector` et `fromElement`. On doit fournit un sélecteur à la première ; ce sélecteur désigne l'élément `<svg>` concerné. La seconde méthode demande un élément du DOM, obtenu par exemple par `getElementByID`. Ces deux méthodes, `fromSelector` et `fromElement`, font quelque chose d'assez étonnant : elles rajoutent des méthodes à l'élément du DOM. Par exemple elles lui ajoutent la méthode `appendElement`. Cette technique d'ajouter les méthodes d'une autre classe à un objet existant s'appelle un *mixin*, voir ces explications. Ici, ça facilite la création d'arborescences SVG.

Revenons au bout de code précédent. La première instruction, ligne 1, récupère l'élément `<svg>` sous forme d'une instance de `SVGelement`. La méthode `SVGelement.fromSelector("#dessin")` cherche l'élément désigné et le transforme en *mixin*, c'est à dire qu'elle lui ajoute des méthodes supplémentaires, comme `appendElement`.

Ensuite ligne 2, on construit une instance de `Chose`. Dans cette classe, le paramètre du constructeur, ligne 4, `parent` est le `SVGelement`, c'est à dire l'élément `<svg>` du document. Cet élément possède maintenant la méthode `appendElement` qui permet, ligne 6, d'ajouter un élément enfant dont on indique le nom de balise. Quand on doit dessiner plusieurs choses, on les place dans un groupe `<g>`. Cet élément enfant, `<g>` est lui aussi un *mixin*, donc on peut créer très facilement une hiérarchie, ligne 8.

La classe `SVGelement` est relativement complexe. Téléchargez le fichier svgelement.js (clic droit, enregistrer sous) et ajoutez cette ligne au fichier `td1_exo3.htm` si elle n'y est pas déjà.

```
<script src="svgelement.js"></script>
```

3.2. Application de ce mécanisme

L'exercice consiste à transformer ce qui dessine le paysage et le voilier en classes. l'élément `<svg>` doit redevenir vide et vous devrez définir une classe `Voilier` et une classe `Paysage` ; la seconde devant utiliser la première :

```
<script src="svgelement.js"></script>

<script>
class Voilier {
  constructor(parent) {
    // groupe pour englober le voilier
    const g = parent.appendElement("g")
    // tracés
    g.appendElement("path", {
      d: "...",
      ...
    })
    ...
```

```
  }
}

class Paysage {
  constructor(parent) {
    // groupe pour le paysage
    const g = parent.appendElement("g")
    // tracés
    g.appendElement("rect", {...})
    ...
    // voilier
    const voilier = new Voilier(g)
  }
}

const svg = SVGelement.fromSelector("#dessin")
new Paysage(svg)
```

On pourrait aussi laisser le voilier en dehors du paysage. Avec ce mécanisme, tout est possible facilement.

NB: attention, en JS, quand un nom de propriété contient un tiret, on doit le mettre dans une chaîne :

```
  {
      stroke-width: 0.02,       // NON !
      "stroke-width": 0.02,     // OUI
  }
```

3.3. Transformation sur le voilier (optionnel)

Dupliquez `td1_exo3.htm` et appelez-le `td1_exo4.htm`.

Nous allons modifier la classe Voilier pour pouvoir le déplacer facilement dans le dessin.

Appliquez une transformation sur le groupe défini dans le voilier :

```
class Voilier {
  constructor(parent, x, y) {
    // groupe pour englober le voilier
    const g = parent.appendElement("g")
    // position du voilier
    g.setAttributes({ transform: "translate(4 -1)" })
    // tracés
    ...
  }
}
```

Cette translation est appliquée à tous les éléments du groupe.

Actuellement, le voilier est dessiné à son emplacement final dans le dessin, $x = 6$, $y = 5$. Toutes les coordonnées des rectangles et chemins sont absolues. On voudrait faire en sorte que le voilier soit dessiné autour d'un point de référence situé à la base de son mât.

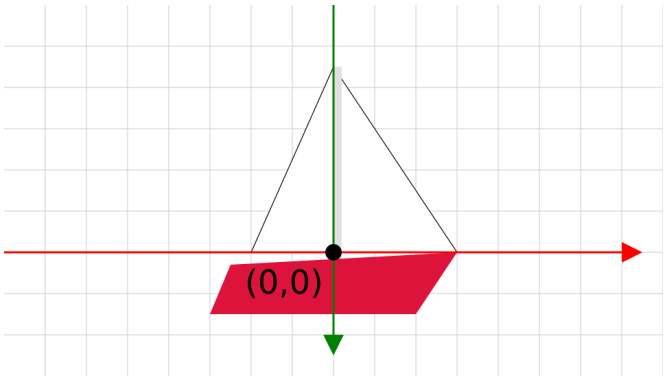

Figure 3: voilier centré en (0,0)

Pour commencer, modifiez la translation du groupe en `transform: "translate(6 5)"`. Elle va directement définir la position du voilier.

Par contre, maintenant le voilier est décalé vers la droite et le bas. Donc il faut corriger toutes les coordonnées : soustrayez 6 de toutes les abscisses et 5 de toutes les ordonnées des éléments du groupe. Le voilier doit revenir à sa place initiale.

Pour finir, on voudrait définir les coordonnées du voilier sous forme de paramètres passés au constructeur :

```
<script src="svgelement.js"></script>

<script>
class Voilier {
  constructor(parent, x, y) {
    // groupe pour englober le voilier
    const g = parent.appendElement("g")
    // position du voilier
    g.setAttributes({ transform: ... })
    // tracés
    ...
  }
}

class Paysage {
  constructor(parent) {
    ...
    // voilier
    const voilier = new Voilier(g, 6, 5)
  }
```

```
}

const svg = SVGelement.fromSelector("#dessin")
new Paysage(svg)
```

Le x=6 et le y=5, paramètres du constructeur doivent être insérés dans une chaîne `translate(...)`. Pour cela, il faut utiliser une chaîne *template* en JS, délimitée avec des accents graves :

```
  constructor(parent, x, y) {
    // groupe pour englober le voilier
    const g = parent.appendElement("g")
    // position du voilier
    g.setAttributes({ transform: `translate(${x} ${y})` })
    ...
```

4. Bilan

On a fait en sorte que les tracés soient réalisés par programmation et non plus par définition statique de balises SVG. L'intérêt est de pouvoir configurer dynamiquement, et animer les dessins. Ce sera l'objet du TD2.