Projet : Memory game

Table des matières

I. Contexte

Durée : 120 minutes

Environnement de travail : Visual Studio Code[1]. Afin de faciliter le test et la lecture du code, vous trouverez dans ce cours les exemples de code sur Replit. Vous pourriez réaliser ce projet sur Replit, mais il est conseillé de le faire sur Visual Studio Code pour commencer à travailler avec un vrai environnement de développement professionnel.

Prérequis : avoir des bases de CSS et de JavaScript. Savoir gérer des boucles, gérer des fonctions en JavaScript.

Contexte

Dans ce projet, nous allons créer un jeu de memory en JavaScript. Le développement de ce jeu nous permettra d'aborder des notions clés telles que la manipulation du DOM pour créer des éléments de façon dynamique, la gestion des événements pour interagir avec les éléments, et les fonctions pour structurer notre code. Le tout en s'amusant !

Attention

Pour avoir accès au code et à l'IDE intégré de cette leçon, vous devez :

1) Vous connecter à votre compte sur https://replit.com/ (ou créer gratuitement votre compte)

2) Rejoindre la Team Code Studi du module via ce lien : https://replit.com/teams/join/mmurvlgiplpxuasordloklllbbqskoim-programmer-avec-javascript

Une fois ces étapes effectuées, nous vous conseillons de rafraîchir votre navigateur si le code ne s'affiche pas.

En cas de problème, redémarrez votre navigateur et vérifiez que vous avez bien accepté les cookies de connexion nécessaires avant de recommencer la procédure.

Pour accéder au code dans votre cours, cliquez sur le nom du lien Replit dans la fenêtre. Par exemple :

II. La création du plateau

A. Création du plateau

Dans un cadre professionnel, il y a peu de chances que vous deviez réaliser un memory. Cependant, cette mécanique en JavaScript pourra vous être utile, notamment pour des applications web, où des règles métiers doivent être gérées côté *front*.

Fondamental **Les règles du memory**

Avant de commencer la réalisation de ce jeu, il faut évidemment connaître les règles. Les voici :

- Le jeu se compose de cartes disposées face cachée sur une surface. Chaque carte est une image ou un symbole identique à une autre carte du jeu. Ainsi, toutes les cartes sont associées en paires.
- Lors de chaque tour, le joueur retourne 2 cartes de son choix sur la face visible.
- Si les 2 cartes retournées forment une paire (c'est-à-dire qu'elles ont la même image ou le même symbole), le joueur les garde et continue à jouer en retournant 2 autres cartes.

1 https://code.visualstudio.com/

- Si les 2 cartes retournées ne forment pas une paire, elles sont de nouveau retournées face cachée.
- Le jeu continue jusqu'à ce que toutes les paires de cartes aient été trouvées.

Créer le projet

Cette partie du cours nous montrera comment créer le projet sur Visual Studio Code et comment créer les fichiers qui seront nécessaires au bon fonctionnement de notre jeu de memory.

1. Créez un dossier appelé « *MemoryGame* ».
2. Ouvrez ce dossier avec VS Code.
3. Créez un fichier « *index.html* ».
4. Créez le contenu de base du fichier HTML. Pour cela, vous pouvez utiliser un raccourci VS Code. Tapez « *!* » dans le fichier HTML, et cliquez sur tabulation sur votre clavier. Vous avez une base de document HTML déjà créé.
5. Créez un fichier « *style.css* » et un fichier « *script.js* ».
6. Appelez ces 2 fichiers depuis votre HTML
7. Ajoutez un H1.

```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">
    <title>Memory game</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Jeu de Memory</h1>
    <script src="script.js"></script>
</body>
</html>
```

[cf.]

Créer la zone de jeu en HTML et CSS

Nous allons devoir créer notre zone de jeu dans le HTML, et injecter les cartes dans cette zone de jeu. Nous créons donc un div vide dans notre HTML. Il aura pour identifiant `game-board`.

```html
  <div id="game-board"></div>
```

Notre zone de jeu devra répondre à un style bien particulier. Nous voulons des cartes de 100 px par 100 px. Nous pouvons donc ajouter ce CSS pour que notre zone de jeu réagisse bien à l'ajout des cartes. Nous ajoutons une police et un text-center sur le body pour que le titre soit plus esthétique. Vous pourrez, si vous le souhaitez, définir une couleur de background et choisir une police plus originale.

```css
body {
    font-family: Arial, sans-serif;
    text-align: center;
}

#game-board {
  display: grid;
  grid-template-columns: repeat(4, 100px);
  grid-template-rows: repeat(4, 100px);
  grid-gap: 10px;
```

```
    justify-content: center;
    margin: 30px auto;
}
```

[cf.]

Maintenant que notre zone de jeu est créée, il nous faut injecter dans cette zone de jeu nos différentes cartes.

Création d'une cartes en JS

Pour commencer, nous allons créer une méthode permettant de créer une carte et de l'ajouter à notre zone de jeu. Depuis le JavaScript, nous allons créer le HTML de la carte et nous allons l'injecter dans notre `div#game-board` pour l'afficher dans la page.

Voici le résultat final d'une carte voulue en HTML :

```
<div class="card" data-value="urlVersImage">
<img class="card-content" src="https://picsum.photos/id/243/100/100”>
</div>
```

[cf.]

Nous voulons donc créer une méthode `createCard` prenant en paramètre la valeur de l'URL de l'image et retournant l'objet HTML de cette carte.

```
function createCard(CardUrl) {
    const card = document.createElement('div');
    card.classList.add('card');
    card.dataset.value = CardUrl;

    const cardContent = document.createElement('img');
    cardContent.classList.add('card-content');
    cardContent.src = `${CardUrl}`;
    card.appendChild(cardContent);
    return card;
  }

//Code pour tester notre fonction
const gameBoard = document.getElementById('game-board');
gameBoard.appendChild(createCard('https://picsum.photos/id/243/100/100'));
```

[cf.]

Créer plusieurs cartes

Maintenant que nous avons créé une carte, il va falloir créer le jeu complet, donc **16 cartes**. Pour commencer, nous avons besoin d'images pour nos cartes. Nous allons utiliser le site picsum.photos, qui est un site proposent des photos libres de droits. Vous pourriez utiliser vos propres images en passant le chemin d'accès à votre image au lieu de l'URL vers picsum.photos. Voici un tableau comprenant 8 URL d'images destinées à notre jeu du memory :

```
const cards = [
    'https://picsum.photos/id/237/100/100',
    'https://picsum.photos/id/238/100/100',
    'https://picsum.photos/id/239/100/100',
    'https://picsum.photos/id/240/100/100',
    'https://picsum.photos/id/241/100/100',
    'https://picsum.photos/id/242/100/100',
    'https://picsum.photos/id/243/100/100',
    'https://picsum.photos/id/244/100/100'
  ];
```

Maintenant, il nous faut parcourir ce tableau pour créer nos 16 cartes. Étant donné que nos images sont en double, il va falloir créer 2 images par URL du tableau. Notre tableau contient 8 éléments, en dupliquant ces éléments, nous aurons nos 16 éléments.

Pour commencer, nous créons donc une méthode retournant le tableau card, mais dupliqué.

Voici cette méthode :

```
function duplicateArray(arraySimple) {
    let arrayDouble = [];
    arrayDouble.push(...arraySimple);
    arrayDouble.push(...arraySimple);
      return arrayDouble;
 }
```

Maintenant que nous pouvons dupliquer ce tableau d'URL, il est possible, pour les 16 URL générées dans ce nouveau tableau, de créer 16 cartes en HTML et de les ajouter à notre gameboard.

Voici le code qui réalise cela :

```
const allCards = duplicateArray(cards);

allCards.forEach(card => {
    const cardHtml = createCard(card);
    gameBoard.appendChild(cardHtml);
})
```

[cf.]

Mélanger un tableau

Nos cartes sont maintenant affichées sur notre page web, mais elles ont toujours la même disposition. Le jeu est donc trop facile, l'utilisateur va vite se rendre compte que les cartes sont positionnées de manière logique, et donc il trouvera toutes les paires aisément. Il nous faut mélanger nos cartes avant de les positionner.

Par conséquent, avant de parcourir le tableau `allCards` pour ajouter toutes les cartes, il faut mélanger ce même tableau.

Il faut créer une méthode `shuffle` permettant de mélanger un tableau.

Voici le code modifié :

```
Début de listing Informatique : JS
function shuffleArray(arrayToshuffle){
    const arrayShuffled = arrayToshuffle.sort(() => 0.5 - Math.random());
    return arrayShuffled;
}

let allCards = duplicateArray(cards);
//Mélanger le tableau
allCards = shuffleArray(allCards);
allCards.forEach(card => {
    const cardHtml = createCard(card);
    gameBoard.appendChild(cardHtml);
})
```

[cf.]

La fonction `shuffleArray` permet de mélanger aléatoirement les éléments d'un tableau passé en paramètre. Nous utilisons ici la fonction `sort` de JavaScript, qui trie les éléments du tableau selon un ordre donné. Dans ce cas, la fonction de tri utilise une fonction de comparaison qui retourne un nombre aléatoire entre - 0,5 et 0,5. Cela signifie que pour chaque paire d'éléments, la fonction de comparaison retournera un nombre positif ou négatif de manière aléatoire, ce qui entraînera un ordre de tri aléatoire pour les éléments. On aura donc un tableau mélangé aléatoirement !

Masquer les cartes

Maintenant que les cartes sont bien positionnées, il faut les masquer à l'utilisateur pour qu'il puisse les deviner. Cette opération, nous allons la faire en CSS.

Voici le code CSS permettant de masquer les cartes :

```css
.card {
    background-color: #ccc;
    border-radius: 5px;
    cursor: pointer;
}

.card .card-content {
    opacity: 0;
 }
```

[cf.]

III. L'ajout de la mécanique de jeu

A. Ajout de la mécanique de jeu

Retourner une carte

Maintenant que toutes les cartes sont positionnées et cachées, l'objectif est de pouvoir afficher l'image lorsque l'on clique dessus. Pour cela, on peut gérer une classe en CSS que l'on nommera `flip`. Cette classe, ajoutée à une carte, permettra d'afficher l'image. On va donc mettre l'opacité du `card-content` à 1.

```css
.card.flip .card-content{
    opacity: 1;
}
```

Maintenant que le CSS est géré, on peut ajouter la classe `flip` sur une carte lorsque l'on clique dessus. On crée donc une méthode contenant le code à exécuter lorsque l'événement `Click` est déclenché sur une carte.

```javascript
function onCardClick(e){
    const card = e.target.parentElement;
    card.classList.add("flip");
}
```

Dans la méthode `createCard(cardUrl)`, on ajoute l'écouteur d'événements pour gérer le clic sur chaque `Card`.

```javascript
card.addEventListener('click', onCardClick);
```

[cf.]

Vérifier les correspondances

Nous avançons dans le jeu : maintenant que nous pouvons afficher des cartes, nous voulons chercher à vérifier si nous avons trouvé une paire de cartes.

Pour cela, nous allons avoir besoin de stocker les 2 cartes qui sont actuellement visibles. Nous créons donc une variable `selectedCards` qui sera un tableau contenant les 2 cartes retournées en jeu. Lorsque nous cliquons sur une carte, nous ajoutons donc cette carte à la variable `selectedCards`.

Si au clic sur une carte, la variable `selectedCards` contient 2 cartes, nous pouvons comparer les 2 data attributes `data-value`. 2 cas possibles :

- Si les 2 sont équivalents, eh bien nous avons trouvé une paire. On ajoute la classe `matched` à ces éléments, on supprime leurs event listener, et on vide la variable `selectedCards`.
- Si ce n'est pas le cas, nous nous sommes trompés. On retire donc la classe `flip` à ces 2 éléments pour les remettre en jeu, et on vide la variable `selectedCards`.

CSS ajouté :

```css
.card.matched {
  border: solid 4px rgb(0, 255, 0);
  box-sizing: border-box;
  cursor: default;
}
.card .card-content {
    max-width: 100%;
}
```

Code JavaScript complet à ce stade du projet :

```javascript
const cards = [
    'https://picsum.photos/id/237/100/100',
    'https://picsum.photos/id/238/100/100',
    'https://picsum.photos/id/239/100/100',
    'https://picsum.photos/id/240/100/100',
    'https://picsum.photos/id/241/100/100',
    'https://picsum.photos/id/242/100/100',
    'https://picsum.photos/id/243/100/100',
    'https://picsum.photos/id/244/100/100'
  ];
const gameBoard = document.getElementById('game-board');
let selectedCards = [];

function createCard(cardUrl){
    const card = document.createElement('div');
    card.classList.add('card');
    card.dataset.value = cardUrl;

    const cardContent = document.createElement('img');
    cardContent.classList.add('card-content');
    cardContent.src= cardUrl;

    card.appendChild(cardContent);

    card.addEventListener('click', onCardClick);
    return card;
}

function duplicateArray(arraySimple){
    let arrayDouble = [];
    arrayDouble.push(...arraySimple);
    arrayDouble.push(...arraySimple);

    return arrayDouble;
}

function shuffleArray(arrayToshuffle){
    const arrayShuffled = arrayToshuffle.sort(() => 0.5 - Math.random());
    return arrayShuffled;
}

function onCardClick(e){
    const card = e.target.parentElement;
    card.classList.add('flip');

    selectedCards.push(card);
```

```
    if(selectedCards.length == 2){
        if(selectedCards[0].dataset.value == selectedCards[1].dataset.value){
            //on a trouvé une paire
            selectedCards[0].classList.add("matched");
            selectedCards[1].classList.add("matched");
            selectedCards[0].removeEventListener('click', onCardClick);
            selectedCards[1].removeEventListener('click', onCardClick);
        }
        else{
            //on s'est trompé
            selectedCards[0].classList.remove("flip");
            selectedCards[1].classList.remove("flip");
        }
        selectedCards = [];
    }
}

let allCards = duplicateArray(cards);
//Mélanger le tableau
allCards = shuffleArray(allCards);
allCards.forEach(card => {
    const cardHtml = createCard(card);
    gameBoard.appendChild(cardHtml);
})
```

[cf.]

Gestion du timing

Maintenant que nous gérons les paires de cette application, on se rend compte qu'il y a un problème. Lorsque l'on clique sur une deuxième carte qui ne correspond pas à la première, la deuxième carte n'a pas le temps de s'afficher puisqu'elle est masquée immédiatement. Il faut donc mettre un délai entre le moment où on clique sur la carte et le moment où on vérifie. Pour cela, il est possible d'utiliser la méthode `setTimeout`.

Méthode `onCardClick(e)` modifiée :

```
function onCardClick(e){
    const card = e.target.parentElement;
    card.classList.add('flip');

    selectedCards.push(card);
    if(selectedCards.length == 2){
        setTimeout(() => {
            if(selectedCards[0].dataset.value == selectedCards[1].dataset.value){
                //on a trouvé une paire
                selectedCards[0].classList.add("matched");
                selectedCards[1].classList.add("matched");
                selectedCards[0].removeEventListener('click', onCardClick);
                selectedCards[1].removeEventListener('click', onCardClick);
            }
            else{
                //on s'est trompé

                    selectedCards[0].classList.remove("flip");
                    selectedCards[1].classList.remove("flip");
            }
            selectedCards = [];
        }, 1000)
    }
```

```
}
```

[cf.]

Gérer la victoire du joueur

À l'heure actuelle, notre jeu du memory fonctionne. Il ne nous reste plus qu'à gérer le moment où le joueur a trouvé toutes les paires, et donc a gagné la partie.

Il existe plusieurs solutions pour cela, nous pourrions :

- Avoir un compteur de paires trouvées, et l'incrémenter à chaque fois qu'on trouve une paire. S'il y a x paires à trouver, et que le compteur vaut x, on a gagné.
- À chaque paire trouvée, vérifier s'il reste des cartes sans la classe matched. S'il n'existe pas de carte n'ayant pas la classe matched, c'est gagné.
- Avoir un compteur de paires à trouver, et le décrémenter à chaque paire trouvée. Si on arrive à 0, on a gagné.
- Il existe d'autres possibilités.

Pour ce cours, nous allons réaliser la deuxième option, le but est de récupérer *via* le sélecteur CSS les cartes qui ne possèdent pas la classe matched : `.card:not(.matched)`.

À chaque fois que nous trouvons une paire, nous faisons un `queryselectorAll` pour récupérer ces cartes. Si on en récupère 0, on peut dire à l'utilisateur que c'est gagné.

Voici le code à ajouter lorsqu'on a trouvé une paire :

```
const allCardNotFinded = document.querySelectorAll('.card:not(.matched)');
if(allCardNotFinded.length == 0){
  //Le joueur a gagné
  alert('Bravo, vous avez gagné');
}
```

[cf.]

Ajouter des animations pour rendre le jeu plus agréable

Le jeu est fonctionnel, mais ce serait plus agréable à utiliser si les cartes se retournaient avec une animation, comme si on retournait de vraies cartes. Plusieurs options existent pour obtenir ce rendu. Nous pouvons faire cela facilement en CSS, avec `transform: rotateY(180deg)`.

Le passage de `transform: rotateY(0deg)` à `transform: rotateY(180deg)` simule le retournement d'une carte. Pour que l'animation soit visible, il faudra aussi ajouter une transition.

Voici le code à ajouter :

```
.card.flip{
    transform: rotateY(180deg);
}
.card .card-content {
    transition: all 0.3s;
}
.card {
    /*animation*/
    transform: rotateY(0deg);
    transition: all 0.5s;
}
```

Voici le code final de notre application :

index.html :

```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">
    <title>Memory game</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>Memory game</h1>
    <div id="game-board">

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

style.css :

```css
body{
    font-family: Arial, sans-serif;
    text-align: center;
}

#game-board{
    display: grid;
    grid-template-columns: repeat(4, 100px);
    grid-template-rows: repeat(4, 100px);
    grid-gap:10px;
    margin: 30px auto;
    justify-content: center;
}

.card {
    background-color: #ccc;
    border-radius: 5px;
    cursor: pointer;

    /*animation*/
    transform: rotateY(0deg);
    transition: all 0.5s;
}

.card .card-content {
    opacity: 0;
    transition: all 0.3s;
}

.card.flip .card-content {
    opacity: 1;
    max-width: 100%;
}

.card.flip{
    transform: rotateY(180deg);
```

```
}

.card.matched{
    border: solid 4px green;
    box-sizing: border-box;
    cursor: default;
}
```

script.js :

```
const cards = [
    'https://picsum.photos/id/237/100/100',
    'https://picsum.photos/id/238/100/100',
    'https://picsum.photos/id/239/100/100',
    'https://picsum.photos/id/240/100/100',
    'https://picsum.photos/id/241/100/100',
    'https://picsum.photos/id/242/100/100',
    'https://picsum.photos/id/243/100/100',
    'https://picsum.photos/id/244/100/100'
  ];
const gameBoard = document.getElementById('game-board');
let selectedCards = [];

function createCard(cardUrl){
    const card = document.createElement('div');
    card.classList.add('card');
    card.dataset.value = cardUrl;

    const cardContent = document.createElement('img');
    cardContent.classList.add('card-content');
    cardContent.src= cardUrl;

    card.appendChild(cardContent);

    card.addEventListener('click', onCardClick);
    return card;
}

function duplicateArray(arraySimple){
    let arrayDouble = [];
    arrayDouble.push(...arraySimple);
    arrayDouble.push(...arraySimple);

    return arrayDouble;
}

function shuffleArray(arrayToshuffle){
    const arrayShuffled = arrayToshuffle.sort(() => 0.5 - Math.random());
    return arrayShuffled;
}

function onCardClick(e){
    const card = e.target.parentElement;
    card.classList.add('flip');

    selectedCards.push(card);
    if(selectedCards.length == 2){
        setTimeout(() => {
            if(selectedCards[0].dataset.value == selectedCards[1].dataset.value){
```

```
                //on a trouvé une paire
                selectedCards[0].classList.add("matched");
                selectedCards[1].classList.add("matched");
                selectedCards[0].removeEventListener('click', onCardClick);
                selectedCards[1].removeEventListener('click', onCardClick);

                const allCardsNotMatched = document.querySelectorAll('.card:not(.matched)');
                console.log(allCardsNotMatched.length);
                if(allCardsNotMatched.length == 0){
                    //Le joueur a gagné
                    alert("Bravo, vous avez gagné");
                }
            }
            else{
                //on s'est trompé
                selectedCards[0].classList.remove("flip");
                selectedCards[1].classList.remove("flip");
            }
            selectedCards = [];
        }, 1000);
    }
}

let allCards = duplicateArray(cards);
//Mélanger le tableau
allCards = shuffleArray(allCards);
allCards.forEach(card => {
    const cardHtml = createCard(card);
    gameBoard.appendChild(cardHtml);
})
```

[cf.]

IV. Pour aller plus loin

Grâce à ce projet, nous avons pu aborder plusieurs aspects clés du développement web, notamment l'utilisation de JavaScript. Cette compétence est particulièrement importante pour ajouter des fonctionnalités engageantes à vos sites, c'est-à-dire pour rendre vos sites plus attractifs et plus fluides pour vos clients. Vous avez pu constater l'utilité de l'API DOM, des tableaux, des boucles, des sélecteurs CSS, et des fonctions en JavaScript.

Même si ce projet ne rentre pas dans un cadre professionnel, il résume très bien les compétences fondamentales en développement front.

Il reste encore beaucoup de fonctionnalités potentielles à implémenter, en voici quelques-unes :

- **Mettre un chronomètre :** lancer un chronomètre en JS au début de la partie, et afficher le temps à la fin de la partie pour donner un score à l'utilisateur.
- **Ajouter la possibilité de recommencer une partie sans rafraîchir la page :** pour cela, il suffit d'ajouter un bouton permettant de régénérer le plateau et de remettre à 0 toutes vos variables de jeu.
- **Bloquer les clics pendant la seconde d'attente :** pendant la seconde d'attente avant de valider une paire, ou de retourner les cartes, on peut encore cliquer sur les autres cartes, c'est un bug qui pourrait compromettre le bon déroulement de la partie.
- **Stocker les précédents scores en cookie :** pour faire une moyenne des scores et établir un meilleur score personnel à battre, on pourrait stocker les scores en cookie pour pouvoir faire des statistiques dessus.