TP 4

JavaScript : dessins, animations et formulaires

Ce TP est dédié à JavaScript. Sauf mention contraire, aucune modification des fichiers HTML fournis n'est autorisée.

EXERCICE

Exercice 4.1 (Dessin animé)**.** L'objectif est de réaliser (en JavaScript) un dessin similaire à celui de la figure 4.1a.

Question 1 : Télécharger le fichier `dessin.html`. Créer un fichier `dessin.js`. On ne travaillera que dans ce fichier JavaScript.

Implémenter une méthode `dessine()` qui dessine la forme de la figure 4.1a grâce à JavaScript. Quelques fonctions utiles : `fillStyle`, `fillRect`, `beginPath`, `moveTo`, `closePath`, `lineTo`, `arc`... (Voir le cours.)

Le carré noir fait 500 px de côté (on pourra utiliser une constante).

Comme d'habitude, appeler cette méthode dans un auditeur d'évènement sur l'évènement `'load'`, par exemple :

```
window.addEventListener("load", () => {
  dessine();
}
```

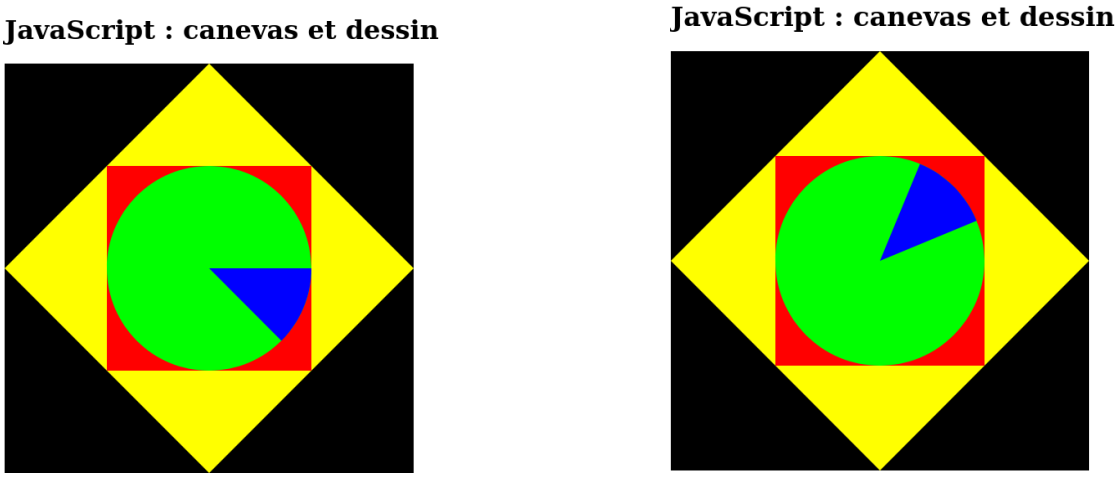
Question 2 : Ajouter des gestionnaires d'évènement pour que :

- un clic de souris décale l'arc de cercle bleu de $\frac{\pi}{8}$ dans le sens trigonométrique (antihoraire) ;
- la pression de la touche « `l` » décale l'arc de cercle bleu de $\frac{\pi}{8}$ dans le sens trigonométrique ;
- la pression de la touche « `d` » décale l'arc de cercle bleu de $\frac{\pi}{8}$ dans le sens antitrigonométrique (sens horaire).

Indications : ajouter une variable globale `theta` qui encode l'angle initial de l'arc. Puis implémenter deux méthodes `decaleAntiTrigo()` et `decaleTrigo()` qui changent la valeur de `theta`, puis rappellent `dessine`. Enfin, ajouter dans les auditeurs concernés les appels à ces deux fonctions, par exemple :

```
document.addEventListener('keypress', (e) => {if(e.key === 'g'){
    decaleTrigo(); }});
```

Un exemple de rendu après 3 clics de souris est donné sur la figure 4.1b.



(a) Dessin initial

(b) Dessin après 3 clics

FIGURE 4.1 – Dessin

Question (optionnelle) 3 : Ajouter un contrôle par les flèches du clavier, en plus des touches « l » et « g ».

Question (optionnelle) 4 : Ajouter un auditeur responsable du fait que, toutes les 2 secondes, l'arc de cercle bleu se décale dans le sens trigonométrique.

Question (optionnelle) 5 : Ajouter deux boutons radio « Sens horaire » et « Sens antihoraire » dans la page HTML (qu'il est autorisé de modifier). (Aucun bouton « envoyer » n'est demandé, ni même un élément `<form>`.)

En fonction de la sélection du bouton, les clics de souris suivants doivent occasionner une rotation horaire ou antihoraire.

■

EXERCICE

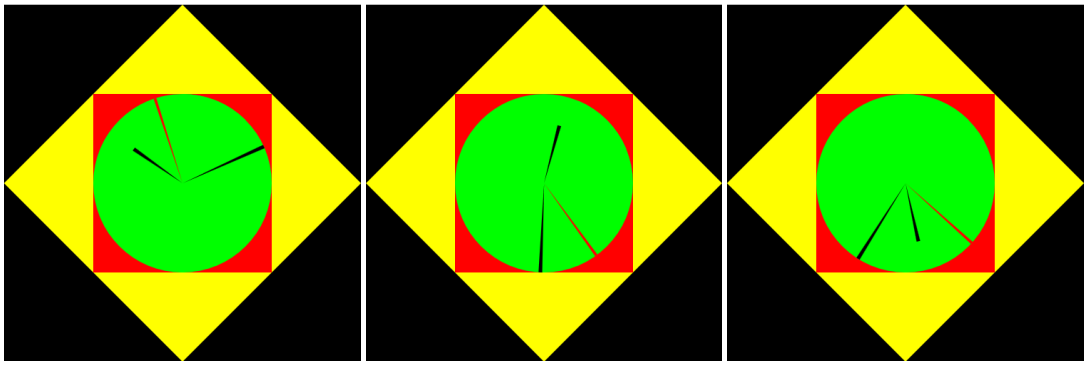
Exercice 4.2 (Horloge)**.** L'objectif est de réaliser (en JavaScript) une horloge similaire à celle de la figure 4.2, à partir de l'exercice précédent.

Question 1 : Dupliquer les fichiers `dessin.html` et `dessin.js` (afin de ne pas supprimer votre exercice précédent) et les nommer `horloge.html` et `horloge.js` (ne pas oublier d'appeler depuis `horloge.html` votre bon fichier JavaScript, donc `horloge.js`).

Supprimer l'arc bleu (ainsi que toute méthode ou variable associée à cet arc) ; celui-ci n'est plus utile dans cet exercice.

Question 2 : Ajouter trois variables globales encodant les angles courants des trois aiguilles de l'horloge :

```
let theta_aiguille_secondes;
let theta_aiguille_minutes;
let theta_aiguille_heures;
```



(a) À 10h10 (b) À 12h30 (c) À 17h35

FIGURE 4.2 – Horloge

Question 3 : Dessiner les trois aiguilles sous forme de trois arcs de cercle (en suivant la technique de l'arc bleu à l'exercice précédent). Varier les couleurs, longueurs et épaisseurs.

Sur la figure 4.2, l'aiguille des secondes est très fine (largeur d'arc : $\frac{\pi}{100}$) et rouge, l'aiguille des minutes est fine ($\frac{\pi}{75}$) et noire, et l'aiguille des heures est plus épaisse ($\frac{\pi}{50}$) et un peu plus courte ($\frac{2}{3}$ de la longueur des autres aiguilles).

Pour tester le rendu, affecter les valeurs des 3 variables encodant les angles initiaux (définies à la question précédente) à des valeurs différentes arbitraires.

Question 4 : Créer une méthode `actualiseHeure()` qui actualise les angles des trois aiguilles en fonction de l'heure exacte. Regarder la documentation de `Date`. Programmer ensuite l'exécution de cette méthode toutes les secondes. Ne pas oublier de redessiner toute l'horloge à la fin de l'exécution de `actualiseHeure()`.

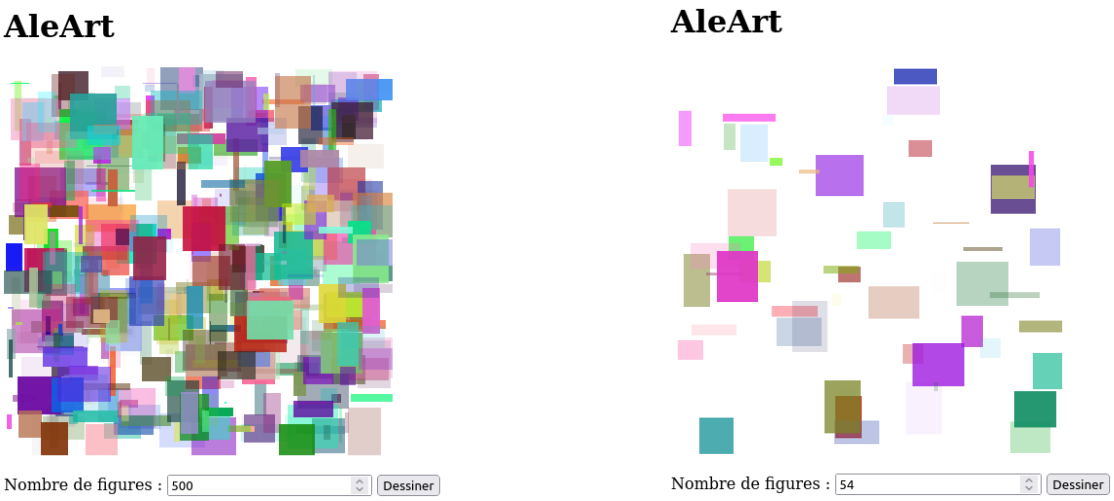
Note : pour programmer un évènement récurrent, on utilise la construction suivante :

```
// Variable globale
let refreshIntervalIdAiguille;

// ...

window.addEventListener('load', function() {
  // ...
  clearInterval(refreshIntervalIdAiguille);
  refreshIntervalIdAiguille = window.setInterval(methode_a_appeler,
    periode_en_millisecondes);
  // ...
}
```

Question 5 : Normalement, pendant la première seconde de l'affichage de la page Web, l'horloge sera à la mauvaise heure. Que faut-il faire (en une seule ligne de code) pour que, dès l'affichage de la page Web, l'horloge s'affiche à la bonne heure ?

Question 6 : Dans la méthode `actualiseHeure()`, ajuster précisément les angles des aiguilles des heures et des minutes afin que ceux-ci dépendent aussi des minutes et des secondes (c'est-à-dire que l'aiguille des heures ne bouge pas d'un cran toutes les heures, mais s'ajuste précisément chaque seconde).

Par exemple, noter la bonne position de l'aiguille des heures (à mi-chemin entre 12h et 13h) sur l'horloge affichant 12h30 (figure 4.2b).



(a) 500 rectangles aléatoires

(b) 54 rectangles aléatoires

FIGURE 4.3 – Dessin

Question (optionnelle) 7 : Ajouter des animations personnalisées, par exemple une aiguille des secondes qui avance de façon continue (et non par secondes), ou bien qui fonctionne par à-coups (c'est-à-dire que, à chaque seconde, l'aiguille avance, mais avec une légère vibration, de façon analogue à une horloge physique).

■

EXERCICE

Exercice 4.3 (Aleart)**.**

Question 1 : Créer un fichier HTML `aleart.html`, contenant un simple canevas de dimensions 400 × 400, et le lier à un fichier `aleart.js`. Écrire une fonction `aleArt` qui affiche 500 rectangles de tailles et couleurs aléatoires (`r,g,b` et transparence) à une position aléatoire dans le canevas. À chaque clic de l'internaute sur le canevas, la fonction est à nouveau appelée, et le dessin est entièrement régénéré.

Un exemple de rendu est donné sur la figure 4.3a (ne pas tenir compte du formulaire à ce stade).

Question (optionnelle) 2 : Ajouter à la page HTML un champ de formulaire pour paramétrer le nombre de rectangles. Lors de l'appui sur un bouton dédié, la page se régénère avec le nombre de rectangles entré.

Un exemple de rendu est donné sur la figure 4.3b.

Question (optionnelle) 3 : Ajouter à la page HTML un menu déroulant permettant de choisir entre des ellipses ou des rectangles. Lors de l'appui sur un bouton dédié, la page se régénère avec le nombre de formes entré.

Question (optionnelle) 4 : Ajouter un formulaire permettant à l'internaute de choisir la taille du canevas de façon dynamique.

■

EXERCICE

Exercice 4.4 (Emojis (exercice pour les plus rapides))**.**

Question (optionnelle) 1 : Créer une page HTML `emojis.html` avec un simple formulaire, et deux boutons « + » et « - ». Voir rendu sur la figure 4.4a (sans les émojis, pour l'instant).

Question (optionnelle) 2 : Lier la page HTML à un fichier JavaScript `emojis.js` qui doit permettre de générer n emojis aléatoires, où n est la valeur du champ. Il doit être possible de modifier n en appuyant sur l'un des deux boutons (auquel cas la valeur du champ progresse elle aussi), ou en entrant une valeur directement dans le champ puis en appuyant sur la touche « entrée ».

Attention : les emojis ne doivent pas tous être regénérés à chaque modification du champ. Par exemple, s'il y a 20 emojis et que l'on entre 13 dans le champ, 7 emojis doivent être enlevés, mais les 13 premiers sont conservés.

Exemple d'utilisation :

1. initialement, le champ vaut 0 ;
2. on appuie ensuite 5 fois sur le bouton « + » : un résultat possible est donné sur la figure 4.4a ;
3. on entre ensuite « 20 » dans le champ, et on appuie sur la touche « entrée » : un résultat possible est donné sur la figure 4.4b (noter que les 5 premiers émojis ont été conservés) ;
4. on entre ensuite « 13 » dans le champ, et on appuie sur la touche « entrée » : un résultat possible est donné sur la figure 4.4c (noter que les 13 premiers émojis ont été conservés).

Indications :

— maintenir un tableau (constant) `TOUS_EMOJIS` d'une sélection d'une dizaine d'émojis de votre choix, parmi lesquelles on choisira dans ce qui suit les émojis aléatoires ;
— maintenir deux variables globales : la valeur du champ (initialement 0), et le tableau courant `tableau_emojis` des émojis (initialement vide) ;
— créer une fonction `actualise_tableau()` qui actualise le tableau `tableau_emojis` en fonction de sa taille courante, et de la valeur du champ ; si la taille courante est supérieure à la valeur, supprimer suffisament d'éléments de `tableau_emojis` ; sinon, en ajouter suffisamment, en piochant à chaque fois un élément de façon aléatoire dans `TOUS_EMOJIS` ;
— créer une fonction `regenere_emojis()` qui regénère dans la page HTML le paragraphe contenant les émojis à partir de `tableau_emojis` (on pourra utiliser `tableau_emojis.join('')`) ;
— ajouter des auditeurs sur les boutons « + » et « - », qui vont simplement actualiser la valeur du champ, puis appeler `actualise_tableau()` et enfin `regenere_emojis()` ;
— ajouter un auditeur sur le champ, qui se déclenche sur la touche « entrée », et fait les actualisations nécessaires à partir des fonctions prédéfinies. Pour vérifier que la chaîne entrée est bien un nombre, on peut utiliser :

```
function est_un_nombre(chaine){
  return /^\d+$/.test(chaine);
}
```

Question (optionnelle) 3 : S'assurer que le bouton « - » est désactivé si la valeur du champ est < 1.

(a) 5 emojis (b) 20 emojis (c) 13 emojis

FIGURE 4.4 – Emojis aléatoires

■