1. Cadre de travail

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

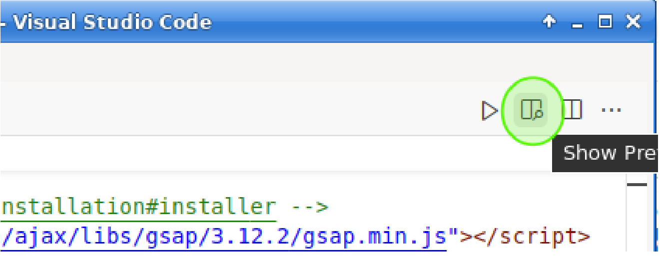

Figure 1: Bouton Show Preview

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.

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://cdnjs.cloudflare.com/ajax/libs/gsap/3.12.2/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"/>
<line x1="190" y1="10" x2="10" y2="190" stroke="green"/>
```

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

Remplir l'image avec un rectangle bleu clair (`LightCyan`) placé derrière les lignes. Vérifier qu'il remplit totalement l'image en altérant temporairement la `viewBox`.

Enlevez les tracés des deux lignes.

2.2. Dessins simples

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

Dessiner la scène suivante (définissez les positions et les couleurs à votre goût).

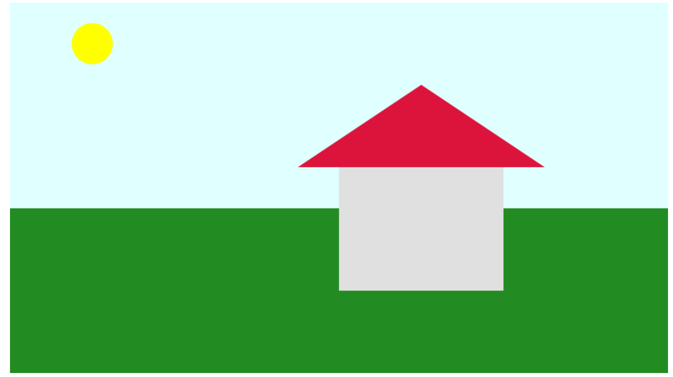

Figure 2: image à reproduire

2.3. Groupement

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

Faites un groupe `<g>` pour englober les murs et le toit de la maison.

Appliquez une transformation sur le groupe : ajoutez-lui un attribut `transform="translate(1 -1)"`. Vérifiez que la translation est appliquée aux deux éléments du groupe.

Maintenant, faites en sorte que les murs et le toit soient définis par rapport au point $(0, 0)$. Ce point devant être au centre bas de la maison et qu'on positionne la maison uniquement en appliquant une translation sur le groupe pour retrouver l'image précédente. Voir le schéma suivant :

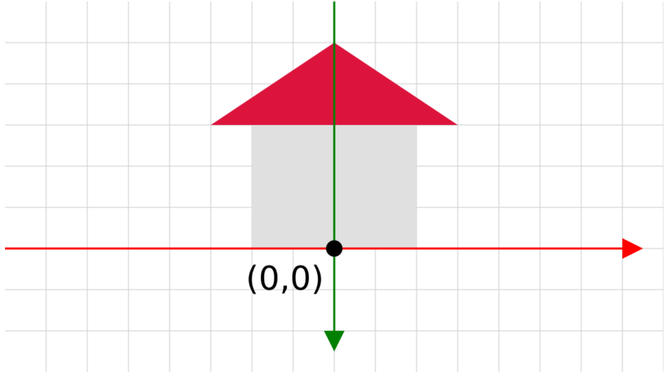

Figure 3: maison centrée en (0,0)

3. Classes pour dessiner

L'objectif est maintenant de dessiner par programmation, afin de pouvoir animer l'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_exo3.htm` et appelez-le `td1_exo4.htm`.

3.1. Classes pour des éléments graphiques

Voici comment on se propose de travailler :

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

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

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 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 on construit une instance de `Chose`. Dans cette classe, le paramètre du constructeur, `parent` est le `SVGelement`, c'est à dire l'élément `<svg>` du document. Cet élément possède maintenant la méthode `appendElement` qui permet d'ajouter un élément enfant dont on indique le nom de balise. Cet élément enfant, `<g>` est lui aussi un *mixin*, donc on peut créer très facilement une hiérarchie.

La classe `SVGelement` est relativement complexe. Téléchargez le fichier svgelement.js (clic droit, enregistrer sous) et ajoutez cette ligne au fichier `td1_exo4.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 la maison en classes. l'élément `<svg>` doit redevenir vide et vous devrez définir une classe `Maison` et une classe `Paysage` ; la seconde devant utiliser la première :

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

<script>
class Maison {
  constructor(parent) {
```

```
    // groupe pour englober la maison
    const g = parent.appendElement("g")
    // tracés
    g.appendElement("rect", {...})
  }
}

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

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

Vous allez sûrement hésiter pour positionner la maison dans le paysage :

- (nul) soit la maison a un emplacement fixe, défini en dur dans son code,
- (facile) soit le constructeur de la maison demande des paramètres pour savoir où il faut la dessiner,
- (difficile) soit le paysage s'occupe de la positionner en lui ajoutant des attributs après création : `maison.setAttributes({transform: "..."})`, mais il faut définir la méthode `setAttributes` dans la classe `Maison` et faire en sorte qu'elle agisse sur l'élément `<g>`, donc qu'il devienne une variable membre.

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.