Tic Tac Toe Games en JS

Le jeu JavaScript Tic-Tac-Toe est un exemple simple de jeux qu’on peu programmer en JavaScript.

La création de jeux est la branche la plus populaire de la programmation. C'est aussi un moyen étonnant d'apprendre les bases d'un langage de programmation tout en créant un projet simple et intéressant.

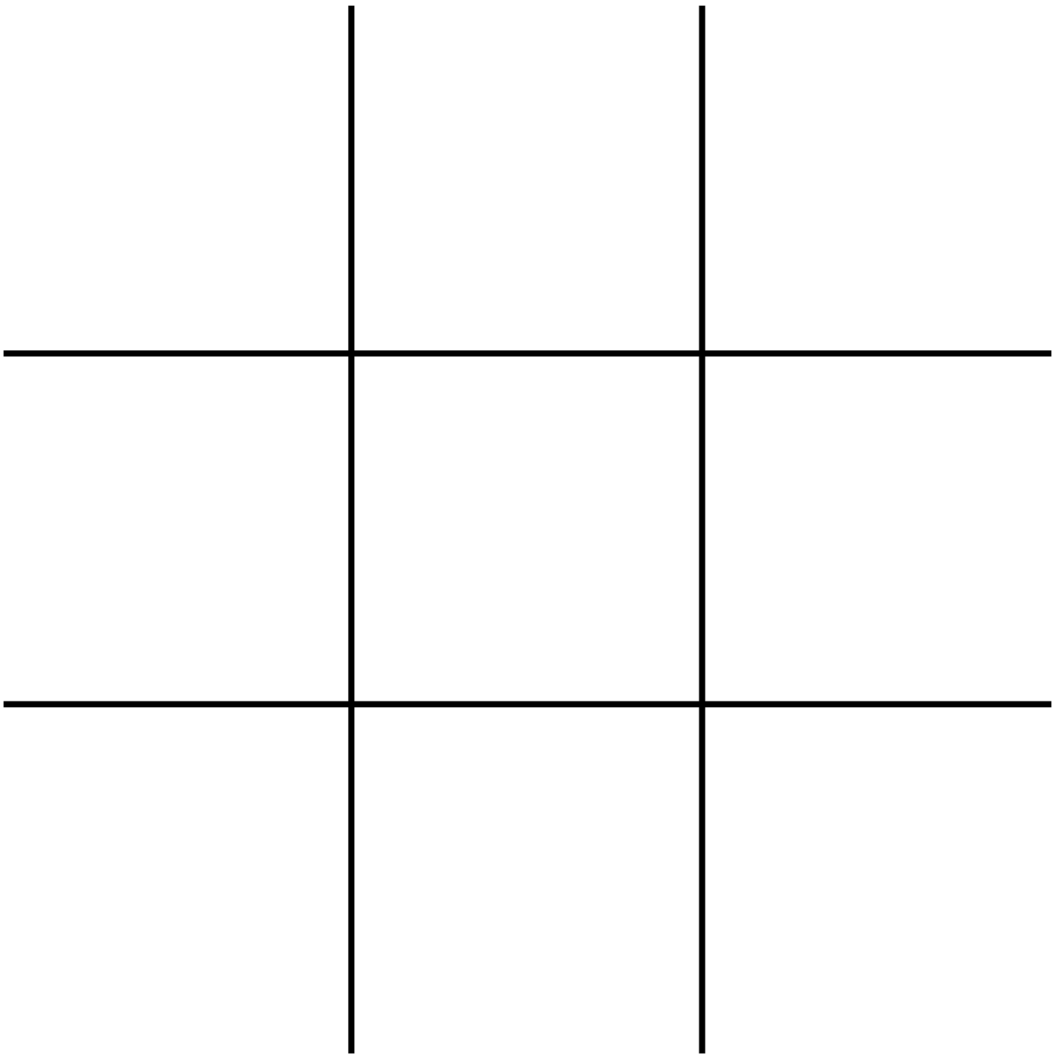
À commencer par le code HTML, qui permet d'attribuer des balises id et des classes aux éléments de notre jeu, tels que le plateau et les cellules. La partie la plus longue de la réalisation du jeu de morpion est le code JavaScript, qui rend le jeu interactif. Enfin, nous vous montrerons comment personnaliser le jeu à l'aide de CSS.

Nous devons séparer différentes parties de code dans différents fichiers. Pour notre jeu JavaScript Tic-Tac-Toe, nous n'utiliserons que trois fichiers différents car il s'agit d'un jeu simple. Dans le fichier index.html, nous attribuerons des classes à tous les constructeurs distincts de notre jeu. Nous allons styliser notre jeu dans le style.css. Enfin, nous écrirons notre script dans le script.js.

Le jeu commencera avec le personnage x, ce qui fait automatiquement de x le premier joueur.

A commencer par HTML.

ce jeu est fait d'un plateau avec des cellules à l'intérieur, nous allons construire les cellules avec l'élément . Cela signifie que nous allons simplement créer des conteneurs génériques que nous styliserons plus tard. Notre conseil sera aussi simple que possible.



un jeu de tic-tac-toe nécessite 9 cases (3x3). Au lieu d'un menu complet, nous ajouterons simplement un message à la fin avec le bouton de redémarrage. C'est ce que nous allons créer dans le corps de notre HTML.

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta http-equiv="X-UA-Compatible" content="IE=edge">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <link rel="stylesheet" href="style.css">
 <title>Tic-Tac-Toe - Malkiel VADNAI </title>
</head>
<body>

</body>
```

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

et ensuite dans le corps, nous avons créé notre tableau et nommé sa classe et son identifiant. Nous avons également attribué la classe "cellule" aux cellules à l'intérieur du tableau. Nous avons ajouté quelques éléments supplémentaires pour notre message gagnant et le bouton de redémarrage.

```
<body>
 <div class="board" id="board">
    <div class="cell" data-cell></div>
    <div class="cell" data-cell></div>
    <div class="cell" data-cell></div>
    <div class="cell" data-cell></div>
    <div class="cell" data-cell></div>
    <div class="cell" data-cell></div>
    <div class="cell" data-cell></div>
    <div class="cell" data-cell></div>
    <div class="cell" data-cell></div>
  </div>
  <div class="winning-message" id="winningMessage">
    <div id="winningMessageText"></div>
    <button id="restartButton">Restart</button>
  </div>
</body>
```

Nous créons maintenant du code pour le jeu JavaScript Tic-Tac-Toe. Cette partie est ce qui rend le jeu ludique

```
const PLAYER_X_CLASS = 'x'
const PLAYER_O_CLASS = 'circle'
const WINNING_COMBINATIONS = [
  [0, 1, 2],
  [3, 4, 5],
  [6, 7, 8],
  [0, 3, 6],
  [1, 4, 7],
  [2, 5, 8],
  [0, 4, 8],
  [2, 4, 6]
]

const cellElements = document.querySelectorAll('[data-cell]')
const boardElement = document.getElementById('board')
const winningMessageElement = document.getElementById('winningMessage')
const restartButton = document.getElementById('restartButton')
```

```
const winningMessageTextElement = document.getElementById('winningMessageText')
let isPlayer_O_Turn = false
```

Nous avons utilisé les balises d'identification que nous avons attribuées dans le `index.html` pour enregistrer les valeurs de tous les éléments du tableau, le message gagnant et le bouton de redémarrage. Pour cela nous avons utilisé la méthode JavaScript `getElementById()`. Pour l'élément de texte du message gagnant, nous avons utilisé la `querySelector()` méthode qui renvoie le premier élément du document qui correspond au sélecteur spécifié. Nous avons également utilisé les crochets ([]) pour cibler l' `data-cell` attribut.

Passons au fonction maintenant , une fois que nos identificateur ont été paramétré

```
startGame()

restartButton.addEventListener('click', startGame)

function startGame() {
  isPlayer_O_Turn = false
  cellElements.forEach(cell => {
    cell.classList.remove(PLAYER_X_CLASS)
    cell.classList.remove(PLAYER_O_CLASS)
    cell.removeEventListener('click', handleCellClick)
    cell.addEventListener('click', handleCellClick, { once: true })
  })
  setBoardHoverClass()
  winningMessageElement.classList.remove('show')
}
```

Pour démarrer le jeu appelée `startGame()`. Nous définissons la `isPlayer_O_Turn` variable sur `false`, ce qui signifie que le premier à jouer sera un caractère x. Le reste de la fonction supprime tous les personnages restants du gameplay précédent. Ici, nous déclenchons également les événements qui peuvent se produire sur notre tableau, à savoir les clics de souris.

```
function handleCellClick(e) {
  const cell = e.target
  const currentClass = isPlayer_O_Turn ? PLAYER_O_CLASS : PLAYER_X_CLASS
  placeMark(cell, currentClass)
  if (checkWin(currentClass)) {
    endGame(false)
  } else if (isDraw()) {
```

```
    endGame(true)
  } else {
    swapTurns()
    setBoardHoverClass()
  }
}
```

`handleClick` nous gére les événements de clic de souris pour les cellules du tableau. La plupart des fonctions appelées ici seront expliquées séparément. En bref, la `currentClass` variable enregistre le personnage (X ou O) dont c'est le tour en ce moment. Une autre fonction est utilisée dans l'instruction if pour vérifier si quelqu'un a déjà gagné en comparant les combinaisons gagnantes au gameplay. De cette façon, il détermine s'il y a un match nul ou non.

```
function endGame(draw){
  if (draw){
    winningMessageElement.innerText = "It's a draw!"
  }
  else {
    winningMessageElement.innerText = 'Player with ${isPlayer_O_Turn ?'O's' :"X's"} wins !"
  }
  winningMessageElement.classList.add('show')
}
```

La `gameEnd()` fonction a été mentionnée précédemment. C'est la fonction qui termine le jeu. La fonction peut soit afficher un message gagnant qui spécifie quel personnage a gagné, soit un message indiquant qu'il n'y a pas de gagnant - c'est un match nul, selon le résultat de l'instruction if.

```
function isDraw() {
  return [...cellElements].every(cell => {
    return cell.classList.contains(PLAYER_X_CLASS) || cell.classList.contains(PLAYER_O_CLASS)
  })
}
```

`placeMark()` et `swapTurns()` sont deux fonctions courtes et simples. Le `placeMark()` place le personnage dans la cellule, `currentClass` soit un X ou un O selon le tour. La deuxième

fonction est celle qui permute les tours après que le personnage soit placé dans une cellule.

Dans la prochaine partie, permettre au joueur de viser plus facilement les cellules. Cela rend également notre jeu un peu plus réactif.

```
function setBoardHoverClass() {
  boardElement.classList.remove(PLAYER_X_CLASS)
  boardElement.classList.remove(PLAYER_O_CLASS)
  if (isPlayer_O_Turn) {
    boardElement.classList.add(PLAYER_O_CLASS)
  } else {
    boardElement.classList.add(PLAYER_X_CLASS)
  }
}
```

Pour qu'un personnage apparaisse dans les cellules en les survolant avec le curseur de notre souris avant de les placer, la fonction setBoardHoverClass() s'occupe de la partie interactive de cela.

```
function checkWin(currentClass) {
  return WINNING_COMBINATIONS.some(combination => {
    return combination.every(index => {
      return cellElements[index].classList.contains(currentClass)
    })
  })
}
```

`checkWin()` qui est appelée pour vérifier si notre tableau correspond à l'une des combinaisons gagnantes.

à ce stade nous avons rien comme affichage

Passons maintenant à partie CSS de notre jeu

Dans la feuille de style, nous utiliserons nos balises d'identification et nos classes pour personnaliser les visuels de notre jeu. Depuis les bordures et la largeur des lignes jusqu'aux couleurs et à la taille du texte.

```
:root {
  --cell-size: 100px;

  --color: #81c3fd; /* for hover */
  --color-set: #0275d8; /* when set */
  --l: 10px; /* X line-width */
}
```

Nous utilisons `:root` element pour définir des variables pour une utilisation ultérieure. Ces variables définissent la couleur de X et O, et la largeur du signe X.

```
body {
  margin: 0;
}

.board {
  width: 100vw;
  height: 100vh;
  display: grid;
  justify-content: center;
  align-content: center;
  justify-items: center;
  align-items: center;
  grid-template-columns: repeat(3, auto)
}
```

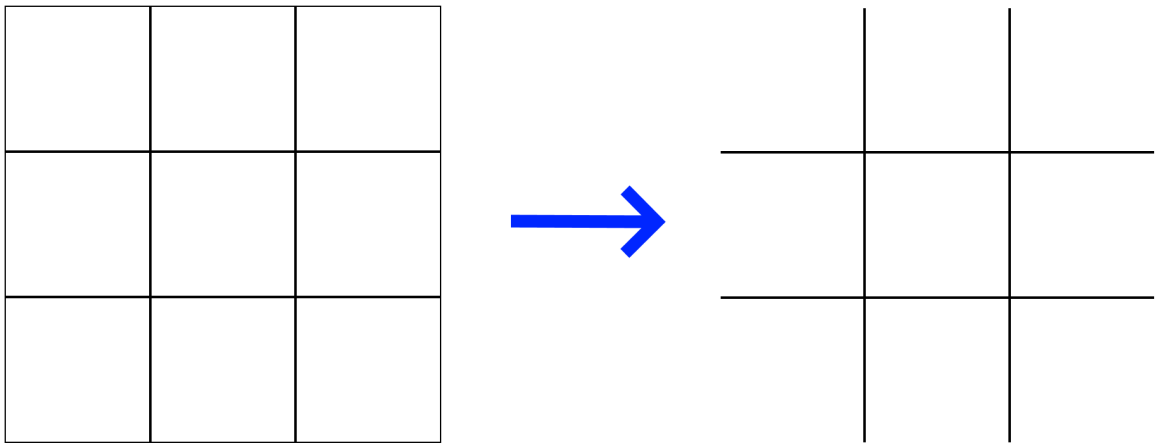
En commençant par visualiser notre HTML, le `margin` crée des espaces entre les éléments avec des bordures définies, ici il est utilisé pour créer des bordures nulles pour tout l'écran. Après cela, nous décrivons l'élément avec notre `board` nom de classe, en spécifiant sa largeur, sa hauteur, comment et où il est affiché. `grid-template-columns` elle spécifie le nombre et la largeur des colonnes dans une disposition en grille. De cette façon, nous fabriquons nos 9 cellules pour jouer.

```
.cell {
  width: var(--cell-size);
  height: var(--cell-size);
  border: 1px solid black;
  display: flex;
  justify-content: center;
```

```
  align-items: center;
  position: relative;
  cursor: pointer;
}
```

Ici, nous spécifions à quoi nous voulons que nos cellules ressemblent. Commençant par `width` et `height`, suivi de `width` et `color` des bordures, `display`, `position` et `cursor`.

Ensuite, faire ressembler notre table à un jeu de Tic-Tac-Toe. Nous excluons les cellules extérieures d'avoir une bordure complète, donc notre tableau peut ressembler à ceci :



Pour une cellule donnée, nous spécifions qu'il ne devrait pas y avoir de bordure. Nous sélectionnons une cellule à l'aide du `nth-child` sélecteur où Nous écrirons le numéro d'une cellule pour sélectionner cet élément/cellule.

```
/* supprimer la bordure pour les bords */
.cell:nth-child(1), .cell:nth-child(2), .cell:nth-child(3) {
  border-top: none;
}

.cell:nth-child(1), .cell:nth-child(4), .cell:nth-child(7) {
  border-left: none;
}

.cell:nth-child(3), .cell:nth-child(6), .cell:nth-child(9) {
  border-right: none;
}

.cell:nth-child(7), .cell:nth-child(8), .cell:nth-child(9) {
  border-bottom: none;
}
```

Pour le survol

```
.cell.x, .cell.circle {
  cursor: not-allowed;
}
```

À l' aide `linear-gradient` de nous définissons les couleurs en 3 étapes blanc, bleu (pour la croix) et blanc encore une fois. Mais attention au début on définit aussi l'angle à l'aide de mots `to top right` sous lesquels tout se passe. Les paramètres d'arrière-plan sont là pour faire de l'espacement et pour centrer la croix.

```
/* for cross */
.board.x .cell:not(.circle):not(.x):hover {
  background: linear-gradient(to top right, transparent calc(50% - var(--l) / 2), var(--co
lor) calc(50% - var(--l) / 2) calc(50% + var(--l) / 2), transparent calc(50% + var(--l) /
 2)),
        linear-gradient(to bottom right, transparent calc(50% - var(--l) / 2), var(--colo
r) calc(50% - var(--l) / 2) calc(50% + var(--l) / 2), transparent calc(50% + var(--l) /
 2));
  background-size: 80% 80%;
  background-repeat: no-repeat;
  background-position: center;
}
```

Pour la couleur, nous utilisons notre variable `--color-set`. Nous pouvez voir que nous avons utilisé `var` function pour cela . Il existe une autre fonction appelée `calc`). Il s'agit d'une fonction spéciale car elle prend toutes sortes de mesures (comme px, pt, %) et les additionne.

Dans le cas où la cellule est vide, l'effet de survol utilisera une variable pour la couleur nommée `--color` et si nous y estampons un x ou un o, ce sera `--color-set`. Ici, Nous pouvoir voir le x estampé à côté de l'effet de survol du o.

Voici le code pour X qui ne flotte pas. Le seul changement est le nom de la couleur.

```
/* for cross (set) */
.cell:not(.circle).x {
  background: linear-gradient(to top right, transparent calc(50% - var(--l) / 2), var(--co
lor-set) calc(50% - var(--l) / 2) calc(50% + var(--l) / 2), transparent calc(50% + var(--
l) / 2)),
        linear-gradient(to bottom right, transparent calc(50% - var(--l) / 2), var(--color
-set) calc(50% - var(--l) / 2) calc(50% + var(--l) / 2), transparent calc(50% + var(--l) /
2));
  background-size: 80% 80%;
  background-repeat: no-repeat;
  background-position: center;
}
```

Dessiner un cercle est beaucoup plus facile, cela ne prend qu'une ligne. Mais le même principe est utilisé, juste avec un gradient radial.

```
/* for circle */
.board.circle .cell:not(.circle):not(.x):hover {
  background: radial-gradient(var(--color) 60%, transparent 60%);
}

/* for circle (set) */
.cell:not(.x).circle {
  background: radial-gradient(var(--color-set) 60%, transparent 60%);
}
```

Cette fois, la façon dont nous utilisons les couleurs lorsqu'un panneau flotte et lorsqu'il est "gravé dans la pierre" est plus transparente.

Le style d'un message gagnant est plus simple. Ici, nous définissons les polices et la couleur du texte. Rendre le bouton un peu arrondi. Et ajoutez un effet de survol à la fin.

```
.winning-message {
  display: none;
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: var(--color-set);
  justify-content: center;
  align-items: center;
  color: white;
  font-size: 5rem;
  font-family: 'Courier New', Courier, monospace;
  flex-direction: column;
}
```

Et voici le code du bouton avec un effet de survol ajouté :

```
.winning-message button {
  border-radius: 10px;
  font-size: 3rem;
  background-color: white;
```

```
  border: 1px solid var(--color-set);
  padding: .25em .5em;
  cursor: pointer;
}

.winning-message button:hover {
  background-color: var(--color-set);
  color: white;
  border-color: white;
}

.winning-message.show {
  display: flex;
}
```