Ndour Coumba
Totaro Pierre-A.

Encadrants: Olivier Barais - Manuel Leduc
Année 2015/2016

M1 MIAGE

TP JAVASCRIPT ET HTML5 - SIR

1. Interaction : le glisser-déposer (Drag-n-Drop)

Dans le fichier Interaction.js, on créé une classe Dnd contenant 4 attributs initialisés à 0.

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

Puis, on déclare trois fonctions à cette classe correspondant aux trois évènements suivants:

- pression
- déplacement
- relâchement

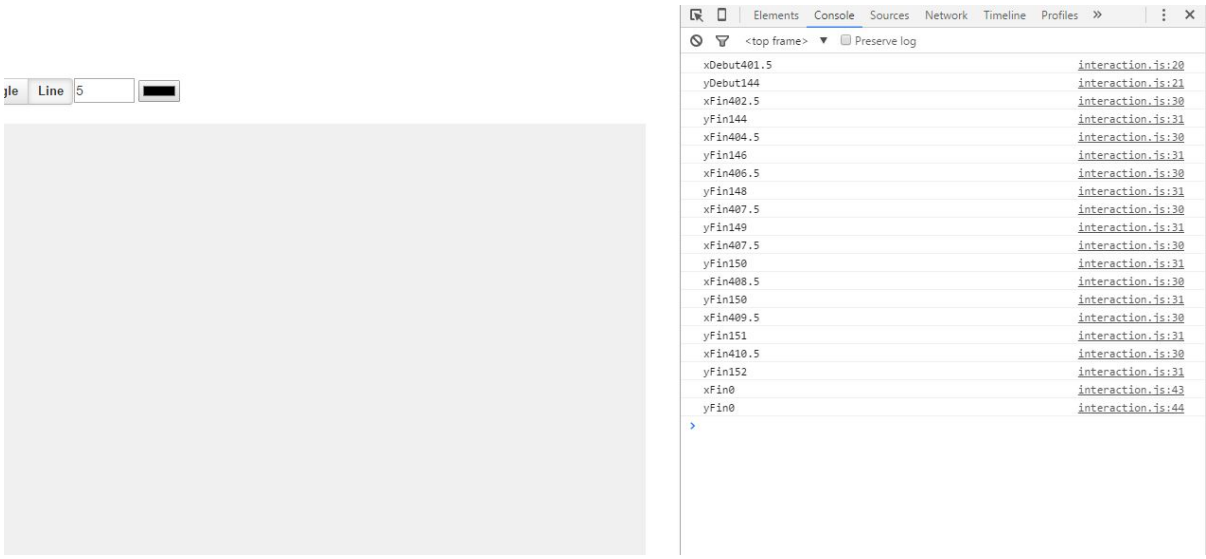
```
this.Pression=function(evt){
    if (this.mouseDown==false){
        this.mouseDown=true;
        this.xDebut=getMousePosition(canvas, evt).x;
        this.yDebut=getMousePosition(canvas, evt).y;
        console.log("xDebut"+this.xDebut);
        console.log("yDebut"+this.yDebut);
    }
}.bind(this);

this.Deplacement=function(evt){
    if (this.mouseDown==true){
        //this.mouseDown=true;
        this.xFin=getMousePosition(canvas, evt).x;
        this.yFin=getMousePosition(canvas, evt).y;
        console.log("xFin"+this.xFin);
        console.log("yFin"+this.yFin);
    }

}.bind(this);

this.Relachement=function(evt){
    if (this.mouseDown==true){
        this.mouseDown=false;
        this.xFin=0;
        this.yFin=0;
        this.yDebut=0;
        this.xDebut=0;
        console.log("xFin"+this.xFin);
        console.log("yFin"+this.yFin);
    }
}.bind(this);
```

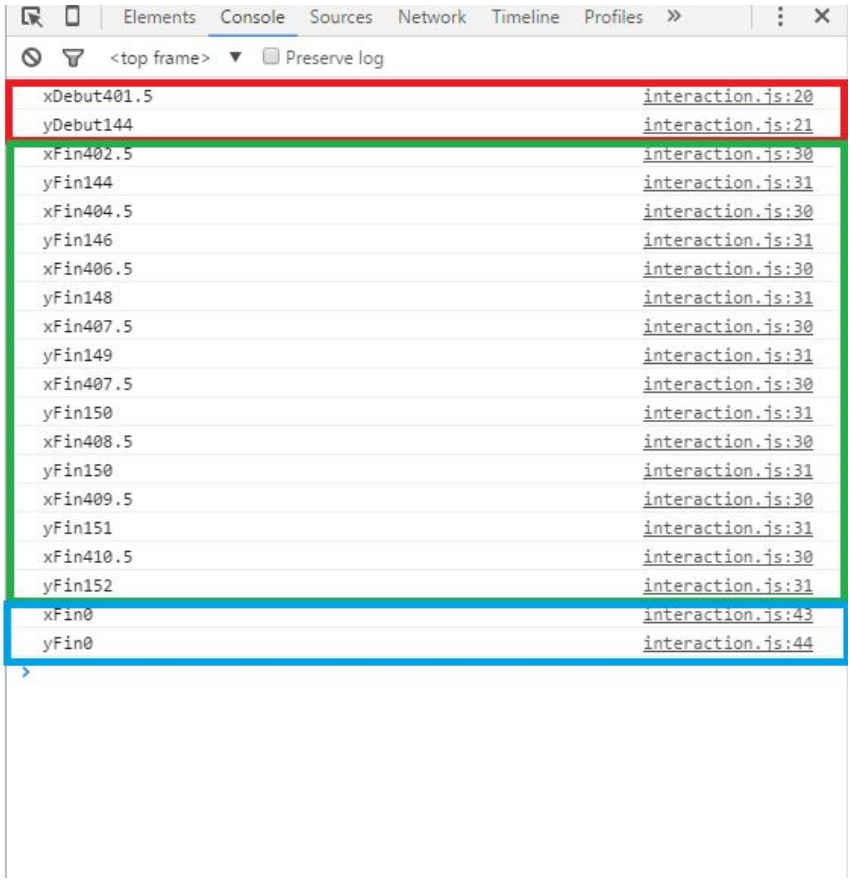
Ces trois fonctions prennent en paramètre un évènement, et on n'oublie pas de lier chaque fonction à la classe DnD grâce à Bind().

Après cela, on enregistre chaque fonction auprès du Canvas. grâce à addEventListener, qui va écouter chaque évènement entre les trois cités, et appeler chaque fonction en conséquence.

```
// Associer les fonctions précédentes aux évènements du canvas.

var Canvas = document.getElementById("myCanvas");
canvas.addEventListener('mousedown', this.Pression, false);
canvas.addEventListener('mousemove', this.Deplacement, false);
canvas.addEventListener('mouseup', this.Relachement, false);
```

On ajoute enfin des Console.log() dans ces fonctions pour pouvoir lister chaque coordonnées du Canvas enregistrées au moment de l'évènement. Pour récupérer les logs, on utilisera la console du navigateur.





Sur cette image, nous voyons la console du navigateur Google Chrome.

Les trois couleurs correspondent aux logs enregistrés lors des trois différents évènements:

- en rouge → Pression sur la souris (clic du bouton)
- en vert → déplacement de la souris (le bouton est toujours maintenu)
- en bleu → relâchement du bouton (on "lâche" la souris), les attributs des coordonnées sont réinitialisés.

2. Le modèle

il faut maintenant s'occuper de la création d'un modèle, dans model.js. Le modèle est un dessin qui est une forme avec une épaisseur et une couleur.

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

Un rectangle est une forme ayant en supplément les coordonnées de son point haut-gauche.

```
function Rectangle(X, Y, largeur, hauteur, epaisseur, couleur){
    Form.call(this, couleur, epaisseur);
    this.X=X;
    this.Y=Y;
    this.largeur=largeur;
    this.hauteur=hauteur;
}
```

Une ligne est une forme ayant en supplément les coordonnées de ses deux points.

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

3. La vue

Il faut maintenant compléter le fichier View.js qui permettra l'affichage des formes dans le canvas par la fonction Paint().

On ajoute donc les fonctions qui ont été proposées dans l'énoncé.

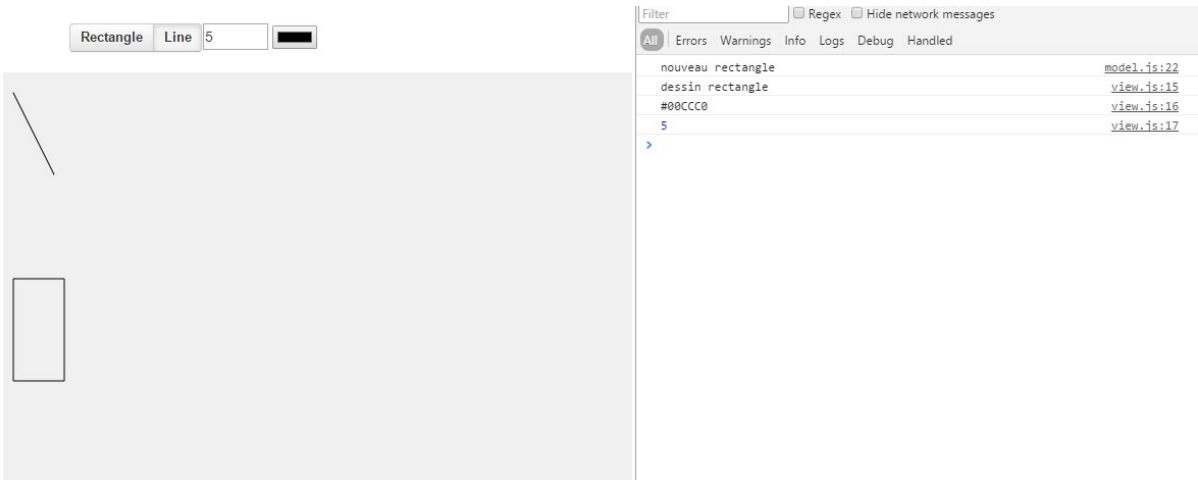
```
Drawing.prototype.paint = function(ctx) {
    console.log("test");
    console.log(this.getForms());
    ctx.fillStyle = '#F0F0F0'; // set canvas' background color
    ctx.fillRect(0, 0, canvas.width, canvas.height);
    this.forms.forEach(function(eltDuTableau) {
        // now fill the canvas
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

Rectangle.prototype.paint = function(ctx) {
    console.log("dessin rectangle");
    console.log(this.epaisseur);
    console.log(this.couleur);

    ctx.lineWidth=this.epaisseur;
    ctx.strokeStyle=this.couleur;

    ctx.rect(this.X, this.Y, this.largeur, this.hauteur);
    ctx.stroke();
};

Line.prototype.paint = function(ctx) {
    ctx.beginPath();

    ctx.lineWidth=this.epaisseur;
    ctx.strokeStyle=this.couleur;


    ctx.moveTo(this.xA, this.yA);
    ctx.lineTo(this.xB, this.yB);
    ctx.stroke();
};
```

Puis, dans le Main.js, il faut décommenter le code proposé à des fins de test d'affichage.

```
// Code temporaire pour tester l'affiche de la vue
var rec = new Rectangle(10, 202, 50, 100, 5, '#00CCC0');
rec.paint(ctx);
var ligne = new Line(10, 20, 50, 100, 5, '#00CCC0');
ligne.paint(ctx);
```

Le résultat attendu est censé représenter un rectangle et une ligne d'épaisseur 5 et de couleur turquoise (d'après le code couleur Html).

Seulement, nous voyons ceci en guise de résultat:



Les formes apparaissent sans problème et aux positions attendues (modifications volontaires pour tester les positions).

Nous nous sommes permis d'ajouter des fonctions Log() dans Rectangle.prototype (voir image view.js) pour vérifier si les couleurs et épaisseurs étaient bien transmises. Le résultat est concluant mais le dessin n'affiche pas ces deux paramètres. A ce stade, nous ne pouvons pas déterminer d'où provient le problème.

4. Le controleur

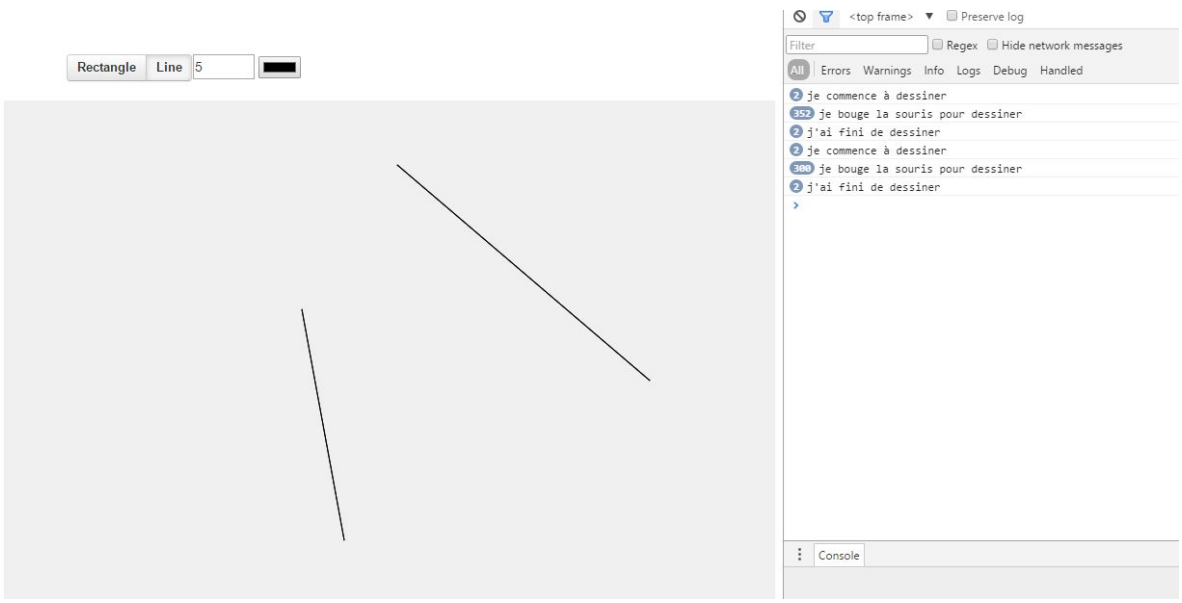
Maintenant, il faut modifier le controller.js qui sera lié à interaction.js.

L'image suivante montre l'action du contrôleur sur le drag-and-drop, par rapport à la création d'une ligne.

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

    // Liez ici les widgets à la classe pour modifier les attributs présents ci-dessus.

    new DnD(canvas, this);

    // Implémentez ici les 3 fonctions onInteractionStart, onInteractionUpdate et onInteractionEnd

    this.onInteractionStart= function(DnD) {
        console.log("je commence à dessiner");
        var butRect = document.getElementById('butRect');
        var butLine=document.getElementById('butLine');
        var spinnerWidth=document.getElementById('spinnerWidth');
        var colour=document.getElementById('colour');


        this.currLineWidth= spinnerWidth.value;
        this.currColour=colour.value;

        this.currentShape = new Line(DnD.xDebut, DnD.yDebut, DnD.xFin, DnD.yFin, this.currLineWidth, this.currColour);
    }.bind(this);

    this.onInteractionUpdate= function(DnD) {
        console.log("je bouge la souris pour dessiner");
        this.currentShape = new Line(DnD.xDebut, DnD.yDebut, DnD.xFin, DnD.yFin, this.currLineWidth, this.currColour);
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        drawing.paint(ctx);
        this.currentShape.paint(ctx);
    }.bind(this);

    this.onInteractionEnd= function(DnD) {
        console.log("j'ai fini de dessiner");
        this.currentShape = new Line(DnD.xDebut, DnD.yDebut, DnD.xFin, DnD.yFin, this.currLineWidth, this.currColour);
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        drawing.addForm(this.currentShape);

        drawing.paint(ctx, canvas);
    }.bind(this);
```

Nous avons inséré des logs pour tester les fonctions en cas de non-affichage.

Le résultat est le suivant:



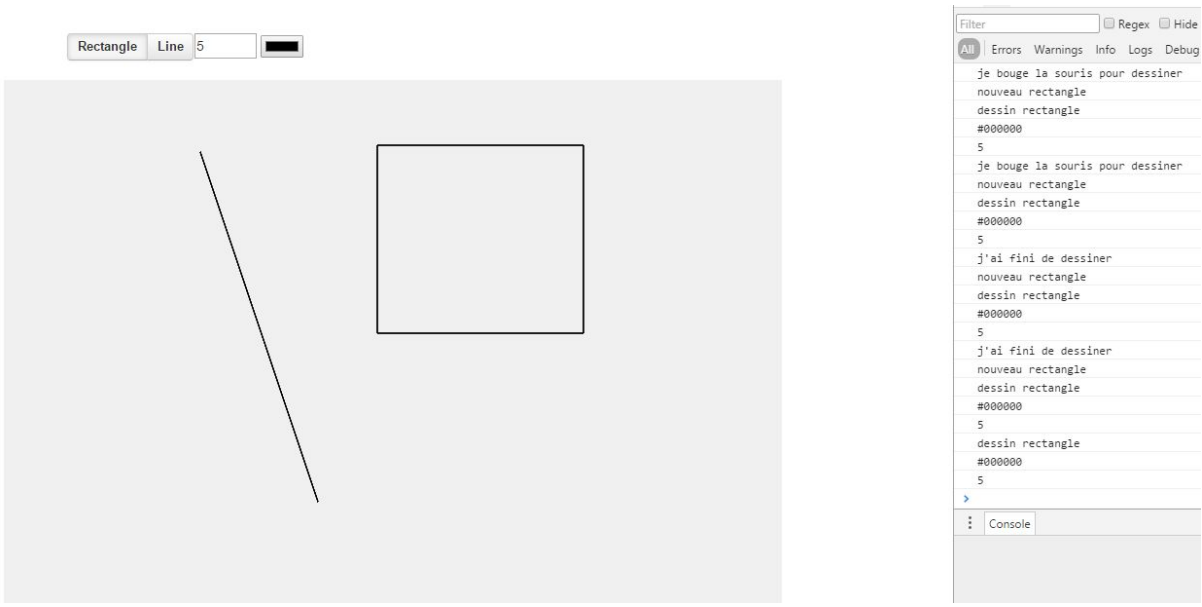
Nous avons dessiné deux lignes, et les logs se sont logiquement affichés.

On rajoute donc les fonctions pour l'affichage d'un rectangle en récupérant les différentes positions du bouton rectangle-ligne et des couleurs ainsi que de l'épaisseur.

Nous rajoutons donc le code du rectangle.

```
this.onInteractionStart= function(DnD) {
    console.log("je commence à dessiner");


    var butRect = document.getElementById('butRect');
    var butLine=document.getElementById('butLine');
    var spinnerWidth=document.getElementById('spinnerWidth');
    var colour=document.getElementById('colour');

    this.currLineWidth= spinnerWidth.value;
    this.currColour=colour.value;

    if(butRect.checked){
        this.currEditingMode=editingMode.rect;
        this.currentShape = new Rectangle(DnD.xDebut, DnD.yDebut, (DnD.xFin-DnD.xDebut), (DnD.yFin-DnD.yDebut), this.currLineWidth, this.currColour);
    }
    else{
        this.currEditingMode=editingMode.line;
        this.currentShape = new Line(DnD.xDebut, DnD.yDebut, DnD.xFin, DnD.yFin, this.currLineWidth, this.currColour);
    }

}.bind(this);

this.onInteractionUpdate= function(DnD) {
    console.log("je bouge la souris pour dessiner");
    switch(this.currEditingMode){
        case editingMode.rect:{
            this.currentShape = new Rectangle(DnD.xDebut, DnD.yDebut, (DnD.xFin-DnD.xDebut), (DnD.yFin-DnD.yDebut), this.currLineWidth, this.currColour);
            break;
        }
        case editingMode.line:{
            this.currentShape = new Line(DnD.xDebut, DnD.yDebut, DnD.xFin, DnD.yFin, this.currLineWidth, this.currColour);
            break;
        }
    }

    ctx.clearRect(0, 0, canvas.width, canvas.height);
    drawing.paint(ctx);
    this.currentShape.paint(ctx);
}.bind(this);

this.onInteractionEnd= function(DnD) {
    console.log("j'ai fini de dessiner");
    switch(this.currEditingMode){
        case editingMode.rect:{
            this.currentShape = new Rectangle(DnD.xDebut, DnD.yDebut, (DnD.xFin-DnD.xDebut), (DnD.yFin-DnD.yDebut), this.currLineWidth, this.currColour);
            break;
        }
        case editingMode.line:{
            this.currentShape = new Line(DnD.xDebut, DnD.yDebut, DnD.xFin, DnD.yFin, this.currLineWidth, this.currColour);
            break;
        }
    }
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    drawing.addForm(this.currentShape);

    drawing.paint(ctx, canvas);
}.bind(this);
```

Et le résultat nous affiche cela:



Nous avons bien dessiné une ligne puis un rectangle. Malheureusement à ce stade, les couleurs et épaisseurs ne correspondent toujours pas au résultat, mais les logs disent le contraire, comme montré dans l'image ci-dessus.

Ce problème non encore résolu reste le plus grand mystère de ce TP, à ce stade.

5. Liste des modifications

oblème non encIl faut maintenant lister les modifications du Canvas en les affichant dans un tableau par exemple.

On créé la fonction UpdateShapeList() qui sera appelée à la fin de la création de la forme. Voici son code:

```
Drawing.prototype.updateShapeList = function(form){

    var tableRef = document.getElementById('myTable').getElementsByTagName('tbody')[0];
    var colour=document.getElementById('colour');
    var spinnerWidth=document.getElementById('spinnerWidth');
    var x = tableRef.childNodes.length;
    var newRow   = tableRef.insertRow(tableRef.rows.length);
    var newCell1  = newRow.insertCell(0);
    var newCell2  = newRow.insertCell(1);
    var newCell3  = newRow.insertCell(2);
    var newCell4  = newRow.insertCell(3);

    if(form instanceof Rectangle){

        var newText1  = document.createTextNode('Rectangle');
    }else if (form instanceof Line){

        var newText1  = document.createTextNode('Ligne');

    }
    var newText3  = document.createTextNode(spinnerWidth.value);
    var newText2  = document.createTextNode(colour.value);


    var newButton = document.createElement('button');
    newButton.id = x;
    newButton.setAttribute('class','btn btn-default');
    newButton.setAttribute('type', 'button');
    newButton.setAttribute('onClick', 'drawing.removeShapeFromList(id)');

    newCell1.appendChild(newText1);
    newCell2.appendChild(newText2);
    newCell3.appendChild(newText3);
    newCell4.appendChild(newButton);
};
```

Cela ajoute quatre cellules pour chaque ligne du tableau bootstrap. Pour nous faire pardonner du problème de couleur et d'épaisseur, nous avons décidé de l'afficher dans ce tableau, ainsi qu'un bouton de suppression, qui permettra d'effacer la figure concernée du canvas.

Voici une partie du résultat:

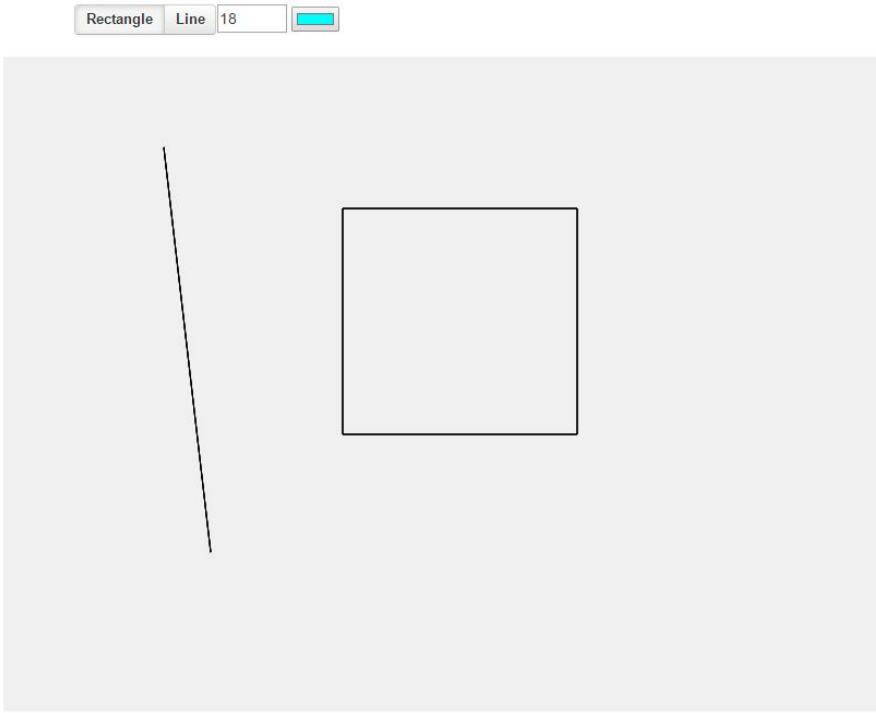


Tableau des formes

Contenant des rectangles ou des lignes

Forme	Couleur	Epaisseur	Supression ?
Ligne	#000000	5	
Ligne	#000000	5	
Rectangle	#00ffff	18	
Rectangle	#00ffff	18	

Il ne reste plus qu'à concevoir la fonction de suppression au clic sur le bouton de chaque ligne du tableau

```
Drawing.prototype.removeShapeFromList = function(index) {

    this.removeForm(index);
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    drawing.paint(ctx, canvas);
    var shapeList = document.getElementById('myTable').getElementsByTagName('tbody')[0];
    while( shapeList.firstChild) {
        shapeList.removeChild( shapeList.firstChild);
    }
    for(var x= 0, nb=drawing.forms.length;x<nb;x++){
        drawing.updateShapeList(drawing.forms[x]);
    }
}
```

Outre le problème permanent des couleurs et de l'épaisseur, l'appel du bouton supprime bien la figure concernée.

Problèmes recontrés lors du TP:

- non respect de la couleur sélectionné mais gardée correctement en mémoire.
- non respect de l'épaisseur mais gardée correctement en mémoire

Bug graphique trouvé (ou erreur de de code ?)

Lors de l'affichage d'un nouveau canvas, si l'on sélectionne d'office un rectangle et qu'on commence à dessiner, nous aurons une multitude de rectangle à chaque mouvement de souris. Mais il n'y aura bien qu'un seul rectangle enregistré au final.