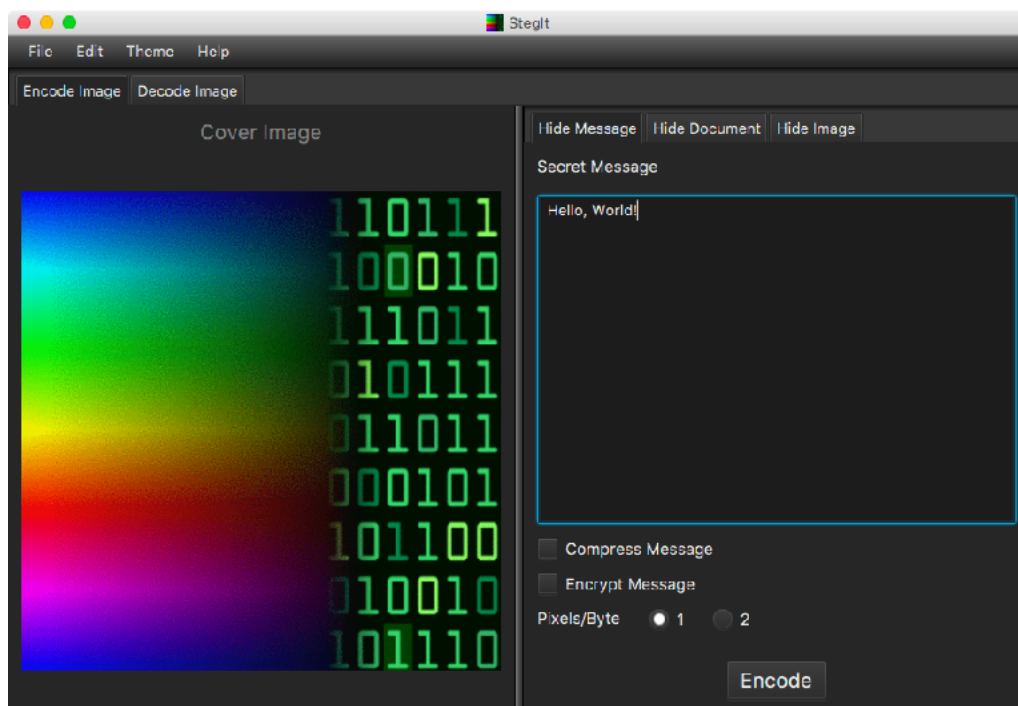


# Projet : Réalisation d'une application de steganographie.

---

- Projet d'été ENSI 2018.
- Réalisé par : Feres Gaaloul, Ilyes Hamrouni.
- Proposé par : Mehdi Hajji.
- Environnement : Java (Version 9.0.4) + JavaFX8 (Interface Graphique)
- Contenu :
  - StegIt.zip (Contient le fichier executable .jar)
  - src.zip (Code source)
  - documentation (Dossier contenant la documentation JavaDoc relative au projet (allez a index.html pour naviguer cette documentation))
  - Package Steganography.png : Diagramme de classe du projet (Pour faciliter la comprehension des différentes classes du projet et leurs relation)
  - Algorithme.pdf : Explication du déroulement de l'application et des algorithmes utilisés.



## **Presentation de l'application:**

StegIt est une application développée en Java (v. 9.0.4) avec JavaFX pour le développement de l'interface graphique qui permet de dissimuler un message, document ou une image dans une autre image et de récupérer le contenu caché et utilise l'algorithme de bits de poids faible (least significant bit).

L'image couvrante peut être dans le format .png, jpg, bmp ou gif (dans le cas d'un gif, la dissimulation d'une image n'est pas supportée)

Le document à cacher peut être dans n'importe quel format de texte brut (.txt, .java, .cpp, .py, .html...) et .xml et même certains formats de texte riche (.rtf et certains .doc).

Les documents ou messages à cacher peuvent avoir une taille maximale de 16777215 octets (~ 16 Mb).

De plus, l'application offre à l'utilisateur le choix de compresser et/ou crypter (avec un mot de passe ou une autre image) le message ou document à cacher.

L'image à cacher peut être dans le format .png, .bmp ou .jpg.

## **Utilisation de l'application:**

Lors de l'ouverture de l'application, l'utilisateur est amené à choisir soit une image couvrante (New Cover Image) ou une image steganographique (New Steganographic Image). Le choix du document, image ou message à cacher est désactivé jusqu'à la sélection d'une image couvrante.

L'application est séparée en 2 onglets (soit pour encoder ou decoder une image).

L'onglet d'encodage est séparé en 2 parties : l'image couvrante et 3 onglets (cacher un message, un document ou une image).

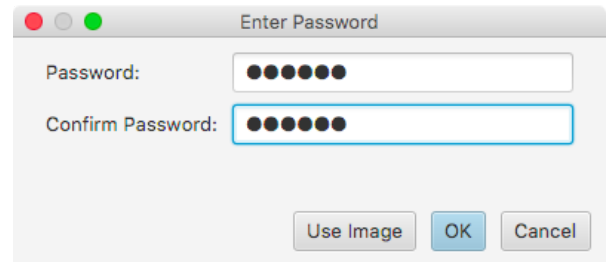
Le menu File permet le choix des différentes images et documents de l'application, ou de quitter l'application.

Le menu Edit permet d'interagir entre la zone de texte du message à cacher et le presse-papier du système.

Le menu Theme permet de basculer entre un thème clair ou sombre.

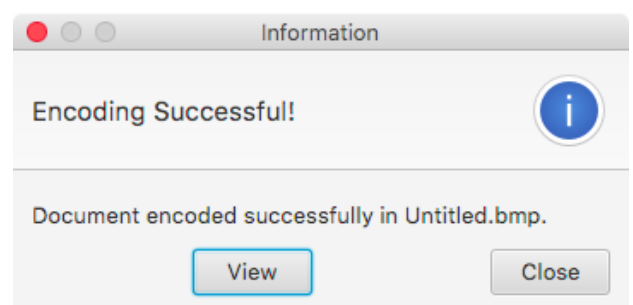
Le menu About permet d'obtenir des informations relatives à l'application.

Des cases à cocher sont disponibles dans les onglets message et document afin de compresser le contenu à cacher et/ou crypter le contenu. la case de cryptage ouvre une fenêtre qui permet de valider un mot de passe ou choisir une image comme mot de passe.

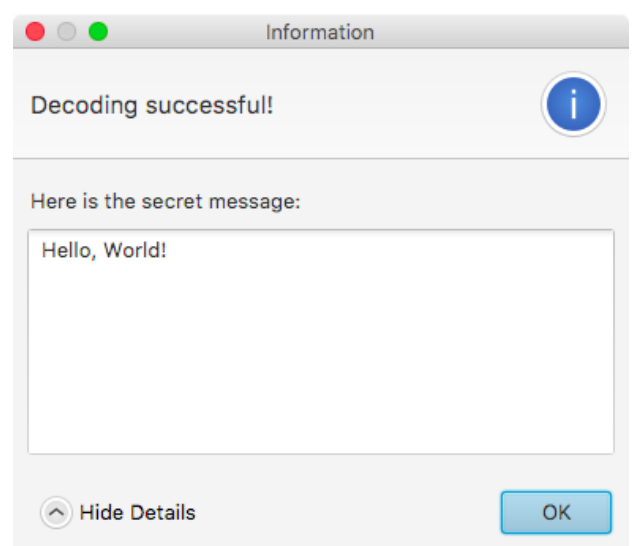


Des boutons radio sont disponibles dans les 3 types d'encodage pour choisir le nombre de pixels par octet (ou pixels par pixel). Dans le cas d'une image, il permet de choisir entre une meilleure qualité ou une meilleure capacité.

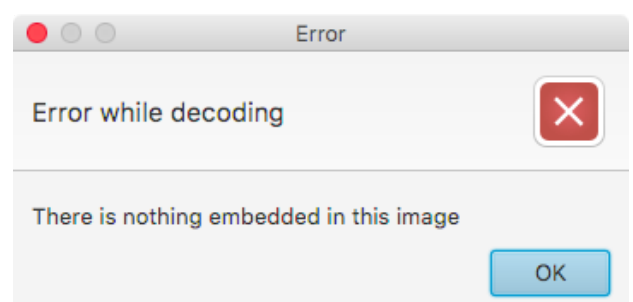
Ensuite, le bouton Encode permet de choisir la destination de l'image encodée. Si l'opération est réussie, une fenêtre nous informera de cela et le bouton View nous permet de visualiser la nouvelle image dans l'application par défaut.



Dans l'onglet de décodage et suite au choix de l'image, le bouton Decode permet de choisir la destination du contenu caché (s'il s'agit d'un document ou d'une image). Une fenêtre s'affichera pour choisir un mot de passe ou une image si le contenu est chiffré. Si l'opération est réussie, une fenêtre nous permet de visualiser le message caché. Si le contenu caché est un document ou une image, le bouton View permettra de le visualiser dans l'application par défaut.

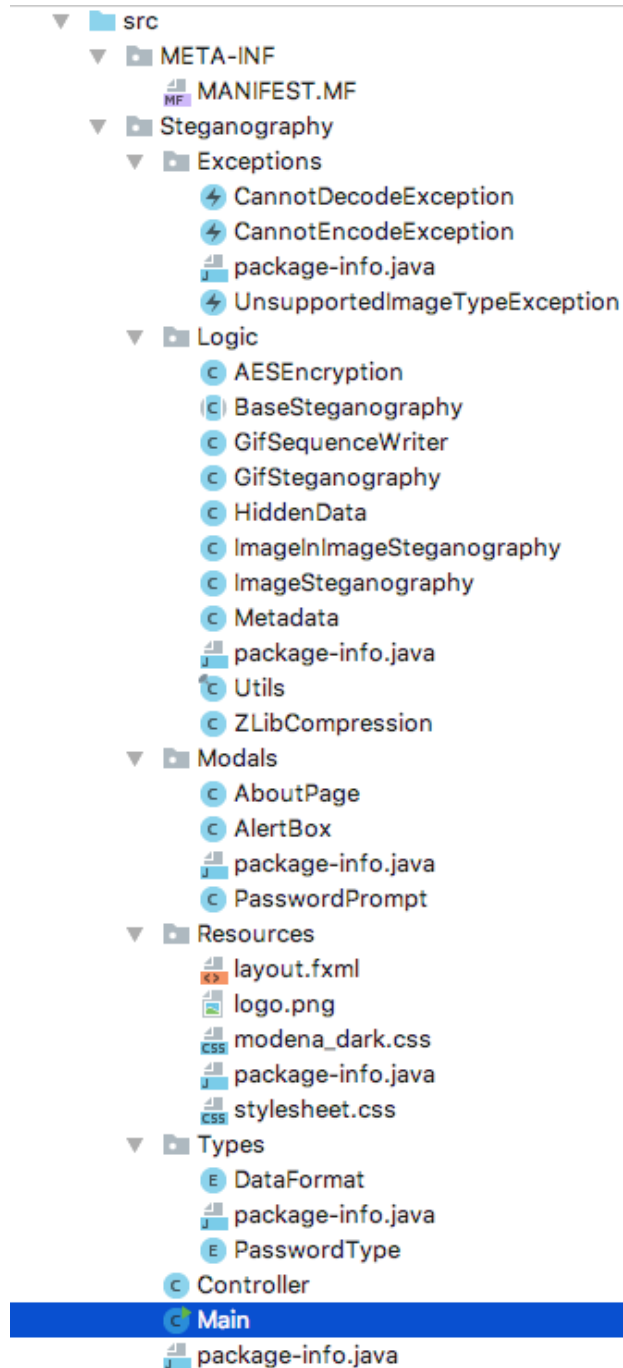


Une fenêtre d'alerte sera affiché si une erreur a été commise lors de l'opération d'encodage ou de décodage.



# Structure du projet:

Le projet est contenu dans le package Steganography:



## • Steganography:

- Main.java : démarre l'application.
- Controller.java : Contrôle l'interface graphique principale.

## • Steganography.Logic: Algorithmes de l'application.

- BaseSteganography.java, ImageSteganography.java, GifSteganography.java, ImageInImage.java: Contrôle le processus de steganographie.
- GifSequenceWriter.java, Metadata.java : nécessaires pour recréer une image gif.
- HiddenData.java: Décrit un fichier/message/ image caché.
- AESEncryption.java: Contrôle le (de)chiffrement.
- ZLibCompression.java: Contrôle la (de)compression.
- Utils.java: Méthodes divers.

## • Steganography.Modals: Fenêtres pop-up.

- PasswordPrompt.java: saisie mot de passe.
- AboutPage.java: Page à propos.
- AlertBox.java: Alertes de succès/erreur.

## • Steganography.Exceptions: Exceptions spécifiques à l'application.

- UnsupportedImageTypeException.java, CannotEncodeException.java, CannotDecodeException.java.

- **Steganography.Types:** Nouveaux types (enums) nécessaires à l'application.
- **Steganography.Resources:** layout, stylesheet et logo de l'application.




Le diagramme de classe dans Package Steganography.png détaille les relations entre les classes (dépendances, héritage).

## Algorithmes utilisés:

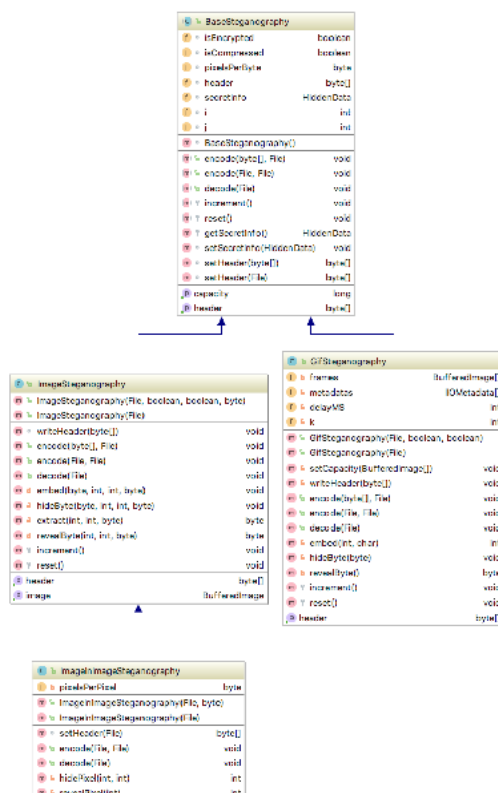
### Steganographie:

L'application repose principalement sur l'algorithme de Least Significant Bit (LSB).

Il s'agit de dissimuler le contenu à cacher dans les bits de poids faible des composants red, green et blue de chaque pixel d'une image. Ce processus est invisible à l'œil nu comme démontré ci-dessous:

Color (Green)	Base 10	Binary	Change
	238	11101110	+3
	235	11101011 (base)	
	232	11101000	-3

Les classes qui permettent l'opération d'encodage/décodage sont:



Les classes ImageSteganography et GifSteganography héritent de la classe abstraite BaseSteganography et implémentent certaines de ses fonctions.

La classe ImageInImageSteganography hérite de ImageSteganography et assure le processus de steganographie pour l'image dans une image.

Ces classes encodent un header contenant des informations relatives au contenu à cacher puis l'image.

Le header prend l'une de ses formes:

**Message:** ['M', Encryption ('E' | 'C'), Compression ('C' | 'U'), Pixels/Byte (1|2), Message length (3 bits), '!']

**Exemple:** [01001101, 01000100, 01010101, 00000001, 00000000, 00110011, 11001100, 00100001]

Corresponds à: ['M' (Message), 'E' (isEncrypted), 'U' (!isCompressed), 1 Pixel/Byte, 13260 bytes, '!']

**Document:** ['D', Encryption ('E' | 'C'), Compression ('C' | 'U'), Pixels/Byte (1|2), File length (3 bits), File extension, '!']

Exemple: [01000100, 01000101, 01000011, 00000010, 01010101, 10101010, 01010101, 01001010, 01000001, 01010110, 01000001, 00100001]

Corresponds à: ['D' (Document), 'E' (isEncrypted), 'C' (isCompressed), 2 Pixel/Byte, 5614165 bytes, .java extension, '!']

**Image:** ['I', pixels/pixel (1|2), Width (2 bytes), Height (2 bytes), '!']

Exemple: [01001001, 00000010, 00000101, 00000000, 00000010, 11010000, 00100001]

Corresponds to: ['I' (Image), 2, 1280, 720, '!']

## Algorithme d'encodage d'un message/fichier dans une image:

Au préambule, le message sera compressé et/ou crypté selon le choix de l'utilisateur dans cet ordre.

### Algorithme d'encodage d'un message/fichier dans une image 24 bit:

- 1) On crée une copie de l'image dans un BufferedImage.
- 2) On lit le message/fichier byte par byte.
- 3) On crée un tableau de byte contenant le header et on l'insère dans les premiers pixels de l'image.
- 4) Les 8 bits de chaque octet du fichier seront dissimulés de la manière suivante:
  - 3 bits de poids faible du composant rouge, 2 bits de poids faible du composant vert et 3 bits de poids faible du composant bleu du pixel (i, j) si l'utilisateur choisit 1 pixel/byte (Deviation maximale : 3 - 7).
  - 1 bit de poids faible du composant rouge, 2 bits de poids faible du composant vert et 1 bit de poids faible dans le composant bleu. Les 4 bits de poids fort seront dissimulés dans le pixel (i, j) et les 4 bits de poids faible dans le pixel (hauteur -i, largeur -j) (Deviation maximale: 1 - 3).
- 4) On enregistre l'image résultante.

### **Algorithme d'encodage d'un message/fichier dans une image gif:**

- 1) On récupère chaque image composant le gif dans un tableau de BufferedImage.
- 2) On récupère les Metadata de chaque image du gif dans un tableau de IIOMetadata. Il contiendra des informations comme le délai, les offsets, le colormap...
- 3) On crée un tableau de byte contenant le header et on l'insère dans les premiers pixels de l'image.
- 4) Les 8 bits de chaque octet du fichier seront dissimulés de la manière suivante:
  - comme les images gif ne contiennent que 255 couleurs, on dissimule chaque octet dans 8 pixels consécutifs (on ne change que le bit de poids faible).
- 5) Si l'on atteint la fin d'une image, on passe à l'image suivante.
- 6) On enregistre un nouveau .gif à partir du tableau de BufferedImage altéré et du tableau de IIOMetadata en utilisant GifSequenceWriter.

### **Algorithme d'encodage d'une image dans une image:**

- 1) On transforme l'image à cacher et l'image couvrante en BufferedImage.
- 2) On crée un tableau de byte contenant le header et on l'insère dans les premiers pixels de l'image.
- 3) à partir de la ligne suivante, on dissimule l'image de cette manière:
  - Si l'utilisateur choisit 1 pixel/pixel (Capacité maximale), les 3 bits de poids fort du pixel (i, j) de l'image à cacher seront dissimulés dans les 3 bits de poids faible du pixel (i+offset, j) de l'image couvrante.
  - Si l'utilisateur choisit 2 pixels/pixels, les 3 prochains bit (4 -> 6) seront dissimulés dans les 3 bits de poids faible du pixel (hauteur-i-offset, largeur-j).
- 4) On enregistre l'image résultante contenant l'image cachée.

## **Algorithme de décodage d'un message/fichier dans une image:**

### **Algorithme de décodage d'un message/fichier dans une image 24 bit:**

- 1) On crée une copie de l'image dans un BufferedImage.
  - 2) On récupère le header du fichier en récupérant l'octet dissimulé dans 3R,2G,3B jusqu'à atteindre '!'.
  - 3) A partir du header, on récupère le nombre de pixels/byte et la longueur du message.
  - 4) On reconstruit le fichier de cette manière:
    - Si le nombre de pixels/byte dans le header est égal à 1, on récupère chaque byte à partir des bits de poids faible 3R,2G,3B du pixel (i, j).
    - Si le nombre de pixels/byte est égal à 2, les 4 bits de poids fort seront dans les bits de poids faible 1R,2G,1B du pixel (i,j) et les 4 bits de poids faible seront dans les bits de poids faible 1R,2G,1B du pixel (hauteur-i, largeur-j)
- Jusqu'à atteindre la fin du message obtenue à partir du header.
- 5) On reconstruit le fichier en utilisant l'extension adéquate se trouvant dans le header.

### **Algorithme de décodage d'un message/fichier dans une image gif:**

- 1) On crée un tableau de BufferedImage à partir du gif.
- 2) On récupère le header du fichier/message jusqu'à atteindre '!'.
- 3) On récupère la taille du fichier/message à partir du header.
- 4) On récupère chaque octet du message jusqu'à atteindre sa longueur.

L'opération de récupération de chaque bit consiste à rassembler le bit de poids faible de chaque 8 bits consécutifs.

- 5) On reconstruit le fichier en utilisant l'extension adéquate se trouvant dans le header.



Dans les 2 cas, suite à l'opération de décodage seront déclenchés les opérations de décryptage et/ou décompression selon le choix lors de l'encodage.

### **Algorithme de décodage d'une image dans une image:**

- 1) On crée une copie de l'image dans une BufferedImage ainsi qu'un nouveau BufferedImage.
- 2) On récupère le header à partir des premiers pixels de l'image pour récupérer la longueur et la largeur de l'image cachée ainsi que le nombre de pixels/pixel.
- 3) On reconstruit l'image pixel par pixel de cette manière:
  - Si le nombre de pixels/pixels est 1, les 3 bits de poids fort de chaque couleur du pixels (i, j) seront les 3 bits de poids faible de chaque couleur du pixel (i+offset, j) de l'image steganographique.
  - Si le nombre de pixels/pixel est 2, les bits de poids (4 -> 6) de chaque couleur du pixel (i, j) seront les 3 bits de poids faible de chaque couleur du pixel (i+offset, j) de l'image steganographique.
- 4) On enregistre la nouvelle image (au format .png).

### **Algorithme de compression:**

On utilise la bibliothèque ZLib de compression dans Java. (Inflater et Deflator).

### **Algorithme de cryptage:**

On utilise l'algorithme de chiffrement symétrique AES (Advanced Encryption Standard) en mode CBC (Cipher Block Chaining) 128 bit.

Le mot de passe de cryptage sera soit tapé par l'utilisateur, soit une image (dans ce cas, le mot de passe sera la somme des pixels diagonaux).

Dans les 2 cas, le mot de passe passera par la fonction de hachage SHA (Secure Hash Algorithm) pour convenir aux besoins du chiffrement (clé 128 bit).

**Fin.**