PROGRAMMATION WEB

JavaScript

POO & LOCALSTORAGE

charles.meunier@u-bourgogne.fr

OBJECTIFS

Réaliser un Memory, ce jeu où il faut retrouver des pairs de cartes présentées initialement face retournée.

A travers cet exercice vous manipulerez les notions suivantes :

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

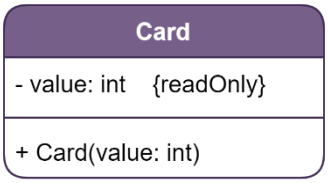
PREPARATION

- Créez un dossier pour le projet.
- Téléchargez et décompresser dans ce dossier les fichiers de l'archive « JS-LocalStorage.zip »
- Ouvrez le projet du dossier avec Visual Studio Code.

UNE CARTE

MODELE

- Dans le dossier `js/models` du projet, implémentez la classe suivante :



L'attribut `value` disposera d'un getter.

CONTROLEUR

- Dans le fichier `js/controllers/controller-memory.js`, ajoutez un attribut `card` de type `Card` à la classe `ControllerMemory`. L'attribut `card` disposera d'un getter.
- Ajoutez une méthode `createCard` qui initialise l'attribut `card` avec une instance de `Card` ayant une valeur aléatoire comprise entre `0x1F90C` et `0x1F9FF`, puis notifie les observateurs du contrôleur.

> Note : lorsqu'une valeur est précédée de `0x` cela signifie qu'elle est exprimée en hexadécimal (base 16).

VUE

- Dans le fichier `js/views/view-memory.js`, ajoutez une méthode `displayCard` à la classe `ViewMemory`. Cette méthode ajoutera à la balise `cards` de la page `index.html`, un élément HTML de type `div` ayant la classe CSS `card`.
  La valeur de la carte sera utilisée pour générée un caractère spécial HTML qui sera affiché dans la div créé précédemment :

```
<p>Voici le caractère spécial qui affiche un smiley pas content &#x1f92c</p>
```

> Note : il est possible de convertir un entier en une chaine de caractères représentant cet entier dans la base souhaitée :
>
> ```
> // Représentation en base 8 (octale)
> const myString = myInt.toString(8);
> ```

- Appelez la méthode `displayCard` dans la méthode `notify` de `ViewMemory`.

APPLICATION

- Dans le fichier `js/application/application-memory.js`, appelez la méthode `createCard` du contrôleur dans le constructeur de la classe `ApplicationMemory`, après l'appelle de la fonction `#initViews`.

TEST

- Sélectionnez le fichier `index.html` de votre projet et démarrez LiveServer.
- Une carte devrait apparaître dans le centre de votre écran. Si ce n'est pas le cas observez la console de développement et déboguez votre programme pour trouver l'origine du problème.

DES CARTES

- Modifiez le code de votre programme pour qu'une nouvelle carte soit générée à chaque fois que l'on clique sur la carte à l'écran.

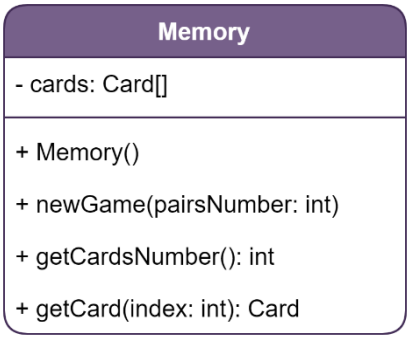
  Une seule fonction devra être modifiée et la modification apportée ne devra pas excédée 5 lignes de code.

- Testez le bon fonctionnement.

LE JEU

MEMORY

- Dans le dossier `js/models`, implémentez la classe `Memory` ci-dessous :



  - Le constructeur de la classe `Memory` initialisera l'attribut `cards` avec un tableau vide.

  - La méthode `newGame` créera un certain nombre de paires de carte en fonction du paramètre `pairsNumber`.

- Les valeurs des cartes seront définies dans l'ordre à partir de la valeur `0x1F90C`.

- La méthode `getCardsNumber` retournera le nombre de cartes présentes dans le tableau `cards`.

- La méthode `getCard` retourne la carte située à la position `index` du tableau `cards`.

CONTROLLER

- Retirez l'attribut `card` ainsi que la méthode `createCard` du contrôleur `ControllerMemory`.
- Ajoutez un attribut `memory` qui sera initialisé par le constructeur du contrôleur avec une nouvelle instance de la classe `Memory`. L'attribut `memory` disposera d'un `getter`.
- Ajoutez une méthode `newGame` qui appellera la méthode `newGame` de memory pour 10 paires de cartes et notifiera les observateurs du contrôleur.

VIEW

- Modifiez la méthode `displayCard` de la classe `ViewMemory` afin qu'elle affiche une carte qui lui sera passée en paramètre.
- Ajoutez une méthode `displayCards` qui appellera la fonction `displayCard` pour chacune des cartes gérées par le `Memory` du `Contrôleur`.
- Modifiez la méthode `notify` de `ViewMemory` afin de n'y appeler que la méthode `displayCards`.

APPLICATION

- Modifiez le constructeur de `ApplicationMemory` afin de remplacer l'appel à `createCard` par `newGame`.
- Testez le bon fonctionnement.

UN PEU D'ALEATOIRE

- Modifiez la méthode `newGame` de `Memory` afin que les cartes soient ajoutées dans un ordre aléatoire dans le tableau `cards` et non plus regroupées par paire comme à présent.

> Note : les tableaux JavaScript possèdent une méthode `splice` qui a plusieurs rôle (voir la doc) dont celui d'insérer des éléments dans un tableau à un indice spécifique :
>
> ```
> const array = [1, 2, 3];
>
> // Insérer la valeur 5 avant l'élément présent à l'indice 1 de array.
> array.splice(1, 0, 5);
> ```
>
> Le second paramètre doit être à `0`. Il indique le nombre d'éléments à supprimer à partir de l'indice donné. C'est une autre fonctionnalité de `splice`.

- Testez le bon fonctionnement

ENREGISTREMENT

Une partie de Memory peut prendre du temps, surtout si le nombre de cartes à trouver augmente. Or le joueur peut être interrompu par un appel voire une coupure de courant qui l'obligerait à tout reprendre de zéro. Le drame !

Nous allons donc ajouter une fonctionnalité qui enregistrera l'état de la partie à chaque changement. Ainsi, lorsqu'il rechargera la page, le joueur retrouvera sa partie là où il l'avait laissée.

CONTROLLER

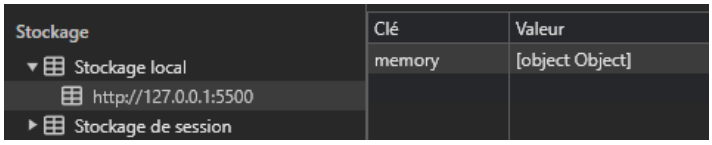
- Ajoutez une méthode `saveGame` au contrôleur qui enregistrera la partie de Memory en cours.

> Note : les navigateurs disposent d'un espace local dans lequel les sites Internet et applications Web peuvent enregistrer des informations. En fonctions des navigateurs, 5 à 10 Mo sont ainsi disponible pour stocker des données sur l'ordinateur de l'utilisateur.
>
> ```
> // Enregistrer une information dans le LocalStorage
> localStorage.setItem("data_name", myData);
>
> // Charger une donnée depuis le LocalStorage (retourne false si aucune donnée
> // n'est trouvée)
> const myData = localStorage.getItem("data_name");
> ```

- Appelez la méthode `saveGame` à la fin de la méthode `newGame`.

  Actualisez la page de votre navigateur, ouvrez les outils de développement et rendez-vous dans la rubrique `Appli` puis `Stockage local` et enfin `http://127.0.0.1`.

  Vous devriez voir quelque chose comme ceci :



Vous pouvez 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. On ne peut pas y stocker directement des objets complexes tels qu'un tableau ou une instance de classe.

JSON

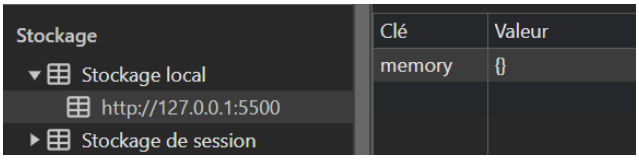
Heureusement, JavaScript nous offre une API pour créer une chaine de caractères au format JSON à partir d'un objet JavaScript.

```
// Convertir une donnée JavaScript en chaine de caractères JSON
const myDataAsString = JSON.stringify(myDataObject);

// Convertir une chaine de caractères JSON en donnée JavaScript
const myDataObject = JSON.parse(myDataAsString);
```

- Modifiez la méthode `saveGame` de `ControllerMemory` pour enregistrer les données du `Memory` au format JSON.
- Testez à nouveau.

Vous devriez obtenir le résultat suivant :



En JSON, `{ }` représente un objet... vide. C'est à dire qu'il n'a aucun attribut. Pourtant, `Memory` bien un attribut `cards`. Mais `cards` est un attribut privé, il ne peut donc pas être vu par la méthode `JSON.stringify`.

SERIALISATION / DESERIALISATION

Il est souvent complexe de sérialiser automatiquement les données d'une instance de classe. Les attributs privés n'apparaissent pas, quand d'autres données peuvent être exportées alors que cela n'est pas nécessaire. Bref, il est souvent préférable de gérer manuellement les données que l'on souhaite voir sérialiser.

CARD

- Ajoutez une méthode `toData` à la classe `Card` qui retournera un objet JavaScript "de base" contenant les données de l'instance.

```
// Créer un objet JavaScript de base
const myObject = {
    firstAttributeName: firstAttributeData,
    secondAttributeName: secondAttributeData,
    // ...
}
```

MEMORY

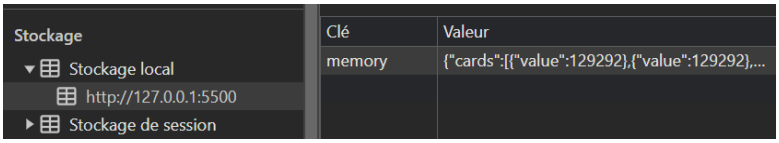
- Ajoutez une méthode `toData` à la classe `Memory` qui retournera un objet JavaScript "de base" contenant les données de l'instance, soit le tableau de cartes. Plus exactement, un tableau d'objets JavaScript représentant des cartes et non des instances de `card`.

  Allons, allons, en réfléchissant un peu, vous allez y arriver :-)

CONTROLLER

- Modifiez la méthode `saveGame` du contrôleur pour enregistrer les données sérialisables du Memory.
- Testez une nouvelle fois.

Vous devriez obtenir cela :



CHARGEMENT

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

MEMORY

- Ajoutez une méthode `fromData` à la classe `Memory` qui prendra en paramètre un objet JavaScript de base contenant les données d'un Memory.

  Il s'agit, en réalité, de faire l'opération inverse de `toData`.

  Pensez à vider le tableau de cartes avant de créer les nouvelles cartes issues des données reçues en paramètre.

CONTROLLER

- Ajoutez une méthode `loadGame` qui appellera chargera les données enregistrées dans le `LocalStorage`.

  La méthode `loadGame` retournera `true` si des données ont bien été chargées et false sinon.

- Ajoutez une méthode start au contrôleur qui appellera la méthode `loadGame`. Si cette dernière renvoie false, les méthode start appellera `newGame` pour démarrer une nouvelle partie.

APPLICATION

- Modifiez le `controleur` de la classe `ApplicationMemory` afin d'appeler start au lieu de `newGame`.
- Testez le bon fonctionnement.

  A chaque rechargement de la page, vos cartes ne doivent plus changer de position.

SESSIONSTORAGE

- Essayez de charger votre page depuis un nouvel onglet. Puis fermez complètement votre navigateur (toutes les instances) et démarrez-le à nouveau pour charger votre page. Toujours les données ?
- Remplacez dans votre code `localStorage` par `sessionStorage` et refaites les tests précédents. Que remarquez-vous ?

BONUS DU DEVELOPPEUR

CARTE

- Ajoutez à la classe `Card` un attribut `faceHidden` initialisé à `true`. Cet attribut disposera d'un getter.
- Ajoutez deux méthodes `show` et `hide` qui affecte respectivement les valeurs `false` et `true` à l'attribut `faceHidden`.
- Modifiez la classe `ViewMemory` afin que les cartes ayant leur attribut `faceHidden` à `true` aient la classe CSS `hidden`.
- Pensez à modifiez les méthodes `toData` de `Card` et `fromData` de `Memory`
- Testez le fonctionnement.

MEMORY

- Ajoutez à la classe Memory une méthode `showCard` qui prendra en paramètre l'indice de la carte à montrer.

- Si la carte est déjà retournée, la fonction n'a aucun effet.
- Si la carte est la première d'une série à être retournée elle reste retournée jusqu'à ce qu'une autre carte soit également retournée.
- Si la carte est la seconde d'une série à être retournée et qu'elle est identique à la première de la série, alors les deux cartes restent visibles jusqu'à la fin de la partie, sinon, les deux cartes sont masquées au bout d'une seconde pour que l'utilisateur ait le temps de voir la carte.
- Si toutes les cartes sont retournées, nouvelle partie !

CONTROLLER

- Ajoutez une méthode `showCard` à la classe `ControllerMemory`. Cette méthode prendra en paramètre l'indice de la carte à montrer. Elle appellera la méthode `showCard` de Memory et notifiera ensuite les observateurs.

VIEW

- Modifiez la classe `ViewMemory` pour que la méthode `showCard` du contrôleur soit appelée lorsque l'on clique sur une carte.
- Testez le bon fonctionnement et que la partie est bien sauvegardée à chaque étape.

BONUS DU PRO DU CSS

- Animez le retournement de la carte à l'aide de l'attribut CSS `transform`.