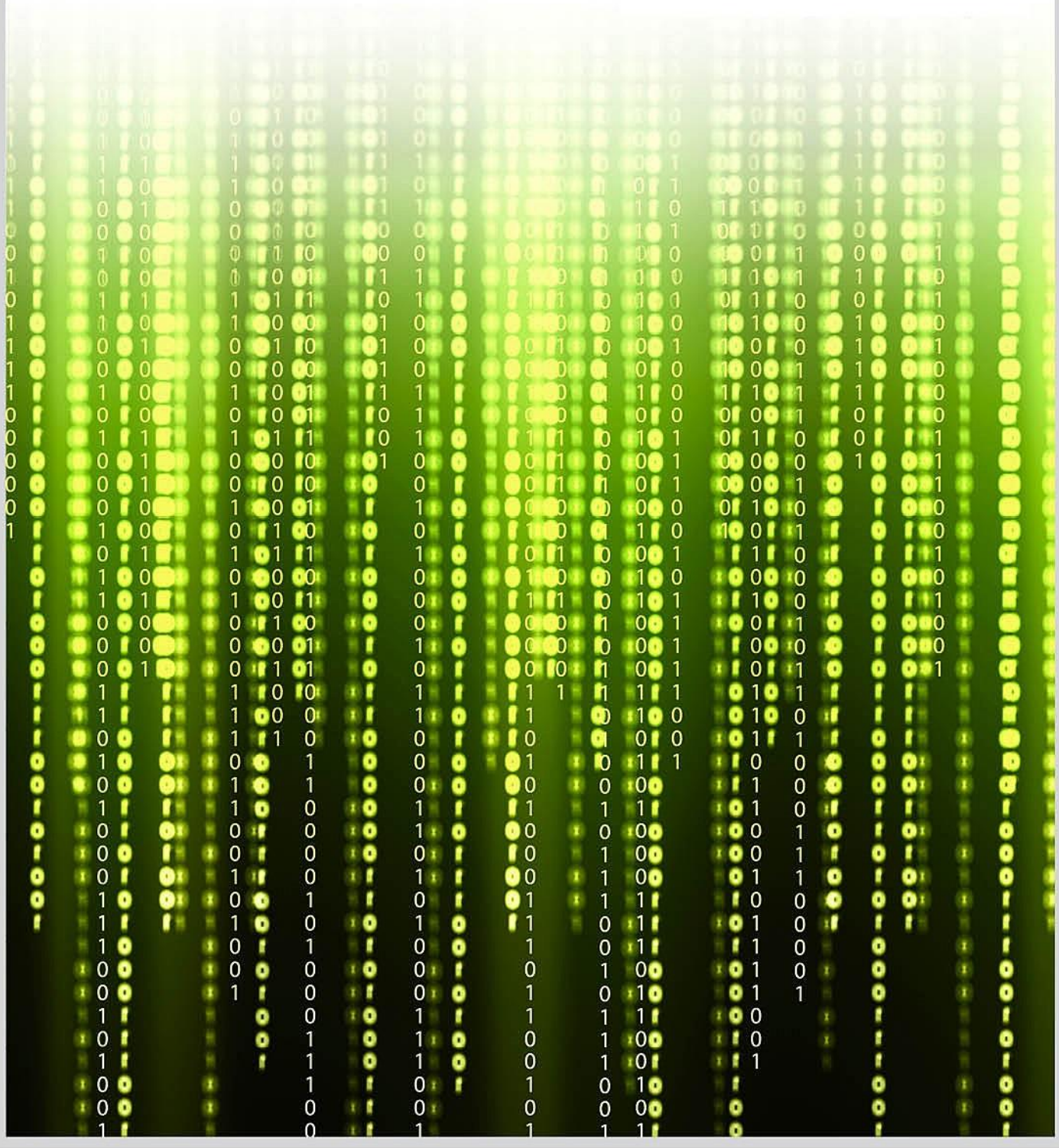
Projet ISN – Année 2015/2016

Stéganographie

Codage & Décodage

Arthur Guyader & Ludovic Richoux Terminale S2 M. Merlet & M. Marchet



Sommaire

Cahier des charges

Problématique : Dissimuler un message secret dans une image numérique sans qu'il soit visible et décoder le ensuite.

Objectifs principaux :

- Proposer un choix de **qualité de stéganographie** (nombre de bits à utiliser)
- Définir une **zone de texte** pour **entrer** le **message à coder**
- **Afficher l'image** pour valider son traitement
- **Décoder l'image** en connaissant le nombre de bit utilisés.

Objectifs facultatifs :

- Coder et décoder en partant des **bits de poids fort ou des bits de poids faible**.
- **Arrêter de déchiffrer** le message, une fois qu'il a été détecté dans son intégralité.

Répartition des tâches

Ludovic s'est occupé de mettre en place toute l'interface graphique à partir du code déjà existant pour que celle-ci s'adapte au mieux à notre projet et réponde à nos besoins et avait aussi débuté le décodage.

Pendant ce temps, je me suis occupé du développement de la partie codage, consistant à placer le code dans une image.

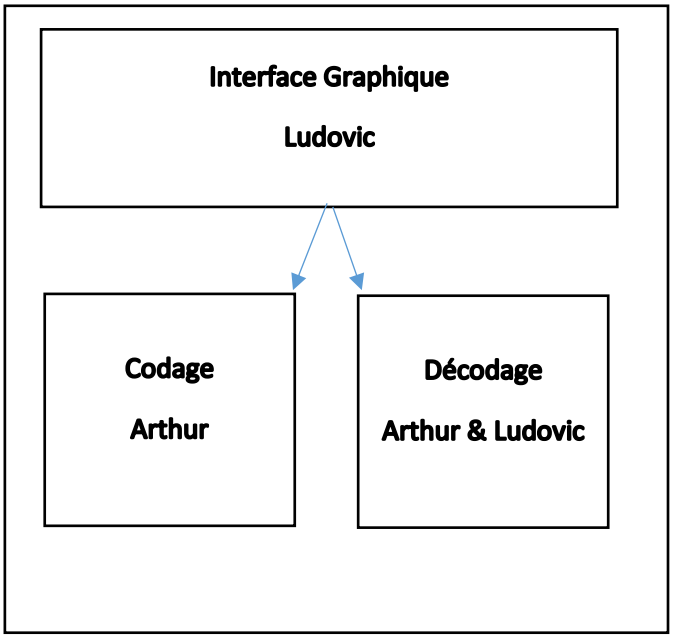
Cependant, nos parties codage et décodage s'allongeaient et n'aboutissaient pas.

Sur les conseils de M. Merlet, j'ai donc décidé de recommencer le codage et le décodage. C'est ainsi que j'ai programmé deux premiers codes qui débutaient par les bits de poids fort.

Malgré tout, ce code n'était pas très dur à modifier pour pouvoir commencer des bits de poids faibles, alors qu'il permettait de multiplier les possibilités de codage par deux. En effet, il suivait la même structure algorithmique, juste quelques valeurs devaient être modifiées.

J'ai par la suite ajouté quelques lignes de code pour permettre d'arrêter le déchiffrement de l'image, une fois que le message est passé.

Enfin nous avons mis nos deux parties en commun pour réaliser l'application finale.



Réalisation – Développement

Pour mener à bien notre projet, nous avons travaillé sur Eclipse un environnement de développement en Java.

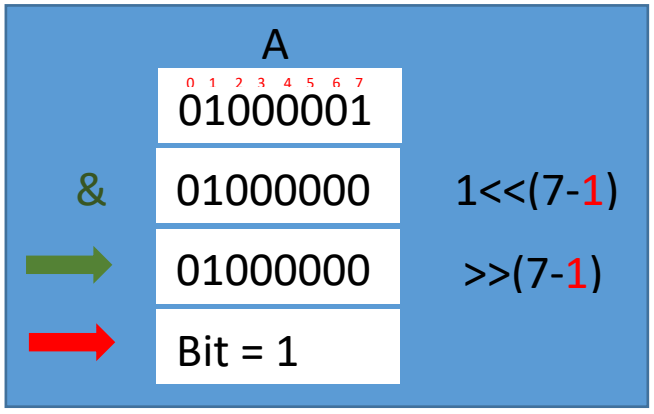
Codage

Après un départ très laborieux (Voir Figure 1 dans l'annexe), qui ne cessait d'allonger le programme, j'ai décidé de repartir à 0 sur la partie codage, sur les conseils de M. Merlet.

En effet, j'ai pu apprendre qu'en programmation un très long code n'est pas meilleur mais souvent moins bon, car il entraîne une perte de temps et une énorme difficulté à débugger.

Pour commencer le codage, on copie toute l'ancienne image dans la nouvelle pour éviter tout problème. Ensuite il était nécessaire de récupérer le code sous forme d'une suite de bit. Pour ceci il fallait d'abord trouver un moyen de parcourir le message secret caractère par caractère et bit par bit. Pour répondre à ce problème, deux boucles for restaient la meilleure solution. La première pour parcourir chaque caractère la seconde pour parcourir chaque bit du caractère.

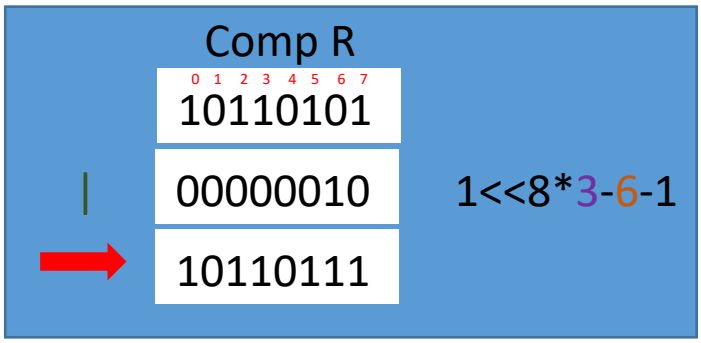
La fonction .charAt (), quant à elle, permet d'extraire le caractère approprié de la chaîne de caractères contenant le message. Ce caractère est ensuite traité pour isoler chaque bit. (Voir Figure 2).



Ce bit doit ensuite être placé dans la bonne composante et sur le bon bit. Deux possibilités peuvent être envisagées :

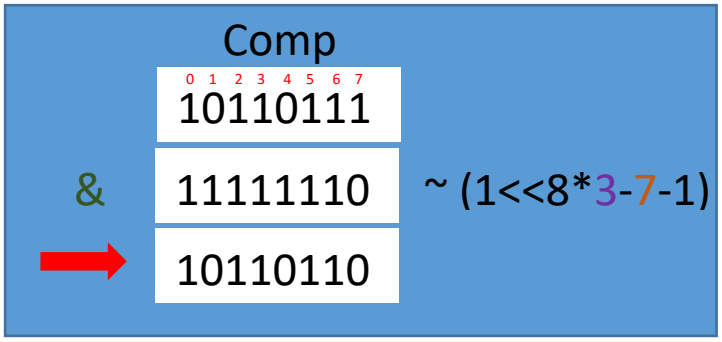
- Le bit à placer est égal 1 :

Dans l'exemple, on cherchera à placer ce bit sur le 7ème bit de la composante rouge. (Voir Figure 3)



- Le bit à placer est égal 0 :

Dans l'exemple, on cherchera à placer ce bit sur le 8ème bit de la composante rouge. (Voir Figure 4)



Une fois que l'on savait comment traiter les différents bits, il ne restait plus qu'à savoir où le faire et cela de manière automatique.

R : 3 **V : 2** **B : 1**

0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7

10110001 11000111 10101001

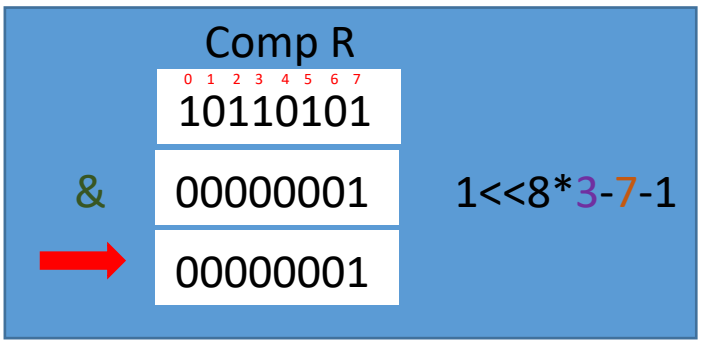
On commence donc pour l'exemple par la 3ème composante, et le 7ème bit. Quand la première opération est finie, on incrémente une variable nous indiquant la place du bit. On traite donc le 8ème bit, cette fois on ne peut pas incrémenter la variable nous indiquant la place du bit car nous sommes arrivés au dernier. On désincrémente donc la variable nous indiquant la place de la composante. On continue cette suite d'opération jusqu'au dernier bit de la composante bleu. A ce moment, on change le pixel à traiter. Et on répète toutes ces opérations jusqu'à que tous les bits du message soit placés. (Voir Figure 5).

Enfin il faut s'assurer de bien copier les informations du dernier pixel codé, à sa place, car si le code s'arrête avant la dernière composante, le pixel n'est pas copié. (Voir Figure 6)

Décodage

Cette fois à l'inverse du codage, on cherche d'abord à parcourir toute l'image, pixel par pixel, composante par composante et tous les bits qui nous intéressent. Pour ceci l'utilisation de quatre boucles for est le moyen le plus utile.

Ensuite, on applique un masque sur le pixel, en connaissant l'emplacement du bit à récupérer:



Une fois ce traitement accompli, il ne reste plus qu'à tester la valeur de la variable obtenue. Si elle est différente de 0, alors le bit que nous voulions était égal à 1. Sinon il était égal à 0. (Voir Figure 7)

Une fois le bit récupéré, on le place dans une variable caractère qui au bout de 8 bits, s'ajoute dans une variable texte puis se remet à 0, en attendant de nouveaux bits. (Voir Figure 8).

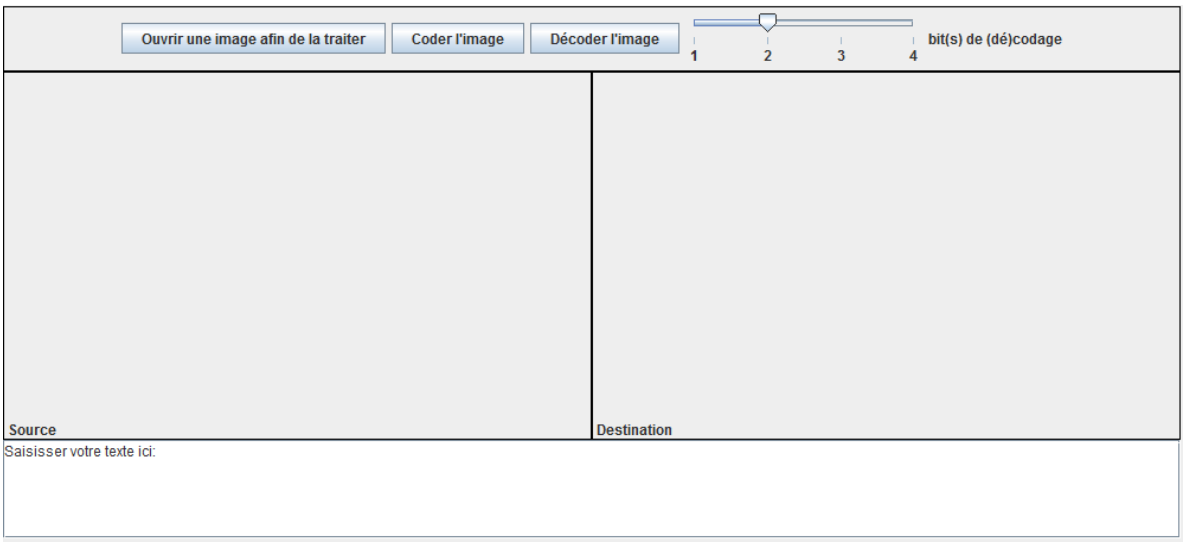
Cependant, si on arrête le code à ce moment, il traite toute l'image et ne s'arrête qu'une fois qu'il a fini ce qui peut prendre un certain temps. Pour réduire ce temps, nous avions imaginé dès le début de placer à la fin de notre code un message permettant d'arrêter le décodage.

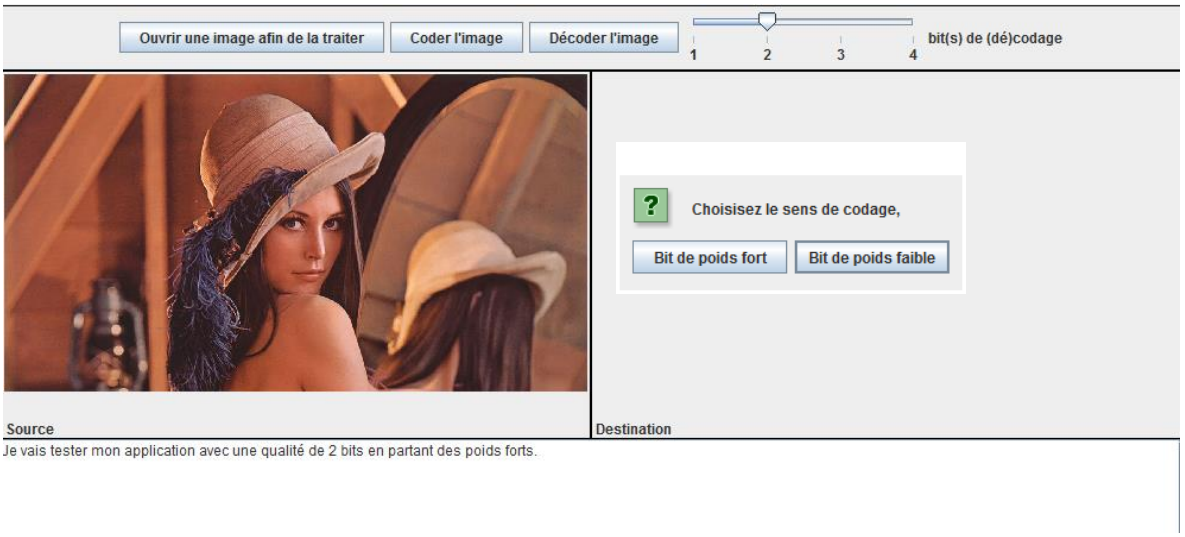
Pour l'expliquer brièvement, on place un message à la fin de notre code qui ne sera sans doute jamais écrit par l'utilisateur : « !3wZ ». Puis, on crée une variable qui voit passer chaque caractère. Dès qu'elle voit le message, elle arrête les boucles for. (Voir Figure 9)

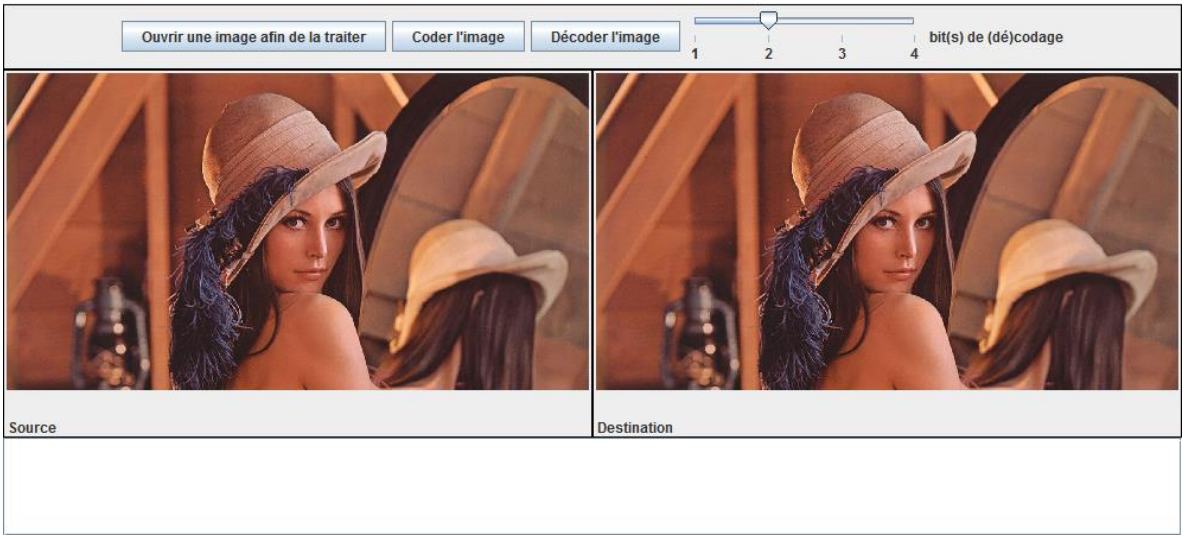
Cependant, cette technique ajoute le message d'arrêt à la fin du texte à rendre. Pour l'enlever, il suffit de récupérer la taille du texte avec le message d'arrêt et de le parcourir en le mettant dans une nouvelle variable mais en s'arrêtant avant le message non désiré. (Voir Figure 10)

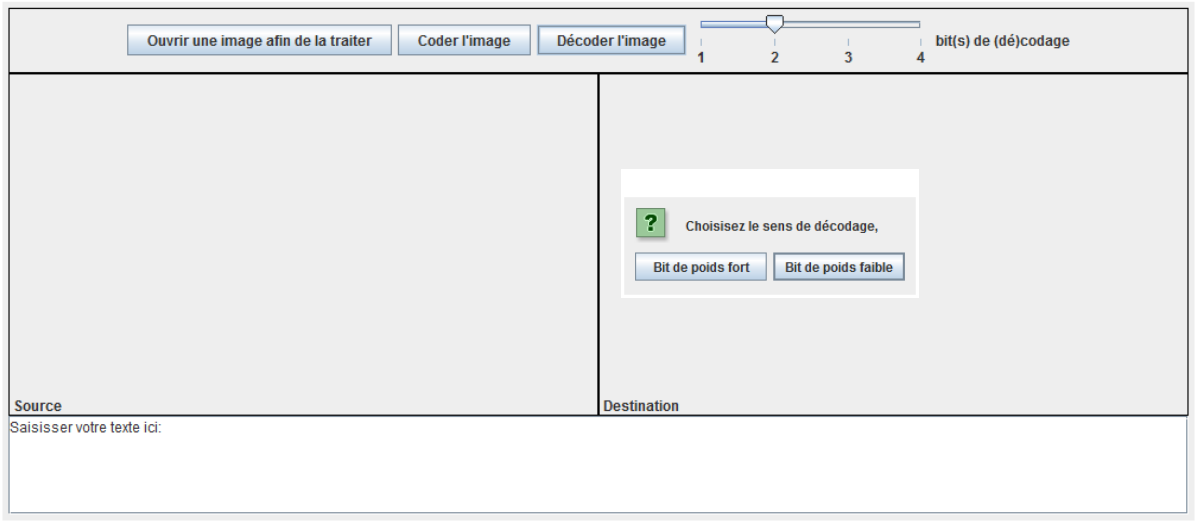
Une fois terminé, nous avons mis l'application, le codage et le décodage en commun.

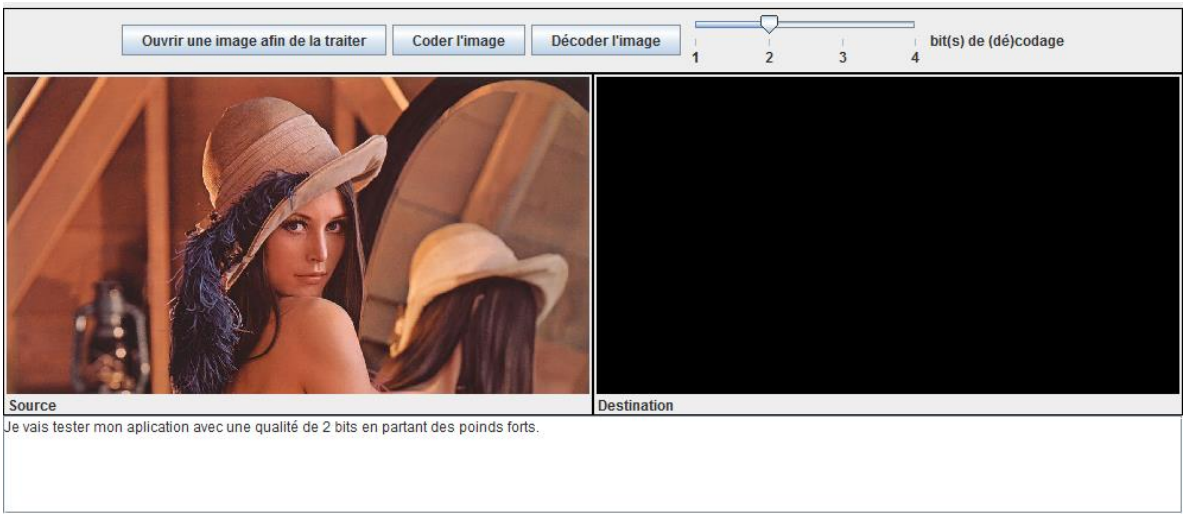
Production finale











Voilà une démonstration de notre application, qui répond au cahier des charges et permet de coder et décoder n'importe quelle image. Contrairement à ce que nous avions fait au départ car notre application traitait aussi la composante transparente, ce qui fût une source d'erreur car toutes les images ne la possèdent pas.

Bilan et perspective

Ce projet m'a été vraiment très instructif, en effet j'ai appris non seulement comment fonctionne la stéganographie mais aussi certaines erreurs typiques en informatique. Par exemple il ne faut pas créer des codes trop longs, car d'une part l'application va être ralentie et d'autre part le débuggage est beaucoup plus difficile. Une autre erreur est de ne pas commencer par les poids faibles, les traitements car on doit dans ce cas fonctionner à l'envers. Enfin ce projet m'a aussi permis d'apprendre à travailler en groupe car si nous avions fait chacun quelque chose de notre côté, nous n'aurions pas été en mesure de trouver certaines erreurs ou de mettre en commun nos productions.

Un projet peut toujours être amélioré, voici différentes pistes :

- Coder à partir de n'importe quel endroit dans l'image

- Renommer et enregistrer où on le souhaite l'image codée

Annexe

```
        //---------------------------------Steganographie-------------------------
-------

        for (c=0;c<=14;c++) // On parcourt tous les caractères
        {

            int bit= new Byte(bytes[c]).intValue(); // On met la valeur receuilli
par getBytes dans un int
            for(i=0;i<=7;i++) { // On extrait chaques bits et on le mets dans un
tableau

                if (i==0){
                bits[i] = (bit & 0x80)>>7;
                }
                else if (i==1){
                bits[i] = (bit & 0x40)>>6;
                }
                else if (i==2) {
                bits[i] = (bit & 0x20)>>5;
                }
                else if (i==3){
                bits[i]= (bit & 0x10)>>4;
                }
                else if (i==4){
                bits[i] = (bit & 0x8)>>3;
                }
                else if (i==5){
                bits[i] = (bit & 0x4)>>2;
                }
                else if (i==6){
                bits[i]= (bit & 0x2)>>1;
                }
                else {
                bits[i] = (bit & 0x1)>>0;
                }

                System.out.println("BIT CODE "+ bits[i]);




            int compoT = (rgbs[y*w+x]&(0xFF000000))>>24;
            // char essai=(char)compoT;
            int compoR = (rgbs[y*w+x]&(0x00FF0000))>>16;
            int compoV = (rgbs[y*w+x]&(0x0000FF00))>>8;
            int compoB = (rgbs[y*w+x]&(0x000000FF))>>0;

            System.out.println(rgbs[y*w+x]);
            System.out.println(compoR);
            System.out.println(essai);


            for (a=0;a<=7;a++){// On met sous forme de bits, les composantes dans
des tableaux
                    if (a==0){
                        compT[a] = (compoT & 0x80)>>7;
                        compR[a] = (compoR & 0x80)>>7;
                        compV[a] = (compoV & 0x80)>>7;
                        compB[a] = (compoB & 0x80)>>7;
                    }
                    else if (a==1){
                        compT[a] = (compoT & 0x40)>>6;
                        compR[a] = (compoR & 0x40)>>6;
                        compV[a] = (compoV & 0x40)>>6;
                        compB[a] = (compoB & 0x40)>>6;
                    }
                    else if (a==2) {
                        compT[a] = (compoT & 0x20)>>5;
                        compR[a] = (compoR & 0x20)>>5;
                        compV[a] = (compoV & 0x20)>>5;
                        compB[a] = (compoB & 0x20)>>5;
                    }
                    else if (a==3){
                        compT[a]=  (compoT & 0x10)>>4;
                        compR[a]=  (compoR & 0x10)>>4;
                        compV[a]=  (compoV & 0x10)>>4;
                        compB[a]=  (compoB & 0x10)>>4;
                    }
                    else if (a==4){
                        compT[a] = (compoT & 0x8)>>3;
                        compR[a] = (compoR & 0x8)>>3;
                        compV[a] = (compoV & 0x8)>>3;
                        compB[a] = (compoB & 0x8)>>3;
                    }
                    else if (a==5){
                        compT[a] = (compoT & 0x4)>>2;
                        compR[a] = (compoR & 0x4)>>2;
                        compV[a] = (compoV & 0x4)>>2;
                        compB[a] = (compoB & 0x4)>>2;
                    }
                    else if (a==6){
                        compT[a]= (compoT & 0x2)>>1;
                        compR[a]= (compoR & 0x2)>>1;
                        compV[a]= (compoV & 0x2)>>1;
                        compB[a]= (compoB & 0x2)>>1;
                    }
                    else {
                        compT[a] = (compoT & 0x1)>>0;
                        compR[a] = (compoR & 0x1)>>0;
                        compV[a] = (compoV & 0x1)>>0;
                        compB[a] = (compoB & 0x1)>>0;
                    }
                }

                if (j<=nbits) {        // On change le bit d'une composante
par celui de notre message
                    compT[8-nbits+m]=bits[i];
                    j++;
                    m++;
                    System.out.println(compT[i]);

                        if (j==nbits)
                        {
                            m=0;
                        }

                }

                else if (j>nbits && j <= (2*nbits)){
                compR[8-nbits+m]=bits[i];
                j++;
                m++;

                        if (j==nbits*2)
                        {
                            m=0;
                        }
                }
                else if (j>nbits*2 && j <= (3*nbits)){
                compV[8-nbits+m]=bits[i];
                j++;
                m++;

                        if (j==nbits*3)
                        {
                            m=0;
                        }
                }
                else
                {
                compB[8-nbits+m]=bits[i];
                j++;
                m++;


                        if (j==nbits*4)
                        {
                            m=0;
                            j=0;

                            System.out.println(compT[0]);
                            System.out.println(compT[1]);
                            System.out.println(compT[2]);
                            System.out.println(compT[3]);


                            for (int k=0; k<7;k++){ // On remet sous la
forme d'un int notre composante
                                valeurT= valeurT & (compT[k]<<(7-k));
                                valeurR= valeurR & (compR[k]<<(7-k));
                                valeurV= valeurV & (compV[k]<<(7-k));
                                valeurB= valeurB & (compB[k]<<(7-k));
                                }

                               System.out.println(valeurT);
                                System.out.println(valeurR);
                                System.out.println(valeurV);
                                System.out.println(valeurB);

                                newrgbs[y*w+x]= (valeurT<<24) |
(valeurR<<16) | (valeurV<<8) | (valeurB);

                                                                valeurT= 0;
                                valeurR= 0;
                                valeurV= 0;
                                valeurB= 0;

                                        x++;

                                        if (x==w){
                                            y++;
                                            x=0;
                                        }
                        }
                }
            }
        }
```

Figure 1

```
bit= (carac & (1<<(7-i)))>>(7-i);
```

Figure 2

```
if (bit==1){
masque = 1<<(8*numcomp-1-numbits);
comp = comp | masque;
}
```

Figure 3

```
else{
masque = ~(1<<(8*numcomp-1-numbits));
comp = comp & masque;
}
```

Figure 4

```
    if (numbits==7){
        numbits = 8 - nbits;

        if (numcomp==1){
            numcomp=4;
            newrgbs[y*w+x]=comp;
            x++;

                    if (x==w){
                        y++;
                        x=0;
                    }
             comp = rgbs[y*w+x];
           }
        else {
            numcomp --;
        }
    }
    else {
        numbits ++;
    }
```

Figure 5

```
  if (numcomp!=1){
      newrgbs[y*w+x]=comp;
  }
```

Figure 6

```
if(bit_carac>=0){
        carac=(char) (carac+(char)(bit<<bit_carac));
        bit_carac--;


}
if (bit_carac < 0) {

    texte = texte + carac;
```

Figure 7

```
masque=   1<< (8*numcomp-numbit-1);
brecup= rgbs[y*w+x] & masque;
if (brecup==0)
 bit=0;
else
 bit=1;
```

Figure 8

```
carac4=carac3;
carac3=carac2;
carac2=carac1;
carac1=carac;


test=""+carac4 +carac3 + carac2 + carac1;


if (test.equals("!3wZ")){
    numbit=9;
    numcomp=0-1;
    x=w;
    y=h;
```

Figure 9

```
int taille_text=texte.length();

  for (int i=0;i<taille_text-4;i++){
      char caracfinal = texte.charAt(i);
      textefinal = textefinal + caracfinal;
  }
```

Figure 10