Compte Rendu SIR

Tp4 : JavaScript

Crée par :

-Kaoutar Bennouna

-Mohammed Ayoub Boukadida

Encadré par Mr:

-Olivier Barais

M1 MIAGE

21 février 2016

Compte Rendu SIR

Tp4 : JavaScript

Table des matières

I- Intégration : le glisser-Déposer (Drag-n-Drop)

1- Question 1 :

Créer une classe DnD contenant les 4 attributs suivants initialisé à 0 : les coordonnées de la position initial du DnD ; celles de la position finale sur le fichier interaction.js :

```
function DnD(canvas, interactor) {


   this.xI = 0;
   this.yI= 0;
   this.xF= 0;
   this.yF= 0;

...
}
```

2- Question 2 :

Déclarer 3 fonctions à cette classe correspondant aux 3 types d'événements à gérer, gestion de la pression, du déplacement et du relâchement. Ces fonctions vont nous permettre de connaitre la position de la souris de l'utilisateur et de les afficher à la console.

```
this.maFctGerantLaPression= function(evt) {
        if(this.boutonPressee==false) {
            this.boutonPressee=true;

            this.xI = getMousePosition(canvas,evt).x;
            this.yI = getMousePosition(canvas,evt).y;
            this.xF = getMousePosition(canvas,evt).x;
            this.yF = getMousePosition(canvas,evt).y;

            console.log("x initial"+this.xI);
            console.log("y initial"+this.yI);
            console.log("x final"+this.xF);
            console.log("y final"+this.yF);


        }
    }.bind(this) ;

    this.maFctGerantLeDeplacement=function(evt) {
      if(this.boutonPressee==true){

          this.xF=getMousePosition(canvas,evt).x;
          this.yF=getMousePosition(canvas,evt).y;

          console.log("x initial"+this.xI);
          console.log("y initial"+this.yI);
          console.log("x final"+this.xF);
          console.log("y final"+this.yF);

      }
    }.bind(this);

    this.maFctGerantLeRelachement=function(evt) {
```

```
            console.log("x initial"+this.xI);
            console.log("y initial"+this.yI);
            console.log("x final"+this.xF);
            console.log("y final"+this.yF);
            if(this.boutonPressee==true){
                this.boutonPressee=false;


                this.xI = 0;
                this.yI =0;
                this.xF = 0;
                this.yF =0;
        }
    }.bind(this) ;
```

3- Question 3 :

Implémenter ces fonctions :

```
function getMousePosition(canvas, evt) {
  var rect = canvas.getBoundingClientRect();
  return {
    x: evt.clientX - rect.left,
    y: evt.clientY - rect.top
  };
};
```

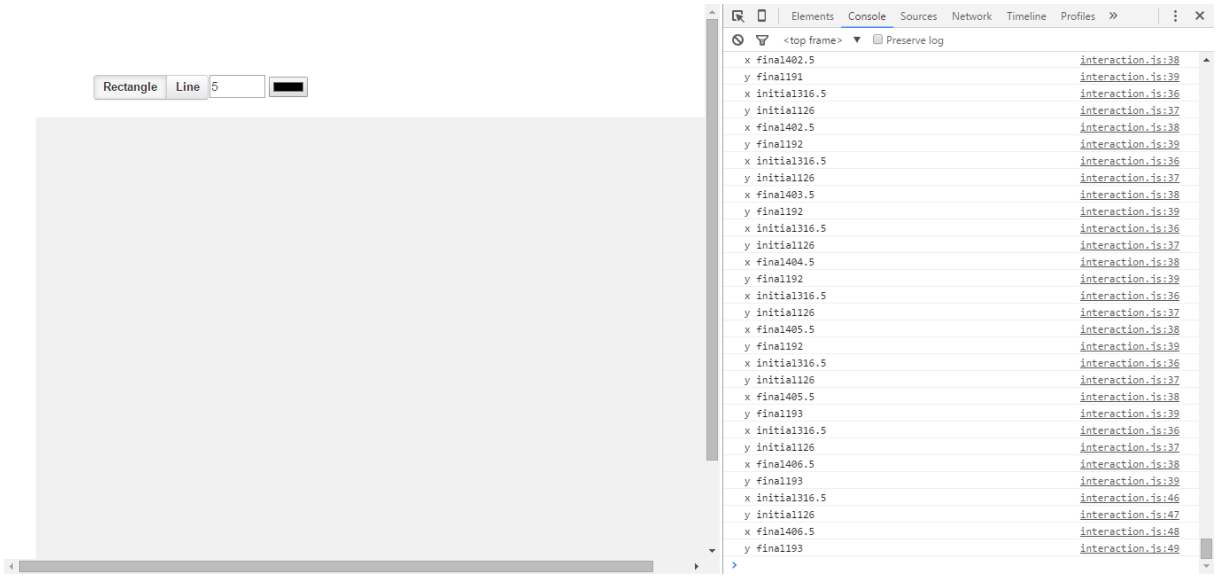
4- Question 4:

Enregistrer chaque fonction auprès du canvas :

```
    canvas.addEventListener('mousedown', this.maFctGerantLaPression,
false);
    canvas.addEventListener('mousemove',
this.maFctGerantLeDeplacement, false);
    canvas.addEventListener('mouseup', this.maFctGerantLeRelachement,
false);
};
```

5- Question 5 :

Affichage du résultat sur la console (F12) du canvas.html après avoir fait appel à console.log et en faisant bouger la souris sur le rectangle gris nous avons obtenu les résultats suivant sur la console :



II- Le modèle :

Le modèle au sens MVC, définit le modèle de données du logiciel. Sur le mini éditeur il va définir les concepts du dessin dans le fichier model.js et en utilisant la classe / fonction Drawing (formes, dessin …)

6- Question 6 :

Implémenter les 4 classes nécessaires pour définir le modèle dans le fichier *model.js*. Pensez à dé-commenter les lignes nécessaires dans le fichier *main.js*.

```
function Drawing() {

    this.forme = new Array();

    this.addForme = function(forme){
    this.forme.push(forme);
    };

    this.removeForme = function(index){
    this.forme.splice(index,1);
    };

};


function forme(epaisseur, couleur) {
    this.epaisseur=epaisseur;
    this.couleur=couleur;
};


function Rectangle(orgX, orgY, larg, haut, epaisseur, couleur) {
    forme.call(this, epaisseur, couleur);
    this.orgX=orgX;
    this.orgY=orgY;
    this.larg=larg;
```

```
    this.haut=haut;
};
Rectangle.prototype = new forme();


function Line(xA, yA, xB, yB, epaisseur, couleur) {
    forme.call(this, epaisseur, couleur);
    this.xA=xA;
    this.yA=yA;
    this.xB=xB;
    this.yB=yB;
};
Line.prototype = new forme();
```

III- La vue:

La vue est une representation graphique possible du modèle. Pour le mini éditeur la vue va consister à peindre dans un canvas les formes du modèle.

7- Question 7:

Ajouter les fonctions d'affichage (fonction paint) dans le fichier view.js et dans chacune des classes. La fonction paint de la classe Form configurera juste la couleur et l'épaisseur du trait du contexte du canvas.

View.js

```
Rectangle.prototype.paint = function(ctx) {

    ctx.lineWidth=this.epaisseur;

    ctx.strokeStyle=this.couleur;
    ctx.beginPath() ;
    ctx.rect(this.orgX, this.orgY, this.larg, this.haut);
    ctx.stroke();
};

Line.prototype.paint = function(ctx) {

    ctx.lineWidth=this.epaisseur;
    ctx.strokeStyle=this.couleur ;
    ctx.beginPath();
    ctx.moveTo(this.xA, this.yA);
    ctx.lineTo(this.xB, this.yB);
    ctx.stroke();
};

Drawing.prototype.paint = function(ctx) {
    ctx.fillStyle = '#F0F0F0'; // changer la couleur du background
    ctx.fillRect(0, 0, canvas.width, canvas.height);
    this.forme.forEach(function(eltDuTableau) {

        eltDuTableau.paint(ctx);
    });
};
```

8- Question 8 :

Tester le code en dé commentant dans le fichier main.js la partie suivante :

```
// Code temporaire pour tester l'affiche de la vue
var rec = new Rectangle(10, 20, 50, 100, 5, '#00CCC0');
rec.paint(ctx);
var ligne = new Line(10, 20, 50, 100, 5, '#00CCC0');
ligne.paint(ctx);
```

Le résultat est en dessinant un rectangle et une ligne :

IV- Le contrôleur :

Le contrôleur est la partie interactive du patron. Il a pour but de transformer les interactions réalisées par l'utilisateur en commandes allant modifier le modèle.

Dans cette partie nous allons définir un interacteur , qui sera une sorte d'outil que l'utilisateur va manipuler afin de réaliser des actions cette interacteur sera nommé Pencil, et sera développé dans le fichier controler.js

9- Question 9 :

Lier la fonction DnD dans le fichier interaction.js avec Pencil dans le controleur.js. Ajouter un appel aux fonctions (onInteraction : Start, Update, End)

```
function DnD(canvas, Pencil) {
                    …
  this.maFctGerantLaPression= function(evt) {
     …
        Pencil.onInteractionStart(this);
     }
  }.bind(this) ;

  this.maFctGerantLeDeplacement=function(evt) {
          …
     Pencil.onInteractionUpdate(this);
   }
```

```
}.bind(this);

this.maFctGerantLeRelachement=function(evt) {
    …

    Pencil.onInteractionEnd(this);
 }

        }.bind(this) ;
```

10- Question 10 et 11:

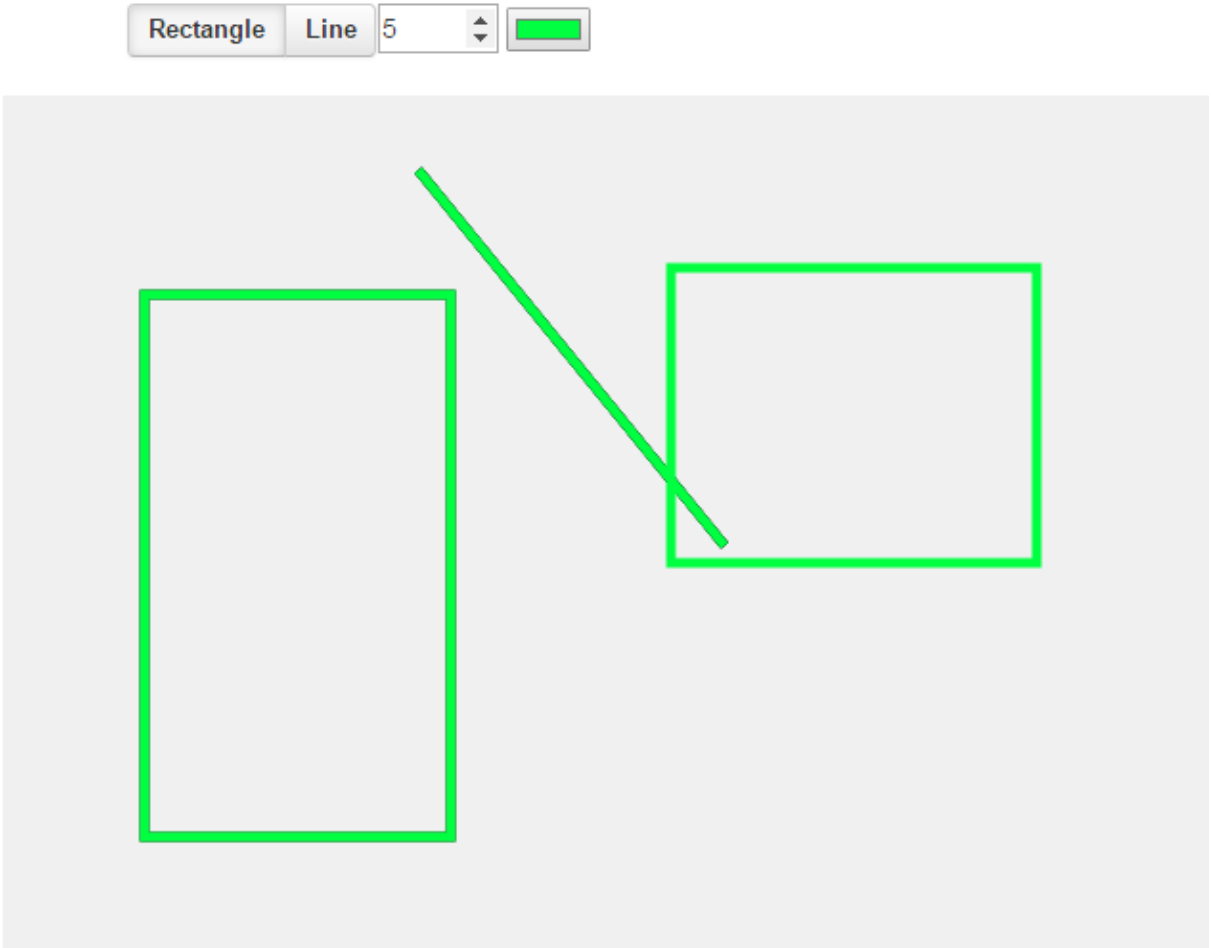
Implémenter les 3 fonctions dans Pencil:

```
function Pencil(ctx, drawing, canvas) {
   this.currEditingMode = editingMode.line;
   this.currLineWidth = 5;
   this.currColour = '#000000';
   this.currentShape = 0;

   new DnD(canvas, this);


//onInteractionStart
 this.onInteractionStart=function(DnD){
   //recuperer les boutons depuis canvas.html afin de verifier
l'�tat des boutons
   var butRect= document.getElementById('butRect');
   //bouton line
   var butLine= document.getElementById('butLine');
   //recuperer la largeur
   var spinnerWidth= document.getElementById('spinnerWidth');
   //recuperer la couleur
   var colour= document.getElementById('colour');

   // leur donner les valeurs definie dans canvs.html
   this.currLineWidth=spinnerWidth.value;
   this.currColour= colour.value;

   //selon le bouton checker rectangle ou encore ligne edditer le
mode
   if(butRect.checked){
      this.currEditingMode=editingMode.rect;
   }else if(butLine.checked){

      this.currEditingMode=editingMode.line;
   }else{
   console.log('la selection nest pas valide');
   }

   // selon la frome selectionnee on va creer les formes
   switch(this.currEditingMode){
      case editingMode.rect:{
      //si c'est un rectangle : en cr�er un
      var larg = DnD.xF-DnD.xI;
      var haut =DnD.yF-DnD.yI;
      this.currentShape = new Rectangle(DnD.xI, DnD.yI, larg, haut,
this.currLineWidth, this.currColour);
         break;
```

```
      }
      case editingMode.line: {
            //si c'est une ligne :
            this.currentShape = new Line(DnD.xI, DnD.yI, DnD.xF,
DnD.yF, this.currLineWidth, this.currColour);
            break;
         }
      default:
            console.log("la forme nexiste pas.");
   }


 }.bind(this);


//onInteractionUpdate

 this.onInteractionUpdate= function(DnD) {
      if(butRect.checked) {
         //Rectangle
         var larg = DnD.xF-DnD.xI;
         var haut = DnD.yF-DnD.yI;
         // la forme actuelle prend un rectangle avec les parametre
suivant
         this.currentShape = new Rectangle(DnD.xI, DnD.yI, larg,
haut, this.currLineWidth, this.currColour);
      }else if(butLine.checked){
         //Ligne
         //la forme actuelle prend une line
         this.currentShape = new Line(DnD.xI, DnD.yI, DnD.xF, DnD.yF,
this.currLineWidth, this.currColour);

      }else{
         console.log('La selection est invalide');
      }
      ctx.clearRect(0, 0, canvas.width, canvas.height);
      drawing.paint(ctx);
      //Dessiner
      this.currentShape.paint(ctx);
   }.bind(this);


//onInteractionEnd

 this.onInteractionEnd= function(DnD) {
      if(butRect.checked) {
         //Rectangle
         var larg = DnD.xF-DnD.xI;
         var haut = DnD.yF-DnD.yI;
         this.currentShape = new Rectangle(DnD.xI, DnD.yI, larg,
haut, this.currLineWidth, this.currColour);
      }else if(butLine.checked){
         //Ligne
         this.currentShape = new Line(DnD.xI, DnD.yI, DnD.xF, DnD.yF,
this.currLineWidth, this.currColour);

      }else{
         console.log('La selection est invalide');
      }
      // lon vide le canvas
      ctx.clearRect(0, 0, canvas.width, canvas.height);
```

```
            //Ajout de la forme dans la liste de dessins du canvas
            drawing.addForme(this.currentShape);
            //la liste de dessins du canvas
            drawing.paint(ctx, canvas);
              //Mise  jour de la liste de formes
               drawing.updateShapeList(this.currentShape);
        }.bind(this);


    };
```

Test :



V- Liste de modification :

11- Question 12 et 13:

Ajouter une fonction updateShapeList dans la vue pour afficher la liste des formes.

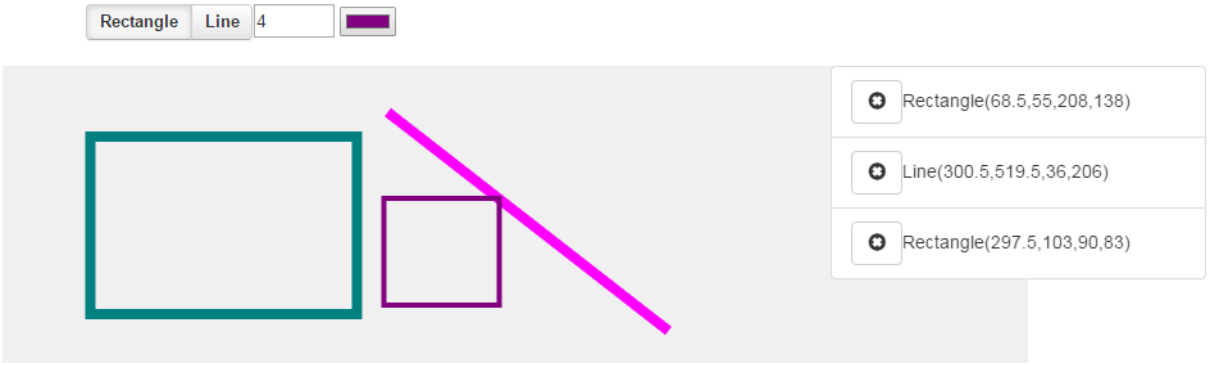
```
Drawing.prototype.updateShapeList = function(forme){
    // Obtenir la liste des formes :shapes
    var myShapeList = document.getElementById('shapeList');
    // Creation d'un li element de la lite
   var li = document.createElement('li');

   var id = myShapeList.childNodes.length;
   // le bouton pour supprimer les formes
   var bouton = document.createElement('button');
   //Creation d'un span
   var span = document.createElement('span');

    // obtenir les coordonées de la position des formes ligne ou
rectangle
    var butRect= document.getElementById('butRect');
    var butLine= document.getElementById('butLine');
    //si c'est une ligne
    if(butLine.checked){
        var rx = forme.xA;
       var ry = forme.xB;
       var rl = forme.yA;
        var rh = forme.yB;
    } else { //si c'est un rectangle
        var rx = forme.orgX;
       var ry = forme.orgY;
       var rl = forme.larg;
        var rh = forme.haut;
    }
    //Changer l'id du bouton
   bouton.setAttribute('id', id);

    bouton.setAttribute('class','btn btn-default');
    //ajouter la croix pour supprimer
    span.setAttribute('class','glyphicon glyphicon-remove-sign');
    //ajouter le span au bouton
    bouton.appendChild(span);

     bouton.setAttribute('onClick', 'drawing.deleteShape('+id+')');
    //ajouter le bouto à li
    li.appendChild(bouton);



       if (forme instanceof Rectangle){
          li.appendChild(document.createTextNode('Rectangle' +'('+
rx+','+ry+','+rl+','+rh+')'));

       } else if(forme instanceof Line){

          li.appendChild(document.createTextNode('Line' +'('+
rx+','+ry+','+rl+','+rh+')'));
       }
```

```
            li.setAttribute('id', 'li'+id);
            li.setAttribute('class', 'list-group-item');
            //ajouter l'element li a note liste myshape
            myShapeList.appendChild(li);

    };
```

12- Question 14 :

Supprimer la forme correspondante dans le dessin.

```
Drawing.prototype.deleteShape = function(id){
//obetenir l'id
   var li = document.getElementById('li'+id);
   var index= $(li).index();
    //supprimer l'element qui contient le bouton et les informations
de la forme
    li.remove();
    //Supprimer la forme
    this.removeForme(index);
    //supprimer le contenu du canvas
   ctx.clearRect(0, 0, canvas.width, canvas.height);
   //redessiner
   drawing.paint(ctx);

};
```



13- Résultat :

On cliquant sur le bouton la forme est supprimée.

Difficultés rencontrées :

- Bug : Dessin du rectangle :

Quand on commence à dessiner avec le rectangle au moment où on bouge la souris il y'a beaucoup de rectangle à l'intérieur et dès qu'on dessine une ligne, on a plus que le rectangle de base. (Résolu)

- DeleteShape

Au premier commit de la dernière partie, je ne supprimais que ce qui était à l'intérieur du rectangle, et pour la ligne un rectangle ailleurs sur le canvas (Résolu)

- Dernière partie (Delete)

Quand je dessinais des dessins; je supprime, et puis je recommence avec un rectangle, il y'a un dessin qui réapparait (le plus récent) →(Résolu) .beginpath sur rectangle, il y'avait aussi un changement de couleur du dernier dessin du réapparaissait

- Sur la console

J'ai cette erreur qui s'affiche dans la console, ('Uncaught TypeError: Cannot read property 'onInteractionStart' of undefined') mais la fonction se lie normalement et s'exécute.