Sanaa Mairouch

Frédéric Rochard

M1 MIAGE

Année 2015-2016

Compte rendu du TP 4
Retour sur les base du développement d'une application Web
Système d'Information Réparti

Encadrant TP : Olivier Barais

Q. 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. Ces attributs seront utilisés plus tard lors de la création de formes.*

On crée la classe DnD dans le fichier *interaction.js*.

```
function DnD(canvas, interactor) {
   // Attributs de la 'classe'
    this.xInitial = 0;
    this.yInitial= 0;
    this.xFinal= 0;
    this.yFinal= 0;
}
```

Q. 2 *Déclarer 3 fonctions à cette classe correspondant aux 3 types d'événements à gérer. Pensez à lier (bind) chaque fonction à la classe. Ces fonctions prennent en entrée un attribut evt correspondant à l'événement produit lors de l'utilisation de la souris. Un tel événement possède les attributs x et y.*

On ajoute les fonctions *maFctGérantLaPression*, *maFctGérantLeDéplacement* et *maFctGérantLeRelâchement* à la classe DnD.

```
function DnD(canvas, interactor) {
   // Attributs de la 'classe'
    this.xInitial = 0;
    this.yInitial= 0;
    this.xFinal= 0;
    this.yFinal= 0;

    this.maFctGérantLaPression= function(evt) {
    }.bind(this) ;
    this.maFctGérantLeDéplacement=function(evt) {
    }.bind(this);
    this.maFctGérantLeRelâchement=function(evt) {
    }.bind(this) ;
}
```

Q. 3 Implémenter ces fonctions. Elles ont pour but de définir la valeur des attributs. Cela va nécessiter de définir un nouvel attribut dans la classe car le code des fonctions liées aux déplacements et au relâchement doit être exécuté uniquement si une pression a été effectuée au préalable. Utiliser la fonction *getMousePosition* qui va placer le point de l'événement relativement à la position du *canvas*.

On implémente les fonctions en affectant aux attributs de la classe les coordonnées récupérés avec la fonction *getMousePosition*.

```
function DnD(canvas, interactor) {
   // Attributs de la 'classe'
    this.xInitial = 0;
    this.yInitial= 0;
    this.xFinal= 0;
    this.yFinal= 0;
    this.boutonPressee=false;

   // Developper les 3 fonctions gérant les événements
    this.maFctGérantLaPression= function(evt) {
```

```
        if(this.boutonPressee==false) {
            this.boutonPressee=true;
            this.xInitial = getMousePosition(canvas, evt).x;
            this.yInitial = getMousePosition(canvas, evt).y;
            this.xFinal=getMousePosition(canvas,evt).x;
            this.yFinal=getMousePosition(canvas,evt).y;
        }
    }.bind(this) ;

    this.maFctGérantLeDéplacement=function(evt) {
      if(this.boutonPressee==true){
          this.xFinal=getMousePosition(canvas,evt).x;
          this.yFinal=getMousePosition(canvas,evt).y;
      }
    }.bind(this);

    this.maFctGérantLeRelâchement=function(evt) {
        if(this.boutonPressee==true){
            this.boutonPressee=false;
            pencil.onInteractionEnd(this);
            //Réinitialisation des coordonnées pour les Drag'n drop
            this.xInitial = 0;
            this.yInitial =0;
            this.xFinal = 0;
            this.yFinal =0;
      }
    }.bind(this) ;

};
```

Q. 4 *Enregistrer chaque fonction auprès du canvas (addEventListener).*

On créer les listener pour la classe DnD pour le canvas pour des évènements correspondant aux actions sur la souris :

```
var Canvas = document.getElementById("myCanvas");
canvas.addEventListener('mousedown', this.maFctGérantLaPression, false);
canvas.addEventListener('mousemove', this.maFctGérantLeDéplacement, false);
canvas.addEventListener('mouseup', this.maFctGérantLeRelâchement, false);
```

Q. 5 *Appeler dans chacune des 3 fonctions console.log pour afficher dans la console Javascript de votre navigateur les coordonnées de chaque événement lors de l'exécution de l'interaction. Vous pouvez maintenant tester le bon fonctionnement de l'interaction en ouvrant la page canvas.html dans votre navigateur, en affichant la console Web et en exécutant un DnD sur la zone de dessin (en gris).*

Nous avons ajouté le code correspondant dans nos 3 fonctions :

```
console.log("********Pression**********");
console.log("x initial"+this.xInitial);
console.log("y initial"+this.yInitial);
console.log("x final"+this.xFinal);
console.log("y final"+this.yFinal);
```

Nous récupérons bien les coordonnées de la souris pour les 3 évènements.

En revanche, le listener détecte 2 évènements ‘mousedown’ lorsqu’on presse le bouton de la souris et 2 évènements ‘mouseup’ lorsqu’on relâche le bouton de la souris.

********Pression********* interaction.js:20:13

x initial78.5 interaction.js:21:13

y initial68 interaction.js:22:13

x final78.5 interaction.js:23:13

y final68 interaction.js:24:13

********Pression********* interaction.js:20:13

x initial78.5 interaction.js:21:13

y initial68 interaction.js:22:13

x final78.5 interaction.js:23:13

y final68 interaction.js:24:13

********Mouvement********* interaction.js:33:11

x initial78.5 interaction.js:34:11

y initial68 interaction.js:35:11

x final78.5 interaction.js:36:11

y final69 interaction.js:37:11

********Mouvement********* interaction.js:33:11

x initial78.5 interaction.js:34:11

y initial68 interaction.js:35:11

x final78.5 interaction.js:36:11

y final69 interaction.js:37:11

********Mouvement********* interaction.js:33:11

x initial78.5 interaction.js:34:11

y initial68 interaction.js:35:11

x final78.5 interaction.js:36:11

y final70 interaction.js:37:11

********Mouvement********* interaction.js:33:11

x initial78.5 interaction.js:34:11

y initial68 interaction.js:35:11

x final78.5 interaction.js:36:11

y final70 interaction.js:37:11

********Mouvement********* interaction.js:33:11

x initial78.5 interaction.js:34:11

y initial68 interaction.js:35:11

x final78.5 interaction.js:36:11

y final71 interaction.js:37:11

********Mouvement********** interaction.js:33:11

x initial78.5 interaction.js:34:11

y initial68 interaction.js:35:11

x final78.5 interaction.js:36:11

y final71 interaction.js:37:11

********Relachement********** interaction.js:42:9

x initial78.5 interaction.js:43:9

y initial68 interaction.js:44:9

x final78.5 interaction.js:45:9

y final71 interaction.js:46:9

********Relachement********** interaction.js:42:9

x initial78.5 interaction.js:43:9

y initial68 interaction.js:44:9

x final78.5 interaction.js:45:9

y final71

Q. 6 *Implémenter les 4 classes nécessaires pour définir le modèle dans le fichier model.js. Pensez à décommenter les lignes nécessaires dans le fichier main.html.*

```
//Classe Drawing
function Drawing() {
    //Déclarer un array
    this.forms = new Array();
    //Méthode pour l'ajout dune forme dans le tableau
    this.addForm = function(form) {
        this.forms.push(form);
    };
    this.removeForm = function(index) {
        this.forms.splice(index,1);
    };
};

//Classe Form
function Form(epaisseur, couleur) {
    this.epaisseur=epaisseur;
    this.couleur=couleur;
};

//Classe Rectangle
function Rectangle(originX, originY, width, height, epaisseur, couleur) {
    Form.call(this, epaisseur, couleur);
    this.originX=originX;
    this.originY=originY;
    this.width=width;
    this.height=height;
};
```

```
//Classe Line
function Line(xA, yA, xB, yB, epaisseur, couleur) {
    Form.call(this, epaisseur, couleur);
    this.xA=xA;
    this.yA=yA;
    this.xB=xB;
    this.yB=yB;
};
```

Q. 7 *Utiliser cette fonctionnalité pour ajouter les fonctions d'affichage (fonction paint) dans chacune des classes. La fonction paint de la classe Forme configurera juste la couleur et l'épaisseur du trait du contexte du canvas. Ces fonctions paint prendront donc en paramètre le contexte du canvas. La fonction paint de la classe Dessin peindra la fond du canvas en gris clair (copier-coller les lignes 10 et 11 du fichier main.js). Vous pouvez utiliser pour dessiner votre rectangle.*

> ctx.rect(initX,initY,finalX,finalY);

> ctx.stroke();

Nous avons créé la fonction *paint* pour chaque forme :

```
// Implémentation des fonctions paint à ajouter dans chacune des classes du
modèle.
Rectangle.prototype.paint = function(ctx) {
    //Affectation couleur et epaisseur
    ctx.lineWidth=this.epaisseur;
    ctx.strokeStyle=this.couleur;
    ctx.rect(this.originX, this.originY, this.width, this.height);
    ctx.stroke();
};

Line.prototype.paint = function(ctx) {
    ctx.beginPath();
    //Affectation couleur et epaisseur
    ctx.lineWidth=this.epaisseur;
    ctx.strokeStyle=this.couleur;
    ctx.moveTo(this.xA, this.yA);
    ctx.lineTo(this.xB, this.yB);
    ctx.stroke();
};

Circle.prototype.paint = function(ctx) {
    ctx.beginPath();
    //Affectation couleur et epaisseur
    ctx.lineWidth=this.epaisseur;
    ctx.strokeStyle=this.couleur;
    ctx.arc(this.xCenter,this.yCenter,this.radius,0,2*Math.PI,true);
    ctx.stroke();
};

Drawing.prototype.paint = function(ctx) {
    ctx.fillStyle = '#F0F0F0'; // set canvas' background color
    ctx.fillRect(0, 0, canvas.width, canvas.height);
    this.forms.forEach(function(eltDuTableau) {
        // now fill the canvas
        eltDuTableau.paint(ctx);
    });
};
```

Q. 8 *Tester votre code avec le code mis en commentaire dans le fichier main.js.*

Nous testons avec le code ci-dessous dans le fichier *main.js*.

```
var rec = new Rectangle(10, 20, 50, 100, 5, '#00CCC0');
rec.paint(ctx);
var ligne = new Line(100, 20, 150, 100, 5, '#00CCC0');
ligne.paint(ctx);
```

Q. 9 La première étape consiste à lier une interaction DnD à Pencil. Dans chacune des 3 fonctions de DnD, ajouter un appel aux fonctions *interactor.onInteractionStart(this);*, *interactor.onInteractionUpdate(this);*, *interactor.onInteractionEnd(this);*. Ces fonctions n'existent pas encore mais elles ont pour but de notifier l'interacteur d'une modification de l'interaction.

Après ajout des appels, nous obtenons les fonctions ci-dessous :

```
// Developper les 3 fonctions gérant les événements
   this.maFctGérantLaPression= function(evt) {
       if(this.boutonPressee==false) {
           this.boutonPressee=true;
           this.xInitial = getMousePosition(canvas, evt).x;
           this.yInitial = getMousePosition(canvas, evt).y;
           this.xFinal=getMousePosition(canvas,evt).x;
           this.yFinal=getMousePosition(canvas,evt).y;
           pencil.onInteractionStart(this);
           /*console.log("********Pression**********");
           console.log("x initial"+this.xInitial);
           console.log("y initial"+this.yInitial);
           console.log("x final"+this.xFinal);
           console.log("y final"+this.yFinal);*/
       }
   }.bind(this) ;

   this.maFctGérantLeDéplacement=function(evt) {
     if(this.boutonPressee==true){
```

```
        this.xFinal=getMousePosition(canvas,evt).x;
        this.yFinal=getMousePosition(canvas,evt).y;
        pencil.onInteractionUpdate(this);
        /*console.log("********Mouvement**********");
        console.log("x initial"+this.xInitial);
        console.log("y initial"+this.yInitial);
        console.log("x final"+this.xFinal);
        console.log("y final"+this.yFinal);*/
    }
  }.bind(this);

  this.maFctGérantLeRelâchement=function(evt) {
      /*console.log("********Relachement**********");
      console.log("x initial"+this.xInitial);
      console.log("y initial"+this.yInitial);
      console.log("x final"+this.xFinal);
      console.log("y final"+this.yFinal);*/
      if(this.boutonPressee==true){
          this.boutonPressee=false;
          pencil.onInteractionEnd(this);
          //Réinitialisation des coordonnées pour les Drag'n drop
          this.xInitial = 0;
          this.yInitial =0;
          this.xFinal = 0;
          this.yFinal =0;
    }
  }.bind(this) ;
```

Q. 10 ***Et Q.11*** *Implémenter ces 3 fonctions dans la classe Pencil. Elles devront créer une forme (en utilisant l'attribut this.currentShape, la forme créée sera un Rectangle ou Ligne en fonction de l'attribut this.currEditingMode), la mettre à jour lorsque l'utilisateur bouge la souris et l'ajouter au dessin lors du relâchement. N'oubliez pas d'appeler la fonction paint de la classe Dessin pour mettre à jour la vue à chaque étape. / Pour modifier le style des formes créées, l'utilisateur doit pouvoir utiliser les widgets fournis (boutons et spinner). Dans la classe Pencil, lier chacun de ces widgets à une fonction modifiant les attributs de la classe Pencil (this.currLineWidth, this.currColour et this.currEditingMode).*

On implémente les 3 fonctions dans la classe Pencil du fichier *Controler.js*.

```
// Implémentation des 3 fonctions onInteractionStart, onInteractionUpdate et
onInteractionEnd
this.onInteractionStart= function(DnD) {
   //Test de la forme
   var butRect = document.getElementById('butRect'),butLine =
document.getElementById('butLine'),

spinnerWidth=document.getElementById('spinnerWidth'),colour=document.getElementB
yId('colour');
   this.currLineWidth= spinnerWidth.value;
   this.currColour=colour.value;
   //Test de la forme sélectionnée
   if(butRect.checked) {
      this.currEditingMode=editingMode.rect;
   }else if(butLine.checked){
      this.currEditingMode=editingMode.line;
   }else if(butCircle.checked){
```

```
      this.currEditingMode=editingMode.circle;
   }else{
      console.log('La sélection de la forme est invalide');
   }
   //Création de la forme sélectionnée
   switch(this.currEditingMode){
   case editingMode.rect: {
      //Création du rectangle
      var largeur = DnD.xFinal-DnD.xInitial;
      var hauteur = DnD.yFinal-DnD.yInitial;
      this.currentShape = new Rectangle(DnD.xInitial, DnD.yInitial, largeur,
hauteur, this.currLineWidth, this.currColour);
      break;
      }
      case editingMode.line: {
         //Création de la ligne
         this.currentShape = new Line(DnD.xInitial, DnD.yInitial, DnD.xFinal,
DnD.yFinal, this.currLineWidth, this.currColour);
         break;
      }
      case editingMode.circle: {
         //Calacul du rayon
         var rayon = Math.sqrt(Math.pow(DnD.xFinal-
DnD.xInitial,2)+Math.pow(DnD.yFinal-DnD.yInitial,2))
         //Création de la ligne
         this.currentShape = new
Circle(DnD.xInitial,DnD.yInitial,rayon,this.currLineWidth,this.currColour);
         break;
      }
      default:
         console.log("la forme sélectionnée n'existe pas.");
   }
}.bind(this) ;

this.onInteractionUpdate= function(DnD) {
   if(butRect.checked) {
      //Création du rectangle
      var largeur = DnD.xFinal-DnD.xInitial;
      var hauteur = DnD.yFinal-DnD.yInitial;
      this.currentShape = new Rectangle(DnD.xInitial, DnD.yInitial, largeur,
hauteur, this.currLineWidth, this.currColour);
   }else if(butLine.checked){
      //Création de la ligne
      this.currentShape = new Line(DnD.xInitial, DnD.yInitial, DnD.xFinal,
DnD.yFinal, this.currLineWidth, this.currColour);
   }else if(butCircle.checked){
      //Calacul du rayon
      var rayon = Math.sqrt(Math.pow(DnD.xFinal-
DnD.xInitial,2)+Math.pow(DnD.yFinal-DnD.yInitial,2))
      //Création de la ligne
      this.currentShape = new
Circle(DnD.xInitial,DnD.yInitial,rayon,this.currLineWidth,this.currColour);
   }else{
      console.log('La sélection de la forme est invalide');
   }
   ctx.clearRect(0, 0, canvas.width, canvas.height);
   drawing.paint(ctx);
   this.currentShape.paint(ctx);
   }.bind(this) ;

this.onInteractionEnd= function(DnD) {
   if(butRect.checked) {
      //Création du rectangle
      var largeur = DnD.xFinal-DnD.xInitial;
```

```
      var hauteur = DnD.yFinal-DnD.yInitial;
      this.currentShape = new Rectangle(DnD.xInitial, DnD.yInitial, largeur,
hauteur, this.currLineWidth, this.currColour);
   }else if(butLine.checked){
      //Création de la ligne
      this.currentShape = new Line(DnD.xInitial, DnD.yInitial, DnD.xFinal,
DnD.yFinal, this.currLineWidth, this.currColour);
   }else if(butCircle.checked){
      //Calacul du rayon
      var rayon = Math.sqrt(Math.pow(DnD.xFinal-
DnD.xInitial,2)+Math.pow(DnD.yFinal-DnD.yInitial,2))
      //Création de la ligne
      this.currentShape = new
Circle(DnD.xInitial,DnD.yInitial,rayon,this.currLineWidth,this.currColour);
   }else{
      console.log('La sélection de la forme est invalide');
   }
   //On reinitialise le canvas
   ctx.clearRect(0, 0, canvas.width, canvas.height);
   //Ajout de la forme à la liste de dessins du canvas
   drawing.addForm(this.currentShape);
   //On recree la liste de dessins du canvas
   drawing.paint(ctx, canvas);
       //Mise à jour de la liste de formes du dessin
       drawing.updateShapeList(this.currentShape);
}.bind(this);
```

1. Liste des modifications

Vous allez compléter l'application avec une liste des modifications apportées au dessin. Cela va permettre de supprimer une forme précédemment dessinée. Pour implémenter cette nouvelle fonctionnalité, il va falloir modifier la page actuelle via Javascript. On utilisera uniquement l'API de base que propose Javascript pour manipuler le Document Object Model (DOM). Le DOM est la représentation arborescente du contenu de la page.

Q. 11 *Ajouter une fonction updateShapeList dans la vue pour afficher la liste des formes. La liste sera contenue dans l'élément dont l'ID est shapeList. Chaque forme aura un affichage différent en fonction de son type (ligne, rectangle). Adapter le contrôleur pour appeler cette nouvelle fonction à chaque fois qu'une nouvelle forme est dessinée.*

```
Drawing.prototype.updateShapeList = function(form){
  //Récupération de l'élément html à alimenter
  var myShapeList = document.getElementById('shapeList');
  //Calcul de l'identifiant
  var i = myShapeList.childNodes.length;
  //Création de l'élément html à insérer
  var newForm = document.createElement('li');
  //Affectation de l'id de l'élément html inséré
  newForm.id    = 'form'+i;
  //Test du type de forme
  if(form instanceof Rectangle){
    //Affectation des attributs
```

```
        newForm.title = "rectangle";
        newForm.innerHTML=i+" rectangle";
    }else if (form instanceof Line){
        //Affectation des attributs
        newForm.title = "ligne";
        newForm.innerHTML=i+" ligne";
    }else if (form instanceof Circle){
    //Affectation des attributs
        newForm.title = "cercle";
        newForm.innerHTML=i+" cercle";
    }
    //Ajout de la forme dans une ligne de la liste
    myShapeList.appendChild(newForm);
};
```

Q. 12 *Modifier updateShapeList pour ajouter le bouton suivant devant chaque élément de la liste.*

```
<button type="button" class="btn btn-default">
        <span class="glyphicon glyphicon-remove-sign"></span>
</button>
```

On ajoute à la fin de la fonction *updateShapeList* le code ci-dessous :

```
    //Ajout du bouton
    var newButton = document.createElement('button');
    newButton.id = i;
    newButton.setAttribute('class','btn btn-default');
    newButton.setAttribute('type', 'button');
    newButton.setAttribute('onClick', 'drawing.removeShapeFromList(id)');
    newForm.appendChild(newButton);
    var newSpan= document.createElement('span')
    newSpan.setAttribute('class', 'glyphicon glyphicon-remove-sign');
    newButton.appendChild(newSpan);
```

Q. 13 Définir le comportement du bouton pour supprimer la forme correspondante dans le dessin.

```
Drawing.prototype.removeShapeFromList = function(index) {
    //Suppression de la forme dans la liste du dessin
    this.removeForm(index);
    //On reinitialise le canvas
```

```
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    //On recree la liste de dessins du canvas
    drawing.paint(ctx, canvas);
    //Mise à jour de la liste de formes du dessin
    //On suppprime toute la liste de formes
    var shapeList = document.getElementById('shapeList');
    while( shapeList.firstChild) {
        // La liste n'est pas une copie, elle sera donc réindexée à chaque appel
        shapeList.removeChild( shapeList.firstChild);
    }
    //on parcourt la liste de forme du dessin
    for(var i= 0, nb=drawing.forms.length;i<nb;i++){
        //On ajoute la forme à la liste
        drawing.updateShapeList(drawing.forms[i]);
    }
}
```

Q. 14 Si vous avez le temps, compléter l'éditeur avec la création d'autres formes (ellipse, polygone, *etc.*), d'autres paramètre (style de la ligne, *etc.*) ou bien des boutons undo/redo.

Nous avons ajouté une forme circulaire à notre projet :

- Ajout d'un bouton
- Ajout d'une classe Circle dans le fichier *model.js*
- Ajout d'une fonction *paint* dans le fichier *view.js*
- Modification du fichier contrôleur pour prise en compte de la nouvelle forme