Documentation Projet Morpion :

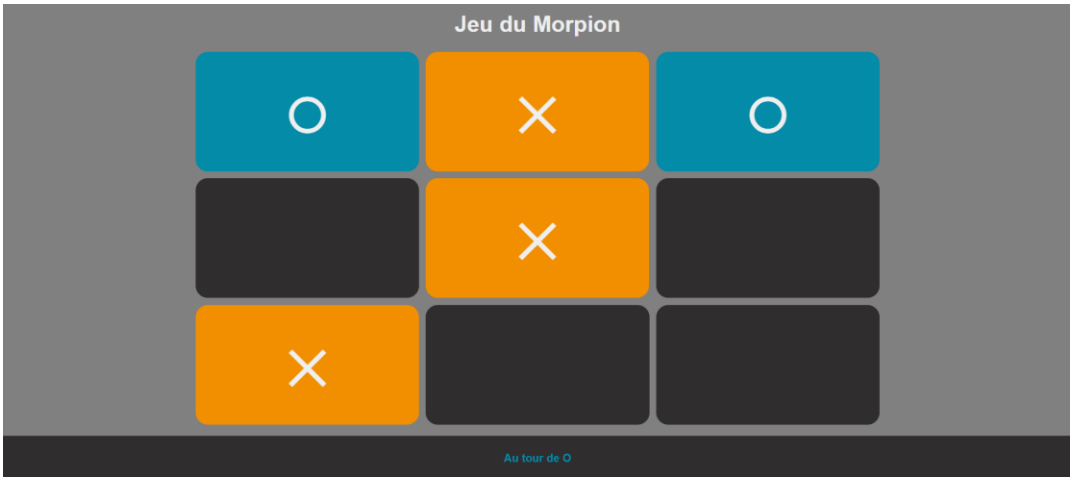
1) Présentation rapide

- Nom du projet : Morpion (Tic-Tac-Toe)
- Objectif : Créer un jeu de morpion fonctionnel pour deux joueurs en utilisant exclusivement les technologies de mise en forme.
- Contexte : Défi technique / Approfondissement des sélecteurs CSS avancés.

2) Fonctionnalités

Le projet simule une application complexe sans aucun script :

- Gestion du tour par tour : Alternance automatique entre les joueurs X et O.
- Marquage des cases : Une fois une case cochée, elle ne peut plus être modifiée.
- Interface intuitive : Design épuré avec un plateau de 3×3.
- Réinitialisation : Utilisation d'un bouton reset natif pour recommencer une partie instantanément.



3) Partie technique simplifiée

Ce projet repose sur une architecture "No-JS", où la logique de programmation est entièrement déportée dans la feuille de style :

- Gestion d'état via Checkbox Hack : Utilisation de 18 input (9 pour le joueur X, 9 pour le joueur O). L'état du jeu est stocké directement dans le DOM grâce à la pseudo-classe :checked.
- Moteur de victoire algorithmique : Détection des lignes, colonnes et diagonales gagnantes via des combinaisons de sélecteurs complexes (ex: input.x:checked + input + input.x:checked + input + input.x:checked ~ .result).
- Système de variables dynamiques : Centralisation de l'UI dans le bloc :root. Les messages de victoire, les couleurs et le tour actuel sont injectés via des variables CSS (--result, --turn).
- Mise en page et animations : * CSS Grid : Structure du plateau 3×3 auto-adaptative.
    - Transformations 3D : Utilisation de rotate3d pour animer l'apparition des symboles lors du clic.
    - Pseudo-éléments : Injection des symboles via ::after et la propriété content.

```
/* X Wins */
input.row.x:checked
  + input
  + input.x:checked
  + input
  + input.x:checked
  ~ div.result,
input.x:checked
  + input
  + input
  + input
  + input
  + input
  + input.x:checked
  + input
  + input
  + input
  + input
  + input
  + input.x:checked
  ~ div.result,
#x1:checked ~ #x5:checked ~ #x9:checked ~ div.result,
#x3:checked ~ #x5:checked ~ #x7:checked ~ div.result {
  --result: "X a gagné la partie!";
  --win-color: var(--x-color);
  display: flex;
}
/* O Wins */
input.row.o:checked
  + input
  + input.o:checked
  + input
  + input.o:checked
  ~ div.result,
input.o:checked
```

Moteur de jeu 100% CSS : Corrélation entre l'état visuel du plateau et la structure logique des sélecteurs complexes pour une application légère et sans JavaScript.

4) Difficultés / Ce que j'ai appris

Ce projet m'a permis de relever des défis de logique "déclarative" peu communs :

- Traduire l'algorithmie en sélecteurs : Le plus grand défi a été de remplacer les boucles if/else par des sélecteurs de frères adjacents (+) et généraux (~).

J'ai dû modéliser mathématiquement chaque combinaison de victoire pour que le CSS "comprenne" quand la partie est finie.

- Gestion de la priorité (Cascade) : J'ai appris à manipuler finement la spécificité des sélecteurs. Par exemple, forcer l'état "Game Over" avec !important sur certaines variables pour outrepasser la logique de tour par tour une fois qu'un gagnant est détecté.
- Architecture HTML rigoureuse : En CSS pur, on ne peut cibler que les éléments qui suivent. J'ai donc dû structurer mon HTML de manière très précise pour que les input placés en haut du document puissent piloter l'affichage des éléments situés plus bas dans le DOM.
- Optimisation de l'UX sans script : J'ai réussi à implémenter un bouton "Nouvelle partie" en utilisant simplement le comportement natif d'un button type="reset", prouvant qu'une connaissance approfondie du HTML standard peut éviter l'usage de scripts inutiles.