Jeu de la vie
Javascript Core

P. REIGNIER
Seoc
February 12, 2018

1 Introduction et mise en place initiale

L'objectif de ce TP est de vous familiariser avec la syntaxe javascript. Le code que vous allez écrire s'exécutera dans le navigateur Chrome afin d'avoir une sortie graphique mais ce n'est pas ici le point principal.

Le thème choisit est le jeu de la vie[1] (simulation d'automates cellulaires). La projet initial vous est fourni. Il contient la page devant être chargée dans le navigateur ainsi que le chargement et l'initialisation de la librairie graphique utilisée : `paper.js`[2].

1. Récupérez l'archive life.zip sur chamilo. Cette archive est un projet netbeans.
2. Ouvrez le projet depuis Netbeans.
3. Vérifiez que le navigateur par défaut est bien chrome ou chromium (figure 1)

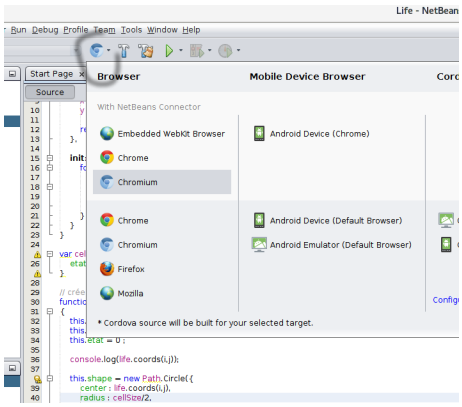

Figure 1: Navigateur utilisé pour exécuter le projet

4. Lancez le projet. Il est possible que netbeans vous demande d'installer l'extension `netbeans connector` sur votre navigateur. Cette extension s'installe depuis le Chrome

[1] `https://fr.wikipedia.org/wiki/Jeu_de_la_vie`
[2] `http://paperjs.org`

Web Store. Une fois le projet "lancé", Netbeans se connecte au navigateur via le connecteur. Cela lui permet d'une part de récupérer la console Javascript de Chrome afin de l'afficher dans sa fenêtre `output`. Cela lui permet également, dès que vous sauvegarder un fichier dans l'IDE, de rafraichir l'affichage dans le navigateur.

Attention : n'oubliez pas de systématiquement utiliser `this` lorsque vous désignez un attribut ou une méthode à l'intérieur d'un objet.

2 Objet life

On va d'abord créer l'environnement. L'environnement est un plateau carré hébergeant les cellules. Il y a `size` cellules horizontalement, et `size` cellules verticalement. Chaque cellule sera représentée graphiquement par un cercle de diamètre `cellSize`. Les cellules seront stockées dans un tableau Javascript à deux dimensions. Un tableau à deux dimensions est un tableau dont les cases sont elle-mêmes des tableaux.

Créez une variable `life` de type objet possédant :

- un attribut `board` de type tableau (valeur initiale vide)
- deux attributs `borderX` et `borderY`. Ces attributs correspondent aux coordonnées graphique dans la page du coin en haut à gauche du plateau contenant les cellules. Ces valeurs sont calculées en tenant compte de la taille du plateau (nombre de cellules * taille de chaque cellule) et de la taille de la zone d'affichage fournie par `view.size.width` et `view.size.height` :

$$\begin{cases} borderX = (view.size.width - size * cellSize)/2 \\ borderY = (view.size.height - size * cellSize)/2 \end{cases}$$

- une fonction `coords(i,j)` retournant un tableau à deux éléments `[x,y]` correspondant aux coordonnées graphiques de la cellule `(i,j)` sur le plateau.

$$\begin{cases} x = this.borderX + i * cellSize; \\ y = this.borderY + j * cellSize; \end{cases}$$

3 Objet cell

Nous allons maintenant créer une fonction constructeur (appel avec `new`) permettant de créer des instances de cellules. Cette fonction `Cell` admet deux arguments (i,j). La cellule créée devra avoir :

- deux attributs i et j correspondant à ses coordonnées sur le plateau (initialisée par les valeurs passées en argument).
- un attribut `state` valant `0`
- un attribut `previousState` valant `0`

- un attribut `shape` contenant la représentation graphique de type cercle. Cette représentation est initialisée par la fonction constructeur `Path.Circle()`. Cette fonction attend un objet comme argument dont les attributs seront les suivants :
  - `center` : le centre du cercle calculé à partir de `(i,j)` grâce à la méthode `coords`,
  - `radius` le rayon du cercle (`cellSize/2`),
  - `fillColor` : la couleur de remplissage (`'white'` pour la cellule morte),
  - `strokeColor` : la couleur du pourtour (`'blue'`).

Afin de vérifier que votre fonction est correcte, créez une cellule en `(1,1)` : elle doit apparaître dans votre page. Supprimez ensuite cet appel de test.

4 Initialisation du plateau

Ajoutez une méthode `init()` à la variable `life`. Cette méthode doit créer $size * size$ cellules et les stocker dans le tableau `board`. Remarque : `board` est initialement un tableau à une seule dimension (cf section 2). Il faut penser à initialiser chaque case à un nouveau tableau vide avant de pouvoir utiliser la "deuxième" dimension.

L'appel à `life.init()` doit maintenant faire apparaître le plateau.

5 Mise à jour de l'état d'une cellule

Complétez la fonction constructeur d'une cellule afin d'ajouter deux méthodes :

1. la méthode `live()` : mise à jour de l'attribut `state` de l'objet et affectation de la couleur de remplissage du cercle (attribut `fillColor` de l'attribut `shape` de la cellule à `'cyan'`.
2. la méthode `die()` : idem avec couleur de remplissage à `'white'`

Remarque : affecter l'attribut `fillColor` provoque le rafraichissement de l'affichage dans la page. Testez ces deux méthodes par invocation directe.

6 Mise à jour du plateau

L'algorithme de mise à jour du plateau est le suivant. A chaque étape, l'évolution d'une cellule est entièrement déterminé par l'état de ses 8 voisines (voir figure 2) :

- une cellule morte possédant exactement 3 voisines vivantes devient vivante
- une cellule vivante possédant 2 ou 3 voisines vivantes reste vivante, sinon elle meurt

Le calcul de l'état des cellules va s'effectuer de manière séquentielle. Les valeurs des cellules à l'instant t est calculé à partir de l'état des cellules à l'instant $t-1$. Il faut donc d'abord geler l'état courant (qui devient celui à $t-1$) et calculer à partir de celui-ci sans le modifier le nouvel état à l'instant t. Cette opération est réalisée par la méthode `saveState` qui sauve pour chaque cellule son état courant dans son attribut `previousState`.

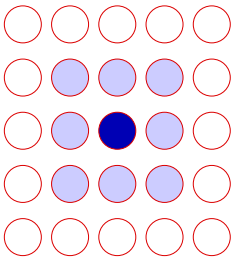

Figure 2: Les 8 voisins d'une cellule

- Complétez la définition de l'objet `life` en ajoutant la méthode `saveState()`

De manière à éviter des cas particuliers dans l'algorithme de mise à jour de l'état des cellules, on considère que tout ce qui est en dehors de plateau est mort.

- Complétez la définition de l'objet `life` afin d'ajouter la méthode `getPreviousSate(i,j)` retournant l'état précédent de la cellule si `(i,j)` est dans le plateau, 0 sinon.
- Ecrire la méthode `iterate()` de `life` réalisant une itération du jeu de la vie (n'oubliez pas lire l'état dans `previousState` et de mettre à jour dans `state`).

7 Animation

Il ne reste plus qu'à animer l'ensemble. Si vous définissez une fonction sans argument dont le nom est `onFrame()`, elle sera alors appelée 60 fois par seconde.

- Ecrivez cette fonction qui devra appeler la méthode `iterate()` de `life()`
- Au lieu d'écrire cette fonction, vous pouvez également si vous le souhaitez faire du pas en pas en n'appelant `iterate()` que sur réception d'un événement clavier :

```
function onKeyUp(event) {
     if (event.key ==’g’) {
         console.log("Step");
         life.iterate();
     }
 }
```

- Testez votre jeu de la vie avec la configuration initiale suivante (barre verticale) :

```
life.board[5][4].live();
life.board[5][5].live();
life.board[5][6].live();
```

  Cette barre doit se transformer en une barre horizontale, puis une barre verticale etc. (voir wikipedia)
- Et maintenant une plus compliquée (explosion) :

```
life.board[5][4].live();
life.board[5][5].live();
life.board[5][6].live();
life.board[5][7].live();
life.board[5][8].live();

life.board[9][4].live();
life.board[9][5].live();
life.board[9][6].live();
life.board[9][7].live();
life.board[9][8].live();

life.board[7][8].live();
life.board[7][4].live();
```