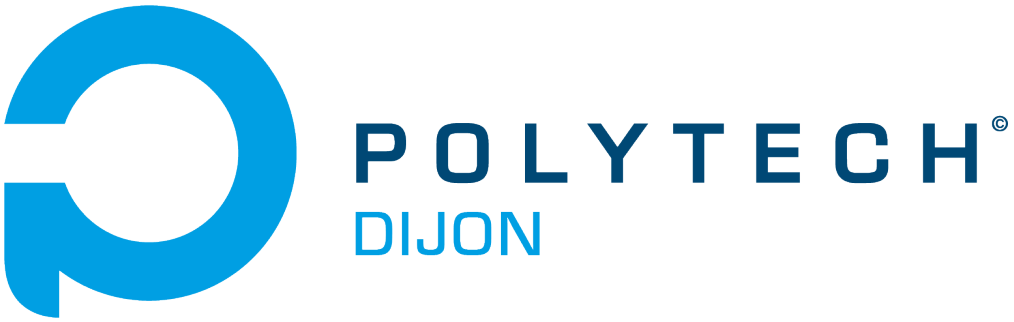
Esirem Informatique & Réseaux

ITC-315 : CR Développement Web

JAVASCRIPT – LocalStorage

Auteur :
BOUQUILLON Erwan

Professeur référent :
MEUNIER Charles



2023-2024

1 JavaScript-LocalStorage

Ce TP a pour but de réaliser un Memory, c'est un jeu où il faut retrouver des paires de cartes présentées initialement face retournée. Au travers de ce TP, on manipulera les notions suivantes :

- Utilisation des modules ES6
- Programmation orientée objets
- Utilisation de l'architecture Modèle/Vue/Contrôleur
- Utilisation de l'API Web LocalStorage

1.1 Une carte

On va dans cette partie, essayer d'implémenter la classe card.

1.1.1 Modèle

Pour cela, on dispose du modèle de la classe à réaliser :

```
export class Card{

    #value;

    constructor(value){
        this.#value = value;
    }

    get value() { return this.#value }
}
```

FIGURE 1 – Class Card

1.1.2 Contrôleur

On ajoute une méthode createCard qui initialise l'attribut card avec une instance de Card ayant une valeur aléatoire comprise entre 0x1F90C et 0x1F9FF, puis on notifie les observateurs du contrôleur. On obtient alors :

```
import { Card } from "../models/card.js";
//import { Memory } from "../models/memory.js";
import { Notifier } from "../patterns/notifier.js";

export class ControllerMemory extends Notifier
{
    #card = null;
    constructor()
    {
        super();

    }

    get card() { return this.#card }

    createCard(){
        const tab1 = ["0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"];
        const tab2 = ["C","D","E","F"];

        const n1 = tab1[Math.floor(Math.random() * 15)]
        const n2 = tab2[Math.floor(Math.random() * 3)]

        var value = "0x1F9" + n1 + n2 ;
        console.log(value)
        value = parseInt(value);
        this.#card = new Card(value);
        this.notify();
    }
}
```

FIGURE 2 – Ajout de CreateCard et de l'attribut card

1.1.3 Vue

La vue va s'occuper d'afficher les cartes en ajoutant des éléments dans le fichier html. On a ainsi le code suivant :

```
displayCard(card){
    const cards = document.querySelector(".cards");
    let enfant = document.createElement("div");
    enfant.classList.add("card");
    enfant.innerHTML = "&#x" + (card.value).toString(16);
    cards.appendChild(enfant);
```

FIGURE 3 – Création de la méthode displayCard

Pour pouvoir respecter le modèle, on fait en sorte que la vue notifie l'application du changement à venir. Pour cela, on rajoute tout simplement this.displayCard() dans la méthode notify() de view-memory.js.

Après ajout de la méthode createCard du contrôleur dans le constructeur de la classe ApplicationMemory. On obtient alors une carte au centre de l'écran avec un émoji qui change à chaque rafraichissement.

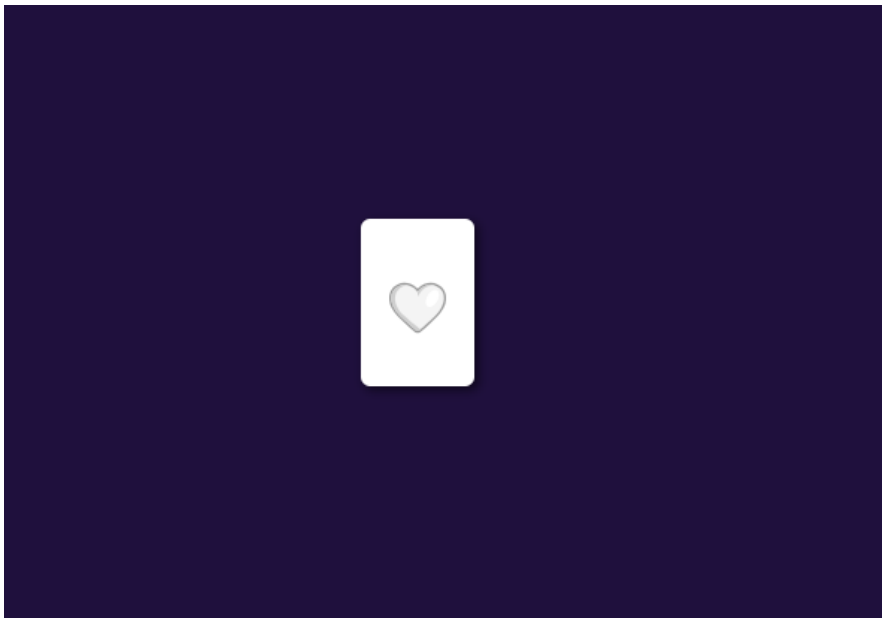


FIGURE 4 – Test du modèle MCV

1.2 Des cartes

On s'apprête à modifier le code pour qu'à chaque nouveau clic sur la carte, une nouvelle carte est créée. Pour cela, on a besoin de rajouter une seule ligne : "enfant.addEventListener('click', () => this.#controllerMemory.createCard()) ;"

```
displayCard(){
    const cards = document.querySelector(".cards");
    let enfant = document.createElement("div");
    enfant.classList.add("card");
    enfant.innerHTML = "&#x" + (this.#controllerMemory.card.value).toString(16);
    cards.appendChild(enfant);
    enfant.addEventListener('click', () => this.#controllerMemory.createCard());
}
```

FIGURE 5 – Modification de displaCard()

Ainsi, après exécution et après avoir spam-click, on obtient ceci :

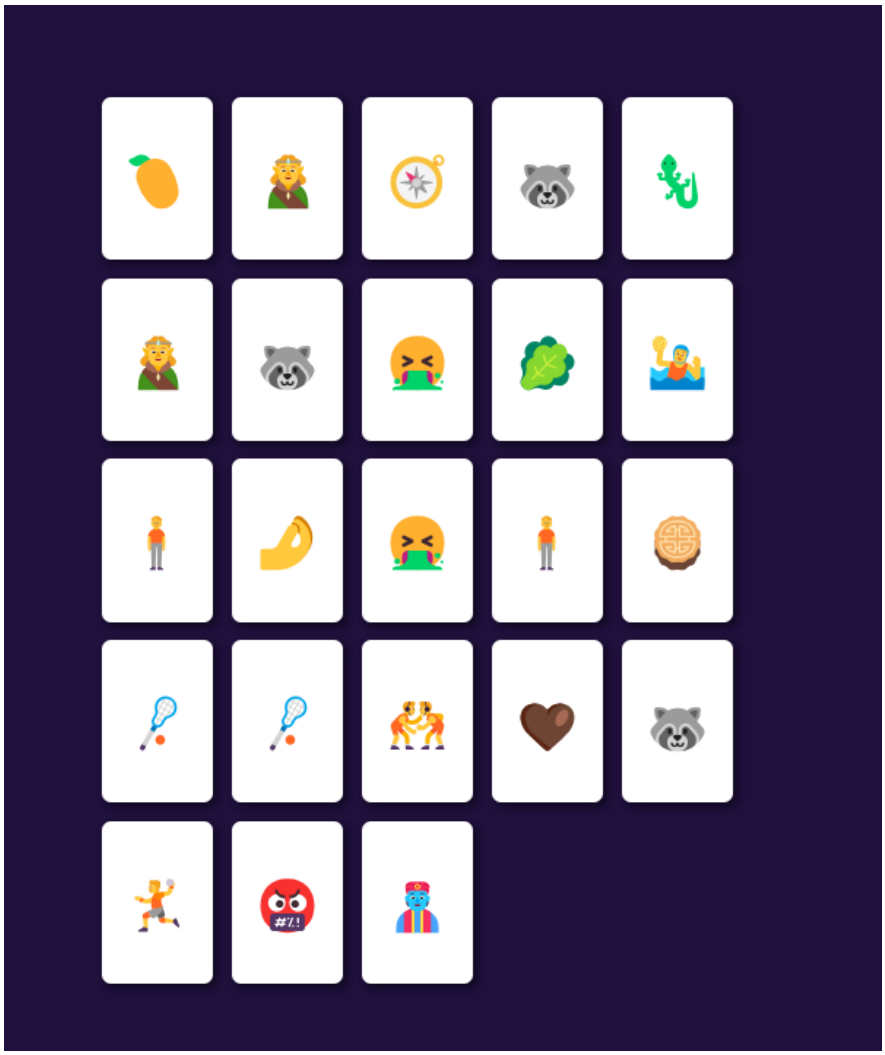


FIGURE 6 – Test click

1.3 Le jeu

On implémente la classe Memory en suivant le modèle établi dans l'énoncé. On obtient le code suivant :

```
import { Card } from "./card.js";

class Memory {
  #cards;

  constructor() {
    this.#cards = [];
  }

  newGame(pairsNumber) {
    for (let i = 0; i < pairsNumber; i++) {
      const value = 0x1f90c + i;
      this.#cards.push(new Card(value), new Card(value));
    }

    this.#cards = this.#cards.sort(() => Math.random() - 0.5);
  }

  getCardsNumber() {
    return this.#cards.length;
  }

  getCard(index) {
    return this.#cards[index];
  }
}
```

FIGURE 7 – Classe Memory

— Le constructeur de la classe Memory initialise l'attribut cards avec un tableau vide.
— La méthode newGame crée un certain nombre de paires de carte en fonction du paramètre "pairsNumber".
— La méthode getCardsNumber retourne le nombre de cartes présentes dans le tableau cards.
— La méthode getCard retourne la carte située à la position index du tableau cards.

On s'intéresse maintenant au controller-memory.js. Il aura pour but d'établir la partie, on applique les modifications et on obtient la classe suivante :

```
import { Memory } from "../models/memory.js";
import { Notifier } from "../patterns/notifier.js";

export class ControllerMemory extends Notifier
{
    #memory;
    //#card = null;
    constructor()
    {
        super();
        this.#memory = new Memory();
    }

    get memory() { return this.#memory };

    newGame() {
        this.#memory.newGame(10);
    }
```

FIGURE 8 – Classe controller-memory.js

On vient modifier également le fichier view-memory.js pour afficher les cartes crées par displayCards().

```
notify()
{
    this.displayCards();
}

displayCard(card){
    const cards = document.querySelector(".cards");
    let enfant = document.createElement("div");
    enfant.classList.add("card");
    enfant.innerHTML = "&#x" + (card.value).toString(16);
    cards.appendChild(enfant);
    //enfant.addEventListener('click', () => this.#controllerMemory.createCard());
}

displayCards(){
    for(let i = 0; i < this.#controllerMemory.memory.getCardsNumber(); i++){
        this.displayCard(this.#controllerMemory.memory.getCard(i));
    }
}
```

FIGURE 9 – Classe view-memory.js

On vient également modifier le constructeur du fichier application pour appeler newGame. On modifie alors par this.#controllerMemory.newGame() ; . En retournant sur la page web, on obtient :

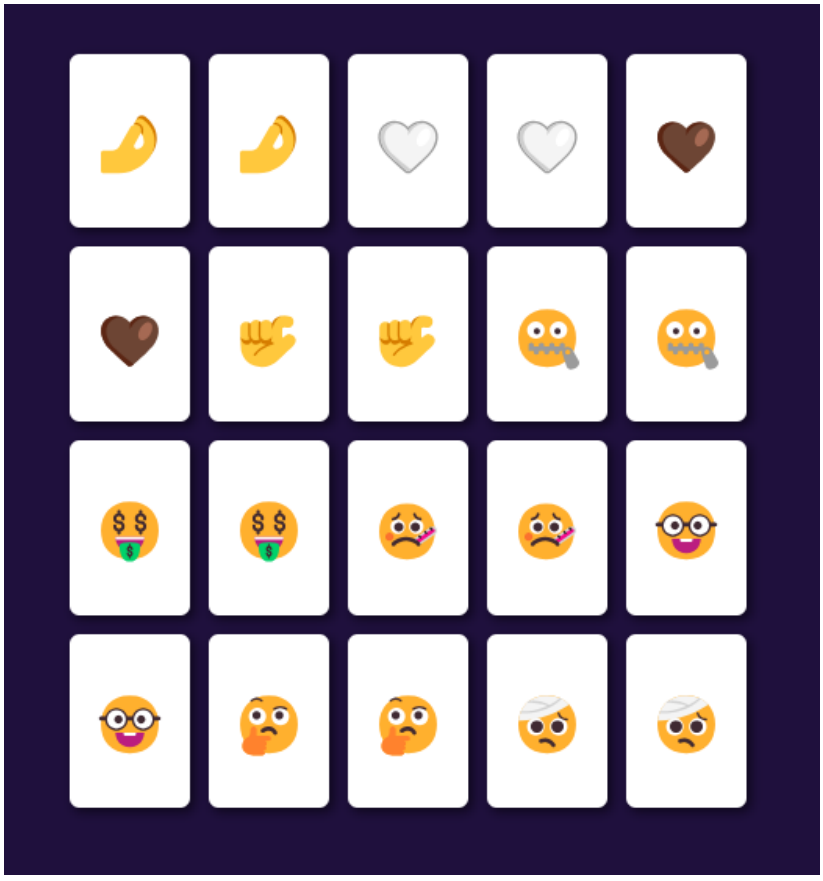


FIGURE 10 – Test des méthodes newGame()

Pour le moment, on obtient un début de memory. Cependant, on remarque que chaque paire se trouve les unes à coté des autres. Par conséquent, on va rajouter une part d'aléatoire afin de le complexifier. Il nous est conseillé d'utiliser le ".splice" cependant je n'y suis pas parvenu. Par conséquent, j'ai simplement utilisé le ".sort" et Math.random() pour y parvenir. Après exécution, on a :

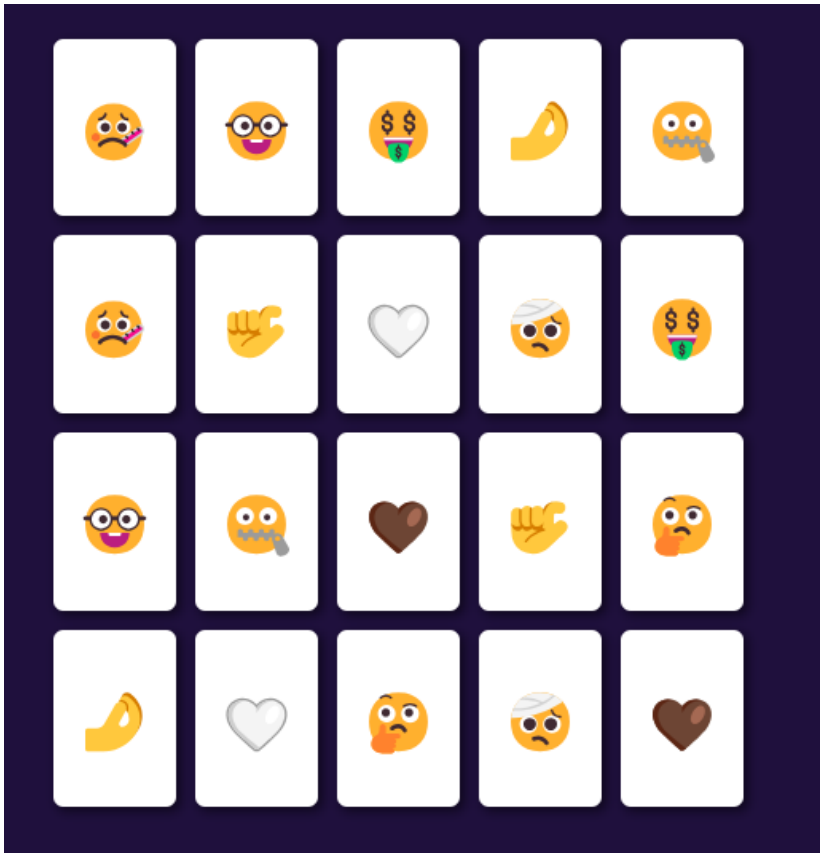


FIGURE 11 – Test de l'aléatoire

On a les 10 paires de cartes mais cette fois ci elle ne se trouve plus les unes à coté des autres mais placés de manière aléatoire.

1.4 Enregistrement

Il serait bien malheureux de perdre notre partie parce que nous avons dû quitter notre poste un instant. C'est pourquoi, on s'intéresse à réaliser un enregistrement de la partie. On va donc ajouter une fonctionnalité qui enregistrera l'état de la partie à chaque changement.
Pour cela, on ajoute une méthode saveGame() dans notre fichier controller :

```
newGame() {
    this.#memory.newGame(10);
    this.saveGame();
    this.notify();
}

saveGame(){
    localStorage.setItem("memory",this.#memory);
}
```

FIGURE 12 – Méthode SaveGame()

Pour vérifier le bon fonctionnement de cette dernière, on se rend sur notre navigateur et on affiche la console (Ctrl + Maj + J). On observe alors :

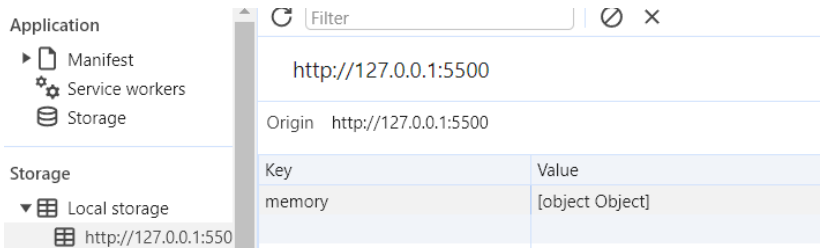


FIGURE 13 – Test de la méthode SaveGame()

On peut voir que la valeur stockée est [object Object], ce qui ne ressemble pas vraiment à un Memory et son attribut cards. Cela vient du fait que le LocalStorage ne peut conserver que des chaines de caractères. Une bonne solution à cela, est d'utiliser le JSON. Pour cela, on vient tout simplement modifier la ligne que nous avons fait précédemment :

```
saveGame(){
    //localStorage.setItem("memory",this.#memory);
    localStorage.setItem("memory",JSON.stringify(this.#memory));
}
```

FIGURE 14 – Modificattion de la méthode SaveGame()

Après exécution, on a :

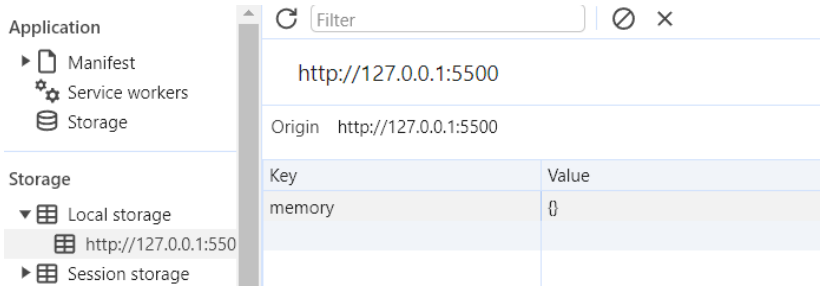


FIGURE 15 – Test de la méthode SaveGame()

On obtient un objet vide c'est-à-dire qu'il n'a aucun attribut. Pourtant, Memory a bien un attribut cards. Mais cards est un attribut privé, il ne peut donc pas être vu par la méthode JSON.stringify.

1.5 Sérialisation/Déserialisation

On a vu précédemment que la méthode saveGame() ne suffisait pas pour renvoyer les informations contenues dans le tableau. Par conséquent, on va faire appel à une nouvelle méthode toData().

```
toData(){
    const myCard = {
        value: this.value,
    }
}
```

FIGURE 16 – Méthode toData() dans card

```
toData(){
    let myCards = [];
    for(let i = 0; i < this.getCardsNumber(); i++){
        const tableau = {
            value: this.getCard(i).value,
        }
        myCards.push(tableau);
    }

    const myMemory = {
        myCards,
    }

    return myMemory;
}
```

FIGURE 17 – Méthode toData() dans memory

On vient modifier la méthode saveGame() pour qu'elle appelle la méthode toData() que l'on vient de créer. On obtient alors :

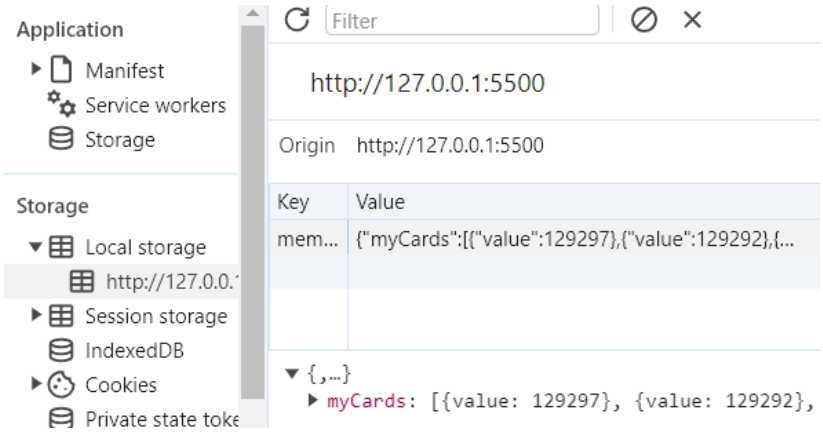


FIGURE 18 – Test des méthodes toData()

1.6 Chargement

Maintenant que le Memory est enregistré, il faut le charger au démarrage de l'application.

```
fromData(myMemory){
    this.#cards = [];
    for(let i = 0; i < myMemory.myCards.length; i++){
        this.#cards.push(myMemory.myCards[i]);
    }
    return this.#cards;
}
```

FIGURE 19 – Méthode fromData()

On va réaliser l'inverse de la méthode toData().

La méthode loadGame va permettre de charger les informations, elle retournera true si des données ont bien été chargées et false sinon.

```
loadGame(){
    const savedData = localStorage.getItem("memory");
    if(savedData){
        this.#memory.fromData(JSON.parse(savedData));
        this.notify();
        return true;
    }
    else{
        return false;
    }
}
```

FIGURE 20 – Méthode loadGame()

On vient maintenant modifier le constructeur pour que ce dernier appel start() au lieu de newGame(). On a :

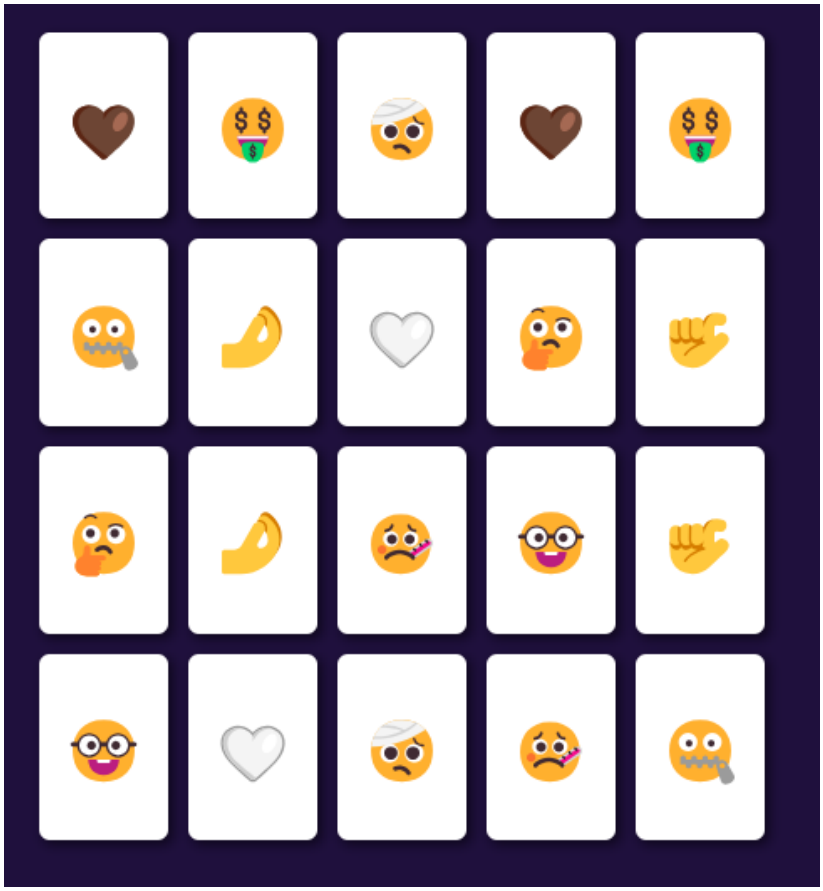


FIGURE 21 – Appel de la fonction start()

On aura beau rafraichir la page web un grand nombre de fois, les cartes ne changeront pas de position.

Lorsque l'on ferme complètement le navigateur (toutes les instances) et qu'on le démarre à nouveau pour charger votre page. On remarque que toutes les données ont disparu par conséquent, il va falloir mettre en place un sessionStorage. On vient alors modifier quelques lignes dans notre code initial.

```
saveGame(){
    //localStorage.setItem("memory",this.#memory);
    //localStorage.setItem("memory",JSON.stringify(this.#memory));
    //localStorage.setItem("memory",JSON.stringify(this.#memory.toData()));
    sessionStorage.setItem("memory",JSON.stringify(this.#memory.toData()));
}

loadGame(){
    //const savedData = localStorage.getItem("memory");
    const savedData = sessionStorage.getItem("memory");
    if(savedData){
```

FIGURE 22 – sessionStorage