Boite de dessin

Benjamin Bonvarlet
Prune Thiry

13 Mars 2014

Sommaire

Première partie

Introduction au Projet

Chapitre 1

Une boite de dessin

1.1 Réaliser un logiciel permettant de dessiner de maniere interactive.

1.1.1 L'idée de ce projet ?

Le projet de conception d'une boite de dessin nous à tout d'abord été proposer par notre professeur en début d'année, en effet, elle pensait que ce serait une bonne idée pour faire une démonstration des facultés du langage Javascript, mais aussi, que ce serait un challenge de conception et d'harmonisation pour un binôme. Un cahier des charges et des tâches nous on par la suite été proposé, ce qui nous à permis de plus cerner l'idée de conception d'un outil de dessin par navigateur, et de comprendre que c'était un travail de long halène, et que nous devrions consacrer un bon nombre d'heure pour la réalisation de celui-ci. La conception de ce projet à durée plus de 3 mois, et plus de 70H d'analyse et de conception ont été nécessaire à la réalisation de ce qui a été appelé le « Paint Toucon ».

1.1.2 Réaliser un logiciel permettant de dessiner de maniere interactive.

La question de société évoqué par ce projet est « Réaliser un logiciel permettant de dessiner de manière interactive ». Il existe une multitude de logiciel permettant de dessiner, pour ne citer que Paint.NET ou encore Adobe Photoshop, mais il n'existe que peu d'outil de dessin par navigateur, car peut être inconnu, ou difficile à réaliser, ce genre d'outil de dessin « portable » n'est que très peu développer sur Internet. C'est pourquoi la réalisation de ce « logiciel » par navigateur est une idée qui pour moi me semble très appréciable, et me permettra d'utiliser pleinement mes compétences dans le domaine de l'informatique.

1.1.3 Comment gérer une palette d'outils de dessin ?

La gestion d'une palette d'outils de dessin se découpe en plusieurs tâches, premièrement une analyse des outils de base, comme le crayon, le segment, la sauvegarde ou l'effacement, mais aussi l'analyse d'outil plus avancé, comme le cercle, ou encore le rectangle ou encore un historique d'actions. Mais je pense que le plus grand challenge de ce projet est de tout faire fonctionner ensemble, et sans problème, mais aussi que le code source soit intuitif, et qu'il puisse être réutilisable à souhait. C'est pourquoi je répondrais à cette problématique par une réponse par analyse et conception d'un outil de dessin.

Deuxième partie

Conception du projet

Chapitre 2

Analyse du cahier des charges.

2.1 Comment gérer une palette d'outils de dessin ?

2.1.1 Présentation du cahier des charges.

Un cahier des charges nous à été fourni en début de projet, la réalisation de l'outil passe donc tout d'abord par l'analyse de celui-ci, et des tâches qui me seront donné.

- Proposer un canevas sur lequel l'utilisateur dessine.
- Permettre le choix de la forme à dessiner : segment, cercle, rectangle, carré.
- Permettre de choisir un pinceau.
- Permettre de choisir une couleur.
- Permettre de choisir le remplissage ou non des formes.

Et par la suite je me suis vu assigné les tâches suivantes.

- Recherche des informations sur la détection du clic de souris.
- Conception de la méthode de détection du clic de souris.
- Conception des algorithmes de dessin à partir du pinceau.
- Écriture des fonctions de détection de clic de souris.
- Écriture des fonctions de relâchement de bouton de souris.
- Écriture des fonctions de dessin d'une ligne.
- Écriture des fonctions de remplissage de la forme.
- Écriture des fonctions de sauvegarde de l'image.

2.1.2 Analyse de la méthode de récupération des coordonnées du clic de souris.

La première tâche à laquelle je me suis attelé est l'analyse de la méthode de récupération des coordonnées de la souris sur le canvas. J'ai tout d'abord rechercher la méthode de récupération d'un clic de souris, et d'un mouvement de souris. Le clic de souris peut être décompensé en deux événements. Celui de l'appui et celui du relâchement, cette précision me sera utile pour plus tard. Le mouvement de souris est aussi un événements à part entière ce qui me facilitera tout aussi bien la tâche pour plus tard.

```
window.addEventListener("mousedown", function(event){});
window.addEventListener("mouseup", function(event){});
window.addEventListener("mousemove", function(event){});
```

Un premier problème se heurte donc à la conception de l'outil de dessin, comment récupérer les coordonnées sur la page et qu'elles puissent être utilisé sur le canvas. L'unique facilité de ce problème étant la récupération des coordonnées, en effet, lors de l'ajout de l'événement sur la fenêtre, un callback est utilisé, celui-ci à comme paramètre "event", un objet comprenant tout un tas d'information sur l'événement venant d'être effectuer dont deux choses importantes, la position x et la position y pour un clique de souris représenté par deux attributs ClientX et ClientY.

```
event.clientX; // Donne les coordonnees x du clique de souris.
event.clientY; // Donne les coordonnees y du clique de souris.
```

Après quelque recherche sur papier, je trouve ma solution au problème, il suffit de soustraire les coordonnées x et y du canvas au coordonnées x et y de la souris obtenu par l'objet event. Les coordonnées nécessaire à cette différence peuvent être récupérer par la commande suivante.

```
canvas.getBoundingClientRect();
```

On pourra donc établir l'algorithme suivant pour obtenir des coordonnées utilisable dans le canvas.

```
// On ajoute l'evenement.
window.addEventListener("mousedown", function(event)
{
  // On recupere la position du canvas.
  var rect = canvas.getBoundingClientRect();
  // On fait les differences pour obtenir les positions.
  var pos = {
    x: (event.clientX - rect.left);
    y: (event.clientY - rect.top);
  };
});
```

2.1.3 Analyse de la gestion des outils d'une palette de dessin.

L'analyse de conception des outils suivants m'a été assigné.

- Le pinceau.
- La ligne (segment).
- Le remplissage.
- La sauvegarde.

Le Pinceau.

J'ai donc rechercher comment dessiner à la manière d'un pinceau sur un canvas, et après avoir réunis tout les informations, j'ai compris qu'il existait deux méthodes pour faire du dessin.

La première consiste à chaque mouvement d'utiliser la fonction de dessin de rectangle mais après moulte essai j'ai remarqué qu'elle avait un problème majeur, un mouvement rapide va laissé des carrées blancs sur le dessin, ce qui n'est pas vraiment ce que je voudrais faire.

```
1 context.fillRect(pos.x, pos.y, width, height);
```

La seconde consiste à créer un tracé puis à chaque mouvement de souris faire une "ligne" sur celui ci, ce qui ressemble bien plus aux outils de dessin comme Paint ou Adobe Photoshop.

```
1 // On demarre le tracer.
2 context.beginPath();
3 // On met le "curseur" sur la bonne position.
4 context.moveTo(pos.x, pos.y);
5 // On fait une ligne vers la bonne position.
6 context.lineTo(pos.x, pos.y);
7 // On applique la ligne au canvas.
8 context.stroke();
```

La Ligne (segment).

Dans un second temps j'ai travaillé sur l'outil de ligne (segment) et pour lui aussi, il existe à mon avis deux solutions.

La première solution consiste à demander deux cliques à l'utilisateur, le premier sera le premier point de la ligne, le second sera le second point de la ligne. Relativement simple à mettre en place, ce n'est pas celui que j'ai choisi, pour la simple et bonne raison qu'il ne me semblait pas naturel de devoir faire deux cliques pour obtenir une simple ligne...

```
1  var secondClique = false;
2  // Si c'est le premier clique.
3  if(secondClique)
4  {
5    context.beginPath();
6    context.moveTo(pos.x, pos.y);
7    secondClique = true;
8  }
9  // Si le second clique.
10 else
11 {
12   context.lineTo(pos.x, pos.y);
13   context.stroke();
14   secondClique = false;
15 }
```

La seconde solution consiste à sauvegarder l'image après le premier clique, et par la suite, à chaque mouvement de souris, remettre l'image d'après le premier clique et de tracer une ligne vers la où le curseur se trouve de la même manière que lors d'un clique.

```
1  if(premierClique)
2  {
3    // On recupere les donnees de l'image pour les stocker dans une
         variable.
4    dataImage = canvas.toDataURL("image/png");
5    premierClique = false;
6    positionPremierCliqueX = pos.x;
7    positionPremierCliqueY = pos.y;
8  }
9  // On creer une image puis on lui integre les donnees recuperer au
       premier clique.
10 var img = new Image();
11 img.src = dataImage;
12 // Quand l'image s'est charger.
13 img.addEventListener("load", function()
14 {
15   // On dessine l'image sur le canvas.
16   context.drawImage(img,0,0);
17   // On commence le tracer.
18   context.beginPath();
19   // On place le premier point aux positions du premier clique.
20   context.moveTo(positionPremierCliqueX, positionPremierCliqueY);
21   // On dessine le premier point.
22   context.stroke();
23   // On place le second point aux positions du mouvement.
24   context.lineTo(positionMouvementX, positionMouvementY);
25   // On dessine la ligne.
26   context.stroke();
27 });
```

Le Remplissage des formes.

Pour dessiner les formes prédéfinis en canvas, il existe deux fonctions.

```
context.stroke();
context.fill();
```

La première fonction va permettre de dessiner la forme sans la remplir et la seconde va dessiner la forme en la remplissant. Il me suffira donc de demander avec un bouton à l'utilisateur de choisir ou non le remplissage, puis de faire une condition sur ce choix, pour savoir laquelle des deux fonctions utiliser.

La Sauvegarde.

Après une simple recherche sur Internet, on remarque qu'il existe une simple solution pour enregistrer son image, il suffit d'ouvrir un nouvel onglet avec comme donnée les données de l'image. L'Utilisateur devra ensuite faire un clique droit et sauvegarder l'image par lui même.

```
window.open(canvas.toDataURL("image/png"));
```

Chapitre 3

Réalisation du cahier des charges.

3.1 Un travail aux multiples facettes.

3.1.1 Un travail en binôme, des outils de partages, l'harmonisation des conceptions.

Le travail en binôme n'est pas quelque chose de facile sur un projet de programmation, en effet, il faut pouvoir partagé les créations en temps réels, et harmoniser les conceptions des différentes personnes travaillant sur celui-ci. Pour le partage des créations nous avons décidé d'utiliser le logiciel Dropbox, permettant de stocker des données en temps réel sur un Cloud. Pour l'harmonisation des conceptions, le projet est programmé tel qu'il est modulable à souhait, principe fondateur de la programmation orienté objet.

3.1.2 Conception de la méthode de récupération des coordonnes du clic de souris.

Après avoir analysé la conception de la méthode de récupération des coordonnées du clic de souris, il reste maintenant à programmer cette fonction. Mon idée est donc de créer trois événements sur la page qui vont correspondre à trois choses, la première un clique de souris, la seconde à un relachement de clique de souris et la troisième à un mouvement de souris. Puis les événements de clique et de mouvement utiliseront la fonction onEventClick permettant de dessiner.

```
1  // Ajout Evenement au click dans le canvas.
2  this.setEventClick = function()
3  {
4    var canvas = this.canvas;
5    // Ecriture des fonctions de detection de clic de souris.
6    window.addEventListener("mousedown", function(event)
7    {
8      Toucon.onEventClick(event);
9      Toucon.clicked = true;
10   });
11   // Ecriture des fonctions de relachement de bouton de souris.
12   window.addEventListener("mouseup", function(event)
13   {
14     Toucon.clicked = false;
15   });
16   // Ecriture des fonctions de mouvement de souris.
17   window.addEventListener("mousemove", function(event)
18   {
19     if(Toucon.clicked)
20     {
21       Toucon.moving = true;
22       Toucon.onEventClick(event);
23     }
24   });
25 }
```

3.1.3 Conception de la gestion des outils d'une palette de dessin.

Pour la conception de la gestion des outils de dessin, il faudra tout d'abord créer des boutons et des événements pour chacun de ses boutons afin de savoir quel outil est utilisé par l'utilisateur.

```
1  // Ajout Evenement au click sur les boutons.
2  this.setEventButton = function()
3  {
4    document.getElementById("undo").addEventListener("mousedown",
         function()
5    {
6      Toucon.beforeAction();
7    });
8    document.getElementById("redo").addEventListener("mousedown",
         function()
9    {
10     Toucon.afterAction();
11   });
12   document.getElementById("brush").addEventListener("mousedown",
         function()
13   {
14     Toucon.buttonID = 0;
15   });
16   document.getElementById("ligne").addEventListener("mousedown",
         function()
17   {
18     Toucon.buttonID = 1;
19   });
20   document.getElementById("rectangle").addEventListener("mousedown"
         , function()
21   {
22     Toucon.buttonID = 2;
23   });
24   document.getElementById("cercle").addEventListener("mousedown",
         function()
25   {
26     Toucon.buttonID = 3;
27   });
28   document.getElementById("pipette").addEventListener("mousedown",
         function()
29   {
30     Toucon.buttonID = 4;
31   });
32   document.getElementById("gomme").addEventListener("mousedown",
         function()
33   {
34     Toucon.buttonID = 5;
35   });
36   document.getElementById("remplissage").addEventListener("
         mousedown", function()
37   {
38     if(Toucon.fill)
39     {
40       Toucon.fill = false;
41       this.className = "";
```

```
42      }
43      else
44      {
45        Toucon.fill = true;
46        this.className = "actif";
47      }
48    });
49    document.getElementById("triangle").addEventListener("mousedown",
          function()
50    {
51      Toucon.buttonID = 6;
52    });
53    // Evenement pour le slider de taille.
54    document.getElementById("largeur_pinceau").addEventListener("
          keydown", this.changeSize);
55    document.getElementById("largeur_pinceau").addEventListener("
          keyup", this.changeSize);
56    document.getElementById("largeur_pinceau").addEventListener("
          mousemove", this.changeSize);
57    document.getElementById("largeur_pinceau").addEventListener("
          mouseup", this.changeSize);
58    // Evenement pour le slider de rouge.
59    document.getElementById("red").addEventListener("keydown", this.
          changeRed);
60    document.getElementById("red").addEventListener("keyup", this.
          changeRed);
61    document.getElementById("red").addEventListener("mousemove", this
          .changeRed);
62    document.getElementById("red").addEventListener("mouseup", this.
          changeRed);
63    // Evenement pour le slider de vert.
64    document.getElementById("green").addEventListener("keydown", this
          .changeGreen);
65    document.getElementById("green").addEventListener("keyup", this.
          changeGreen);
66    document.getElementById("green").addEventListener("mousemove",
          this.changeGreen);
67    document.getElementById("green").addEventListener("mouseup", this
          .changeGreen);
68    // Evenement pour le slider de bleu.
69    document.getElementById("blue").addEventListener("keydown", this.
          changeBlue);
70    document.getElementById("blue").addEventListener("keyup", this.
          changeBlue);
71    document.getElementById("blue").addEventListener("mousemove",
          this.changeBlue);
72    document.getElementById("blue").addEventListener("mouseup", this.
          changeBlue);
73    // Evenement pour le bouton de sauvegarde.
74    document.getElementById("sauvegarde").addEventListener("click",
          this.saveImage);
75    // Evenement pour le bouton de reset.
76    document.getElementById("reset").addEventListener("click", this.
          toolReset);
77  }
```

Pour chaque bouton il existe un événements qui sera appelé lors du clic sur celui-ci. La fonction ci-dessus présente le module complet, avec les outils de changements, de la largeur du pinceau ainsi que les boutons de chaque outils. L'action majeur pour les outils de dessin est le changement du Toucon.buttonID qui est l'identifiant pour chaque outil, celui-ci sera utilisé dans la fonction onEventClick pour pouvoir dynamiquement dessiner avec l'outil sélectionné.

```
1  this.onEventClick = function(event)
2  {
3    var canvas = this.canvas;
4    var context = this.context;
5    context.lineJoin = this.lineJoin; // Ecriture des fonctions de
         dessin des differentes formes.
6    context.lineCap = this.lineCap; // Ecriture des fonctions de
         dessin des differentes formes.
7    context.strokeStyle = this.color.rvb; // Couleur.
8    context.lineWidth = this.size; // Taille.
9    var rect = canvas.getBoundingClientRect();
10   // Calcul des positions.
11   var pos = {
12     x : (event.clientX - rect.left) - 10, // Difference de 10 pour
           les marges.
13     y : (event.clientY - rect.top) - 10 // Difference de 10 pour
           les marges.
14   };
15   switch(this.buttonID)
16   {
17     // Brush.
18     case 0:
19       this.toolBrush(pos);
20       break;
21     // Line.
22     case 1:
23       this.toolLine(pos);
24       break;
25     // Rectangle.
26     case 2:
27       this.toolRect(pos);
28       break;
29     // Circle.
30     case 3:
31       this.toolCircle(pos);
32       break;
33     // Pipette.
34     case 4:
35       this.toolPipette(pos);
36       break;
37     //gomme.
38     case 5:
39       this.toolEraze(pos);
40       break;
41     //triangle.
42     case 6:
43       this.toolTriangle(pos);
44       break;
45   }
46 }
```

On retrouve bien les algorithmes présenté lors de l'analyse du cahier des charges. Après avoir programmer ces deux fonctions, j'ai programmé les fonctions pour les outils qui m'étaient assignés qui sont pour rappel.

- Le pinceau.
- La ligne (segment).
- Le remplissage.
- La sauvegarde.

Le Pinceau.

Pour le pinceau, comme énoncé dans l'analyse de la conception de celui-ci, un fonctionnement dynamique et qui ne laisse pas de carré non utilisé est de rigueur. J'ai donc programmé la fonction toolBrush qui permet de dessiner tel le pinceau de Paint ou de Adobe Photoshop.

```
1  // Fonction de pinceau.
2  this.toolBrush = function(pos)
3  {
4    var context = this.context;
5    if(!this.brushing)
6    {
7      context.beginPath();
8      context.moveTo(pos.x, pos.y);
9      context.lineTo(pos.x, pos.y);
10     context.stroke();
11     this.brushing = true;
12   }
13   else
14   {
15     context.lineTo(pos.x, pos.y);
16     context.stroke();
17   }
18 }
```

La Ligne (segment).

Pour l'outil de dessin de ligne, voyant que la méthode des deux points n'est pas ce que j'appelle naturel, et comme précisé dans l'analyse de cette outil, j'utiliserais donc une manière beaucoup plus dynamique pour celui-ci, sauvegarder l'image et la remettre à chaque mouvement, comme pour des logiciels comme Paint ou Adobe Photoshop.

```
1  // Fonction de ligne.
2  this.toolLine = function(pos)
3  {
4    var canvas = this.canvas;
5    var context = this.context;
6    if(this.lineHistory.bool)
7    {
8      this.lineHistory.data = canvas.toDataURL("image/png");
9      this.lineHistory.bool = false;
10     this.lineHistory.x = pos.x;
11     this.lineHistory.y = pos.y;
12   }
13   var img = new Image();
14   img.src = this.lineHistory.data;
15   img.addEventListener("load", function()
16   {
17     context.drawImage(img,0,0);
18     context.beginPath();
19     context.moveTo(Toucon.lineHistory.x, Toucon.lineHistory.y);
20     context.stroke();
21     context.lineTo(pos.x, pos.y);
22     context.stroke();
23   });
24 }
```

La Remplissage.

Pour chaque fonction créer par mon binôme je devais donc faire comme précisé dans l'analyse une condition pour pouvoir obtenir soit un remplissage soit pas de remplissage par rapport au choix utilisateur. En utilisant donc un bouton sur la page, relié à un événements qui va changer la variable booléenne Toucon.fill et une condition dans chaque fonction de forme, j'obtiendrais donc ce que je voulais.

```
1 if(Toucon.fill)
2 {
3   context.fill();
4 }
5 else
6 {
7   context.stroke();
8 }
```

La Sauvegarde.

Comme énoncé dans l'analyse de cet outil, j'ouvre donc une nouvelle fenêtre avec comme donnée les données de l'image du canvas à l'action d'un événement attaché à un bouton de sauvegarde sur la fenêtre de dessin.

```
1 // Fonction de sauvegarde.
2 this.saveImage = function()
3 {
4   var canvas = Toucon.canvas;
5   window.open(canvas.toDataURL("image/png"));
6 }
```

Chapitre 4

La mise en page de la fenêtre de canvas.

4.1 Les boutons pour le choix utilisateur.

4.1.1 Analyse et conception de ces boutons.

Pour les choix utilisateurs, mon binôme et moi avons eux l'idée d'utiliser de simples boutons, sur lequel l'utilisateur va venir cliquer pour obtenir l'outil qu'il souhaite. Un événement sera rattaché à chaque bouton comme énoncé dans la partie Conception de la gestion des outils d'une palette de dessin du Chapitre III de la Partie 2.

4.2 Les sliders pour les couleurs et la largeur du pinceau.

4.2.1 Analyse et conception de ces sliders.

Pour obtenir le changement de couleur et de largeur du pinceau, nous avons optes pour une gestion avec des sliders et un verre de montre, celui-ci changera de taille et de couleur dynamiquement quand l'utilisateur utilisera ces sliders. Comme énoncé dans la partie Conception de la gestion des outils d'une palette de dessin du Chapitre III de la Partie 2, des événements seront attachés à ces sliders, et utiliseront des fonctions pour pouvoir obtenir de l'utilisateur un choix de couleur ou de largeur. Les fonctions suivantes permettront donc grâce aux événements un choix utilisateur dynamique et fonctionnel.

```
1  // Fonction de changement de largeur du pinceau.
2  this.changeSize = function()
3  {
4    document.getElementById("output").innerHTML = document.
         getElementById("largeur_pinceau").value;
5    document.getElementById("montre").style.width = document.
         getElementById("largeur_pinceau").value + "px";
6    document.getElementById("montre").style.height = document.
         getElementById("largeur_pinceau").value + "px";
7    Toucon.size = document.getElementById("largeur_pinceau").value;
8  }
9  // Trois fonctions de changement de composante de couleur.
10 this.changeRed = function()
11 {
12   var composante = (Toucon.forceX2(parseInt(document.getElementById
         ("red").value).toString(16))).toUpperCase();
13   document.getElementById("oRed").innerHTML = composante;
14   Toucon.color.r = composante;
15   Toucon.color.rvb = "#" + Toucon.color.r + Toucon.color.v + Toucon
         .color.b;
16   document.getElementById("montre").style.background = Toucon.color
         .rvb;
17 }
18 this.changeGreen = function()
19 {
20   var composante = (Toucon.forceX2(parseInt(document.getElementById
         ("green").value).toString(16))).toUpperCase();
21   document.getElementById("oGreen").innerHTML = composante
22   Toucon.color.v = composante
23   Toucon.color.rvb = "#" + Toucon.color.r + Toucon.color.v + Toucon
         .color.b;
24   document.getElementById("montre").style.background = Toucon.color
         .rvb;
25 }
26 this.changeBlue = function()
27 {
28   var composante = (Toucon.forceX2(parseInt(document.getElementById
         ("blue").value).toString(16))).toUpperCase();
29   document.getElementById("oBlue").innerHTML = composante;
30   Toucon.color.b = composante;
31   Toucon.color.rvb = "#" + Toucon.color.r + Toucon.color.v + Toucon
         .color.b;
32   document.getElementById("montre").style.background = Toucon.color
         .rvb;
33 }
34 // Forcer a avoir 2 nombres pour l'hexa.
35 this.forceX2 = function(number)
36 {
37   if(parseInt(number, 16) < 16)
38   {
39     return "0" + number;
40   }
41   else
42   {
43     return number;
44   }
45 }
```

Chapitre 5

Les tests des fonctionnalités réalisé par mon binôme.

5.1 Réalisation des tests.

5.1.1 Les fonctions de dessins de formes.

Mon binôme avait à réaliser les fonctions de dessins de forme de rectangle et de cercle. Pour réaliser les tests de ces fonctions, j'ai donc utilisé ces outils de différentes façons. Les tests de la fonction de rectangle étant concluant je ne vais pas m'attarder dessus, quand à la fonction de cercle, il y eu dans les premières versions de celle-ci un bug assez important, en effet, le rayon du cercle ne se dessiné que pour un mouvement sur l'axe x et non pas sur l'axe y, ce qui était assez dérangeant, après l'avoir signalé à mon binôme, celle-ci à rechercher d'où venait le dysfonctionnement et reprogrammer la fonction de calcul du rayon. Dans les dernières versions ce bug n'est plus d'actualité.

5.1.2 Les fonctions d'effacement de dessin.

La fonction d'effacement étant assez simple à réaliser, il n'y a pas eu tellement de test sur celle-ci, puisqu'il suffit de faire un rectangle blanc démarrant des coordonnées (0 ;0) et de taille du canvas. Une fonction assez légère et simple, mais qui permet d'optimiser la tâche utilisateur de façon très significative, c'est à dire que l'utilisateur n'a pas a actualisé la page à chaque fois qu'il veut refaire un dessin.

5.2 Amélioration possible et propose a mon binôme.

5.2.1 Une fonction de gomme et une fonction de triangle.

Comme amélioration possible, j'ai proposé à mon binôme la création d'un outil de gomme, et la création d'un outil de forme triangulaire. Celle-ci à relevé le défi avec brio, et ces fonctions sont donc intégré dans la dernière version du projet.

5.2.2 Améliorations possible pour une palette de dessin

Après avoir fini notre projet, mon binôme et moi même avons travailler sur une "suite" du cahier des charges que nous avions accomplis, cette suite ce compose de nouveaux outils indispensable je pense pour un outil de dessin. Je ne détaillerais pas volontairement ces outils dans ce dossier, car je pense qu'ils sont Hors-Sujet par rapport à ce que Mme Mouton nous avait proposé à la base. Ces outils sont néanmoins intégrés à la dernière version du projet. La liste des outils intégrés et non demandé dans le cahier des charges sont :

- L'Historique des actions.
- La Forme Triangulaire.
- La Gomme.
- La Pipette.

Chapitre 6

Avis sur ce projet.

6.1 Un travail en binôme et mon premier projet d'envergure.

6.1.1 Mon avis sur tout ça.

Un outil de dessin est donc un outil complexe qui nécessite beaucoup de temps et beaucoup de réflexion, ce n'est pas quelque chose qu'on entreprend seul. J'étais très content de travailler sur ce projet et j'ai apprécié le faire avec mon binôme, c'était une très bonne idée de projet et j'en remercie Mme Mouton. Mon année d'informatique Science du Numérique fut donc une très bonne expérience pour moi, et j'espère m'aidera pour mes futures études d'informatique.

Bibliographie

[1] Dropbox. 2014. url : http://www.dropbox.com/.

[2] Google. 2014. url : http://google.fr/.

[3] Icons. 2014. url : http://www.fatcow.com/.

[4] OpenClassroom. 2014. url : http://fr.openclassrooms.com/.

[5] Template. 2014. url : http://www.html5webtemplates.co.uk/.

[6] W3Schools. 2014. url : http://www.w3schools.com/.

[7] Wikipédia. 2014. url : http://fr.wikipedia.org/.